Open-Sora 2.0: Training a Commercial-Level Video Generation Model in $200k

Open-Sora Team

HPC-AI Tech

Abstract

Video generation models have achieved remarkable progress in the past year. The quality of AI video continues to improve, but at the cost of larger model size, increased data quantity, and greater demand for training compute. In this report, we present Open-Sora 2.0, a commercial-level video generation model trained for only $200k. With this model, we demonstrate that the cost of training a top-performing video generation model is highly controllable. We detail all techniques that contribute to this efficiency breakthrough, including data curation, model architecture, training strategy, and system optimization. According to human evaluation results and VBench scores, Open-Sora 2.0 is comparable to global leading video generation models including the open-source HunyuanVideo and the closed-source Runway Gen-3 Alpha. By making Open-Sora 2.0 fully open-source, we aim to democratize access to advanced video generation technology, fostering broader innovation and creativity in content creation. All resources are publicly available at: https://github.com/hpcaitech/Open-Sora.

1 Introduction

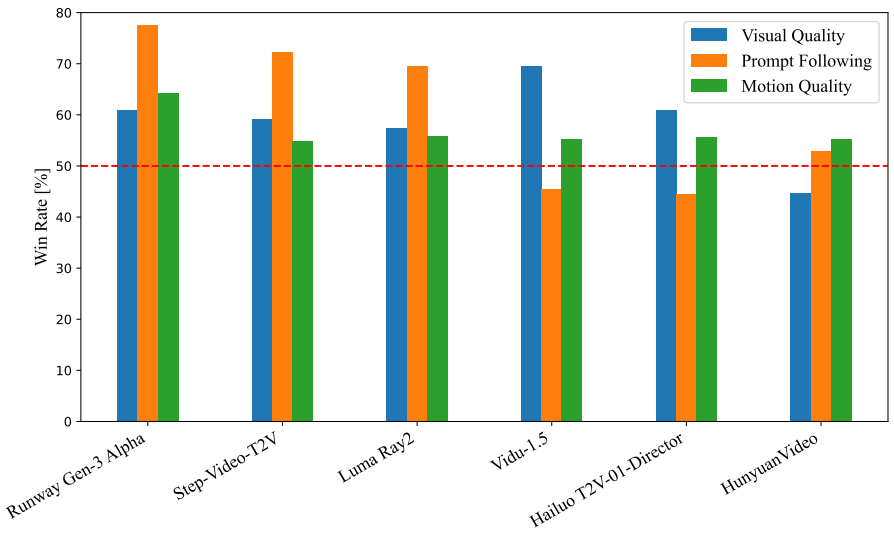


Figure 1: Human preference evaluation of Open-Sora 2.0 against other leading video generation models. Win rate represents the percentage of comparisons where our model was preferred over the competing model. The evaluation is conducted on 100 prompts carefully designed to cover three key aspects: 1) visual quality, 2) prompt adherence, and 3) motion quality. Results show that our model performs favorably against other top-performing models in all three aspects.

The past year has witnessed an explosion of video generation models. Since the emergence of OpenAI's Sora [4] in February 2024, a series of video generation models—either open-source [16, 19, 22, 27, 43] or proprietary [26, 31, 34]—have appeared at an unprecedented pace, each striving to achieve "Sora-level" generation quality. While the quality of generated videos continues to improve, there is a clear trend toward rapid growth in model size, data quantity, and computing resources. Following the success of scaling large language models (LLMs) [15, 18], researchers are now applying similar scaling principles [19] to video generation, converging on similar model architectures and training techniques.

In this report, however, we show that a top-performing video generation model can be trained at a highly controlled cost, offering new insights into cost-effective training and efficiency optimization.. We build Open-Sora 2.0, a commercial-level video generation model trained for only $200k. As shown in Table. 4, the training cost is 5-10 times lower than comparable models like MovieGen [31] and Step-Video-T2V [27] based on a fair comparison. This remarkable cost efficiency stems from our joint optimization of data curation, training strategy, and AI infrastructure—all of which are detailed in the following sections.

