VTinker: Guided Flow Upsampling and Texture Mapping for High-Resolution Video Frame Interpolation

Chenyang Wu[1], Jiayi Fu[1], Chun-Le Guo[1, 2], Shuhao Han[1], Chongyi Li[1, 2] *

[1]VCIP, CS, Nankai University
[2]NKIARI, Shenzhen Futian
{chenyangwu, jiayifu, hansh}@mail.nankai.edu.cn,
{guochunle, lichongyi}@nankai.edu.cn

Abstract

Due to large pixel movement and high computational cost, estimating the motion of high-resolution frames is challenging. Thus, most flow-based Video Frame Interpolation (VFI) methods first predict bidirectional flows at low resolution and then use high-magnification upsampling (e.g., bilinear) to obtain the high-resolution ones. However, this kind of upsampling strategy may cause blur or mosaic at the flows' edges. Additionally, the motion of fine pixels at high resolution cannot be adequately captured in motion estimation at low resolution, which leads to misalignment in task-oriented flows. With such inaccurate flows, input frames are warped and combined pixel-by-pixel, resulting in ghosting and discontinuities in the interpolated frame. In this study, we propose a novel VFI pipeline, VTinker, which consists of two core components: guided flow upsampling (GFU) and Texture Mapping. After motion estimation at low resolution, GFU introduces input frames as guidance to mitigate detail blurring in bilinear upsampling flows, which makes flows' edges clearer. Subsequently, to avoid pixel-level ghosting and discontinuities, Texture Mapping generates an initial interpolated frame, referred to as the intermediate proxy. The proxy serves as a cue for selecting clear texture blocks from the input frames, which are then mapped onto the proxy to facilitate producing the final interpolated frame via a reconstruction module. Extensive experiments demonstrate that VTinker achieves state-of-the-art performance in VFI.

Code — https://github.com/Wucy0519/VTinker

Introduction

Recent advancements in imaging technologies and devices have made high-resolution video more widely accessible. However, due to hardware limitations and network transmission constraints, high-resolution videos are often at a low frame rate. To improve the frame rate of high-resolution video, video frame interpolation (VFI) that synthesizes intermediate frames between consecutive frames in a video sequence is commonly applied (Reda et al. 2022).

Benefiting from the advantage of motion estimation, flow-based VFI methods are becoming mainstream. As shown in Fig. 1, flow-based high-resolution VFI can be divided into three interrelated stages. (1) *Motion Estimation (M.E.)*: Given the challenges of larger movements and higher computational demands, motion estimation is performed on the downsampled frames in most high-resolution VFI methods. (2) *Upsampling*: To apply low-resolution motion estimation to high-resolution frames, flow-based methods commonly use high-magnification upsampling (e.g., bilinear or adaptive-kernel based) on the low-resolution flow. (3) *Frame Synthesis*: Depending on the upsampled flow, the frame synthesis model predicts an intermediate frame between two consecutive frames.

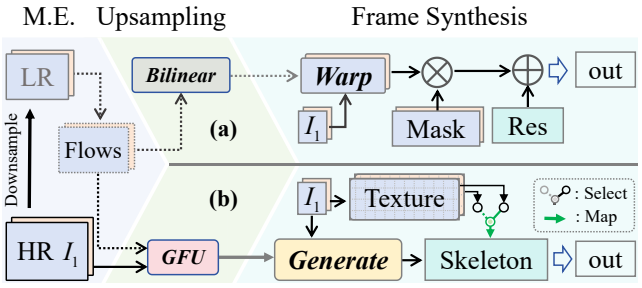

Figure 1: This figure compares VTinker with current flow-based method for high-resolution VFI. After Motion Estimation (M.E.), bidirectional flows are obtained at low resolution. **(a)** The current method employs bilinear upsampling for the high-resolution flows, followed by a pixel-by-pixel synthesis. **(b)** VTinker employs the proposed Guided Flow Upsampling (GFU) and generates the final result through texture mapping.

Unfortunately, there are deficiencies in existing flow-based high-resolution VFI pipelines. Specifically, as shown in Fig. 1(a), in the context of two consecutive high-resolution frames, I_0 & I_1, the current approach involves frame downsampling and performs flow estimation through motion estimation at a low resolution (Li et al. 2023; Liu et al. 2024). Subsequently, flow-based methods (Jin et al. 2023a; Liu et al. 2024) upsample the estimated low-resolution flows using algorithms such as **Bilinear Upsampling or Adaptive-kernel Based Method** that is called adaptive flow upsampling (AFU, proposed in ASFlow (Luo et al. 2021a) for flow estimation task and used in SGM (Liu et al. 2024) for VFI). However, in the face of task-oriented

*Corresponding author.

flow upsampling during end-to-end training, AFU produces discontinuous boundaries, which also makes it less suitable for flow-based VFI models (such as widespread mosaic-like artifacts in SGM's results, as shown in Figs. 4 and 5). The more commonly used method is bilinear upsampling. However, it generally causes the blurring motion boundaries (Luo et al. 2021a) and further leads to blurring or ghosting.

Additionally, the robust application for the task-oriented flow, which is estimated at low resolution but applied to high resolution, presents a significant challenge. In the context of motion estimation at low resolution, it is evident that the motion of fine pixels at high resolution is not adequately captured. The downsampling operation causes the model to prioritize overall motion and leads to neglecting finer details. When flow is upsampled, task-oriented flow encounters difficulties in generalizing to high-resolution applications, which results in biased motion alignment.

Furthermore, in the frame synthesis stage, the given frames I_0 & I_1 are warped to I_t^0 & I_t^1 with obtained flows, respectively. They are then merged into an intermediate frame $\hat{I}_t^{'}$, which incorporates the predicted mask. Then, a network is used to predict a residual term Res and add it to the frame $\hat{I}_t^{'}$ to produce the final output $\hat{I}_t$ at the pixel level (Jin et al. 2023a; Li et al. 2023). This frame synthesis mechanism, called **'Mask&Res'**, produces results at the pixel level with dual-source texture from both I_0 and I_1, as shown in Fig. 1(a). However, the motion estimation is inaccurate due to the blurred boundary and unalignment of the upsampled flows. This results in the interpolation generally exhibiting ghosting, blurring, and discontinuities, because it is based on a fusion of the two misaligned warped frames.

