DisCa: Accelerating Video Diffusion Transformers with Distillation-Compatible Learnable Feature Caching

Supplementary Material

7. Suppelmentary Experiments

7.1. Quantitative comparison for Image-to-Video

As shown in Table 4, besides the discussed experiments on **HunyuanVideo 1.0** (**text-to-video**), we conducted additional evaluations on the **image-to-video** task using the current state-of-the-art video model, **HunyuanVideo 1.5** to verify generalization, evaluating with PSNR, SSIM, and LPIPS. We subjected our proposed methods to **more aggressive settings** for fair comparisons with alternative distillation methods to demonstrate the necessity of our proposed approach. Low-level metrics were calculated on 300 selected high-quality 1K image-text pairs by comparing the accelerated generated images against their corresponding non-accelerated counterparts. For brevity, our proposed Restricted MeanFlow is abbreviated as R-MeanFlow.

Table 4. **Quantitative comparison on different accleration methods** for HunyuanVideo 1.5 on image-to-video (I2V) task.

Method	NFE↓	Latency(s)↓	LPIPS↓	PSNR↑	SSIM↑
HunyuanVideo 1.5 [I2V]	50	778.1	-	-	-
TaylorSeer	8	123.1	0.2050	19.34	0.7137
MagCache	8	127.6	0.2093	19.03	0.7056
AdaCache	8	129.4	0.2084	19.08	0.7070
DCM	8	124.5	0.2496	18.04	0.6616
R-MeanFlow($R = 0.2$)	**8**	**124.5**	**0.1778**	**19.85**	**0.7186**
TaylorSeer	4	79.6	0.2470	18.74	0.6996
MagCache	4	64.5	0.2416	18.73	0.7009
AdaCache	4	64.4	0.2551	18.40	0.6889
DCM	4	**62.3**	0.2450	17.84	0.6686
DisCa ($N = 2, R = 0.2$)	**4**	64.2	**0.1942**	**19.29**	**0.7071**

7.2. User Studies

As shown in Figure 6, we conducted user studies for 4-step and 8-step inference to provide a more comprehensive comparison, demonstrating our method's clear superiority over the baselines. While a performance gap remains between 8-step inference and the original non-accelerated 50-step generation, our approach represents a promising step forward.

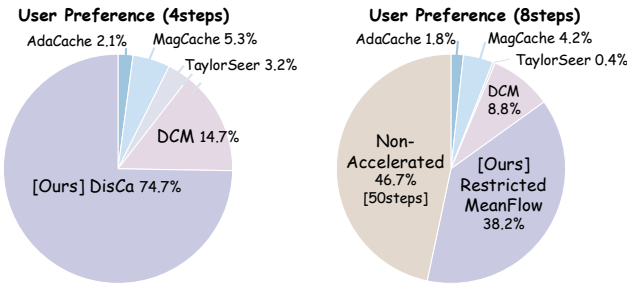

Figure 6. **User Studies** for **DisCa** on HunyuanVideo 1.5 [I2V].

7.3. Ablation Study for Distillation

As shown in Table 5, we provide additional experiments using the predictor directly on multi-step models without distillation to demonstrate how distillation facilitates predictor construction. While the 'w/o distillation' baseline yields acceptable performance, it exhibits noticeable degradation as inference steps are reduced. In contrast, DisCa benefits from the smoother trajectories induced by distillation, which enable more precise predictions.

Table 5. Ablation study for predictor w/ and w/o distillation

Method	**NFE↓**	**Latency(s)↓**	**LPIPS↓**	**PSNR↑**	**SSIM↑**
Predictor w/o distillation	8	145.5	0.1904	20.03	0.7278
Restricted MeanFlow(R=0.2) (Ours)	8	124.5	0.1778	19.85	0.7186
Predictor w/o distillation	4	90.4	0.2498	18.55	0.6963
DisCa ($N = 2, R = 0.2$) **(Ours)**	4	64.2	0.1942	19.29	0.7071

8. Experiment details

8.1. Training and Distillation settings

Starting with the HunyuanVideo with Classifier-Free-Guidance (CFG), we first completed CFG distillation using a learning rate of $lr = 10^{-5}$. Specifically, instead of inferring the CFG and No-CFG branches separately, we utilized a small FFN to append the input CFG information to the condition vector, thereby learning the behavior under different CFG settings. During this distillation process, the CFG was randomly sampled within the range of 1.0 to 8.0.

