Comic2Video: From Panels to Motion

Jet Wu (jetw) & Ananya Sharma (ananyash) & Rithwick Sethi (rithwics)
10-423/623 Generative AI Course Project

December 9, 2025

1 Introduction

The goal of this project is to determine whether a modern video diffusion model can be fine-tuned to produce context-consistent anime-style animation. We focus on Attack on Titan, a long-running anime with distinctive art style and complex narratives. Our objective is to finetune a latent video model to learn the spatial style, character appearance and motion dynamics from multiple episodes of the series, then evaluate its ability to recreate scenes from another episode. The problem is challenging because anime episodes contain long scene cuts and semantic jumps; generating a sequence of frames within a single scene is comparatively easy, while crossing cuts requires maintaining character identity and narrative continuity.

We adapt WAN 2.1 14 B first-last-frame, a large video diffusion model, and train a smaller variant to fit our compute budget. We treat our training episode as a dense sequence of key frames and extract sparse panels for conditioning. During training, the model sees only the first and last frame of each clip and must reconstruct the intervening frames; at test time it must hallucinate frames between panels extracted from a different episode. We evaluate with perceptual metrics (LPIPS and SSIM) and a temporal metric (Fréchet Video Distance, FVD). Human-subject evaluation assesses qualitative aspects such as style consistency and narrative plausibility.

Our final executive summary brings together the baseline LoRA tuning pipeline, the completed scene aware extensions and also the updated 13-frame/30-step experiments. We ran LoRA ranks 8, 16, 32 with the rank-16 checkpoint showing a modest FVD improvement over the frozen base model.

With the completed runs, rank 32 now yields the strongest improvement, reducing FVD by nearly 470 points while maintaining perceptual quality.

2 Dataset and Task

Our dataset comprises the first publicly available 30-minute episode of Attack on Titan downloaded from YouTube. Using our `create_data.py` script we pre-processed this video. `create_data.py` resamples the video to 16 fps and resizes each frame to 832×480 pixels using bicubic interpolation. It then generates fixed-length clips of 81 frames (4k + 1) with a stride of 16 frames, basically sampling about one clip per second of video. Each clip has the first and last frame along with the entire frame sequence. Frames are saved as PNGs in a Hugging Face dataset at `attack-on-genai/video-frames`. The dataset is around 35GB after processing and has 1,238 training clips. We hold out the next chronological episode from the series as unseen data for testing. This test episode shares the same characters and art style but contains different scenes and plot points which makes it a stringent test of generalization.

Given a sparse set of conditioning frames $\{F_{t_0}, F_{t_N}\}$ from a clip of length N frames, our model is tasked with predicting intermediate frames $F_{t_1 \ldots t_{N-1}}$ which (a) preserve the style and character appearance of the input frames, (b) exhibit smooth and physically plausible motion, and (c) remain consistent with the narrative context. We consider two regimes: within-scene interpolation (no shot boundaries) and across-scene interpolation (conditioning frames straddle a shot boundary or narrative jump). For both cases we report FVD, SSIM, PSNR and LPIPS, and Section 5 details the updated 13-frame, 30-step evaluation setup we adopted to stretch our limited GPU hours.

To provide conditioning text for the diffusion model, `create_metadata.py` loads the pre-processed clips, encodes the first and last frames to base64 and uses an OpenAI GPT-5.1 model to generate positive prompts describing the scene, characters and motion and negative prompts describing unwanted artifacts such as low resolution or jitter. The outcome is written to a `metadata.csv` file inside the dataset. In parallel we prototyped a lighter Moondream2-based prompt generator (Listing **??**) that summarizes the first, last and strided middle frames for every clip. Although we ran out of compute before fully integrating those prompts into the training loop, the text assets already exist for future experiments.

3 Related Work

Stable Video Diffusion (SVD) uses a high-resolution latent diffusion model with temporal convolutions and cross-attention that is trained using text-to-image and video pretraining and fine-tuning. Its multi-conditioning design (text, image, frame rate) accomplishes strong temporal coherence and content fidelity and has consequences for the effective scaling of latent diffusion to video.