To improve the performance of the high-resolution VFI model and yield clearer results, as shown in Fig. 1(b), we propose a novel pipeline, VTinker. VTinker consists of two core components: **Guided Flow Upsampling (GFU) and Texture Mapping**. Inspired by UPFlow (Luo et al. 2021b), which uses high-resolution information as guidance for flow upsampling in flow estimation task, we propose GFU for task-oriented flow refinement. Guided by high-resolution information, the upsampling process using GFU produces sharper flows, aligning more closely with the edges of associated frames. GFU helps to provide better high-resolution motion estimation, which improves the quality of the result and reduces artifacts at the boundaries.

In order to reduce ghosting and discontinuities due to the misaligned flows and the 'Mask&Res' mechanism, VTinker uses texture mapping to synthesize the interpolated frame. Specifically, VTinker first generates an intermediate proxy, and then maps it using blocks of regional textures, which are exclusively selected from frame I_0 or I_1. These continuous block textures, sourced solely from either I_0 or I_1, remain unaffected by incorrect warping, resulting in a higher degree of continuity in the texture of $\hat{I}_t$. Finally, the reconstruction module rebuilds the proxy that is mapped by high-quality textures, producing clearer results. To ensure that the textures are of high quality, we use the weight-shared reconstruction module to reconstruct the extracted textures and supervise them with input frames.

The **contributions** of this paper are as follows:

- To address the boundary errors caused by current upsampling strategies, we propose Guided Flow Upsampling (GFU) inspired by UPFlow (Luo et al. 2021b). It improves flow upsampling by using input frames to guide the refinement of task-oriented flows, effectively reducing ghosting along the boundaries in interpolated results.
- To resolve flow misalignment resulting from the absence of high-resolution and fine-pixel motion, we propose Texture Mapping. It restores unaligned details using clear textures selected from input frames, improving the continuity in interpolated results.
- With our two core components, we propose VTinker, a novel VFI pipeline for high-resolution video, achieving state-of-the-art performance in VFI. In particular, for high-resolution VFI, the proposed VTinker exhibits impressive performance in the details.

Related Work

Flow-based VFI (Jin et al. 2023a; Li et al. 2023) is mainstream, almost all flow-based methods first obtain the bidirectional flows through motion estimation and then use the flows to warp the two input frames. The final output is obtained by fusing the two warped frames with an estimated Mask and Residual Maps at the pixel level, which is called 'Mask&Res'. The texture of the 'Mask&Res' output is pixel-level dual-source and often suffers from ghosting and blurring when motion estimation is inaccurate.

Facing high-resolution video, VFI models often need to deal with motion over a distance of more than a hundred pixels, making motion estimation particularly challenging (Reda et al. 2022). Some recent methods (Jin et al. 2023b,a; Li et al. 2023; Liu et al. 2024) first estimate the flows at low resolution and then warp the original high-resolution frames by the upsampled flows. However, the upsampling methods mostly are bilinear, which results in the upsampled flows being misaligned with the images' boundaries, as illustrated in Fig. 3. ASFlow (Luo et al. 2021a) proposes Adaptive Flow Upsampling (AFU), which obtains the optical flow upsampling results by estimating a weighted kernel for each upsampled region separately. AFU is considered an effective unsupervised method in the optical flow estimation task. However, for VFI, we train the model in an end-to-end manner, which leads to task-oriented flow estimation. Whether through self-supervision or unsupervision, there is no access to the ground truth of task-oriented flow. When the AFU-like upsampling method is used for VFI, the results interpolated by SGM (Liu et al. 2024) (shown in Fig. 5) are poorly generalized to high-resolution frame interpolation, resulting in some mosaic-like appearance of results. Meanwhile, in Fig. 6, ablation experiments demonstrate the discontinuities in the boundary of the flow, which is upsampled by AFU.

Proposed Method: VTinker

As shown in Fig. 2, given two frames I_0 & $I_1 \in \mathbb{R}^{H\times W\times 3}$, where H and W respectively denotes the height and width of

