RobustGait: Robustness Analysis for Appearance Based Gait Recognition (Supplementary)

We provide additional results, implementation details, and expanded tables for various experiments discussed in the main paper. It includes results across different parsing models, perturbation robustness analysis, and further implementation specifics.

- **Section 1**: Discussion on ethical issues and broader impact of the work.
- **Section 2**: Technical details on benchmark setup including segmentation models, training configuration, and evaluation protocol.
- **Section 3**: Limitations of our study.
- **Section 4**: Discussion on societal and ethical implications of our study.
- **Section 5**: Additional results across parsing models and robustness scores on multiple severity levels and noisy training.
- **Section 6**: Training details of proposed approach.
- **Section 7**: Gait model evaluation and training results on the large-scale MEVID benchmark.
- **Section 8**: Implementation specifics and severity schedules for the 15 corruption types in our benchmark.

1. Discussion

Ethical issues: The datasets used in this study: CASIA-B, CCPG, SUSTech1K, and MEVID, are publicly available benchmark datasets, each of which has addressed ethical considerations in their respective publications. Our use of these datasets strictly follows their intended research purposes.
Broader impact: By investigating robustness in real-world surveillance scenarios, this work highlights both opportunities and risks. While the proposed techniques improve the reliability of gait recognition systems, they may also accelerate broader deployment in contexts where privacy, consent, and ethical considerations are critical. As with any biometric technology, careful evaluation of use cases and their societal implications is essential to ensure responsible adoption.

2. Benchmark Details

Segmentation Setup: We use five parsing models for silhouette extraction. SCHP and M2FP both use a ResNet101 backbone; M2FP is initialized with CIHP-pretrained weights. CDGNet also uses a ResNet101 backbone, initialized with LIP-pretrained weights. STVC is configured with a ResNet18 backbone and stride-8 settings. GSAM is included as a foundation segmentation model. All outputs are converted to binary silhouettes for downstream use.
Gait Recognition Setup: We adopt the OpenGait framework and default hyperparameters to train six models: GaitBase, GaitGL, DeepGait, SwinGait, GaitPart, and GaitSet. Input silhouettes are resized to 64×44 with 30-frame sequences (10 for DeepGait on SUSTech1K). Training uses $P \times K$ sampling, standard triplet + cross-entropy loss, and optimizers chosen per model: SGD for DeepGait/GaitBase/GaitSet, Adam for GaitGL/GaitPart, and weighted Adam for SwinGait. Frame skipping is applied as per the original model configurations.
Implementation Details: All experiments were conducted using 2 Tesla V100-PCIE-32GB GPUs (CUDA 12.4). Our implementation is based on the OpenGait repository: `https://github.com/ShiqiYu/OpenGait`
Evaluation Protocol: Gait recognition models are evaluated using a standard probe-gallery setup, where the gallery set contains reference sequences for each identity, and the probe set contains query sequences to be matched against the gallery. The goal is to correctly identify each probe sequence by retrieving the most similar sequence from the gallery. For the **CASIA-B** dataset, the gallery includes sequences captured under the normal walking condition: `nm-01`, `nm-02`, `nm-03`, and `nm-04`. The probe set comprises sequences from varying conditions, including `nm-05`, `nm-06`, `bg-01`, `bg-02`, `cl-01`, and `cl-02`. In the **CCPG** dataset, the gallery consists of sequences `U1_D1`, `U2_D2`, `U3_D3`, `U0_D3`, and `U0_D0`. The probe set includes `U0_D0_BG`, `U0_D0`, `U3_D3`, `U1_D0`, and another instance of `U0_D0_BG` under different conditions. For the **SUSTech1K** dataset, the gallery contains the sequence `00-nm`, while the probe set includes a diverse set of variations: `01-nm`, `bg`, `cl`, `cr`, `ub`, `uf`, `oc`, `nt`, and additional sequences labeled `01` through `04`.

3. Limitations

Due to differences in generalization, not all parsing models were applicable across all three datasets (CASIA-B, CCPG, and SUSTech1K), leading to selective parser usage in certain evaluations. The study focuses on silhouette-based gait recognition, leaving out pose and depth-based methods. At higher noise severities, the data becomes heavily degraded, making meaningful evaluation on those data challenging. Additionally, the computational cost of evaluating multiple parsing and recognition models across datasets and perturbation settings constrained the depth and exhaustiveness of some analyses presented in this work.

4. Broader Impact

This work highlights the need for robust gait recognition systems for practical deployment by systematically evaluating model performance under real-world corruptions such as occlusions, lighting changes, and sensor noise. It demonstrates that parsing and recognition models vary significantly in robustness, encouraging more transparent benchmarking. The released dataset support future research in robust and generalizable biometric representation learning.

While the proposed techniques improve robustness, they may also facilitate broader deployment of gait recognition systems in real-world settings, including those where ethical, privacy, or consent considerations are important. As with any biometric technology, thoughtful evaluation of use cases and societal implications is essential to ensure responsible use.

5. Additional Results

Results on Different Silhouette Extraction Models: Silhouette quality plays an important role in the performance of gait recognition models. We evaluate the impact of different human parsing models on gait recognition across three datasets: CASIA-B, CCPG, and SUSTech1K. Results for each combination of parsing method and gait recognition model are shown in Tables 1, 2, and 3.