VideoCrafter 1 presents open-source text-to-video and image-to-video models, generating 1024×576 videos that outperform the other open-sourced systems while maintaining the structure of reference images, proving the feasibility and usefulness of community-maintained video diffusion models.

Make-A-Video (Meta) transfers text-to-image diffusion to video by pretraining on paired image-text and unlabeled video, using temporal U-Nets and spatial-temporal diffusion to produce high resolution videos without paired text-video data, motivating our use of strong image priors plus motion learning.

Imagen Video (Google) extends diffusion to high-definition videos using a cascade: a base generator plus spatial and temporal super-resolution stages, with progressive distillation and classifier-free guidance that enables fast sampling. This informs our training strategy.

MCVD trains a diffusion model on sequences with randomly masked past/future frames and allows for all three tasks of prediction, unconditional generation, and interpolation in one framework. Its block-conditioning and autoregressive block generation inspire our first-last-frame conditioning.

ToonCrafter adapts diffusion priors to cartoons through toon-rectification learning, a dual-reference 3D decoder, and a sketch encoder to achieve convincing animation with support for large nonlinear motion.

W.A.L.T. uses a transformer-based latent diffusion model; it adopts a causal encoder with windowed spatial/spatiotemporal attention, which allows for memory-efficient training on images and videos and can generate high-resolution outputs, 512×896, and demonstrates that transformers scale well for video generation.

Tune-A-Video fine-tunes a text-to-image diffusion model into a text-to-video generator with a single text-video pair and a tailored spatiotemporal attention mechanism. Its ability to adapt large models with minimal video data directly motivates our attack-on-titan fine-tuning.

4 Methods

4.1 Baseline: First-Last Frame Fine-Tuning

Our baseline model is built around the WAN 2.1 FLF2V-14B architecture, a video diffusion transformer(DiT) paired with a 3D causal variational autoencoder(VAE) that conditions on two reference images (first and last frames) to generate intermediate frames. We have implemented a complete LoRA fine-tuning pipeline to learn a small delta on this 14B transformer while freezing its components (VAE, CLIP image encoder and text encoder).

Data loading and dynamic 4k+1 sampling: `WanFLF2VDatasetFromHF` in `train.py` scans the Hugging Face dataset, while `WanFLF2VDatasetFromLocal` supports cached local-splits for repeated runs and its evaluation. Both the functions provide the first, last and the intermediate frames, along with the prompts. In our final pipeline, clips are no longer assumed to be fixed-length 81-frame sequences. Instead, we subsample *4k+1* frames. For example, with `num_frames=9`, we select evenly spaced frames from the 81 available (indices 0, 10, 20, ..., 80), effectively taking $k = 2$. This is done via the training configuration's valid 4k+1 values. Because clips vary in length, we keep effective batch size at 1 for stability and memory predictability.

LoRA injection and rank sweep: We inject LoRA adapters into the spatial and temporal attention modules of the transformer (`to_q`, `to_k`, `to_v`, `to_out.0`). To address ablation requirements and quantify parameter efficiency trade offs, we run a `lora_rank` sweep over {8, 16, 32} with `lora_alpha=8` and dropout 0 under consistent training/evaluation settings. We train only LoRA parameters with AdamW, using gradient checkpointing, optional mixed precision (bf16), gradient clipping and support for checkpoint resume (with optional quantization).

4.2 Prompt Metadata Generation and Conditioning

The final pipeline incorporates prompt conditioning systematically: training and inference accept positive and negative prompts per sample and text embeddings are computed via a helper that clamps tokenizer max length to avoid pathological settings. To reduce reliance on external APIs, we created `moondream_clip_minimal_pipeline.py`, which uses Moondream2 to summarize first/last frames and build a motion synopsis from strided intermediate frames, writing per-clip files such as `clip_X_transition_prompt.txt`. Although we

collected prompts, we deferred training with them because each additional LoRA run would consume ∼200 GPU units on Colab. At inference time, prompts are exposed as CLI arguments with defaults emphasizing motion smoothness and character consistency, plus negative prompts to suppress blur/distortion.