The model, after CFG distillation, already achieves a theoretical $2\times$ speedup compared to the original. Building on this, we proceeded with MeanFlow distillation (which is the Restricted MeanFlow discussed in this paper) using a learning rate of $lr = 10^{-5}$. We conducted experiments with Restricted MeanFlow at $\mathcal{R} = 0.4$ and $\mathcal{R} = 0.2$. Given that the $\mathcal{R} = 0.2$ version exhibited fewer artifacts, it was adopted as the foundation for DisCa training.

For DisCa training, we first used MSE loss (with a learning rate of 10^{-4}) and random cache reuse intervals $\Delta = (t - t')$ sampled between 0 to $\Delta_{max} = 0.2$ for a 500-iter initialization. We then introduced the discriminator for generative-adversarial training. The learning rate for the predictor was set to 10^{-4} and the discriminator's learning rate was set to 10^{-2}, with the weight of generative-adversarial loss for predictor training $\lambda = 1.0$. Experiments confirmed that these settings ensured a stable adversarial dynamic between the two. The final results were obtained after 1000 iters of GAN training.

8.2. Discussions on VBench

VBench [17] is applied for evaluating the generated videos. VBench consists of 16 sub-dimensions that comprehensively assess video generation quality from multiple aspects. Among these, object class, multiple objects, human action, color, spatial relationship, scene, appearance style, temporal style, and overall consistency constitute the Semantic Score, measuring the model's semantic control capability. Meanwhile, subject consistency, background consistency, temporal flickering, motion smoothness, aesthetic quality, imaging quality, and dynamic degree form the Quality Score, evaluating the overall quality of the generated videos. These two scores are then weighted to produce the Total Score with a ratio of "$\mathbf{Semantic} : \mathbf{Quality} = \mathbf{1} : \mathbf{4}$".

Such a scoring strategy may partially reflect a model's capabilities and the knowledge it encapsulates. However, for critical issues in real-world application scenarios, such as **malformation, blurring, and other fatal flaws**, the Quality score, despite being weighted heavily, fails to respond effectively to them. This can be observed by comparing the visuals in Figure 3 and Figure 4 with the metrics in Table 1 and Table 2: For instance, in Figure 3, the MeanFlow 10-step example exhibits obvious malformation, while the Restricted MeanFlow 10-step example is free of such issues, yet the difference in their Quality scores is only 0.9%. Besides, in Figure 4, the PAB output is completely blurred or even collapsed due to excessive acceleration, but the corresponding Quality score in Table 2 only drops by 6.3% compared to the non-accelerated model, suggesting relying on the Quality score or even the Total score as the primary metric for this paper is clearly unreasonable.

Conversely, the **Semantic score responds well to such fatal issues**, frequently showing the most dramatic decline in the aforementioned scenarios characterized by distinct distortion and quality loss. Therefore, we elect to use the Semantic score as the primary metric, with the Quality and Total scores serving only as supplementary references.

8.3. Break-down Analysis for VBench

We decomposed the variations in VBench Semantic Scores following acceleration. Analysis reveals that while scene-related performance decreased by 28.1%, Spatial Relationship scores improved by 24.4%, demonstrating superior spatial awareness. We leave the mitigation of these scene-related limitations for future exploration.

8.4. Comparison Configurations

The acceleration methods mentioned above in Table 2 can be categorized into the low-speed region (middle) and the high-speed region (lower). We briefly introduce their computational distribution configurations here:

CFG Distilled & (Restricted) MeanFlow: Simply samples with the CFG-distilled model at a specified number of steps, such as 20 and 10 steps. The acceleration ratio is $2\times$ that of the original model (at the same given step count).