We show the human preference evaluation of Open-Sora 2.0 and other global leading video generation models in Figure 1. The models for comparison include proprietary models like Runway Gen-3 Alpha [34], Luma Ray2 [26] and open-source models like HunyuanVideo [19] and Step-Video-T2V [27]. We evaluate these models in three important aspects: 1) visual quality, 2) prompt adherence, and 3) motion quality. Despite the low cost of our model, it outperforms these top-performing models in at least two aspects out of the three, demonstrating its strong potential for commercial deployment.

This report is structured as follows. In Section 2, we elaborate on our data strategy, including our hierarchical data filtering system and annotation methods. Section 3 details our model architecture, covering both our novel Video DC-AE autoencoder design and the DiT architecture. Section 4 explores our cost-effective training methodology, which enables commercial-level quality at just $200k. Section 5 presents our conditioning approaches, including image-to-video and motion control techniques. Section 6 outlines the system optimizations that maximize training efficiency, and Section 7 evaluates our model's performance against leading video generation models. Finally, we conclude with a summary of our contributions and insights for future research.

2 Data

Our goal for data is to build a hierarchical data pyramid, catering to the requirement of the progressive training process. To this end, we develop a collection of filters that function distinctly from each other, aiming to tackle various types of data detections. By progressively strengthening the degree of filtration, we can obtain subsets of smaller sizes but higher purity and quality.

In Section 2.1, we first elaborate the data filtering system that consists of two main components: preprocessing and score filtering. Then, we introduce data annotation methods in Section 2.2. Finally, we present detailed statistics of the whole dataset in Section 2.3.

2.1 Data Filtering

The hierarchical data filtering system is illustrated in Figure 2. We begin by preprocessing raw videos into trainable video clips, then progressively apply a series of filters, ranging from loose to strict, to construct a structured data pyramid.

2.1.1 Preprocessing

The preprocessing phase converts raw videos into short clips suitable for training. During this phase, we first eliminate broken files and raw videos with outlying attributes. Specifically, we filter out videos with a duration of less than 2 seconds, a bit per pixel (bpp) below 0.02, a frame rate (fps) under 16, and an aspect ratio outside the range [1/3, 3], and those with a "Constrained Baseline" profile.

Then, we detect continuous shots within the raw video and segment it into short clips based on the detection result. For shot detection, we apply the libavfilter from FFmpeg [12] to calculate the scene scores, which quantify visual differences among frames. The shot boundaries are identified at points where the change in scene scores exceeds an empirically determined threshold.

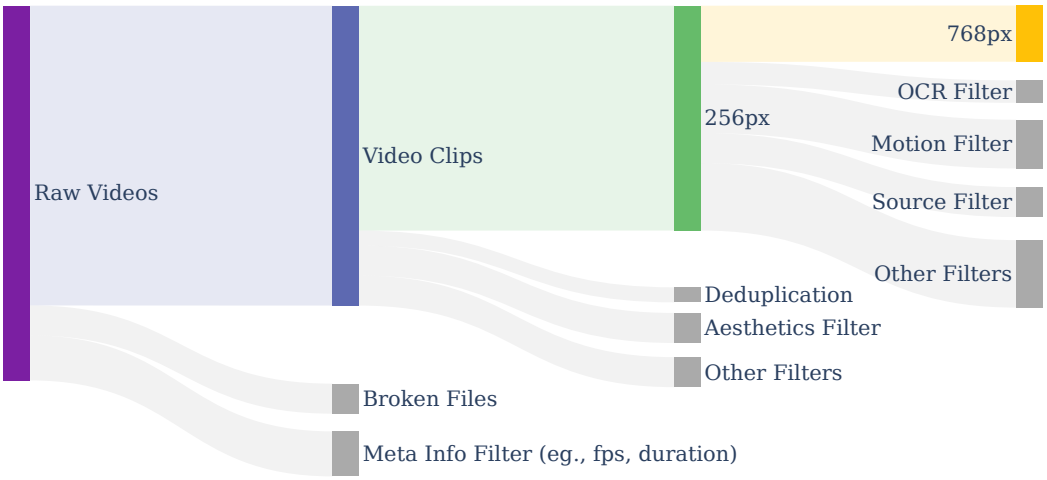


Figure 2: **The hierarchical data filtering pipeline**. The raw videos are first transformed into trainable video clips. Then, we apply various complimentary score filters to obtain data subsets for each training stage.