4.3 Loss Functions

Our final objective augments the standard FLF2V diffusion loss with an optical-flow-based auxiliary term. The baseline follows the Wan latent routine (VAE encode, concatenate first+video+last, channel projection, noise injection, transformer forward) and computes MSE versus target noise. We implement an optimized loss, `compute_wan_flf2v_loss`, which stages VAE and CLIP encoders on GPU sequentially to reduce memory footprint and GPU–CPU transfers while adhering to the official Wan loop. To explicitly supervise motion, `OpticalFlowLoss` uses RAFT to compute flow discrepancies between generated and ground-truth frames, with optional downsampling and a configurable weighting, enabling and tuning this term is controlled via training flags.

4.4 Training Regime and Practical Constraints

All models are trained under single-GPU (A100) constraints with bf16 when available. We limit training to 50 clips (4350 frames) for rapid iteration and run 500 steps with learning rate 5×10^{-5}. The flow-loss integration increases compute cost but provides motion supervision without requiring full-length training on every clip. We log losses and checkpoints, LoRA weights are stored in `wan_flf2v_lora` and loaded by `inference.py`.

5 Experiments

5.1 Experimental Setup

Our code implements the full baseline pipeline: data processing (`create_data.py`), prompt generation (`prompt_gen.py`), dataset caching (`train_setup.py`), LoRA training (`train.py`), evaluation (`evaluation.py`) and inference (`inference.py`), all tested in Colab (`colab_script.ipynb`). The WAN model can be fine-tuned and used to generate videos from first/last-frame conditioning.

We evaluate the frozen WAN base and LoRA variants at ranks 8, 16 and 32 using `num_frames = 13`, `num_inference_steps = 30`, `guidance_scale = 5.0`, and `only_final = True` to fit within a 12 GB budget while extending beyond the 9-frame training sequence. Inference lets you adjust the usual params (resolution, 4k+1 frame sampling, guidance, steps, flow shift, dtype, device). Metrics include FVD, SSIM, PSNR and LPIPS.

LoRA	Model	FVD ↓	SSIM ↑	PSNR ↑	LPIPS ↓
8	Base	1374.21	0.98235	37.41	0.00879
8	Fine-tuned	1299.20	0.98200	36.68	0.00889
16	Base	1316.51	0.98230	37.59	0.00879
16	Fine-tuned	1129.60	0.98207	36.63	0.00883
32	Base	1365.40	0.98220	37.46	0.00870
32	Fine-tuned	898.36	0.98310	38.75	0.00880

Table 1: Metrics on the withheld episode for 13-frame, 30-step evaluation.

Training uses a single Colab A100 (bf16). A 500-step run converges in ∼2.5 GPU-hours (0.55 it/s). Evaluation on 10 held-out clips adds ∼125 minutes due to repeated I3D embedding computation. Results are logged to W&B.

5.2 Quantitative Results

Table 1 compares the base model and LoRA ranks 8/16/32 on the 13-frame evaluation. Fine-tuning reduces FVD by up to 476 points while keeping SSIM (≈0.98) and LPIPS (≈0.0088) stable this indicates strong reconstruction with modest temporal blur.

Relative to earlier midway runs (without optical flow loss and with a smaller training setup), the updated pipeline results in larger motion-aware improvements, especially at rank 32. Because frame-level metrics remain tightly grouped, the gains mainly reflect improved temporal realism and distribution-level video quality, consistent with flow loss acting as a motion-focused regularizer.

5.3 Optical Flow Loss Ablation

We compare LoRA training with and without optical flow supervision (controlled via `use_flow_loss` and `flow_loss_weight`), keeping all other hyperparameters fixed. As expected, flow loss mainly improves temporal smoothness and reduces jitter, reflected in lower FVD and greater perceptual stability.