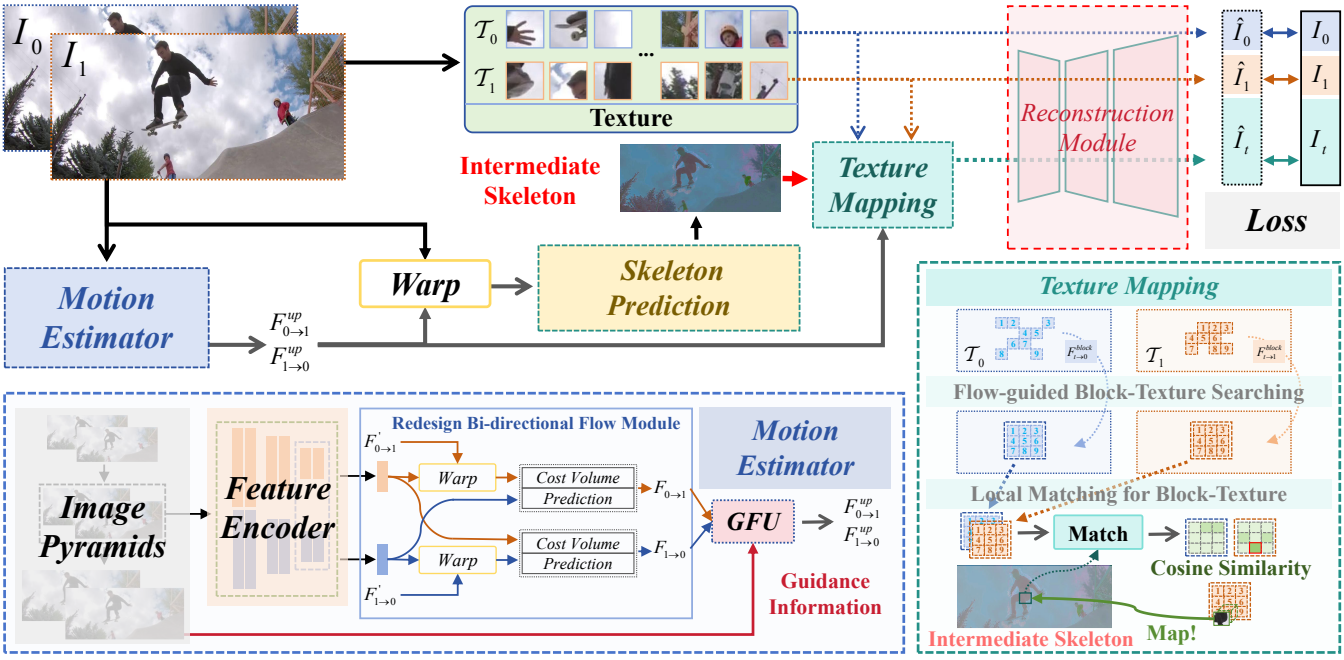

Figure 2: Architecture overview of the proposed VTinker. Given two consecutive frames I_0 & I_1, VTinker first estimates the bi-directional flows $F_{0\to1}$ & $F_{1\to0}$, which are fused by warping to produce an intermediate proxy. Then, after extracting features of the input frames, VTinker divides the features into texture blocks. Through Flow-guided Block-Texture Searching and Local Matching for Block-Texture, texture blocks corresponding to each position of the proxy are selected. Finally, VTinker maps these block-textures to the proxy and rebuilds the interpolated frame by a reconstruction module.

frames, our approach aims to obtain the interpolated frame $\hat{I}_t$ according to the time step t. First of all, based on two input frames, VTinker initially estimates the bidirectional flows at low resolution, and then upsamples them by GFU. Subsequently, VTinker generates an intermediate proxy and maps clear textures onto the proxy. Finally, VTinker obtains the interpolated frame $\hat{I}_t$ after being reconstructed by a reconstruction module.

Motion Estimation with GFU

Benefiting from its pyramid recurrent structure, UPR-Net (Jin et al. 2023a) can serve as a lightweight motion estimator for VTinker. However, the alignment in the UPR-Net (Jin et al. 2023a)'s bi-directional flow module performs in interpolation time step t, while refines flows $F_{0\to1}$ and $F_{1\to0}$. This discrepancy could confuse the coordination, which ultimately leads to a reduction in the efficiency of flow estimation. To achieve more efficient alignment and more accurate flow estimation, as illustrated in Fig. 2, we refer to the structure of PWC-Net (Sun et al. 2018) and redesign the motion estimator.

Specifically, the features of I_0 are warped by predicted flow $F'_{0\to1}$ and aligned with the features of I_1, which are used to update flow $F'_{0\to1}$ and obtain flow $F_{0\to1}$. The same operation is used for flow $F'_{1\to0}$, leading to more accurate motion estimation. We train the redesigned UPR-base following the same training setting as UPR-Net. Using the redesigned UPR-base, we obtain low-resolution bi-directional flows $F_{0\to1}$ and $F_{1\to0}$, and then we upsample them by GFU.

The boundary of the upsampled flow $F^{up}_{0\to1}$ should be aligned with the boundary of the frame I_0. As shown in Fig. 3, to address the inherent challenges associated with flow upsampling, inspired by UPFlow (Luo et al. 2021b), we propose a simple but effective guided flow upsampling (GFU) module, which integrates high-resolution image information to enhance the quality of the upsampled flow. First, GFU uses bilinear flow upsampling to upsample low-resolution flow, producing an initial upsampled flow in which motion edges may appear blurring. Second, the guided information within the input frame is extracted using convolutional layers. Third, GFU employs this guidance information to rectify the blurring details in the bilinearly upsampled flow, enhancing the sharpness of the flow's edges. Ablation experiments (see Fig. 6) demonstrate that GFU offers greater reliability compared to the bilinear upsampling and AFU.

After upsampled by GFU module, flows $F^{up}_{0\to1}$ & $F^{up}_{1\to0}$ warp frames I_0 & I_1 into I^0_t & I^1_t, which can be expressed as:

$$I^0_t = \mathcal{W}(I_0, F^{up}_{0\to t}), I^1_t = \mathcal{W}(I_1, F^{up}_{1\to t}), \tag{1}$$

where $F^{up}_{0\to t} = t \times F^{up}_{0\to1}$ and $F^{up}_{1\to t} = (1-t) \times F^{up}_{1\to0}$ denote the estimated flows $F^{up}_{1\to0}$ & $F^{up}_{0\to1}$ mapped to specific time step t in a linear function.

Texture Mapping of VTinker

In this section, Texture Mapping is used to generate the intermediate proxy and map it with selected high-quality textures. VTinker first predicts the intermediate proxy and extracts texture blocks from two input frames. By flow guidance and calculated similarity , VTinker indexes the high-quality mapping blocks.

Proxy Prediction and Texture Extraction Based on warped frames I_t^0 & I_t^1, as displayed in Fig. 2, the proxy Prediction module generates the intermediate proxy $\mathcal{Q}$:

$$\mathcal{Q} = Convs(I_t^0, I_t^1), \mathcal{Q} \in {}^{\frac{H}{2}\times\frac{W}{2}\times C}, \tag{2}$$

where C denotes the channel number of $\mathcal{Q}$. We use multilayer convolutions for the proxy prediction. Concurrently, the textures of two consecutive frames are extracted and divided into texture blocks with a specific size setting.

To refine $\mathcal{Q}$ into the interpolated frame $\hat{I}_t$, VTinker uses the obtained proxy $\mathcal{Q}$ to index the textures blocks and map them into the proxy $\mathcal{Q}$. Based on frames I_0, I_1, we obtain the texture $\mathcal{T}_0$, $\mathcal{T}_1$:

$$\mathcal{T}_0 = Convs(I_0), \mathcal{T}_1 = Convs(I_1). \tag{3}$$

We use convolutional layers as an extractor. According to the block size s, we respectively divide the texture $\mathcal{T}_0$ and $\mathcal{T}_1$ into many texture blocks $\mathcal{B}_0^{x,y}$ and $\mathcal{B}_1^{x,y}$, which overlaps with each other. For example, $\mathcal{B}_0^{x,y}$ can be expressed as:

$$\mathcal{B}_0^{x,y} = Split(\mathcal{T}_0|x, y, s), \mathcal{B}_0^{x,y} \in \mathbb{R}^{s\times s\times C}, \tag{4}$$

where $Split(\mathcal{T}|x, y, s)$ denotes that the texture $\mathcal{T}$ is divided into $s \times s$ texture-blocks, with $\mathcal{B}^{x,y}$ representing the texture block located at position index (x, y). In addition, the proxy $\mathcal{Q}$ is similarly divided into blocks for the purpose of texture mapping, represented by $\mathcal{B}_q^{x,y}$:

$$\mathcal{B}_q^{x,y} = Split(\mathcal{Q}|x, y, s), \mathcal{B}_q^{x,y} \in \mathbb{R}^{s\times s\times C}. \tag{5}$$

