Self Forcing

- □□□□□Self Forcing: Bridging the Train-Test Gap in Autoregressive Video Diffusion
- □□□□□https://arxiv.org/abs/2506.08009
- NeurIPS 2025 spotlight

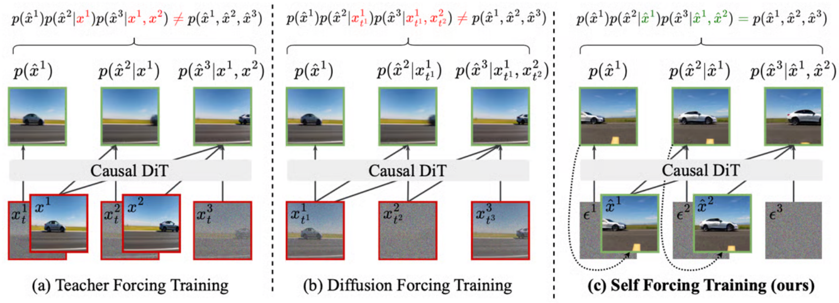

Figure 1: **Training paradigms for AR video diffusion models.** (a) In Teacher Forcing, the model is trained to denoise each frame conditioned on the preceding clean, ground-truth context frames. (b) In Diffusion Forcing, the model is trained to denoise each frame conditioned on the preceding context frames with varying noise levels. Both (a) and (b) generate outputs that do not belong to the distribution the model generates during inference. (c) Our Self Forcing approach performs autoregressive self-rollout *during training*, denoising the next frame based on previous context frames generated by itself. A distribution-matching loss (*e.g.*, SiD, DMD, GAN) is computed on the final output video to align the distribution of generated videos with that of real videos. Our training paradigm closely mirrors the inference process, thereby bridging the train-test distribution gap.

□□□□□Self Forcing□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□GT□□□□□□□□□□□□□□□□□□□□□□□□Self Forcing□□□□□□□kv-cache□□

□□□□□

Algorithm 1 Self Forcing Training

Require: Denoise timesteps $\{t_1, \ldots, t_T\}$
Require: Number of video frames N
Require: AR diffusion model G_θ (returns KV embeddings via G_θ^{KV})

```
loop
    Initialize model output X_θ ← []
    Initialize KV cache KV ← []
    Sample s ~ Uniform(1, 2, ..., T)
    for i = 1, ..., N do
        Initialize x^i_{t_T} ~ N(0, I)
        for j = T, ..., s do
            if j = s then
                Enable gradient computation
                Set x̂^i_0 ← G_θ(x^i_{t_j}; t_j, KV)
                X_θ.append(x̂^i_0)
                Disable gradient computation
                Cache kv^i ← G_θ^KV(x̂^i_0; 0, KV)
                KV.append(kv^i)
            else
                Disable gradient computation
                Set x̂^i_0 ← G_θ(x^i_{t_j}; t_j, KV)
                Sample ε ~ N(0, I)
                Set x^i_{t_{j-1}} ← Ψ(x̂^i_0, ε, t_{j-1})
            end if
        end for
    end for
    Update θ via distribution matching loss
end loop
```

Algorithm 2 Autoregressive Diffusion Inference with Rolling KV Cache

Require: KV cache of size L frames
Require: Denoise timesteps $\{t_1, \ldots, t_T\}$
Require: Number of generated frames M
Require: AR diffusion model G_θ (returns KV embeddings via G_θ^{KV})

```
Initialize model output X_θ ← []
Initialize KV cache KV ← []
for i = 1, ..., M do
    Initialize x^i_{t_T} ~ N(0, I)
    for j = T, ..., 1 do
        Set x̂^i_0 ← G_θ(x^i_{t_j}; t_j, KV)
        if j = 1 then
            X_θ.append(x̂^i_0)
            Cache kv^i ← G_θ^KV(x̂^i_0; 0, KV)
            if |KV| = L then
                KV.pop(0)        ▷ Cache eviction
            end if
            KV.append(kv^i)
        else
            Sample ε ~ N(0, I)
            Set x^i_{t_{j-1}} ← Ψ(x̂^i_0, ε, t_{j-1})
        end if
    end for
end for
return X_θ
```

□□□□□□□□□□□□□□□□□□□□□□□KV-Cache□□□□□□□

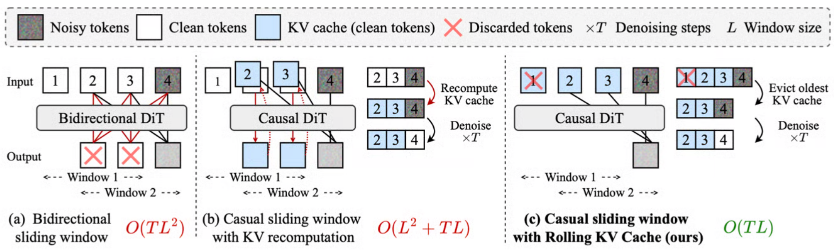

Figure 3: **Efficiency comparisons for video extrapolation.** When performing video extrapolation through sliding window inference, (a) bidirectional diffusion models trained with TF/DF [10, 73] do not support KV cache. (b) Prior causal diffusion models [69, 100] require re-computing KV when shifting the window. (c) Our method does not recompute KV and enables more efficient extrapolation.

- □□□□□□□□
- □□□64 H100
- □□□http://self-forcing.github.io

Older

2026-5-6

CausVid

leicheng © 2022-2025