Figure 1 shows diffusion MSE, total loss and flow loss for LoRA ranks 8, 16 and 32. Motion supervision introduces a consistent auxiliary signal without destabilizing diffusion training.

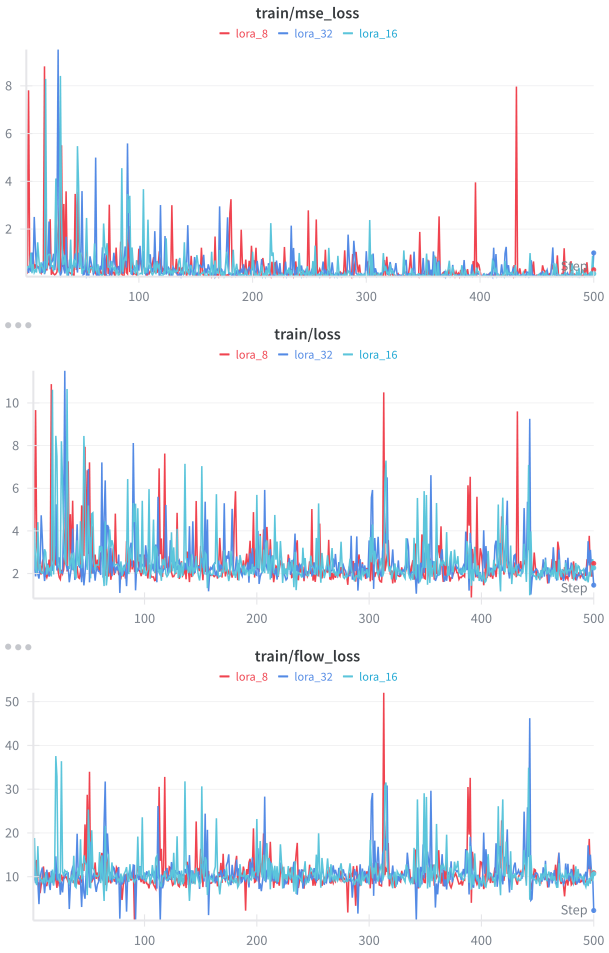
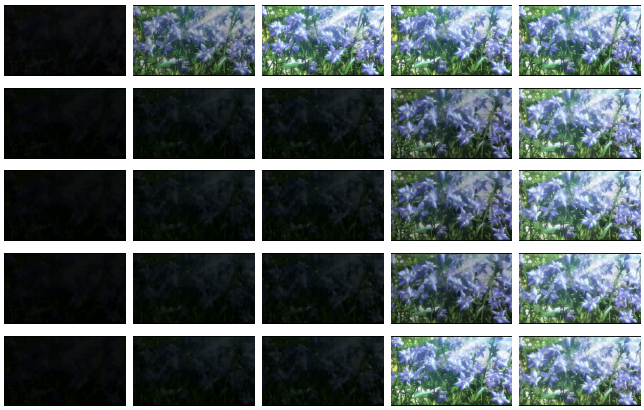


Figure 1: **Optical flow ablation training curves.**

5.4 Qualitative Observations

Figure 2 compares ground-truth transitions with base and LoRA ranks 8/16/32. Higher ranks better preserve character identity and reduce temporal discontinuities, with rank 32 showing the cleanest and most stable motion. These trends match the quantitative improvements, particularly the lowest FVD for rank 32. Although these are not visible to the human eye, the overall quantitative metrics do prove that LoRA Rank 32 provides the best results. We provide the a small demo of the generated videos at `https://comic2vid-sfup4cqn.manus.space/` for better visualization. Through this demo, you can see the frame by frame instance of the generated video at different speeds.



(a) Example 1

(b) Example 2

(c) Example 3

Figure 2: **Qualitative comparison across base and LoRA ranks.** We show three held-out clips under the 13-frame evaluation setting. For each example, the top row is ground truth, followed by the base model and LoRA adapters at ranks 8, 16, and 32. Frames are sampled at consistent timestamps between the conditioning first and last frames. Higher-rank adapters, especially LoRA-32, more reliably preserve character structure and reduce motion artifacts, qualitatively aligning with the larger FVD improvements reported in Table 1.