Flow-guided Block-Texture Searching By estimating the motion between the given frames I_0 & I_1, bidirection flows $F_{0\to t}^{up}$ & $F_{1\to t}^{up}$ can be obtained. Since these flows contain direct motion information, they facilitate texture indexing. Therefore, to obtain preliminary texture block matching, we use flows $F_{0\to t}^{up}$ and $F_{1\to t}^{up}$ for indexing.

Firstly, according to block size $s \times s$, we respectively downsample $F_{0\to t}^{up}$ and $F_{1\to t}^{up}$ to obtain $F_{0\to t}^{block}$ and $F_{1\to t}^{block}$, in 'nearest' mode. To avoid texture mixing that can result from the application of bilinear-based forward or backward warping, we first convert the forward flows $F_{0\to t}^{block}, F_{1\to t}^{block}$ into backward flows $F_{t\to 0}^{block}, F_{t\to 1}^{block}$:

$$F_{t\to 0}^{block} = -1 \times \mathcal{W}(F_{0\to t}^{block}, F_{0\to t}^{block}). \tag{6}$$

Then, we use the '$Grid\_Sample$'(Nearest mode) function in PyTorch (Paszke et al. 2019) to index texture blocks and obtain indexed texture blocks $\mathcal{B}_{0,t}$, which can be expressed as:

$$\mathcal{B}_{0,t} = GridSample(\mathcal{B}_0, F_{t\to 0}^{block}, mode = Nearest). \tag{7}$$

Following Eq. (6) and Eq. (7), we obtain $\mathcal{B}_{0,t}$ and $\mathcal{B}_{1,t}$. These are rearrangement of the texture blocks $\mathcal{B}_0$ and $\mathcal{B}_1$.

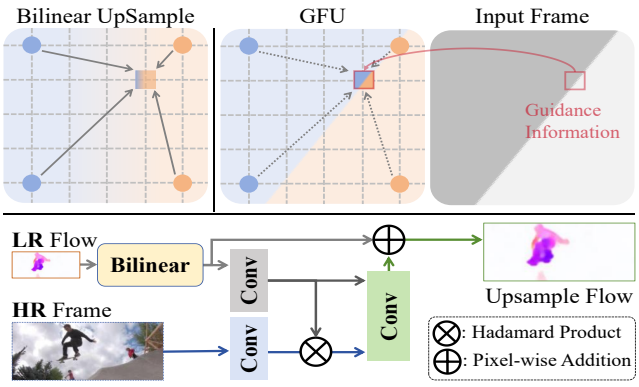

Figure 3: Guided Flow Upsampling (GFU) Module. The difference between bilinear upsampling and GFU is visually shown above the line. The edges of flow upsampled by GFU are more aligned with the input frame than the bilinear. Blue and orange points indicate pixels at low resolution. The framework of GFU is below the line. GFU uses the input frame as guidance information to refine the flow upsampling. Remarkable contrast is shown in Tab. 3 and Fig. 6.

Local Matching for Block-Texture Due to the inaccuracy of the flow estimation, the proxy block $\mathcal{B}_q^{x,y}$ generally corresponds to neither $\mathcal{B}_{0,t}^{x,y}$ nor to $\mathcal{B}_{1,t}^{x,y}$, which would significantly affect the interpolated result. To achieve a better matching, we process the proxy $\mathcal{B}_q$ and texture maps $\mathcal{B}_{0,t}$ and $\mathcal{B}_{1,t}$ into low-resolution index-tensors $\mathcal{K}_q, \mathcal{K}_{0,t}, \mathcal{K}_{1,t}$ by multilayer convolutions:

$$\mathcal{K}_q^{x,y} = Convs(\mathcal{B}_q^{x,y}), \mathcal{K}_q^{x,y} \in \mathbb{R}^{\frac{s}{2}\times\frac{s}{2}\times C'}, \tag{8}$$

where C' denotes the channel number of $\mathcal{K}_q$. The same operation is applied to obtain $\mathcal{K}_{0,t}, \mathcal{K}_{1,t}$. Inspired by QKV mechanism (Vaswani et al. 2017), we convert index-tensors $\mathcal{K}_q^{x,y}$, $\mathcal{K}_{0,t}^{x,y}$ and $\mathcal{K}_{1,t}^{x,y}$ to index-vectors $Q^{x,y}$, $K_0^{x,y}$ and $K_1^{x,y}$ by

$$\begin{aligned} Q^{x,y} &= Mean(Norm(\mathcal{K}_q^{x,y})), \\ K_0^{x,y} &= Mean(Norm(\mathcal{K}_{0,t}^{x,y})), \\ K_1^{x,y} &= Mean(Norm(\mathcal{K}_{1,t}^{x,y})), \end{aligned} \tag{9}$$

where $Q^{x,y}, K_0^{x,y}, K_1^{x,y} \in \mathbb{R}^{1\times 1\times C'}$. Then, based on $Q^{x,y}$, we match the most relevant vector in $N \times N$ neighbors of both two index-vector groups K_0 and K_1 at the position (x, y). $\mathbb{C}^{x,y,(N)}$ denotes a set of correlations with $Q^{x,y}$:

$$\mathbb{C}^{x,y,(N)} = Q^{x,y} \times [\mathbb{K}_0^{x,y,(N)} \cap \mathbb{K}_1^{x,y,(N)}], \tag{10}$$

$$\mathbb{C}^{x,y,(N)} = \{C_e^{x+i,y+j}, -\frac{N}{2} \le i, j \le \frac{N}{2}, e = \{0, 1\}\}, \tag{11}$$

where $C_e^{x+i,y+j} = Q^{x,y} \times K_e^{x+i,y+j}$, and $\mathbb{K}_0^{x,y,(N)}$ denotes a set of $N \times N$ neighbors of $\mathcal{K}_0^{x,y}$:

$$\mathbb{K}_0^{x,y,(N)} = \{K_0^{x+i,y+j}, -\frac{N}{2} \le i, j \le \frac{N}{2}\}. \tag{12}$$

