Accelerating Diffusion Transformers with Token-wise Feature Caching

Chang Zou*, Xuyang Liu*, Ting Liu, Siteng Huang, Linfeng Zhang✉

SJTU & UESTC & SCU & NUTD & ZJU

Overview

TLDR

This paper introduces ToCa, a training-free **Token-wise feature Caching** method designed to accelerate **diffusion transformers** by adaptively selecting tokens for caching based on their sensitivity to feature reuse. ToCa achieves significant speedups with minimal quality loss by leveraging temporal redundancy and error propagation properties.

Contributions

- propose ToCa as a fine-grained feature caching strategy for diffusion transformers. To the best of our knowledge, ToCa **first** introduces the perspective of **error propagation** in feature caching methods.
- introduce four scores to select the most suitable tokens for feature caching in each layer. Besides, ToCa apply **different caching ratios** in layers of different depths and types.
- Abundant experiments on PixArt-α, OpenSora, and DiT have been conducted, which demonstrates that ToCa achieves a **high acceleration ratio** while maintaining **nearly lossless generation quality**.

Limitations of Layer-wise Feature Caching

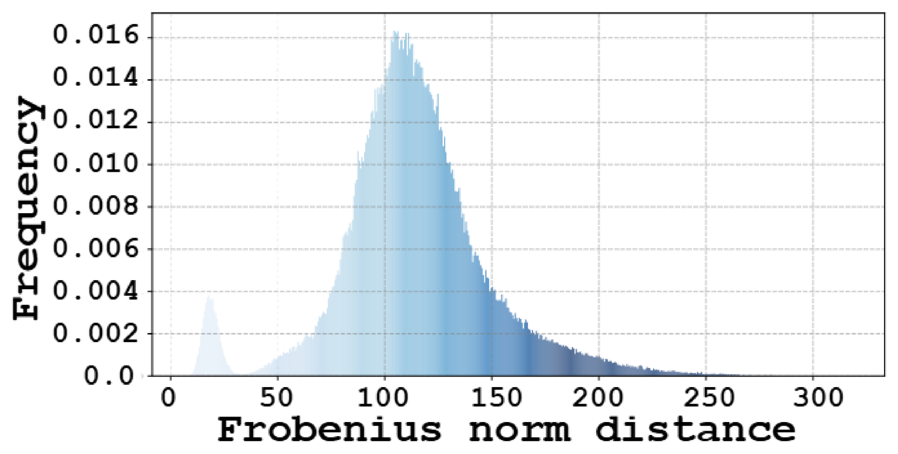


Figure 1: **Temporal Redundancy:** Distribution of the distance between the feature of tokens in the previous and the current timestep.