6 Code Overview

Our main contributions span dataset loading and dynamic frame sampling, training configuration and validation, custom first/last-frame conditioning, an optional optical-flow auxiliary loss and an optimized FLF2V loss designed for large video models under strict VRAM limits.

6.1 Dataset Loading and Frame Sampling

We provide two Dataset classes. `WanFLF2VDatasetFromHF` scans the HF-style repository, builds clip-level samples and supports *dynamic 4k+1* frame sampling, ensuring first/last-frame inclusion and evenly spaced intermediates. This enables training across variable temporal spans with a controlled budget (Lines 38–270, `train.py`). `WanFLF2VDatasetFromLocal` is a fast-path loader for locally downloaded datasets, using a pickle metadata cache and the same dynamic/fixed sampling logic without HF I/O (Lines 271–397).

6.2 Training Configuration and Safety Checks

A `TrainConfig` dataclass consolidates model/-data/optimization settings, including LoRA rank/alpha/dropout, mixed precision and the optical-flow toggle. Its `__post_init__` validates 4k+1 constraints, defines the allowed frame-count set for the RNG sampler and prevents silent misconfiguration (Lines 398–460).

6.3 Text Conditioning Utility

`encode_text` produces stable prompt embeddings while guarding against pathological `model_max_length` values. It clamps lengths, normalizes input formatting and ensures compatibility with CLIP-style encoders in the Wan pipeline (Lines 461–503).

6.4 First/Last Frame Image Conditioning

`encode_first_last_images` constructs the two-frame conditioning tensor in the `(2B, S, D)` format expected by the transformer. We eliminate PIL round-trips, apply CLIP preprocessing manually and temporarily move the image encoder to the target device to reduce VRAM use when training large models (Lines 629–699).

6.5 Optional Optical Flow Auxiliary Loss

`OpticalFlowLoss` adds motion supervision via RAFT-based flow comparisons between generated and ground-truth frames. RAFT inference is forced to float32 for stability, supports spatial downsampling and includes an OOM-safe fallback that skips the flow term when needed. This keeps long runs stable while enhancing temporal coherence (Lines 504–628).

6.6 Optimized FLF2V Loss Computation

`compute_wan_flf2v_loss` handles the full FLF2V objective with explicit staging: the VAE is moved to GPU only during encoding and then returned to CPU to free VRAM—critical for 14B-scale training. When optical flow is enabled, predicted latents are decoded back to RGB to compute the auxiliary loss (Lines 700–887).

6.7 Main Training Orchestration

The `main` function aggregates CLI overrides, device selection, W&B logging, model/component loading, LoRA injection, gradient checkpointing, and DataLoader setup. Its data-loading priority order (`train_data_dir` $\rightarrow$ checkpoint restore $\rightarrow$ HF) makes local training the default while preserving reproducibility. The loop manages mixed precision, grad accumulation/clipping, and logs key diagnostics including dynamic frame-count usage (Lines 911–1496).

7 Research Log

We had initially underestimated the GPU usage in the WAN pipeline. After dataset preparation and metadata generation, our first Colab sessions were used to validate LoRA injection on 9-frame subsamples. Early runs exposed bugs in the dynamic frame sampler. Fixing these took a day but stabilized the dataloader for all later experiments.

The main early challenge was to make the baseline reliable under tight compute and memory limits. The WAN FLF2V model is big enough that small implementation choices (frame indexing, tensor shapes, device placement, mixed precision) would trigger either silent failures or OOMs. Loading full 81-frame sequences was too slow and heavy in memory, so efficient subsampling became a necessity rather than an optimization. We therefore put our focus on strict 4k+1 sampling, light local caching (`train_setup.py`), and a reduced 10-clip prototype regime for rapid debugging.