Then, we evaluate the maximum element $C_e^{x+i,y+i}$ in the set $\mathbb{C}^{x,y,(N)}$, and get the position index $(x + i, y + j, e)$ of this

	Venue	DAVIS(1080p)				DAVIS(4K)				Vimeo90K	R.	F.
		PSNR↑	SSIM↑	LPIPS↓	DISTS↓	PSNR↑	SSIM↑	LPIPS↓	DISTS↓	PSNR↑		
XVFI	ICCV21	25.219	0.794	0.170	0.069	24.799	0.799	0.182	0.071	35.070	69	85
RIFE	ECCV22	25.907	0.803	**0.134**	**0.052**	25.691	0.809	0.143	**0.055**	34.189	**29**	**52**
M2M	CVPR22	26.191	0.812	0.164	0.072	25.900	0.816	0.173	0.076	35.369	61	73
AMT-G	CVPR23	25.950	0.813	0.180	0.079	OOM	OOM	OOM	OOM	**36.352**	120	638
UPR-Large	CVPR23	26.582	0.820	0.156	0.068	26.405	0.825	0.161	0.069	36.115	54	251
UPR-LLarge	CVPR23	26.655	0.821	0.156	0.069	OOM	OOM	OOM	OOM	36.247	65	445
EMA-VFI	CVPR23	26.465	0.816	0.169	0.073	26.351	0.822	0.173	0.075	35.917	44	109
SGM-Local	CVPR24	26.008	0.808	0.159	0.067	24.710	0.784	0.196	0.084	34.286	104	188
SGM-1/2Point	CVPR24	26.927	0.826	0.149	0.068	26.798	0.831	0.154	0.071	35.670	108	188
VTinker-$\mathcal{L}_{cc}$	–	**26.976**	**0.827**	**0.134**	0.059	**26.826**	**0.833**	**0.139**	0.063	35.640	121	765
SoftSplat	CVPR20	25.385	0.784	0.147	0.054	25.018	0.788	0.159	0.058	35.359	117	268
EDSC	PAMI21	24.547	0.768	0.205	0.082	24.195	0.771	0.215	0.087	34.843	**55**	**72**
FILM	ECCV22	26.009	0.801	0.124	0.048	23.078	0.709	0.252	0.099	**35.550**	273	–
PerVFI	CVPR24	26.230	0.808	0.114	0.042	OOM	OOM	OOM	OOM	33.841	285	872
VTinker	–	**26.778**	**0.817**	**0.108**	**0.039**	**26.610**	**0.823**	**0.115**	**0.041**	35.064	121	765

Table 1: Quantitative comparison of state-of-the-art VFI methods on DAVIS (1080P), DAVIS (**4K**), and Vimeo90K (448 × 256). OOM means Out Of Memory. R. and F. donate **R**untime(ms) and **F**LOPs(G). **RED**: best performance, BLUE: the second best. Methods listed above the line are trained without perception loss, while those below are trained with perception loss.

	SNU-FILM								Xiph-2K		Xiph-4K	
	Easy		Medium		Hard		Extreme					
	LPIPS↓	DISTS↓	LPIPS↓	DISTS↓	LPIPS↓	DISTS↓	LPIPS↓	DISTS↓	LPIPS↓	DISTS↓	LPIPS↓	DISTS↓
EDSC	0.020	0.022	0.036	0.032	0.077	0.050	0.150	0.076	0.086	0.046	0.190	0.078
FILM	0.014	0.013	0.023	0.018	0.046	0.026	0.093	0.045	0.033	0.023	0.513	0.194
RIFE	0.014	0.013	0.025	0.019	0.050	0.029	0.101	0.049	0.040	0.020	0.084	0.035
AMT-G	0.020	0.023	0.035	0.034	0.062	0.048	0.124	0.072	0.096	0.053	OOM	OOM
UPR-Large	0.019	0.021	0.035	0.033	0.063	0.048	0.114	0.067	0.099	0.055	0.226	0.101
UPR-LLarge	0.019	0.021	0.035	0.034	0.064	0.049	0.114	0.068	0.101	0.055	OOM	OOM
LDMVFI	**0.013**	–	0.027	–	0.068†	–	0.139†	–	–	–	–	–
EMA-VFI	0.019	0.021	0.035	0.033	0.073	0.050	0.147	0.083	0.096	0.052	0.222	0.097
SGM-Local	0.023	0.024	0.036	0.033	0.068	0.048	0.138	0.078	0.097	0.052	0.217	0.094
SGM-1/2point	0.019	0.021	0.034	0.032	0.065	0.047	0.125	0.072	0.097	0.053	0.222	0.097
PerVFI	0.015	0.012	0.026	0.018	0.049	0.027	0.094	0.044	0.038	0.015	OOM	OOM
VTinker	**0.013**	**0.011**	**0.022**	**0.016**	**0.044**	**0.025**	**0.088**	**0.040**	**0.031**	**0.013**	**0.066**	**0.025**

Table 2: Quantitative comparison of state-of-the-art VFI methods on **720P** dataset (SNU-FILM), **2K** dataset (Xiph-2K), and **4K** dataset (Xiph-4K). The scores of LDMVFI are taken from their paper and are denoted by symbol †.

element. Note that this step ensures that the texture block comes from either I_0 or I_1. Thus, the texture block corresponding to the position (x, y) of the proxy $\mathcal{Q}$ is $\mathcal{B}_e^{x+i,y+j}$, which will be mapped to $\mathcal{Q}^{x,y}$ to recover the detail of the interpolated result. Finally, following the steps above, all the texture block $\mathcal{B}_q^{x,y}$ are traversed.

