IncreFA: Breaking the Static Wall of Generative Model Attribution

Supplementary Material

Table 5. **Composition of IABench**. We collected 28 categories of generative models along with real images sourced from the COCO dataset, and partitioned them into training and test sets.[†]We generate all images following there official github.

Model	Release	Train	Test	Source
Real	——-	20000	5000	[10]
StyleGAN3 [23]	02-2022	3600	900	[28]
StyleGAN-XL [46]	06-2022	3600	900	[28]
LDM [44]	07-2022	20000	5000	[10]
SD1.4 [39]	08-2022	20000	5000	[10]
SD1.5 [39]	10-2022	20000	5000	[10]
SD2 [39]	11-2022	20000	5000	[10]
SD2.1 [39]	12-2022	20000	5000	[10]
Tiny-SD [24]	06-2023	20000	5000	[10]
SD-XL [39]	06-2023	20000	5000	[10]
Small-SD [39]	07-2023	20000	5000	[10]
SSD-1B [20]	07-2023	20000	5000	[10]
E4S [30]	10-2023	3600	900	[10]
PixArt-XL-2 [8]	11-2023	20000	5000	[10]
LCM-SDXL [32]	11-2023	20000	5000	[10]
LCM-SD1.5 [32]	11-2023	20000	5000	[10]
SDXL-Turbo[39]	11-2023	20000	5000	[10]
PG-v2 [27]	12-2023	20000	5000	[10]
MidjourneyV6[34]	12-2023	8000	2000	[10]
SegMoE-SD [39]	02-2024	20000	5000	[10]
GPT-4o [9]	05-2024	8000	2000	[9]
SD3 [39]	06-2024	20000	5000	[10]
FLUX.1-dev [26]	08-2024	20000	5000	[10]
FLUX.1-schnell [26]	08-2024	20000	5000	[10]
Imagen3 [11]	08-2024	3600	900	[28]
R3GAN [22]	12-2024	3600	900	[28]
PG-v2.5[27]	12-2024	20000	5000	[10]
Cogview4 [13]	03-2025	10000	3000	—[†]
Nano-banana [67]	08-2025	9844	3989	[67]
Total		**433844**	**110489**	

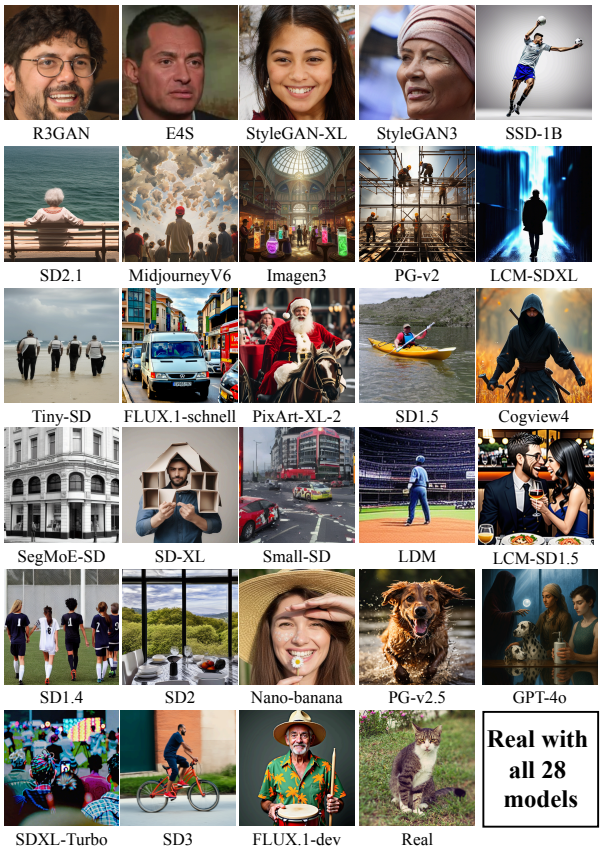

Figure 7. **Samples visualization.**

9. More Details about IABench

We collected generative models from 4 categories of GANs, 2 categories of autoregressive models, and 22 categories of diffusion models. The statistical information for IABench is presented in Tab. 5, including sources. The visualization of the 28 generative model categories is shown in Fig. 7. We gathered 544,333 images spanning from 2022 to 2025 and partitioned them into training and test sets. Among these, our real images are sourced from the COCO dataset. Images from CogView4 were manually generated using its official GitHub repository. Specifically, we employed GPT-4o to randomly generate 3,000 prompts. After removing highly similar prompts through cosine similarity comparison, we generated 7 images for each prompt using different random seeds. Following further manual screening, a total of 13,000 images were ultimately obtained.