Δ-DiT: For the Multi-Modal DiT (MMDiT) structure in HunyuanVideo composed of 20 Double-Stream layers and 40 Single-Stream layers, Δ-DiT skips the 40 Single-Stream layers via a residual-form cache at every cache step. After one complete computation step, it skips $(N-1)$ subsequent steps with cache. The configurations for the low- and high-speed regions are set to $N = 5$ and $N = 8$, respectively.

PAB: In the MMDiT architecture, the traditional distinction between Spatial-Temporal Attention and Cross-Attention does not exist. Therefore, the conventional hierarchical Attention caching strategy degenerates into periodically caching and reusing all Attention outputs. Here, the configurations for the low-speed and high-speed regions are set to $N = 5$ and $N = 8$, respectively.

TeaCache: TeaCache utilizes timestep embedding information to achieve dynamic adjustment of computational allocation. It skips the cache step by performing **one** final reuse at the end. Here, the error threshold ℓ is set to 0.15 for the low-speed region and 0.4 for the high-speed region.

FORA: FORA achieves acceleration by caching and reusing the neural network parts within the residual networks corresponding to Attention and MLP in each DiT Block. Here, the configurations for the low-speed and high-speed regions are set to $N = 3$ and $N = 6$, respectively.

TaylorSeer: TaylorSeer's cache design is similar to FORA's but introduces caching with more derivative tensors, leading to higher VRAM occupancy but noticeably improved performance. Here, we align with the 3-step warm-up strategy from the original paper's code to protect structural information. The configurations for the low- and high-speed regions are $N = 3$ and $N = 6$, respectively.

DisCa: As introduced previously, DisCa performs further acceleration on the Restricted MeanFlow distilled 20-step model with the introduction of a lightweight learnable predictor. N represents the maximum allowed cache interval during inference. For $N = 4, 3, 2$, the inference alternates between 8,11,13 DiT inferences and corresponding 12,9,7 predictor inferences, respectively. Crucially, the inference cost of the predictor in these scenarios is almost negligible.

8.5. Acceleration with Distributed Parallel

8.5.1. VRAM analysis

As shown in Figure 7, we analyzed VRAM consumption for each method, separating it into two parts: model with forward inference and cache occupancy. It can be observed that TeaCache, Δ-DiT, and the proposed **DisCa** all incur negligible extra VRAM overhead. Conversely, caching schemes such as PAB, FORA, and TaylorSeer, which rely on multi-layer caching to provide richer information, gener-

Table 6. **Detailed break-down analysis** for Semantic Score in Table 2.

Method HunyuanVideo 1.0 [T2V]	Semantic Score	Object Class	Multiple Objects	Human Action	Color	Spatial Relationship	Scene	Appearance Style	Temporal Style	Overall Consistency
Original:50steps	73.5(-0.0%)	85.4 (-0.0%)	59.5(-0.0%)	96.0 (-0.0%)	85.3(-0.0%)	58.5(-0.0%)	46.2(-0.0%)	22.1(-0.0%)	24.7(-0.0%)	27.6(-0.0%)
DisCa($R = 0.2, N = 4$)	69.3(-5.7%)	79.1(-7.4%)	56.4(-5.2%)	84.8(-11.7%)	84.8(-0.6%)	72.8(+24.4%)	33.2(-28.1%)	20.8(-5.9%)	22.9(-7.3%)	26.3(-4.7%)

Table 7. **Comparison for the theoretical and actual acceleration of different methods** on HunyuanVideo.