Reconstruction Module and Loss Function

As shown in Fig. 2, to ensure the quality of the texture block $\mathcal{B}_0^{x,y}$ and $\mathcal{B}_1^{x,y}$, we supervise not only the interpolated result $\hat{I}_t$ with Ground Truth I_t but also the texture $\mathcal{T}_0$ and $\mathcal{T}_1$. The reconstruction module is a UNet-like network and aims to transform the latent space into the image space. By using weight-shared reconstruction module, we obtain the result $\hat{I}_0$ and $\hat{I}_1$ by restoring texture $\mathcal{T}_0$ and $\mathcal{T}_1$, and supervise them with the input frames I_0 and I_1. These frames do not require reconstruction during inference.

We train our model with the Style loss $\mathcal{L}_S$ proposed in FILM (Reda et al. 2022):

$$\mathcal{L}_S = w_l \mathcal{L}_1 + w_{VGG} \mathcal{L}_{VGG} + w_{Gram} \mathcal{L}_{Gram}, \quad (13)$$

Three Style losses $\mathcal{L}_S^t$, $\mathcal{L}_S^0$, $\mathcal{L}_S^1$ are computed separately using $\hat{I}_t$, $\hat{I}_0$, $\hat{I}_1$ and I_t, I_0, I_1. We combine these Style losses to obtain the final loss $\mathcal{L}_S^{all}$:

$$\mathcal{L}_S^{all} = w_t \times \mathcal{L}_S^t + w_0 \times \mathcal{L}_S^0 + w_1 \times \mathcal{L}_S^1, \quad (14)$$

where the weights (w_t, w_0, w_1) are set empirically.

Experiments

Experimental Settings

Training Settings. VTinker is trained using the Style loss (Reda et al. 2022), as defined in Eq. (14), with the weights (w_t, w_0, w_1) set to (0.8, 0.1, 0.1). To ensure fair comparisons, using the commonly employed Charbonnier and Census Loss, we retrain VTinker, which is called VTinker-$\mathcal{L}_{cc}$. We utilize the commonly used Vimeo90K (Xue et al. 2019) Triplet as the training set and augment the images by randomly cropping 256 × 256 patches. AdamW optimizer (Loshchilov and Hutter 2017) is used to train VTinker for 0.2M iterations, with a batch size of 16 on 8 GPUs. During training, we train the redesigned UPR-base model following the training setting of UPR-Net (Jin et al. 2023a). Because of its lightweight design and efficiency, we use it as our motion estimator.

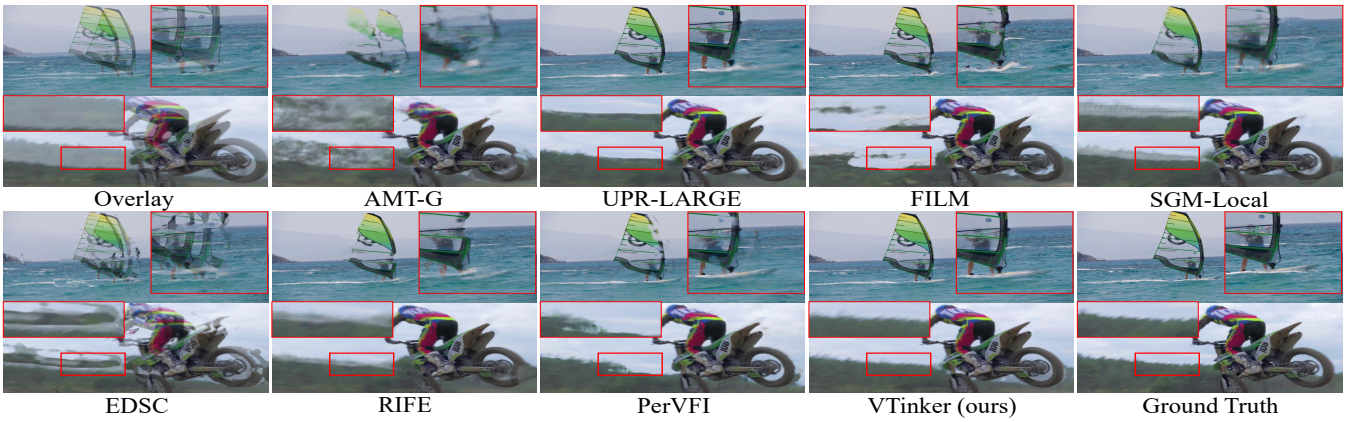

Figure 4: Qualitative comparisons among different methods on 2K resolution. All cases move more than 140 pixels. Overlay is the average of two input frames.

Compared Methods. We compare our VTinker with several state-of-the-art VFI algorithms, including methods trained without perception-based loss (XVFI (Sim, Oh, and Kim 2021), RIFE (Huang et al. 2022), M2M (Hu et al. 2022), AMT (Li et al. 2023), UPR (Jin et al. 2023a), EMA-VFI (Zhang et al. 2023) and SGM (Liu et al. 2024)) and with percetion-based loss (SoftSplit (Niklaus and Liu 2020), EDSC (Cheng and Chen 2021), FILM (Reda et al. 2022), LDMVFI (Jain et al. 2024), and PerVFI (Wu et al. 2024)). The publicly available implementations of those methods are used in comparisons.

Evaluation Metrics. We adopt the commonly used metrics for evaluation, including PSNR, SSIM, LPIPS (Zhang et al. 2018), DISTS (Ding et al. 2022), and FloLPIPS (Danier, Zhang, and Bull 2022). To address high LPIPS scores resulting from overfitting to VGG features, we use VGG19 for calculating loss while employing AlexNet for LPIPS computation. Runtime and FLOPs are evaluated with a frame size of 512×512.

Datasets. To evaluate the performance of the models, we employ commonly used VFI benchmarks: DAVIS (1080P, all full resolution cases are resized to 1080P) (Perazzi et al. 2016), DAVIS(4K, includes large number of 4K cases with large motion), Vimeo90K (Xue et al. 2019) (448×256), SNU-FILM (Choi et al. 2020) (1280×720, contains four subsets: easy, medium, hard and extreme, with increasing motion scales), Xiph (Niklaus and Liu 2020)(2K, 4K), X-Test (Sim, Oh, and Kim 2021), UCF101 (Soomro, Zamir, and Shah 2012) (256×256), and DAVIS (480P, 640×480).