10. More Details about Experiments

10.1. Incremental Baselines

iCaRL. [42] The model was trained for 20 epochs in the initial session with a learning rate of 0.001. Each subsequent incremental session was trained for 20 epochs with a learning rate of 0.001, decayed by a factor of 0.1 at epochs 80 and 120. The exemplar memory size was fixed at 2000 samples in total.

FOSTER. [55] The model was trained for 20 epochs in the initial session with a learning rate of 0.001. Each subsequent incremental session consisted of 20 boosting epochs followed by 20 compression epochs, both using a learning rate of 0.001. The exemplar memory size was fixed at 2000 samples in total.

DualPrompt. [59] DualPrompt employs a pre-trained ViT-Base/16 backbone with blocks, patch embedding, cls token, norm, and positional embedding frozen. The model is trained for 5 epochs per incremental session using a constant learning rate of 0.001 and no exemplar memory. It introduces two types of learnable prompts optimized via prefix tuning: general prompts of length 5 placed at layers 0 and 1, and task-specific prompts placed at layers 2, 3, and 4. A shared batch-wise prompt pool of size 10 is maintained, where each prompt has length 5 and top-k=1 prompt is selected for each input using the CLS token as the embedding key. Prompt masking is applied, a pull constraint with coefficient 0.1 encourages proximity between selected prompts and their pool counterparts, and both the prompt pool and keys are initialized uniformly.

L2P. [60] L2P employs a pre-trained ViT-Large/16 backbone with blocks, patch embedding, cls token, norm, and positional embedding frozen. The model is trained for 5 epochs per incremental session using a constant learning rate of 0.001875 and no exemplar memory. It maintains a batch-wise prompt pool of size 10 where each prompt has length 2. For each input, the top-1 most relevant prompt is dynamically selected using the CLS token as the embedding key. A pull constraint with coefficient 0.1 is applied to regularize the selected prompts toward their corresponding pool entries. Both the prompt pool and prompt keys are initialized uniformly.

SimpleCIL. [73] SimpleCIL employs a pre-trained ViT-Base/16 backbone using shallow visual prompt tuning with 3 learnable prompt tokens prepended to the input sequence. No exemplar memory is used. The model is trained with Adam optimizer, learning rate 0.01, weight decay 0.05, and batch size 256. The initial session with 2 classes and all subsequent incremental sessions with 4 new classes each use the same hyper-parameters and are trained until convergence with no fixed epoch count specified.

APER. [73] APER employs a pre-trained ViT-Base/16 backbone augmented with Adapter modules. Only the adapter parameters are trained while the rest of the backbone remains frozen. The model is trained for 20 epochs per incremental session using the Adam optimizer with a learning rate of 0.001, weight decay of 0.0005, and batch size 48. No exemplar memory is used. Each adapter has a bottleneck dimension of 64.

DGR. [21] DGR combines distillation and generative replay using a ViT-Base/16 backbone. The model is trained for 20 epochs in the initial session and 20 epochs in each subsequent incremental session, both using a learning rate of 0.001 and weight decay of 0.0005. A fixed exemplar memory of 2000 samples is maintained together with a generator that produces pseudo-samples of past classes for replay training. The distillation loss weight λ is set to 1 and the replay consistency weight γ_r is set to 1. Temperature $\mathcal{T}$ for distillation is 2. No exemplar memory per class constraint is enforced during incremental learning.

TUNA. [58] TuNA employs a pre-trained ViT-Base/16 backbone and introduces a low-rank residual adapter (r=16) into every linear layer of the transformer blocks. Only the LoRA parameters are updated during incremental learning while the original pretrained weights remain frozen. The model is trained for 15 epochs per incremental session using the Adam optimizer with cosine annealing learning rate starting at 0.01, weight decay of 0.0005, and batch size 16. A covariance-based contrastive adapter regularization with coefficient 0.001 is applied. No exemplar memory is used.