Robustness Analysis of Gait Recogniton Models under noises: To assess the robustness of modern gait recognition models, we evaluate performance across three benchmark datasets (CASIA-B, CCPG, SUSTech1K) under four broad perturbation categories. For each model and dataset, we compute both absolute robustness (δ_a) and relative robustness (δ_r) as defined in the main paper. Table 4 reports results on the CASIA-B dataset. Table 5 and Table 6 show the corresponding results for CCPG and SUSTech1K, respectively.

Additional results for noises: Tables 7, 8, 9, 10, 11, 12, 13, 14 show performance results of gait models under the different noise types at each of the 5 severity levels.

Choice of Segmentation Model We use the SCHP model for our analysis as it offers the best trade-off in computational efficiency compared to other segmentation and parsing backbones.

Analysis on Noise Severity We show the detailed analysis on how severity of noise impacts silhouettes and model features.

Robustness Analysis with Noisy Gallery: We construct a *fixed noisy gallery* by applying one of the 15 corruption types to each gallery sequence, with severity levels sampled randomly using the probabilities 0.6 (severity 1), 0.3 (severity 2), and 0.1 (severity 3). This noisy gallery is held constant across evaluations. We then evaluate each gait recognition model using 15 perturbed probe sets—each corresponding to one of the 15 corruption types—against this shared noisy gallery. Table 16 reports the absolute (δ_a) and relative (δ_r) robustness scores for each model on the CASIA-B dataset across four corruption categories.

Robustness via Noise-Aware Training Details: We create a perturbed training set with five representative corruption types, covering camera viewpoint changes, temporal distortions, environmental artifacts, and occlusions. These perturbations are applied with severity levels 1, 2, and 3, sampled according to a probability distribution of 0.6, 0.3, and 0.1, respectively. The remaining ten corruption types, which are unseen during training, are used to generate the noisy test set for evaluation. We train gait recognition models with varying ratios of clean and noisy training data (i.e., 100:0, 80:20, 50:50, and 20:80), and evaluate them on both the original clean test set and the noisy test set. The performance under these settings is summarized in Table 17 and Table 18.

6. Training Details: Efficient Distillation

We implement a two-stream distillation framework to improve robustness against noise in gait embeddings. The training procedure involves a fixed teacher network and a learnable student network. The teacher is applied only to clean sequences, while the student is trained on both clean and noisy inputs. We use a contrastive loss between the teacher's embedding (extracted from clean inputs) and the student's embeddings (from both clean and noisy sequences) to align the representational space. Let $E_T(x)$ and $E_S(x)$ denote the embeddings from the teacher and student, respectively. The contrastive losses are defined as:

$$\begin{aligned} \mathcal{L}_{\text{con}}^{\text{clean}} &= \text{Con}(E_T(x_{\text{clean}}), E_S(x_{\text{clean}})), \\ \mathcal{L}_{\text{con}}^{\text{noisy}} &= \text{Con}(E_T(x_{\text{clean}}), E_S(x_{\text{noisy}})). \end{aligned} \tag{1}$$

where $\text{Con}(\cdot,\cdot)$ denotes a normalized temperature-scaled contrastive loss.

Additionally, the student is trained with softmax and triplet losses using both clean and noisy inputs. The softmax losses are given by

$$\mathcal{L}_{\text{softmax}}^{\text{clean}} = \text{CE}(\mathbf{z}_S^{\text{clean}}, y), \tag{2}$$

$$\mathcal{L}_{\text{softmax}}^{\text{noisy}} = \text{CE}(\mathbf{z}_S^{\text{noisy}}, y). \tag{3}$$

	Baseline[ICPR06]			**SCHP**[TPAMI20]			**M2FP**[arXiv23]			**CDGNet**[CVPR22]			**GSAM**[arXiv24]		
IoU	1.00			0.63			0.69			0.58			0.62		
Method	NM	BG	CL	NM	BG	CL	NM	BG	CL	NM	BG	CL	NM	BG	CL
DeepGaitV2 [arXiv23]	97.3	93.8	**78.5**	95.0	89.8	74.6	**97.8**	**93.9**	75.4	88.8	74.6	48.2	94.0	87.3	69.5
GaitGL[ICCV21]	97.4	94.5	**83.6**	93.2	88.1	77.4	**97.6**	**94.7**	**83.6**	83.5	73.3	53.0	92.6	86.6	75.1
GaitBase[CVPR23]	97.6	94.0	77.4	93.6	88.4	72.8	**98.1**	**95.2**	**77.9**	89.3	76.7	49.9	96.0	89.9	74.7
SwinGait[arXiv23]	94.0	87.1	64.4	88.8	80.8	59.2	93.7	85.9	58.3	82.7	64.9	35.7	89.6	79.9	54.8
GaitPart[CVPR20]	**96.2**	**90.6**	78.2	92.5	85.9	73.3	96.0	**90.6**	**76.6**	79.4	65.9	45.2	91.9	81.8	70.8
GaitSet[AAAI19]	95.6	90.2	74.8	91.7	83.2	68.6	**96.5**	**90.8**	**75.2**	80.9	65.6	41.8	92.9	80.8	68.9

Table 1. Comparison of six gait recognition models under different silhouette extraction methods on CASIA-B. Results are reported under three conditions: NM (normal), BG (bag), and CL (clothing). The best result per row and condition is highlighted in bold.