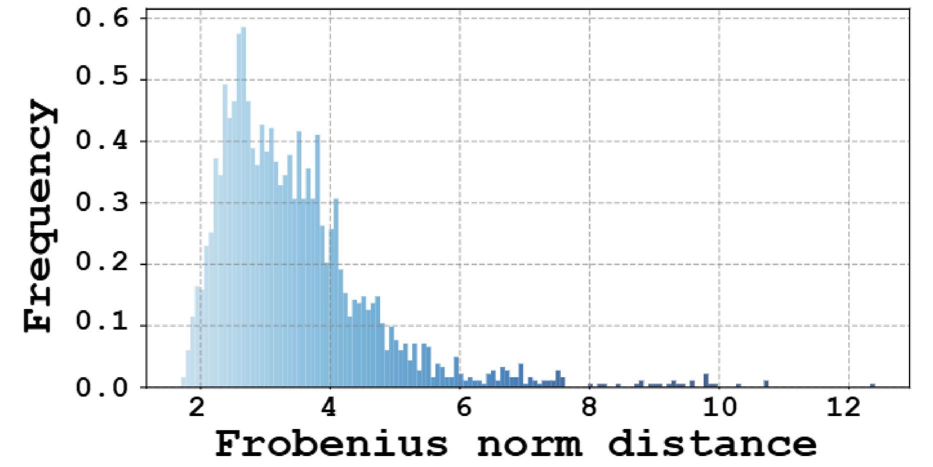
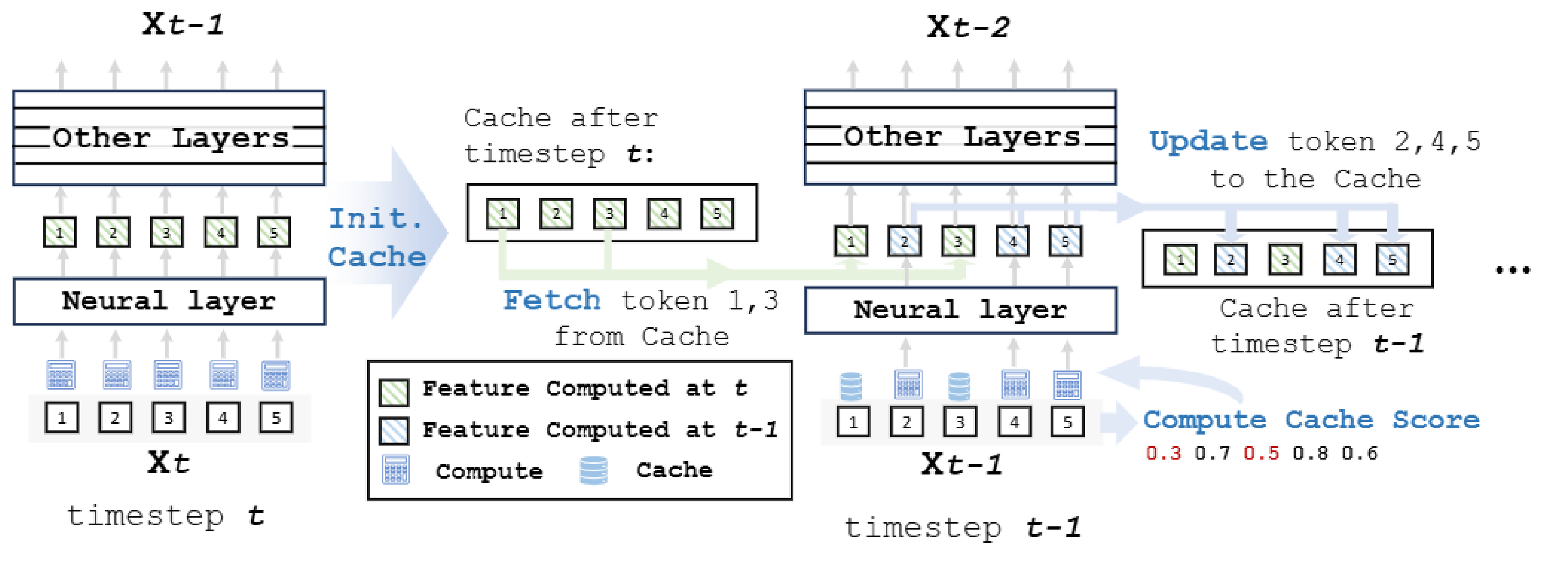


Figure 2: **Error Propagation:** Distribution of the error in the final layer output when the same noise is applied to each token in the first layer.

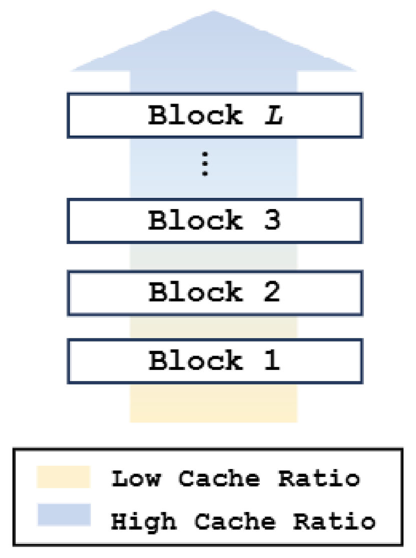
Difference in Temporal Redundancy: Tokens exhibit **varying similarity** across adjacent timesteps. Caching high-similarity tokens reduces computation with minimal quality loss, while low-similarity tokens may degrade generation results if cached.

Difference in Error Propagation: Errors from cached tokens **propagate differently** due to attention mechanisms. Some tokens cause larger errors than others, making token selection critical for minimizing quality impact.