Finally, we segment the video based on the shot detection results, ensuring that the output clips adhere to specific format constraints: frame rate (fps) below 30, longer dimension not exceeding 1080 pixels, and H.264 codec. Additionally, black borders are removed during this process. For those shots exceeding 8 seconds, we further divide them into multiple 8-second clips, while shots shorter than 2 seconds are discarded.

2.1.2 Score Filtering

To address the various defects in raw data, we develop a bag of complimentary filters, each targeting a specific aspect of data quality. These filters work together as a comprehensive and robust purification system. Typically, each filter evaluates a sample by assigning a score based on its respective criteria, and the filtering intensity is controlled by a threshold. We introduce all the score-based filters in the following sections.

Aesthetic Score. We assess the aesthetic quality of a sample using the CLIP+MLP aesthetic score predictor [35]. For a video sample, we extract the first, middle, and last frame, compute their individual scores, and take the average as the final aesthetic score.

Motion Score. The motion intensity of a video is measured using the VMAF motion score from FFmpeg's libavfilter library [12]. Videos with extremely low or excessively high motion scores are filtered out to maintain a balanced motion range.

Blur Detection. To evaluate image clarity, we apply the Laplacian operator from OpenCV [3]. A sample is considered blurry if the variance of the Laplacian image falls below a configurable threshold. For video samples, we extract five uniformly spaced frames and adopt a majority voting approach to determine blurriness.

OCR. We detect text bounding boxes using PaddleOCR [2] and compute the total area of bounding boxes with a confidence score above 0.7. Images or videos containing excessive text are discarded to avoid unwanted overlays.

Camera Jitter Detection. We detect camera jitter using Shot Boundary Detection from the PySceneDetect library [6]. A video is identified as having camera jitter if the average frame-to-frame change exceeds a predefined threshold.

2.2 Data Annotation

For captioning, we utilize the open-source vision-language model LLaVA-Video [42] to annotate 256px videos. We prompt the model to focus on six aspects for a detailed and comprehensive caption, which are 1) main subjects; 2) subjects' actions; 3) background and environment; 4) lighting condition and atmosphere; 5) camera movement; 6) video style, such as realistic, cinematic, 3D, animation, and so on. For high-resolution 768px training data, we leverage a stronger proprietary model Qwen 2.5 Max [38] to generate more accurate and semantically aligned captions. We find that Qwen 2.5 Max produces fewer hallucinations and delivers better semantic consistency than LLaVA-Video.

For both training and inference, we append the motion score after the caption to enable configurable control over motion strength in the generated videos.

2.3 Data Statistics

We conduct a statistical analysis of some key attributes of the video data, including aesthetic scores, duration (seconds), aspect ratios (height/width), and the length of the caption. To further explore the distribution of text content, we visualize common words in video captions using a word cloud.