Quantitative Evaluation

As shown in Tab. 1, according to whether perception-based loss is used in model training or not, we divide methods into two groups, and both versions of VTinker outperform others in PSNR, SSIM, and LPIPS for high-resolution VFI. We also analyze VTinker in SNU-FILM and Xiph comparison with state-of-the-art methods in Tab. 2. The results show our proposed VTinker has the superior perceptual quality. In terms of the larger moving test scenarios (including SNU-FILM hard and extreme subsets, and DAVIS (1080P)), VTinker has more obvious advantages over the compared methods on SNU-FILM easy and medium subsets. VTinker is trained only on Vimeo90K , but performs better than the LDMVFI trained on high-resolution data, especially on SNU-FILM hard and extreme subsets. This also shows that our proposed VTinker has a strong generalizability. For 2K resolution, Xiph-2K, the movement of this group is approximately 100 pixels. In contrast, for 4K resolution (Xiph-4K and DAVIS, the movement is around 200 pixels. Our proposed VTinker achieves an impressive performance in this comparison. For the 4K evaluation, the proposed VTinker obtains a 0.012 (approximately 34%) improvement over the second-ranked method RIFE in terms of metric DISTS on the Xiph-4K dataset. The result highlights the efficacy of the proposed VTinker in the high-resolution VFI task. More results, including comparisons on other benchmarks can be found in the supplementary material.

Qualitative Evaluation

In Fig. 4, all cases move over 140 pixels. Due to the misalignment of the estimated flow, the result of UPR-Large shows a wide range of ambiguities, such as the regions of sea and grass. VTinker uses high-quality texture mapping to recover detailed information in the intermediate proxy, which makes our result clearer. In Fig. 5, motion distances in each case exceed 200 pixels. Facing large movement, some methods (such as UPR-Large, SGM-1/2point, and PerVFI) show blurring and artifacts, and some methods (such as RIFE and FILM) display distortion and ghosting in the results. It is worth noting that the results of the SGM (using AFU-like upsampling method) show discontinuities in the boundaries (‘mosaic-like artifacts), which is due to the discontinuity in the upsampling flow processed by AFU (as displayed in Fig. 6). Notably, the proposed VTinker exhibits impressive performance in the details, such as text.

Ablation Experiments

Guided Flow Upsampling. In the GFU module, we introduce the input frame information as a reference to make the boundary of the upsampled flow more consistent with

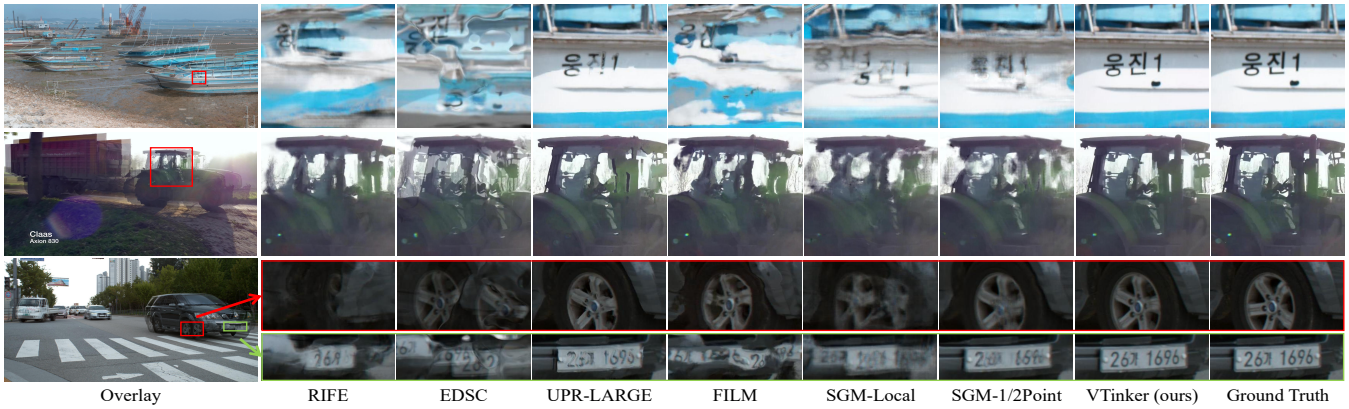

Figure 5: Qualitative comparisons among different methods on 4K. All cases move more than 200 pixels. Zoom in for details.

Figure 6: Comparison between various flow upsampling. Zoom in for mosaic-like and discontinuous details of the AFU. The left is the results, and the right shows the flow.

T.M.	Upsampling	Xiph-4K	Xiph-2K
×	Bilinear	0.079/0.036	0.040/0.019
×	AFU	0.081/0.031	0.039/0.017
×	GFU	0.082/0.033	0.038/0.016
√	Bilinear	0.080/0.034	0.037/0.018
√	AFU	0.078/0.031	0.037/0.017
√	GFU	**0.066/0.025**	**0.031/0.013**
	w/o GFU ⇒ GFU		VS
PSNR(Edge)↑	22.640 ⇒ 25.070		**10.73%↑**
SSIM(Edge)↑	0.7575 ⇒ 0.8143		**7.50%↑**
IoU(Edge)↑	0.2343 ⇒ 0.3926		**67.56%↑**

Table 3: Results of ablation experiments. Above the bold line: we show the metrics "LPIPS ↓ / DISTS ↓" on the dataset Xiph-2K and Xiph-4K (Niklaus and Liu 2020). T.M. denotes Texture Mapping. Below the bold line: comparison between w/o and w/ GFU on edge-related metrics. **RED**: best performance

the input frame. To verify the validity of the GFU module, we replace the GFU module with AFU (Luo et al. 2021a) and bilinear upsampling to train our model. According to Tab. 3, the use of GFU leads to better performance than AFU and bilinear upsampling. As shown in Fig. 6, the AFU estimates a 3×3 sampling kernel for each upsampled target pixel, which leads to discontinuities in the boundaries of both upsampling flow and result. Due to the model with AFU being trained on Vimeo90K, which is low-resolution, it does not generalize well to high resolution. In contrast, although VTinker is also trained on Vimeo90K, GFU achieves better high-resolution upsampling results, demonstrating its strong generalization capability. Our GFU module is modified based on the bilinear upsampling, and we introduce the input frame information to refine the upsampled boundary. When GFU and Texture Mapping are used together, the overall gain from GFU is minimal when evaluated with whole-image metrics. However, its advantages become more apparent in edge-region evaluations and visualizations. To assess performance in edge regions, we first apply the Canny operator to extract an edge mask from the image. We then compute PSNR, SSIM, and IoU metrics within this masked region. In Tab. 3, it shows that GFU improves the quality of the edge region of the inference result significantly. As displayed in Fig. 6, compared to the retrained model that uses bilinear upsampling, GFU produces sharper flow boundaries, resulting in clearer and more detailed outputs.