MOS. [49] MOS employs a pre-trained ViT-Base/16 backbone with lightweight residual adapter modules inserted after both the attention and FFN layers of each transformer block. Only the adapter parameters are updated while the original backbone remains frozen. The model is trained for 20 epochs per incremental session using the Adam optimizer with cosine annealing learning rate starting at 0.03, weight decay of 0.0005, and batch size 48. An ensemble of the current and momentum-updated adapters with momentum 0.1 is used for inference. A covariance-based contrastive adapter regularization with coefficient 0.1 is applied. Each adapter uses a bottleneck dimension of 16. No exemplar memory is used.

10.2. Attribution Baselines

DNA-Net. [65] DNA-Net is trained from scratch for a maximum of 30 epochs with early stopping using a batch size of 32. The optimizer uses an initial learning rate of 0.001 with exponential decay of factor 0.9 applied, applied every 2500 iterations. The contrastive loss employs a temperature of 0.07. The best model is selected based on validation accuracy.

RepMix. [5] RepMix is trained for 30 epochs using the Adam optimizer with an initial learning rate of 0.0001, weight decay of 0.0005, and multi-step learning rate scheduling with gamma 0.85. The batch size is 32. Training employs a specialized mixing strategy that combines 2 images per batch (mixup samples=2) with beta 0.4 and applies mixing at layer level 5. Images are perturbed using ImageNet-C corruptions with up to 15 sequential transformations applied with probability 1.0.

DE-FAKE. [47] For DE-FAKE, we follow its original pipeline: images are captioned using BLIP, and both the generated captions and the corresponding images are encoded with the CLIP text encoder and visual encoder (ViT-B/16), respectively. The resulting 512-dimensional image and text embeddings are concatenated into a 1024-dimensional joint representation. A lightweight MLP classifier consisting of three fully-connected layers ($1024 \rightarrow 512 \rightarrow 256 \rightarrow 29$) with ReLU activation and 0.5 dropout after the first layer is trained on top of the frozen CLIP embeddings. The entire model is trained for 50 epochs using the Adam optimizer with a learning rate of 0.0003 and cross-entropy loss. Training and testing are performed with batch size 32, and images are resized to 256×256 and normalized with mean and std of 0.5.

POSE. [66] POSE is trained from scratch for a maximum of 30 epochs using batch size 16. The classifier employs an initial learning rate of 0.0001 with exponential decay (gamma=0.9) applied every 500 iterations and uses softmax loss with temperature 0.1. An auxiliary augmentation network is jointly trained with learning rate 0.01 to generate class-specific perturbations in DCT space. Training incorporates a distance-preserving loss weighted by 0.0001, a closeness-to-known-class loss weighted by 0.01 with similarity threshold 0.95, and an MSE lower bound of 0.

Siamese. [1] The Siamese employs an EfficientNet backbone with 512-dimensional embeddings and is trained from scratch for 50 epochs on 380×380 input images. Training follows a few-shot episodic paradigm with 3 classes per batch and 4 images per class using the Adam optimizer at a learning rate of 0.0001.

10.3. Additional Ablations

Ablation on τ. We conducted ablation experiments on τ, with results presented in Tab. 6. When τ is low, it implies stricter detection for unseen instances, whereas detection for seen instances becomes more lenient. Consequently, at lower levels of τ, the accuracy for unseen detection decreases substantially, while the accuracy for seen detection exhibits only minor changes. Conversely, when τ is high, it implies stricter detection for seen instances, with any output below this threshold being classified as unseen. Consequently, as τ increases, the accuracy for seen detection experiences a significant decline. Considering the trade-off between provenance accuracy and unseen detection performance, we set $\tau = 0.65$ as our hyperparameter.

Ablation on Latent Memory Bank. We conducted ablation experiments on different sizes of the Latent Memory Bank (denoted as N_B), with results presented in Tab. 7.

Table 6. **Ablations on τ.** We report the accuracy within seen categories and the detection rate for unseen instances following the final incremental step.

τ	Seen	Unseen
0.5	78.11	65.01
0.55	78.10	89.66
0.60	77.34	95.78
0.65	78.80	98.93
0.70	72.00	98.98
0.75	69.59	98.53
0.80	55.98	99.02
0.85	54.23	98.97
0.90	41.80	99.52
0.95	39.55	99.79

Table 7. **Ablation Study on the Number of Samples per Model in the Latent Memory Bank.** We report the average accuracy following the completion of the final incremental task in the table.