Method HunyuanVideo 1.0 [T2V][62]	CFG Distilled	Peak VRAM	Theoretical			Actual	
			FLOPs(T) ↓	Speed ↑	Latency(s)↓	Latency(s) ↓	Speed↑
Original: 50 steps	✘	99.23GB	394552.32	1.00×	1155.3	1155.3	1.00×
CFG Distilled: 50 steps	✔	97.21GB	197276.16	2.00×	577.7	581.1	1.99×
Original: 10 steps	✘	99.23GB	78910.46	5.00×	231.1	234.7	4.92×
CFG Distilled: 20 steps	✔	97.21GB	78910.46	5.00×	231.1	234.4	4.93×
Δ-DiT($N = 5$) [7]	✔	97.68GB	92068.57	4.29×	269.3	306.7	3.77×
PAB($N = 5$) [76]	✔	121.3GB	64688.07	6.10×	189.4	216.5	5.34×
TeaCache($l = 0.15$) [31]	✔	97.70GB	75350.10	5.24×	220.5	237.6	4.86×
FORA($N = 3$) [56]	✔	124.6GB	78910.46	5.00×	231.1	265.7	4.35×
TaylorSeer($N = 3, O = 1$) [32]	✔	130.7GB	78910.46	5.00×	231.1	268.3	4.31×
MeanFlow: 20 steps [14]	✔	97.21GB	78910.46	5.00×	231.1	232.7	4.96×
Restricted MeanFlow: 20 steps[Ours]	✔	97.21GB	78910.46	5.00×	231.1	232.4	4.97×
DisCa($\mathcal{R} = 0.2, N = 2$) [Ours]	✔	97.64GB	52239.36	7.55×	153.0	152.8	7.56×
CFG Distilled: 10 steps	✔	97.21GB	39455.23	10.0×	115.5	119.7	9.65×
Δ-DiT($N = 8$) [7]	✔	97.68GB	84178.00	4.69×	246.3	253.7	4.55×
PAB($N = 8$) [76]	✔	121.3GB	58058.68	6.80×	169.9	178.8	6.46×
TeaCache($l = 0.4$) [31]	✔	97.70GB	40779.21	9.68×	119.3	125.3	9.22×
FORA($N = 6$) [56]	✔	124.6GB	35509.71	11.1×	108.3	144.2	8.01×
TaylorSeer($N = 6, O = 1$) [32]	✔	130.7GB	43400.76	9.09×	127.1	166.0	6.96×
Restricted MeanFlow: 9 steps[Ours]	✔	97.21GB	35509.71	11.1×	104.0	108.3	10.7×
DisCa($\mathcal{R} = 0.2, N = 3$) [Ours]	✔	97.64GB	44619.05	8.84×	130.7	130.7	8.84×
DisCa($\mathcal{R} = 0.2, N = 4$) [Ours]	✔	97.64GB	33188.56	11.9×	97.1	97.7	11.8×

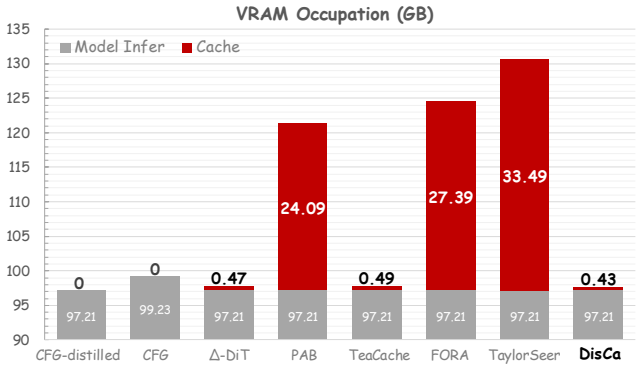

Figure 7. **VRAM Occupation analysis for different acceleration methods.** The VRAM consumption among different caching methods varies significantly; the proposed **DisCa** incurs only about 0.4 GB of extra VRAM overhead, which is negligible.

ate significant VRAM overhead, increasing the demands on the operating device. Specifically, TaylorSeer, which performed optimally among previous methods, consumes an additional 33.49 GB of VRAM. In contrast, **DisCa**, while achieving higher acceleration and higher quality (as shown in Table 2), incurs only 0.43GB of additional VRAM overhead, compressing the extra VRAM cost to just 1.3%.

8.5.2. Cache Architecture Effects Actual Latency

Prior works on feature caching have largely overlooked the relationship between the caching architecture and the discrepancy between theoretical and practical speedups. In this work, we analyze this gap and discuss its implications under the Distributed Parallel conditions.

As shown in Table 7, we calculated the corresponding floating point operations (FLOPs) for each acceleration method and estimated their theoretical speedup ratio by referencing the FLOPs compression, which is then compared against the measured speedup ratio in Table 2.