	Baseline[CVPR23]				**SCHP**[TPAMI20]				**M2FP**[arXiv23]				**CDGNet**[CVPR22]				**GSAM**[arXiv24]			
IoU	1.00				0.96				0.83				0.87				0.88			
Method	CL	UP	DN	BG	CL	UP	DN	BG	CL	UP	DN	BG	CL	UP	DN	BG	CL	UP	DN	BG
DeepGait[arXiv23]	**79.2**	**85.0**	**81.3**	**90.0**	78.9	84.8	81.0	89.7	70.8	76.6	76.0	85.8	67.5	75.8	74.7	84.7	71.3	78.5	77.0	83.6
GaitGL[ICCV21]	**61.8**	**68.1**	**64.6**	**70.2**	61.7	67.7	62.7	70.1	53.9	61.0	59.5	63.8	49.8	56.3	53.8	61.2	55.9	63.6	57.7	66.9
GaitBase[CVPR23]	72.1	75.3	77.2	78.7	**73.8**	**76.9**	**78.1**	**80.7**	70.8	73.1	76.9	75.9	60.6	64.8	68.4	69.3	64.2	69.6	72.6	73.1
SwinGait[arXiv23]	**61.2**	**71.9**	**66.5**	81.5	**61.2**	71.4	66.0	**82.1**	54.0	63.4	64.6	77.0	49.5	61.8	60.1	75.9	55.0	65.4	64.1	76.0
GaitPart[CVPR20]	57.7	63.6	**62.8**	66.4	**58.1**	**63.7**	61.7	**66.7**	51.3	56.9	55.6	58.6	47.8	55.2	53.0	57.0	46.1	52.7	50.7	55.7
GaitSet[AAAI19]	**58.8**	**64.5**	**63.7**	**67.5**	59.2	63.7	**63.9**	**67.1**	50.8	55.8	56.7	58.0	46.4	52.7	53.8	55.1	51.5	57.7	57.0	59.2

Table 2. Comparison of six gait recognition models under different silhouette extraction methods on CCPG. Results are reported under four conditions: CL (full), UP (up), DN (down), and BG (bag). The best result per row and condition is highlighted in bold.

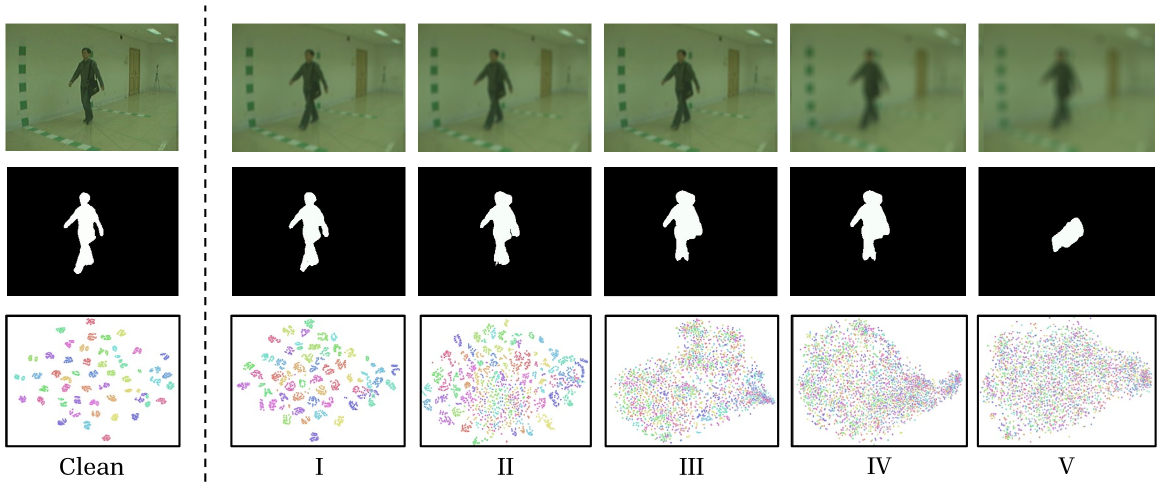

Figure 1. **Qualitative Analysis** of increasing digital noise severity on CASIA-B. *(Top)* Silhouettes from the parsing model degrade visibly with higher noise, losing structural integrity. *(Bottom)* t-SNE of DeepGait features shows reduced cluster separability, leading to weakened identity discrimination.

and the triplet losses are defined as

$$\mathcal{L}_{\text{triplet}}^{\text{clean}} = \text{Triplet}(E_S(x_{\text{clean}}), y), \tag{4}$$

$$\mathcal{L}_{\text{triplet}}^{\text{noisy}} = \text{Triplet}(E_S(x_{\text{noisy}}), y). \tag{5}$$

$$\begin{aligned}\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{con}}^{\text{clean}} + \lambda_2 \mathcal{L}_{\text{con}}^{\text{noisy}} + \lambda_3 \mathcal{L}_{\text{softmax}}^{\text{clean}} \\ + \lambda_4 \mathcal{L}_{\text{softmax}}^{\text{noisy}} + \lambda_5 \mathcal{L}_{\text{triplet}}^{\text{clean}} + \lambda_6 \mathcal{L}_{\text{triplet}}^{\text{noisy}}.\end{aligned} \tag{6}$$

During inference, only the student model is used.