N_B per task	Final Acc.
5	56.04
10	64.68
20	62.32
50	66.30
100	76.99
150	77.80
200	77.82
250	79.23
500	82.00

As N_B increases, IncreFA's final average accuracy gradually improves; however, when N_B exceeds a certain threshold, the accuracy gains from further increasing N_B diminish. Clearly, a larger N_B yields higher incremental accuracy. However, in practical applications, storage costs and other constraints must be considered. Taking into account the diminishing marginal returns of increasing N_B and the actual memory footprint, we adopt $N_B = 150$ as the final memory bank size.

10.4. Robustness & Threshold

We conducted robustness experiments comparing incremental attribution and unseen Acc. across different τ reported in the table below. Our method retains reasonable performance across various degradations. Under degradations, τ has a stronger influence on unseen family Acc. than on incre. attribution, likely because degradation substantially increases the difficulty of attribution.

Table 8. **Experiments under different degradations.**

τ	JPEG-QF = 95	QF = 90	QF = 80	Resizing = 128	256	384
0.60	74.49 / 92.17	67.04 / 77.45	68.32 / 76.52	71.38 / 84.83	**72.99** / 89.71	75.49 / 94.92
0.65	74.31 / 95.00	**69.31** / 83.99	**68.35** / 85.67	71.00 / 89.97	72.80 / 92.49	**77.10** / 95.02
0.70	**75.01 / 95.14**	68.99 / **85.43**	65.32 / **85.77**	**73.85 / 93.90**	72.59 / **94.08**	74.42 / **96.00**

10.5. Unseen & Wrong Families

(a) We conducted two experiments: one in which we masked out one known family during training and treated it as unseen during inference; the other in which we preserved the existing protocol throughout training and introduced newly emerged mixed-architecture (GLM-Image []) as unseen cases. Due to the excessive number of Diffusion models, we randomly masked out 50% of them. We report

the Avg. Acc. (Un. Acc.) across the two tasks for the unseen families scenario in the table below. Unseen families at the family level disrupt inter-model information sharing, which reasonably accounts for the observed performance drop in our method.

Table 9. **Experiments under masked families.**

Unseen type	Diffusions	GANs	ARs	GLM-Image
Increment	69.57 (87.29)	75.41 (92.56)	75.99 (95.84)	77.94 (92.95)
Attribution	90.34 (92.00)	91.70 (94.55)	92.08 (92.49)	94.67 (94.30)

(b) We evaluated the performance impact by intentionally assigning incorrect family labels to randomly selected sets of four generative model families in IABench. The results are reported in the table below. Similarly, incorrect families inevitably lead to performance degradation.

Table 10. **Experiments under wrong families.**

Direction	D→G	G→D	AR→G	AR→D	G→AR	D→AR
Increment	71.47	71.29	72.53	71.28	72.56	73.11
Attribution	82.91	83.87	92.76	88.39	93.08	87.50

10.6. Broader datasets

We conduct experiments on broader datasets. The corresponding results are reported as Acc. (Un. Acc.) in the table below. Due to the notably low resolution of certain samples in the RED116/140, our method experiences a clear performance degradation. Comparisons with prior works will be supplemented in the final version.

Table 11. **Experiments within broader datasets.**

Datasets	RED116	RED140	Co.Fo. (Small)	Scaling (GenImage + Co.Fo)
Increment	65.16 (87.92)	64.56 (84.00)	51.05 (73.00)	51.12 (75.39)
Attribution	79.30 (96.01)	78.91 (97.52)	62.73 (85.19)	64.62 (83.18)

10.7. Large Scale & Compute Resources

(a) We extended the incremental learning process on IABench by successively incorporating more generative models. The corresponding results are presented in the table below. Dataset details can be found in **Q3**. As the number of models grows, source attribution becomes increasingly difficult. **(b)** We also compute the memory cost by fixing $N = 200$ with the results reported in the table below.

Table 12. **Memory cost with more models.**

Trained models	IABench	+ 32 models	+ 64	+116	+140
Incre. Acc.	78.80	72.53	70.00	64.91	64.19
Un. Acc.	98.93	98.14	92.20	89.19	81.20
Memory Cost (GB)	12.74	26.81	40.87	63.72	74.27