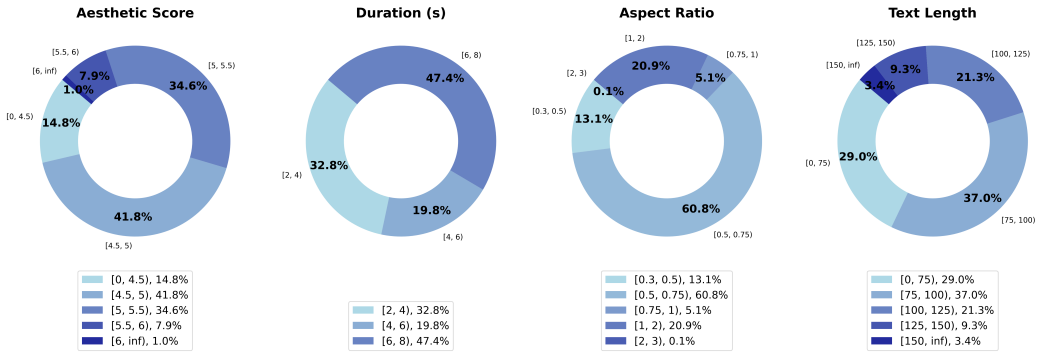


Figure 3: Distribution of key attributes of the whole video dataset.

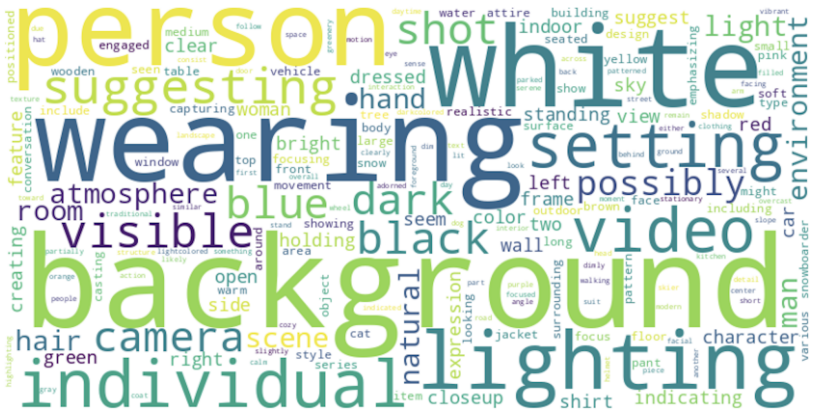


Figure 4: Word cloud of the video captions.

As shown in Fig. 3, the majority of videos have aesthetic scores ranging between 4.5 and 5.5, indicating a generally moderate level of visual appeal. Video durations range from 2 to 8 seconds, with nearly half of the dataset consisting of clips between 6 and 8 seconds, which offer richer temporal information for learning dynamic patterns. The analysis of aspect ratios shows that the majority fall between 0.5 and 0.75, corresponding to the 16:9 format, ensuring adaptability across different formats and enhancing the model's generalization. Additionally, over 70% of video captions exceed 75 words, providing detailed descriptions that contribute to a more informative training process.

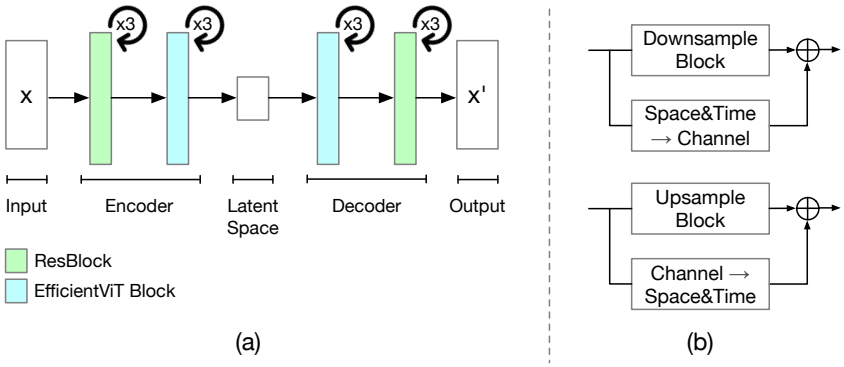


Figure 5: **Architecture of Video DC-AE.** (a) Overview of Video DC-AE: Each block in encoder introduces spatial downsampling, while temporal downsampling occurs at blocks 4 and 5, with a corresponding symmetric structure in the decoder. (b) Residual Connection in Video DC-AE Blocks.