	Baseline[CVPR23]				**SCHP**[TPAMI20]				**M2FP**[arXiv23]				**CDGNet**[CVPR22]				**GSAM**[arXiv24]			
IoU	1.00				0.85				0.99				0.84				0.83			
Method	NM	CL	UM	OVR	NM	CL	UM	OVR	NM	CL	UM	OVR	NM	CL	UM	OVR	NM	CL	UM	OVR
DeepGaitV2[arXiv23]	89.1	76.9	86.2	85.9	86.0	68.3	82.7	81.3	89.0	51.4	87.8	84.4	88.7	45.6	86.6	85.3	82.2	58.9	88.0	85.3
GaitBase[CVPR23]	80.4	62.9	74.9	77.7	88.1	77.3	83.7	83.6	88.7	56.4	84.7	83.6	86.4	49.6	80.2	81.5	81.2	58.2	83.9	82.8
SwinGait[arXiv23]	88.1	80.6	84.4	84.9	79.7	53.7	70.8	71.6	86.3	46.3	82.7	80.8	81.6	37.8	77.5	76.8	80.6	49.7	83.9	81.5
GaitGL[ICCV21]	83.6	70.2	76.9	78.8	85.0	78.2	81.5	80.6	87.5	74.8	83.8	84.4	81.2	58.2	83.9	82.8	75.7	56.7	79.5	77.3
GaitSet[AAAI19]	62.6	27.7	62.1	64.7	72.6	39.9	63.1	66.5	78.9	40.7	71.1	73.9	79.7	53.7	70.8	71.6	69.1	31.8	64.9	66.7
GaitPart[CVPR20]	63.7	29.8	61.3	64.8	69.6	46.8	62.5	65.4	76.5	47.5	69.1	72.5	67.0	29.9	58.2	63.3	63.7	34.7	62.9	65.9

Table 3. Comparison of six gait recognition models under different silhouette extraction methods on SUSTech1K. Results are reported under NM (normal), CL (clothing), UM (umbrella), and OVR (overall).

CASIAB[ICPR06]	**Camera**		**Temporal**		**Environmental**		**Occlusion**	
	δ_a	δ_r	δ_a	δ_r	δ_a	δ_r	δ_a	δ_r
DeepGaitV2[arXiv23]	0.42	**0.33**	0.66	0.61	0.67	**0.62**	0.63	**0.57**
GaitGL[ICCV21]	0.38	0.28	0.52	0.44	0.62	0.56	0.54	0.46
GaitBase[CVPR23]	0.37	0.26	0.72	0.67	0.63	0.56	0.56	0.49
GaitSet[AAAI19]	0.42	0.25	0.77	0.70	0.64	0.53	0.59	0.46
GaitPart[CVPR20]	0.40	0.27	0.76	0.70	0.63	0.55	0.57	0.47
SwinGait[arXiv23]	**0.49**	0.32	**0.81**	**0.75**	**0.71**	0.61	**0.67**	0.56

Table 4. Absolute (δ_a) and relative (δ_r) robustness scores of gait models on the CASIA-B dataset across different perturbation types. Higher is better.

CCPG[CVPR23]	**Camera**		**Temporal**		**Environmental**		**Occlusion**	
	δ_r	δ_a	δ_r	δ_a	δ_r	δ_a	δ_r	δ_a
DeepGaitV2[arXiv23]	0.63	0.55	0.61	0.68	0.71	0.76	**0.53**	0.61
GaitGL[ICCV21]	0.70	0.55	0.52	0.68	0.64	0.76	0.50	0.67
GaitBase[CVPR23]	0.65	0.54	0.61	0.69	0.65	0.73	0.50	0.61
GaitSet[AAAI19]	0.71	0.55	0.61	0.75	0.63	0.76	0.49	0.68
GaitPart[CVPR20]	**0.72**	0.54	0.58	0.74	0.62	0.76	0.48	**0.68**
SwinGait[arXiv23]	0.69	**0.56**	**0.69**	**0.78**	**0.72**	**0.80**	0.52	0.67

Table 5. Absolute (δ_a) and relative (δ_r) robustness scores of gait models on the CCPG dataset across different perturbation types.

SUSTech1k[CVPR23]	**Camera**		**Temporal**		**Environmental**		**Occlusion**	
	δ_r	δ_a	δ_r	δ_a	δ_r	δ_a	δ_r	δ_a
DeepGaitV2[arXiv23]	0.19	0.38	0.76	0.82	0.69	0.76	**0.34**	0.50
SwinGait[arXiv23]	0.19	**0.47**	0.83	0.89	**0.74**	**0.83**	0.28	**0.53**
GaitBase[CVPR23]	0.19	0.33	**0.89**	**0.91**	0.70	0.75	0.32	0.43

Table 6. Absolute (δ_a) and relative (δ_r) robustness scores of gait models on the SUSTech1k dataset across different perturbation types.