As in Table 7, the theoretical and actual speedup ratios for CFG Distilled, MeanFlow, and Restricted MeanFlow are extremely close. Among the cache acceleration methods, the theoretical and actual speedup ratios for TeaCache and Δ-DiT are also well aligned. However, schemes such as PAB, FORA, and TaylorSeer exhibit a significant difference between their theoretical and actual speedup ratios. The closeness of the theoretical and actual speedup ratios for CFG Distilled, MeanFlow, and Restricted MeanFlow is obviously reasonable because they can be understood as simply reducing the number of computation steps, while the discrepancy observed among the cache acceleration methods, however, is worth considering.

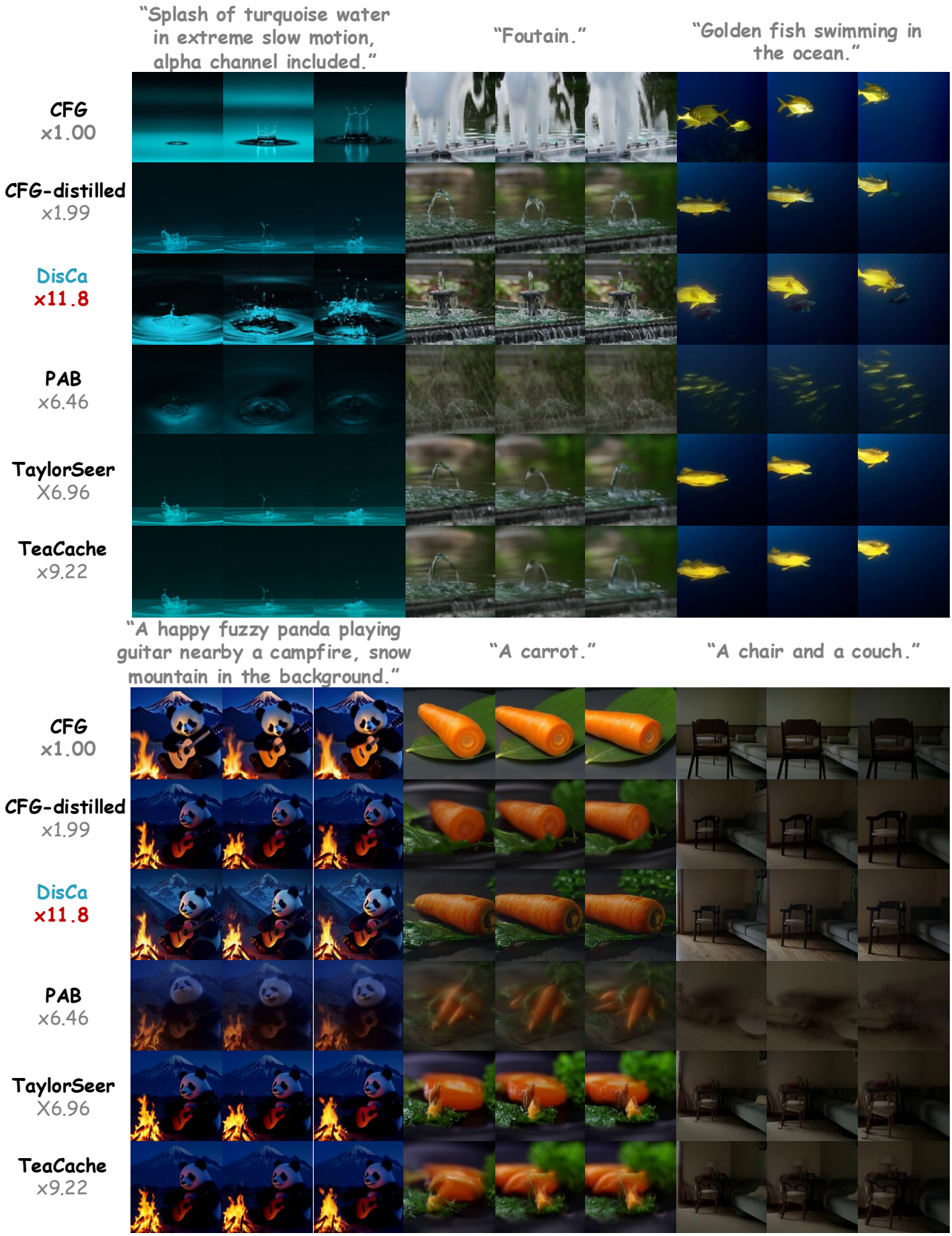

Figure 8. **More visualization examples for HunyuanVideo.** The numerous examples clearly demonstrate that the proposed **DisCa** not only significantly surpasses previous acceleration schemes in terms of speed but also achieves *immense advantages across multiple criteria, including structural semantics, detail fidelity, temporal consistency, adherence to physical plausibility, and aesthetic quality*.

Those with a significant difference between theoretical and actual speedup in Table 7, PAB, FORA, and TaylorSeer, all have a multi-layer cache structure. Specifically, they are multi-layer feature caching schemes that cache the output of each neural network layer individually, retaining the residual structure of the DiT Block to utilize richer features for computation. They prepare a set of cached tensors for every DiT Block: for PAB, it is the output of the attention calculation; for FORA, it is the output of both the attention and MLP calculations; and for TaylorSeer, it additionally includes their multi-order derivatives.

As mentioned above, these schemes impose a particularly large VRAM overhead on the device. Furthermore, such a multi-layer cache structure generates frequent memory accesses during the cache step, and reusing these caches leads to multiple, albeit minor and especially sparse, calculations. In previous scenarios where sequence parallelism was not enabled, computational resources on the device were tight, and these memory accesses and sparse computations were optimized by low-level hardware libraries. Therefore, the extra overhead, though present, was not severe enough to warrant action.