Figure 4 presents a word cloud illustrating the vocabulary distribution in video captions. The captions encompass not only main subjects such as *"person"* and their actions like *"wearing"*, but also background elements (*"background"*, *"setting"*, *"atmosphere"*) and lighting conditions (*"lighting"*). This aligns with the six key aspects of our VLM prompting strategy introduced in Sec. 2.2. Furthermore, the frequent occurrence of *"person"* and *"individual"* suggests that a substantial portion of the dataset features human subjects.

3 Model Architecture

The two key components of a video generation model are the autoencoder and the diffusion transformer. For the autoencoder, our model is initially trained on HunyuanVideo's VAE and later adapted to our Video DC-AE, whose structure is detailed in Sec. 3.1. The architecture of the diffusion transformer is presented in Sec. 3.2.

3.1 3D Autoencoder

We begin by leveraging the open-source HunyuanVideo VAE [19] as the initial autoencoder for our model. To further reduce both training and inference costs, we develop a video autoencoder with deep compression (Video DC-AE, named after [9]) that improves efficiency while maintaining high reconstruction fidelity.

HunyuanVideo VAE achieves a compression ratio of $4 \times 8 \times 8$, which allows an 8-second, high-resolution (1280×720) video at 16 FPS to be transformed into a latent representation of 32×160×90. However, even with a patch size of 2×2, our generation model still needs to process around 115K tokens per training video, resulting in considerable computational demands.

To address the above challenge, we identify a potential redundancy in the spatial dimension and propose to increase the spatial compression ratio to 32 while keeping the same temporal compression ratio at 4. This adjustment effectively reduces the number of spatial tokens while preserving essential motion features in our Video DC-AE. Since our AE training data consists of 32-frame, 256px videos,

Model	Down. (TxHxW)	Info. Down.	Channel	Causal	LPIPS↓	PSNR↑	SSIM↑
Open Sora 1.2 [43]	$4 \times 8 \times 8$	192	4	✓	0.161	27.504	0.756
StepVideo VAE [27]	$8 \times 16 \times 16$	96	64	✓	0.082	28.719	0.818
HunyuanVideo VAE [19]	$4 \times 8 \times 8$	48	16	✓	**0.046**	30.240	0.856
Video DC-AE	$4 \times 32 \times 32$	96	128	×	0.051	30.538	0.863
Video DC-AE	$4 \times 32 \times 32$	48	256	×	0.049	**30.777**	**0.872**

Table 1: **Auto-encoder reconstruction performance comparison**. The Video DC-AE highlighted with yellow background is selected for generative model adaptation.

this new compression ratio allows us to train the autoencoder at a latent shape of $8 \times 8 \times 8$, which maintains the same level of information in all dimensions.

Chen *et al.* [9] introduce DC-AE as an effective approach to achieve higher downsampling ratios while maintaining reconstruction quality. While the original DC-AE is primarily optimized for image encoding, we extend its architecture to support video encoding by incorporating temporal compression mechanisms.

As shown in Figure 5(a), our Video DC-AE encoder consists of three residual blocks followed by three EfficientViT blocks [5], with the decoder adopting a symmetrical structure. The first five encoder blocks and the last five decoder blocks function as downsampling and upsampling layers, respectively.

To adapt DC-AE for video encoding, we replace 2D operations (e.g., convolutions, normalization) with 3D operations. Additionally, we introduce temporal compression in the last two downsampling blocks (blocks 4 and 5) of the encoder, and temporal upsampling in the first two upsampling blocks (blocks 2 and 3) of the decoder to reconstruct temporal information effectively.

Moreover, we follow DC-AE to introduce special residual blocks to connect the downsample and upsample blocks. This is because Chen *et al.* [9] identifies gradient propagation issues in these blocks, such that without the residuals, it is especially difficult to train high-compression autoencoders.

As shown in Figure 5(b), our downsample residual blocks follow DC-AE's pixel-shuffling strategy, redistributing pixels from the spatial and temporal dimensions into the channel dimension, followed by channel-wise averaging to achieve the desired compression. The upsample residual blocks perform the inverse operation by first duplicating channels, and subsequently redistributing them back into the spatial and temporal dimensions to reconstruct the original structure.