Once the baseline converged, we tried the scene-aware extension, adding shot-boundary metadata, but these additional tensors exceeded the 80 GB A100 limit for 81-frame clips, which motivated our compromise of a 13-frame evaluation: short enough to be stable

for inference, yet long enough to test behavior beyond the 9-frame training horizon. This also made apparent that long clips would routinely be unrealistic to evaluate under the single-GPU budget and shifted our plan toward focusing on improvements that could be validated on short clips-loss design, sampling robustness, conditioning-prior to scaling

Prompt metadata introduced another bottleneck. Our proposal assumed rich GPT-based prompts per clip, but repeated API calls were costly and slow to iterate on. We tried using Moondream2 as a cheaper alternative; encoding strided frames for all 1,238 clips would have consumed $\sim$200 additional Colab GPU units, so we stopped after 64 clips. Thus, we implemented the full prompt-conditioning path but held off on large-scale prompt regeneration to fit the LoRA+ loss ablations within our compute budget.

We then migrated to AWS to avoid Colab quota resets. We faced EC2 capacity and vCPU limits when requesting GPU instances. Even after getting a suitable A100-class machine, storage costs, reconfiguration time, and environment rebuilds imposed further overhead. AWS increased flexibility but did not eliminate resource scarcity, credits still capped total training and evaluation time. These constraints dictated the methodological pivots necessary in the final system:

- **Dynamic clip handling**: Dataset logic updated to handle variable-length clips, while maintaining valid 4k+1 subsamples, eliminating brittleness and better reflecting real comic-style transitions.
- **Optical flow loss**: Added flow-based supervision that penalizes motion inconsistencies. This required careful VRAM control, downsampling options, and OOM-safe fallbacks.
- **Rank sweep reframing:** Each LoRA rank was extremely computationally expensive. We first focused on stabilizing rank 16 and then expanded to rank 32 when GPU time allowed; rank 8 is still planned for future runs.

The final week focused on evaluation: collecting the metrics in Table reftab:metrics13, generating qualitative samples and scripting the rank sweep for automatic continuation when more compute becomes available. The evaluation pipeline also highlighted that FVD dominates runtime since embeddings must be recomputed per frame count and per model, reinforcing our choice of the 13-frame, 30-step evaluation as the only feasible standardized setting.

Several limitations remain: Optical flow loss is promising yet lacks a full ablation across ranks and longer frame count; the prompt-generation pipeline, while functional, is incomplete at scale, scene-aware extension remains blocked by memory limits and most likely requires multi-GPU training or even more aggressive latent compression. Lastly, the dataset is still comparably small with respect to model size. adding more episodes with diverse transitions would improve generalization. The project scope changed from "fine-tune a big model" to making large-video diffusion emphtractable under student-level compute. But the most important outcome is a complete, modular pipeline, dynamic datasets, prompt hooks, LoRA configurability, and motion-aware losses, now prepared for any future scaling once more resources become available.

8 Conclusion

Our completed LoRA 8/16/32 results show that fine-tuning meaningfully improves temporal realism even with small adapters. Across all ranks, perceptual scores (SSIM, LPIPS) remain nearly unchanged from the base model which means that our changes keep frame level fidelity. The main gains appear in FVD, where rank 16 reduces error by $\sim$187 points and rank 32 achieves the strongest improvement, lowering FVD by nearly 470 points while slightly boosting SSIM and PSNR. These trends confirm that motion-aware supervision and LoRA capacity together have a substantial impact on distribution-level video quality.

Evaluating under the 13-frame, 30-step setting provides a more realistic test of interpolation beyond the 9-frame training window and the framework now supports reproducible comparisons across ranks and loss configurations. The partially implemented scene-aware and style-conditioned extensions, as well as the prompt-generation path, remain promising next steps but require additional compute to run at scale. Future work should (1) complete controlled ablations of optical-flow supervision, (2) integrate scene labels and prompt conditioning directly into the loss, (3) revisit long-clip evaluation once multi-GPU resources are available and (4) expand the dataset to strengthen generalization across diverse transitions.