7. Scaling to Large Real-World Dataset MEVID

To assess model robustness and scalability in unconstrained settings, we evaluate on the MEVID dataset. MEVID is a large-scale video-based re-identification benchmark comprising 8,092 tracklets of 158 subjects recorded across 73 days in 33 camera views spanning 17 locations. Each subject appears in multiple sessions with varied clothing (598 outfits in total), motion styles, and environments (indoor/outdoor), making it well-suited for testing gait models under real-world challenges. The tracklets average 590 frames each and include significant variations in background clutter, occlusion, lighting, and viewpoints. MEVID ensures diversity in geography and activity context. We train each model from scratch on the MEVID training set using standard classifica-

Perturbations	Clean	Sev 1	Sev 2	Sev 3	Sev 4	Sev 5
gaussian_noise	86.5	72.5	35.0	2.8	2.4	0.0
defocus_blur	86.5	66.5	32.7	2.7	1.5	1.8
zoom_blur	86.5	35.3	19.4	8.5	4.1	2.6
impulse_noise	86.5	15.5	3.0	1.9	0.5	0.0
impulse_noise2	86.5	64.8	15.0	4.3	2.5	1.7
speckle_noise	86.5	81.9	75.7	63.1	44.0	25.2
shot_noise	86.5	85.3	81.5	69.9	45.8	6.4
zoom_in	86.5	83.5	77.4	57.5	33.8	23.7
freeze	86.5	77.3	62.4	45.7	30.1	44.9
sampling	86.5	84.4	69.7	48.0	36.4	25.2
low_light	86.5	86.1	86.6	86.6	86.5	86.5
rain	86.5	83.4	74.0	25.2	5.5	2.3
snow	86.5	72.1	69.3	66.5	68.8	72.1
fog	86.5	47.3	25.1	23.1	9.3	2.8
Static	86.5	73.5	65.6	54.6	36.4	14.9

Table 7. DeepGait robustness scores on CASIA-B dataset (rounded to one decimal place).

Perturbations	Clean	Sev 1	Sev 2	Sev 3	Sev 4	Sev 5
gaussian_noise	84.9	62.3	22.9	1.8	1.7	0.0
defocus_blur	84.9	54.6	17.9	2.2	1.4	1.7
zoom_blur	84.9	20.6	11.1	5.3	3.2	2.7
impulse_noise	84.9	10.0	1.8	2.1	0.0	0.0
impulse_noise2	84.9	54.1	10.1	2.8	1.6	1.5
speckle_noise	84.9	76.0	67.4	50.6	29.7	14.5
shot_noise	84.9	82.0	75.3	59.5	31.8	3.8
zoom_in	84.9	80.3	68.2	40.4	19.8	12.7
freeze	84.9	79.8	70.3	51.9	29.4	51.1
sampling	84.9	82.5	76.1	61.4	41.7	22.1
low_light	84.9	84.6	84.9	84.9	85.0	85.0
rain	84.9	78.4	65.2	17.0	4.9	1.5
snow	84.9	64.2	59.9	55.6	58.4	62.8
fog	84.9	31.6	13.8	12.6	5.1	2.3
Static	84.9	66.1	57.0	44.7	27.7	11.3

Table 8. GaitBase robustness scores on CASIA-B dataset (rounded to one decimal place).

tion and metric learning objectives.

8. Details on Corruptions

We present the details of each and every noises and how it is implemented.

Gaussian Noise The Gaussian Noise function introduces Gaussian-distributed noise to each frame in an array of video frames. The noise severity is determined by a predefined scale corresponding to different severity levels: 0.08, 0.12, 0.18, 0.26, and 0.38. Each frame is normalized to a [0, 1] range before noise addition. After applying the noise, the pixel values are clipped to ensure they remain within the [0, 1] range, and then the frames are rescaled back to [0, 255]. Finally, the processed frames are converted back to an unsigned 8-bit integer format.

Speckle Noise Each frame is normalized to a [0, 1] range before noise addition. Speckle noise is generated by multi-

Perturbations	Clean	Sev 1	Sev 2	Sev 3	Sev 4	Sev 5
gaussian_noise	75.4	61.6	30.9	3.0	2.2	0.0
defocus_blur	75.4	55.0	25.7	2.3	1.5	1.9
zoom_blur	75.4	29.3	16.0	6.6	3.6	2.4
impulse_noise	75.4	15.2	2.7	2.0	0.4	0.0
impulse_noise2	75.4	57.8	15.0	4.1	1.5	1.2
speckle_noise	75.4	70.5	65.7	54.6	39.2	23.9
shot_noise	75.4	73.7	70.3	60.5	40.1	7.2
zoom_in	75.4	72.0	66.1	49.3	28.6	20.5
freeze	75.4	70.2	63.1	53.0	39.6	52.8
sampling	75.4	74.0	68.9	57.6	49.6	35.8
low_light	75.4	75.0	75.3	75.3	75.3	75.3
rain	75.4	72.0	61.7	23.9	7.1	2.1
snow	75.4	59.6	57.4	55.9	58.1	60.8
fog	75.4	39.3	21.0	19.7	8.1	2.4
Static	75.4	63.1	56.2	47.4	31.4	12.7

Table 9. SwinGait robustness scores on CASIA-B dataset (rounded to one decimal place).