Overall Framework of Token-wise Feature Caching



(a) Overview of ToCa in timestep t and $t-1$



(b) ToCa in layers at different depth

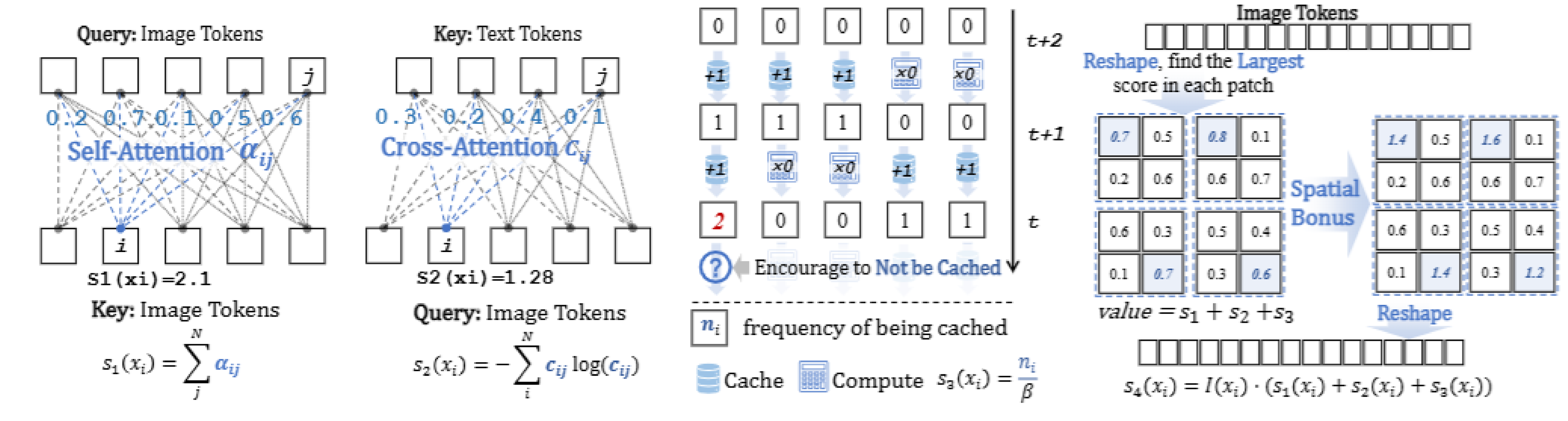
Step 1: In the ***fresh step***, the model performs full computation on all tokens in all layers and updates the computed results into the cache.

Step 2: In the ***cache step***, the model firstly compute the **importance score**, deciding the tokens to be cached. In this example, token 1 and 3 are cached.

Step 3: Then, the tokens 2,4,and 5 are computed through the neural layer. The output of cached token 1,3 are fetched from cache.

Step 4: Then, the calculated tokens 2,4,and 5 are used to update the cache.

Token Selection for Feature Caching



(I) Influence to Other Tokens (II) Control Ability (III) Cache Frequency (IV) Uniform Spatial Distribution

(I) Influence to Other Tokens: Self-attention weights identify **highly influential** tokens, which are less suitable for caching.

(II) Control Ability: Cross-attention weights and entropy are used to avoid caching image tokens with strong influence on **text tokens (conditions)**.

(III) Cache Frequency: **Repeatedly** cached tokens are deprioritized.