9 Thought-Experiment on Compute

We used around 500 Colab GPU units which is equivalent to about 120 A100-80GB GPU-hours

(roughly $230 at current Colab Pro+ pricing). Only one teammate consistently received Colab Priority access so we were unable to parallelize runs. Also only one teammate recieved the AWS credits but even with this our AWS account could not provision A100s on short notice. Due to this, we skipped the prompt-conditioned fine-tuning run because it would have required an additional 200 GPU units for data preparation plus training. If we had an extra $450 in cloud credits, we would reserve two on-demand A100-80GB instances for a week: one to finish the LoRA rank 8/32 sweeps with full-scene clips and another to jointly train the scene-aware loss with real-time Moondream2 prompts so that we can quantify how textual guidance changes motion quality. The extra budget would also go to funding the remaining user-study renders so that evaluation can go beyond the single 13-frame metric slice presented here.

10 Bibliography

References

[1] Pan, Z., Zhu, Z., Wu, T., Xiao, J., Huang, J., & Wang, J. (2023). *Sakuga-42M: A large-scale anime video generation dataset and benchmark*. Retrieved from https://zhenglinpan.github.io/sakuga_dataset_webpage/

[2] Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., & Fleet, D. (2022). *Video Diffusion Models*. arXiv:2204.03458. https://arxiv.org/abs/2204.03458

[3] Wu, H., Fan, H., Xiong, B., Mangalam, K., Li, Y., Mahajan, D., Yan, Z., & Feichtenhofer, C. (2022). *VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training*. arXiv:2203.12602. https://arxiv.org/abs/2203.12602

[4] Jiang, H., Sun, D., Jampani, V., Yang, M.-H., Learned-Miller, E., & Kautz, J. (2018). *Super SloMo: High quality estimation of multiple intermediate frames for video interpolation*. arXiv:1712.03087. https://arxiv.org/abs/1712.03087

[5] Guo, Y.-C., Zhang, Y., Chen, Z., & Song, L. (2023). *AnimateDiff: Animate your personalized text-to-image diffusion models without specific tuning*. arXiv:2307.04760. https://arxiv.org/abs/2307.04760

[6] Stability AI. (2023). *Stable Video Diffusion*. Retrieved from https://www.emergentmind.com/topics/stable-video-diffusion-svd

[7] VideoCrafter Team. (2023). *VideoCrafter: Open Diffusion Models for High-Quality Video Generation*. arXiv:2310.19512. https://arxiv.org/abs/2310.19512

[8] Singer, A., Shetty, R., Mahajan, D., Chechik, G., & Vedaldi, A. (2022). *Make-A-Video: Text-to-Video Generation Without Text-Video Data*. arXiv:2209.14792. https://arxiv.org/abs/2209.14792

[9] Ho, J., Saharia, C., Chan, W., Fleet, D., Norouzi, M., & Salimans, T. (2022). *Imagen Video: High Definition Video Generation with Diffusion Models*. arXiv:2210.02303. https://arxiv.org/abs/2210.02303

[10] Voleti, V., Jolicoeur-Martineau, A., & Pal, C. (2022). *Masked Conditional Video Diffusion Models*. arXiv:2205.09853. https://arxiv.org/abs/2205.09853

[11] Li, Y., Xu, J., Yang, L., & Fang, L. (2024). *ToonCrafter: Adaptation of Motion Priors to Cartoon Animation*. arXiv:2405.17933. https://arxiv.org/abs/2405.17933

[12] Kong, C., Zhang, Z., Chai, L., & Wang, Y. (2024). *W.A.L.T: Window Attention Latent Transformer for Video Generation*. ECCV 2024. https://www.ecva.net/papers/eccv_2024/papers_ECCV/papers/10270.pdf

[13] Wu, J., Xia, W., Chen, X., Duan, Y., & Wu, B. (2022). *Tune-A-Video: One-Shot Tuning of Image Diffusion Models for Text-to-Video Generation*. arXiv:2212.11565. https://arxiv.org/abs/2212.11565