Perturbations	Clean	Sev 1	Sev 2	Sev 3	Sev 4	Sev 5
gaussian_noise	82.1	82.1	79.0	60.3	37.8	31.9
defocus_blur	82.1	31.6	31.3	31.2	31.2	31.1
impulse_noise	82.1	82.1	79.4	50.7	36.1	31.2
speckle_noise	82.1	38.1	35.1	32.9	31.7	31.3
shot_noise	82.1	45.5	34.4	32.0	31.3	31.2
motion_blur	82.1	66.5	63.8	59.2	58.5	59.5
zoom_in	82.1	72.5	62.5	50.8	49.9	61.0
freeze	82.1	58.5	53.2	48.1	43.1	48.0
snow	82.1	56.6	55.7	54.7	54.3	54.2
fog	82.1	65.8	64.0	62.0	60.7	56.6
Static	82.1	52.6	50.4	42.0	36.6	35.4

Table 10. DeepGait robustness scores on CCPG dataset (rounded to one decimal place).

Perturbations	Clean	Sev 1	Sev 2	Sev 3	Sev 4	Sev 5
gaussian_noise	77.4	77.4	74.5	56.7	35.5	29.9
defocus_blur	77.4	29.8	29.6	29.5	29.4	29.1
impulse_noise	77.4	77.3	74.9	47.6	34.0	29.3
speckle_noise	77.4	35.9	33.1	30.9	29.8	29.6
shot_noise	77.4	42.7	32.4	30.0	29.0	29.3
motion_blur	77.4	57.4	54.5	50.5	51.8	54.6
zoom_in	77.4	67.8	57.4	43.3	39.5	49.0
freeze	77.4	54.3	50.4	45.0	39.7	44.7
snow	77.4	48.8	47.9	47.1	46.6	46.3
fog	77.4	57.0	55.2	53.6	52.6	48.4
Static	77.4	45.4	43.7	36.6	33.4	33.6

Table 11. GaitBase robustness scores on CCPG dataset (rounded to one decimal place).

Perturbations	Clean	Sev 1	Sev 2	Sev 3	Sev 4	Sev 5
gaussian_noise	70.2	26.9	31.8	51.2	67.6	70.2
defocus_blur	70.2	26.2	26.3	26.4	26.5	26.7
impulse_noise	70.2	26.3	30.6	42.9	67.9	70.2
speckle_noise	70.2	26.4	26.7	27.7	29.5	32.1
shot_noise	70.2	26.3	26.3	26.5	29.0	38.5
motion_blur	70.2	56.6	54.4	50.9	51.0	51.4
zoom_in	70.2	53.4	45.5	43.6	54.7	62.5
freeze	70.2	53.7	51.3	47.4	42.7	47.1
snow	70.2	49.2	48.4	47.6	47.3	47.1
fog	70.2	55.8	54.3	52.8	51.7	47.7
Static	70.2	44.4	43.8	35.1	30.8	29.9

Table 12. SwinGait robustness scores on CCPG dataset (rounded to one decimal place).

plying the normalized image by Gaussian noise scaled by predefined severity levels: 0.15, 0.2, 0.25, 0.3, and 0.35. The np.random.normal function generates Gaussian-distributed noise, which is added to each pixel of the frame. The noisy image is then clipped to the [0, 1] range and rescaled back to [0, 255], before being converted to an unsigned 8-bit integer format.

Shot Noise Each frame is normalized to a [0, 1] range before noise addition. The noisy image is generated by scaling the normalized image by predefined severity levels: 250, 100, 50, 30, and 15, then passed to the np.random.poisson function, which adds Poisson-distributed noise. This noise models the random variation of photon count in low-light conditions. The resulting image is then divided by the severity level value to normalize it. The noisy image is clipped to the [0, 1] range and rescaled back to [0, 255], before being converted to an unsigned 8-bit integer format.

Impulse Noise Each frame is normalized to a [0, 1] range before noise addition. Salt-and-pepper noise is introduced using the skimage.util.random_noise function in 's&p' mode. This function randomly sets a proportion of pixels to either 0 or 1, based on severity levels: 0.03, 0.06, 0.09, 0.17, and 0.27. The noisy image is then clipped to the [0, 1] range and rescaled back to [0, 255], before being converted to an unsigned 8-bit integer format.

Defocus Blur Each frame is normalized to a [0, 1] range before blur addition. A disk-shaped kernel is created using the disk function, with radius values based on severity levels: 3, 4, 6, 8, and 10, and an alias blur of 0.1 to 0.5. The kernel simulates out-of-focus blur and is applied to each color channel of the frame using cv2.filter2D. The blurred image is clipped to the [0, 1] range and rescaled back to [0, 255], before being converted to an unsigned 8-bit integer format.

Zoom Blur Each frame is normalized to a [0, 1] range before blur addition. The frames are zoomed by factors defined by severity levels: 1-1.11, 1-1.16, 1-1.21, 1-1.26, and 1-1.31. This is done using the scipy.ndimage.zoom function, which interpolates the image to create a zoom effect. For each severity level, a range of zoom factors is applied, and the resulting images are averaged to create a smooth blur effect. Specifically, scipy.ndimage.zoom is used to resample the image at different scales. Finally, the frames are rescaled back to [0, 255] and converted to an unsigned 8-bit integer format.