However, with the sequence parallelism technique enabled here, computational resources become relatively abundant. As a result, these sparse memory accesses and computations are not fully optimized by the low-level hardware libraries and the GPU device, leading to unexpectedly greater time consumption. In contrast, TeaCache and Δ-DiT generate only one memory access and one simple, reused extra computation during a single cache-step inference, so their additional overhead is almost negligible.

Summary: *The cache structure is particularly critical in high-resolution, long-sequence video generation scenarios. It not only determines VRAM overhead but also significantly impacts computational efficiency in a parallel environment. Clearly, only reusing the cache in the final layer, instead of a complex multi-layer cache structure, is more appropriate.*

The proposed DisCa employs this single-final-layer-cache structure. Although the introduction of the additional small neural network predictor results in higher extra computation during the cache step than methods like TaylorSeer, which use simple calculations for prediction, this computational overhead remains negligible compared to a complete DiT inference. Furthermore, because the predictor calculation introduced by DisCa is inherently *highly parallelized and imposes low memory access pressure*, it can achieve an acceleration ratio of up to $\mathbf{11.8}\times$, *with the difference from the theoretical acceleration even in the margin of error*.

8.6. More Visualization Results

In this section, we provide additional visualization examples to further substantiate the superior improvement achieved by the proposed **DisCa** compared to previous acceleration schemes, as shown in Figure 8.

9. Detailed Algorithms

In this section, we present detailed algorithms as pseudocode for the aforementioned CFG Distillation, Restricted MeanFlow Distillation, and Generative Adversarial Training process for the Predictor to facilitate understanding.

Algorithm 1 CFG distillation

Input: max and min cfg scale g_{max}, g_{min},
- data-noise-text pairs $\{x_0, \epsilon, c\}_i$,
- CFG Model $\mathcal{M}^{CFG}$.

Init: CFG Distilled Model $\mathcal{M}^*_\theta = \mathcal{M}^{CFG}$.

while Training **do**
  Sample $t \sim U(0,1)$, $g \sim U(g_{min}, g_{max})$.
  Sample $x_t = (1-t) \cdot x_0 + t \cdot \epsilon$.
  Compute $v_c = \mathcal{M}^{CFG}(x_t, t, c)$.
  Compute $v_{uc} = \mathcal{M}^{CFG}(x_t, t, None)$.
  Compute $v_{target} = g \cdot v_c + (1-g) \cdot v_{uc}$.
  Compute $v_\theta = \mathcal{M}^*_\theta(x_t, t, c)$.
  Compute `loss` $= \|v_\theta - v_{target}\|_2^2$.
  `loss.backward(),optimizer.step()`
  # Loss backward & optimize parameter θ.
end while
return CFG Distilled Model $\mathcal{M}^*_\theta$.