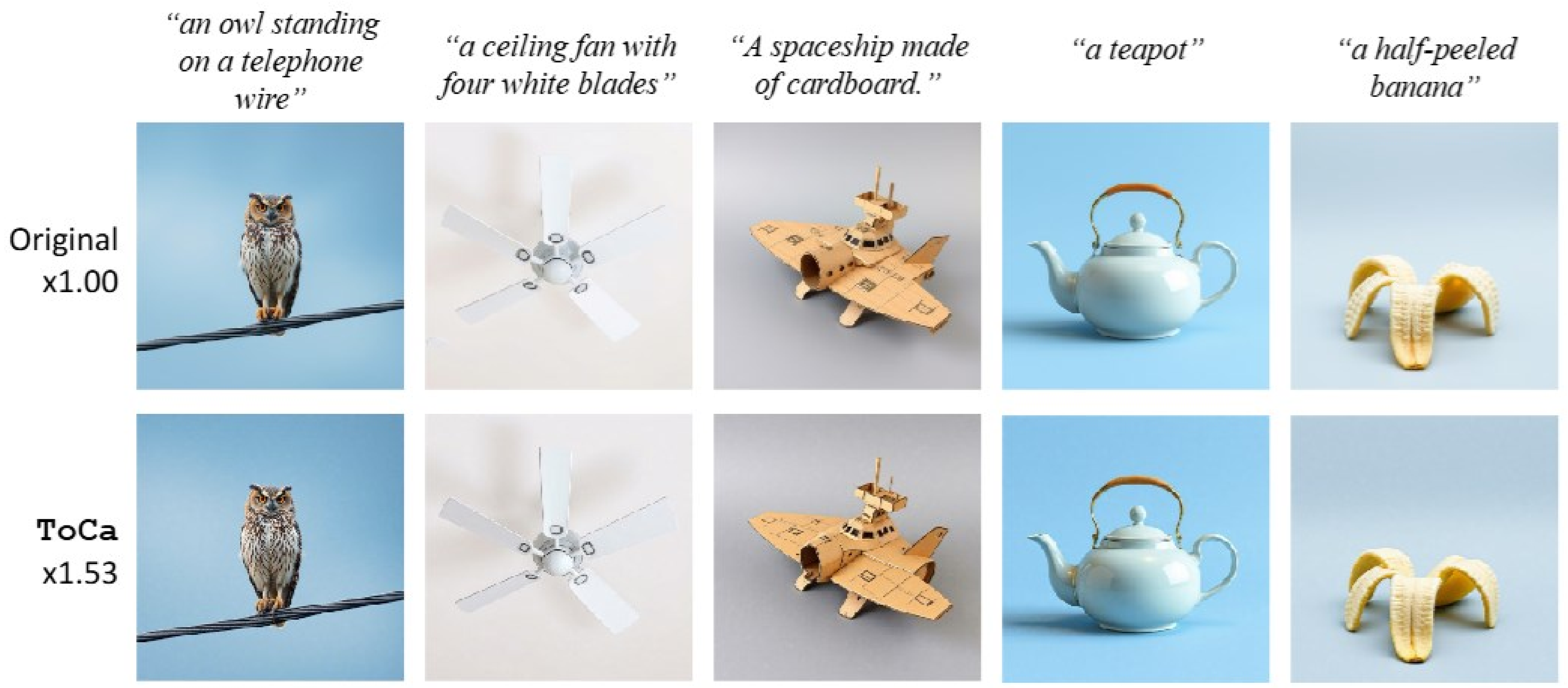
(IV) Uniform Spatial Distribution: Cache scores are adjusted to promote **spatially uniform** token caching.

Quantitative Results on PixArt-α and OpenSora

Method	Latency(s) ↓	FLOPs ↓	Speed ↑	MS-COCO2017 FID-30k ↓	MS-COCO2017 CLIP ↑	PartiPrompts CLIP ↑
PixArt-α (Chen et al., 2024a)	0.682	11.18	1.00×	28.09	16.32	16.70
50% steps	0.391	5.59	2.00×	37.46	15.85	16.37
FORA($\mathcal{N}=2$) (Selvaraju et al., 2024)	0.416	5.66	1.98×	29.67	16.40	17.19
FORA($\mathcal{N}=3$) (Selvaraju et al., 2024)	0.342	4.01	2.79×	29.88	16.42	17.15
ToCa ($\mathcal{N}=3, R=60\%$)	0.410	6.33	1.77×	**28.02**	**16.45**	17.15
ToCa ($\mathcal{N}=3, R=70\%$)	0.390	5.78	1.93×	28.33	16.44	17.75
ToCa ($\mathcal{N}=3, R=80\%$)	0.370	5.05	2.21×	28.82	16.44	**17.83**
ToCa ($\mathcal{N}=3, R=90\%$)	0.347	4.26	2.62×	29.73	16.45	17.82

Method	Latency(s) ↓	FLOPs(T) ↓	Speed ↑	VBench(%) ↑
OpenSora (Zheng et al., 2024)	81.18	3283.20	1.00×	79.13
Δ-DiT* (Chen et al., 2024b)	79.14	3166.47	1.04×	78.21
T-GATE* (Zhang et al., 2024b)	67.98	2818.40	1.16×	77.61
PAB[1*] (Zhao et al., 2024)	60.78	2657.70	1.24×	78.51
PAB[2*] (Zhao et al., 2024)	59.16	2615.15	1.26×	77.64
PAB[3*] (Zhao et al., 2024)	56.64	2558.25	1.28×	76.95
50% steps	42.72	1641.60	2.00×	76.78
FORA(Selvaraju et al., 2024)	49.26	1751.32	1.87×	76.91
ToCa($R=80\%$)	43.52	1439.70	2.28×	**78.59**
ToCa($R=85\%$)	43.08	1394.03	**2.36**×	78.34

Qualitative Results of Text2Image on FLUX



ToCa achieves **nearly lossless speedups** of 1.51×, 1.93×, and 2.36×, and 2.75× on FLUX, PixArt-α, OpenSora, and DiT-XL models respectively, while maintaining **generation quality**.