Motion Blur The motion blur function simulates motion in video frames by applying a motion blur effect using the Wand library's MagickMotionBlurImage function. Each frame is converted to the WandImage format and processed with a motion blur effect defined by varying radii and sigma values: (10, 3), (15, 5), (15, 8), (15, 12), and (20, 15). The motion blur method uses these parameters to create a directional blur, simulating motion at different speeds and angles. The frames are then converted back to numpy arrays, clipped to the [0, 255] range, and returned as unsigned 8-bit integers.

Zoom In The zoom in function simulates a gradual zoom-in effect on each frame. Zoom factors are determined by predefined severity levels: 1.5, 2.0, 2.5, 3.0, and 3.5. For each frame, a zoom matrix is created using cv2.getRotationMatrix2D, which specifies the center of the zoom and the scaling factor. The cv2.warpAffine function is then used to apply this transformation to the frame, effectively zooming in. The frames are processed incrementally, creating a smooth zoom effect over time. Finally, the processed frames are normalized to [0, 255] and converted to an unsigned 8-bit integer format.

Freeze The freeze function mimics the effect of a frame freeze by randomly selecting and repeating certain frames. The severity levels determine the proportion of frames to be repeated: 40%, 20%, 10%, 5%, and 10%. The function ensures the transition between frozen and regular frames is smooth by duplicating the selected frames in a way that the sequence of repeated frames appears natural. This is achieved by selecting the frames at random intervals and ensuring that the duplicates are seamlessly integrated with the regular frames. After processing, the frames are clipped to [0, 255] and converted to an unsigned 8-bit integer format.

Sampling The sampling function reduces the frame rate by downsampling and then upsampling the frames. Severity levels define the downsampling rates: 2, 4, 8, 16, and 32. The frames are first downsampled by selecting every nth frame and then upsampled by repeating these frames to match the original frame count. This mimics the effect of a lower frame rate, simulating scenarios with limited bandwidth or processing power. The processed frames are clipped to [0, 255] and converted to an unsigned 8-bit integer format.

Low Light The low light function applies a vignette effect to simulate low light conditions. The severity levels, defined by vignette strength: 1, 2, 3, 4, and 5, determine the darkness of the edges. A mask is created using np.mgrid to simulate

Perturbations	Clean	Sev 1	Sev 3	Sev 5
gaussian_noise	75.9	7.3	6.4	3.2
impulse_noise	75.9	6.4	5.3	3.8
speckle_noise	75.9	12.3	14.1	15.7
motion_blur	75.9	64.2	27.4	5.7
freeze	75.9	68.6	51.4	52.2
rain	75.9	69.0	41.5	9.4
snow	75.9	66.2	64.5	62.4
fog	75.9	77.8	76.0	69.2
Static	75.9	50.1	26.0	1.2

Perturbations	Clean	Sev 1	Sev 3	Sev 5
gaussian_noise	65.5	5.2	5.3	2.8
impulse_noise	65.5	3.7	3.8	3.4
speckle_noise	65.5	9.1	12.4	13.7
motion_blur	65.5	59.4	24.6	4.7
freeze	65.5	62.0	50.1	51.0
rain	65.5	65.5	39.3	8.1
snow	65.5	60.9	59.7	58.1
fog	65.5	68.2	66.4	61.5
Static	65.5	35.9	17.7	1.1

Table 13. Robustness scores of **DeepGait** (left) and **SwinGait** (right) on the **SUSTech1K** dataset across selected perturbation severities.

Perturbations	Clean	Sev 1	Sev 3	Sev 5
gaussian_noise	83.6	9.6	9.4	4.0
impulse_noise	83.6	6.7	5.3	3.8
speckle_noise	83.6	14.0	15.2	16.3
motion_blur	83.6	71.9	33.6	5.6
freeze	83.6	81.5	70.8	71.4
rain	83.6	76.0	48.2	9.8
snow	83.6	73.7	72.5	71.2
Static	83.6	51.7	26.9	1.4

Table 14. GaitBase Robustness Scores on SUSTech1K dataset.

a light source effect, darkening the edges while keeping the center bright. This mask is applied to each frame, adjusting the brightness to create a realistic low-light environment. The vignette mask decreases linearly from the center to the edges, simulating the effect of a light source fading out. Finally, the frames are clipped to [0, 255] and converted to an unsigned 8-bit integer format.

Fog The fog function uses the Albumentations library to simulate fog effects in video frames. The severity of the fog is controlled by fog coefficients, which determine the density and intensity of the fog. These coefficients are predefined for each severity level: 0.49, 0.59, 0.69, 0.79, and 0.89. The A.RandomFog function is employed, which creates a fog effect by overlaying a semi-transparent white layer over the image, reducing the contrast and adding a hazy appearance. The alpha_coef parameter controls the transparency of the fog, while fog_coef_lower and fog_coef_upper set the range for the fog density.