Texture Mapping. After processing with texture mapping, the region with ghosting and blurring is mapped by a high-quality block-texture, which improves the clarity of the interpolated results. As displayed in Tab. 3, the proposed VTinker trained without texture mapping would lead to considerable degradation of the model's performance. In particular, texture mapping can improve the quality of detail in the interpolated results for high-resolution interpolation.

Conclusion

We proposed VTinker, which consists of two core components: GFU and Texture Mapping. Extensive experiments have demonstrated that VTinker outperforms the state-of-the-art VFI methods and produces high-quality intermediate frames, especially in high-resolution video.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (62306153, 62225604), the Natural Science Foundation of Tianjin, China (24JCJQJC00020), the Young Elite Scientists Sponsorship Program by CAST (YESS20240686), the Fundamental Research Funds for the Central Universities (Nankai University, 070-63243143), and Shenzhen Science and Technology Program (JCYJ20240813114237048). The computational devices are partly supported by the Supercomputing Center of Nankai University (NKSC). This work was also supported by the OPPO Research Fund.

References

Cheng, X.; and Chen, Z. 2021. Multiple video frame interpolation via enhanced deformable separable convolution. *TPAMI*, 44(10): 7029–7045.

Choi, M.; Kim, H.; Han, B.; Xu, N.; and Lee, K. M. 2020. Channel attention is all you need for video frame interpolation. In *AAAI*, volume 34, 10663–10671.

Danier, D.; Zhang, F.; and Bull, D. 2022. FloLPIPS: A Bespoke Video Quality Metric for Frame Interpoation. *CoRR*, abs/2207.08119.

Ding, K.; Ma, K.; Wang, S.; and Simoncelli, E. P. 2022. Image Quality Assessment: Unifying Structure and Texture Similarity. *TPAMI*, 44(5): 2567–2581.

Hu, P.; Niklaus, S.; Sclaroff, S.; and Saenko, K. 2022. Many-to-many splatting for efficient video frame interpolation. In *CVPR*, 3553–3562.

Huang, Z.; Zhang, T.; Heng, W.; Shi, B.; and Zhou, S. 2022. Real-time intermediate flow estimation for video frame interpolation. In *ECCV*, 624–642.

Jain, S.; Watson, D.; Tabellion, E.; Poole, B.; Kontkanen, J.; et al. 2024. Video interpolation with diffusion models. In *CVPR*, 7341–7351.

Jin, X.; Wu, L.; Chen, J.; Chen, Y.; Koo, J.; and Hahm, C.-h. 2023a. A Unified Pyramid Recurrent Network for Video Frame Interpolation. In *CVPR*, 1578–1587.

Jin, X.; Wu, L.; Shen, G.; Chen, Y.; Chen, J.; Koo, J.; and Hahm, C.-h. 2023b. Enhanced bi-directional motion estimation for video frame interpolation. In *WACV*, 5049–5057.

Li, Z.; Zhu, Z.-L.; Han, L.-H.; Hou, Q.; Guo, C.-L.; and Cheng, M.-M. 2023. AMT: All-Pairs Multi-Field Transforms for Efficient Frame Interpolation. In *CVPR*, 9801–9810.

Liu, C.; Zhang, G.; Zhao, R.; and Wang, L. 2024. Sparse Global Matching for Video Frame Interpolation with Large Motion. In *CVPR*, 19125–19134.

Loshchilov, I.; and Hutter, F. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Luo, K.; Luo, A.; Wang, C.; Fan, H.; and Liu, S. 2021a. ASFlow: Unsupervised Optical Flow Learning With Adaptive Pyramid Sampling. *TCSVT*, 32: 4282–4295.

Luo, K.; Wang, C.; Liu, S.; Fan, H.; Wang, J.; and Sun, J. 2021b. Upflow: Upsampling pyramid for unsupervised optical flow learning. In *CVPR*, 1045–1054.

Niklaus, S.; and Liu, F. 2020. Softmax splatting for video frame interpolation. In *CVPR*, 5437–5446.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 8024–8035.

Perazzi, F.; Pont-Tuset, J.; McWilliams, B.; Gool, L. V.; Gross, M. H.; and Sorkine-Hornung, A. 2016. A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation. In *CVPR*, 724–732.

Reda, F.; Kontkanen, J.; Tabellion, E.; Sun, D.; Pantofaru, C.; and Curless, B. 2022. Film: Frame interpolation for large motion. In *ECCV*, 250–266.

Sim, H.; Oh, J.; and Kim, M. 2021. Xvfi: extreme video frame interpolation. In *ICCV*, 14489–14498.

Soomro, K.; Zamir, A. R.; and Shah, M. 2012. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*.

Sun, D.; Yang, X.; Liu, M.-Y.; and Kautz, J. 2018. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *CVPR*, 8934–8943.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *NeurIPS*, 5998–6008.

Wu, G.; Tao, X.; Li, C.; Wang, W.; Liu, X.; and Zheng, Q. 2024. Perception-Oriented Video Frame Interpolation via Asymmetric Blending. In *CVPR*, 2753–2762.

Xue, T.; Chen, B.; Wu, J.; Wei, D.; and Freeman, W. T. 2019. Video enhancement with task-oriented flow. *IJCV*, 127: 1106–1125.

Zhang, G.; Zhu, Y.; Wang, H.; Chen, Y.; Wu, G.; and Wang, L. 2023. Extracting motion and appearance via inter-frame attention for efficient video frame interpolation. In *CVPR*, 5682–5692.

Zhang, R.; Isola, P.; Efros, A. A.; Shechtman, E.; and Wang, O. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*, 586–595.