Algorithm 2 Restricted MeanFlow distillation

Input: data-noise pairs $\{x_0, \epsilon\}_i$,
- CFG distilled $\mathcal{M}^*$,
- Restrict factor $\mathcal{R}$.
- *# text info c is omitted for simplification.*

Init: Restricted MeanFlow model $\mathcal{M}_\theta = \mathcal{M}^*$

while Training **do**
  Sample $t, r =$`sample_t_r(`$\mathcal{R}$`)`:
    # Sampling in Restricted MeanFlow.
    # Sample $\mathcal{I} \sim U(0, \mathcal{R})$.
    # Sample $t \sim U(0, 1)$.
    # Compute $r = \max(0, t - \mathcal{I})$.
  Sample $x_t = (1 - t) \cdot x_0 + t \cdot \epsilon$.
  Compute $v = \mathcal{M}^*(x_t, t)$.
  Compute $u, \mathrm{d}u/\mathrm{d}t =$`jvp(`$\mathcal{M}_\theta, (x_t, r, t), (v, 0, 1)$`)`
  Compute $u_{tgt} = v - (t - r) \cdot \mathrm{d}u/\mathrm{d}t$.
  Compute `loss` $= \|u -$ `stopgrad(`u_{tgt}`)`$\|_2^2$.
  # No gradient backward go through u_{tgt}.
  `loss.backward(),optimizer.step()`
  # Loss backward & optimize parameter θ.
end while
return Restricted MeanFlow Distilled Model $\mathcal{M}_\theta$.

Algorithm 3 Predictor Training

Input: data-noise pairs $\{x_0, \epsilon\}_i$,
- Sampling timestep pairs $\{(t, r)\}_i$,
- Restricted MeanFlow distilled model $\mathcal{M}$,
- max cache interval Δ_{max}.
- Features Extractor $\mathcal{F} := \mathcal{M}$,

Init: random initialized Predictor $\mathcal{P}_{\theta_P}$,
- random initialized Discriminator D_{θ_D}.

while Training **do**
  Sample $\Delta \sim U(0, \Delta_{max})$.
  Compute $(t', r') = \max((0, 0), (t - \Delta, r - \Delta))$.
  Sample $x_t = (1 - t) \cdot x_0 + t \cdot \epsilon$.
  Compute $\mathcal{C} = \mathcal{M}(x_t, r, t)$. *# Initialize Cache*
  Sample $x_{t'} = (1 - t') \cdot x_0 + t' \cdot \epsilon$.
  Compute $u_{pred} = \mathcal{P}_{\theta_p}(\mathcal{C}, x_{t'}, r', t')$.
  Compute $u_{tar} = \mathcal{M}(x_{t'}, r', t')$.
  Compute $x_{t''}^{pred} = x_{t'} - (t' - r') \cdot u_{pred}$.
  Compute $x_{t''}^{tar} = x_{t'} - (t' - r') \cdot u_{tar}$.
  Define $(t'', r'') = (r', r' - (t'' - r''))$.
  Compute $\mathcal{L}_\mathcal{P} = \|u_{pred} - u_{tar}\|_2^2 + \lambda \cdot \max(0, 1 - \mathcal{D} \circ \mathcal{F}(x_{t''}^{pred}))$,
  $\mathcal{L}_\mathcal{P}$`.backward(), optimizer_P.step()`
  Compute $\mathcal{L}_\mathcal{D} = \max(0, 1 - \mathcal{D} \circ \mathcal{F}(x_{t''}^{pred})) + \max(0, 1 + \mathcal{D} \circ \mathcal{F}(x_{t''}^{tar}))$.
  $\mathcal{L}_\mathcal{D}$`.backward(), optimizer_D.step()`
  # Loss backward & optimize parameter θ_P, θ_D.
end while
return Restricted MeanFlow Distilled Model $\mathcal{M}_\theta$.