Rain The rain function also utilizes the Albumentations library to overlay realistic rain streaks on video frames. The severity of the rain is defined by rain types and parameters: "drizzle", "drizzle", None, "heavy", and "torrential", with corresponding brightness coefficients (0.7, 0.7, 0.6, 0.55, and 0.5) and drop lengths (5, 15, 20, 40, and 50). The A.RandomRain function simulates rain by adding streaks and adjusting brightness. The parameters slant_lower and slant_upper set the angle of the rain streaks, drop_length controls the length of each streak, and blur_value determines the blurriness of the rain. The brightness_coefficient adjusts the brightness of the image to simulate the darkening effect of rain. The rain effect is applied to each frame, creating a consistent simulation of rainfall. Finally, the processed frames are converted back to an unsigned 8-bit integer format.

Snow The snow function adds snowflakes and increases brightness using the Albumentations library. The severity of the snow is controlled by snow coefficients: 0.05, 0.1, 0.15, 0.2, and 0.25. The A.RandomSnow function is employed to simulate snow by overlaying white noise on the image and increasing the brightness to mimic the reflective nature of snow. Parameters like snow_point and brightnes_coeff are adjusted to control the density and intensity of the snowfall. The brightness_coefficient increases the overall brightness to simulate the glare and reflection caused by snow. Finally, the processed frames are converted back to an unsigned 8-bit integer format.

Occlusion The occlusion function introduces random obstructions in video frames by overlaying object masks from the COCO dataset. The severity of the occlusion is determined by the extent to which the object covers the frame. Image IDs in the COCO dataset are sorted by the area occupied by objects, and this sorted list is divided into five groups based on severity. Depending on the severity level, a group is selected, and a random object from this group is used. The corresponding mask is retrieved using coco.annToMask, resized to fit the frame dimensions, and applied using PIL.Image.paste, blending the masked objects into the scene. This process simulates partial occlusions by static objects

Model	**Params**	**GFLOPs**	**Inference Time**	**FPS**
SCHP[TPAMI20]	66.7M	87.36	43.75	22.86
CDGNet[CVPR22]	80.9M	162.86	47.71	20.96
M2FP[arXiv23]	63.0M	92.73	78.45	12.75
GSAM[arXiv24]	874M	2984.06	865.99	1.15

Table 15. Comparison of model complexity and efficiency.

Method	**Camera**		**Temporal**		**Environmental**		**Occlusion**	
	δ_a	δ_r	δ_a	δ_r	δ_a	δ_r	δ_a	δ_r
DeepGaitV2[arXiv23]	0.36	0.26	0.75	0.71	0.45	0.37	0.56	0.49
GaitGL[ICCV21]	0.32	0.22	0.72	0.67	0.39	0.29	0.49	0.40
GaitBase[CVPR23]	0.33	0.21	0.77	0.73	0.40	0.30	0.46	0.37
GaitSet[AAAI19]	0.42	0.24	0.83	0.79	0.45	0.29	0.55	0.41
GaitPart[CVPR20]	0.39	0.25	0.84	0.81	0.42	0.29	0.52	0.41
SwinGait[arXiv23]	**0.46**	**0.29**	**0.86**	**0.82**	**0.53**	**0.37**	**0.62**	**0.50**

Table 16. Absolute (δ_a) and relative (δ_r) robustness scores of gait models on CASIA-B with a fixed noisy gallery.

Noise Ratio	**GaitBase**[CVPR23]			**DeepGaitV2**[Arxiv23]			**SwinGait**[Arxiv23]		
	NM	BG	CL	NM	BG	CL	NM	BG	CL
No Noise	69.34	60.30	44.92	69.70	60.04	45.55	67.18	56.06	39.42
20%	73.06	64.13	46.31	75.07	64.06	46.80	71.16	59.76	39.29
50%	73.84	64.34	44.87	75.29	64.27	44.43	71.48	57.81	37.72
80%	73.69	63.40	43.03	74.43	61.64	42.20	71.40	56.47	34.53

Table 17. Rank-1 accuracy (%) on the noisy test set (CASIA-B) across different training noise ratios.

Noise Ratio	**GaitBase**[CVPR23]			**DeepGaitV2**[Arxiv23]			**SwinGait**[Arxiv23]		
	NM	BG	CL	NM	BG	CL	NM	BG	CL
No Noise	95.66	90.40	74.88	94.93	89.60	74.97	90.64	82.52	63.17
20%	94.62	88.23	72.20	94.43	86.78	69.67	90.14	80.48	57.55
50%	93.23	85.62	66.98	91.35	84.75	64.01	89.40	77.82	53.56
80%	91.83	83.42	63.24	90.98	81.22	61.49	88.65	74.63	49.62

Table 18. Rank-1 accuracy (%) on the original clean test set (CASIA-B) across different training noise ratios.

Model	**mAP**	**Top-1**	**Top-5**	**Top-10**	**Top-20**
GaitBase[CVPR23]	11.5	**22.5**	**34.3**	41.6	47.0
GaitGL[ICCV21]	7.1	6.0	16.5	23.8	32.4
DeepGaitV2[Arxiv23]	8.7	16.5	30.2	36.2	45.1
SwinGait[Arxiv23]	**11.7**	16.2	29.5	39.7	**49.2**
GaitSet[AAAI19]	9.6	8.9	27.9	38.1	48.9
GaitPart[CVPR20]	9.1	13.3	27.6	36.8	49.1

Table 19. Training performance of gait models on the MEVID dataset. We report mAP and Top-1/5/10/20 retrieval accuracy (%).