We train Video DC-AE from scratch and evaluate its reconstruction quality, with results summarized in Table 1. Our model is compared against OpenSora 1.2 VAE, the recently open-sourced StepVideo VAE [27], and HunyuanVideo VAE [19]. The results demonstrate that Video DC-AE achieves competitive performance, with only minor degradation in LPIPS [41] scores compared to the best-performing HunyuanVideo VAE, while maintaining strong PSNR and SSIM scores.

Furthermore, while the 256-channel version of Video DC-AE offers higher reconstruction quality, we opt for the 128-channel variant in the video generation model to enable faster adaptation due to its reduced channel size (see Section 4.3 for further details).

3.2 DiT Architecture

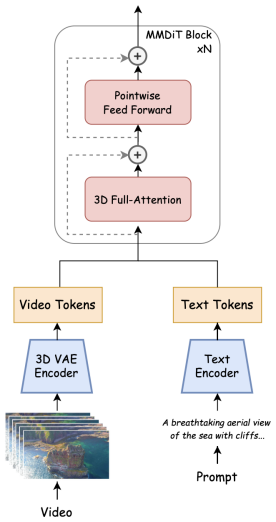


Figure 6: Open-Sora diffusion transformer architecture.

To achieve higher video quality, we employ full attention to capture long-range dependencies effectively. After encoding the input through the autoencoder, we patchify the latent representations to enhance computational efficiency and improve model learning. Sana [39] suggests that a patch size of 1 yields better training stability and finer details in video generation. With Video DC-AE's high compression ratio, we can afford to reduce the patch size to 1 (i.e., no patch at all), whereas a patch size of 2 is still required when the HunyuanVideo autoencoder is used in our generation model.

Inspired by FLUX's MMDiT [20], we employ a hybrid transformer architecture that incorporates both dual-stream and single-stream processing blocks. In the dual-stream blocks, text and video information are processed separately to facilitate more effective feature extraction within each modality. Subsequently, single-stream blocks integrate these features to facilitate effective cross-modal interactions. To further enhance the model's ability to capture spatial and temporal information, we apply 3D RoPE (Rotary Position Embedding) [37], which extends traditional positional encoding to three-dimensional space, allowing the model to better represent motion dynamics across time.

For text encoding, we leverage T5-XXL [10] and CLIP-Large [32], two high-capacity pretrained models known for their strong semantic understanding. T5-XXL captures complex textual semantics, while CLIP-Large improves alignment between text and visual concepts, leading to more accurate prompt adherence in video generation.

An overview of the model architecture is illustrated in Figure 6, and the detailed architectural specifications are presented in Table 2.

Double-Stream Layers	Single-Stream Layers	Model Dimension	FFN Dimension	Attention Heads	Patch Size
19	38	3072	12288	24	2

Table 2: Architecture hyperparameters for Open-Sora 2.0 11B parameter video generation model.

4 Model Training

A commercial-grade video generation model typically requires more than 10 billion parameters and O(100M) training samples. To mitigate the substantial computational cost associated with training, we propose a cost-effective training pipeline, as outlined in Table 3. Our approach consists of three key stages: (1) training a text-to-video model on low-resolution video data, (2) training an image-to-video model on low-resolution video data, and (3) fine-tuning an image-to-video model on high-resolution videos.

Training Stage	Dataset	CP	#iters	#GPUs	#GPU day	USD
256px T2V	70M	1	85k	224	2240	$107.5k
256px T/I2V	10M	1	13k	192	384	$18.4k
768px T/I2V	5M	4	13k	192	1536	$73.7k
Total					4160	$199.6k

Table 3: **Training Configurations and Cost Breakdown**. This table presents the training configurations across different stages and the total cost for a single full training run, assuming the rental price of H200 is $2 per GPU hour.

4.1 Efficient Training Strategy

To develop a high-quality video generation model within a constrained budget, our training strategy emphasizes the following four key aspects:

Leveraging Open-Source Image Models Prior research [16, 28, 43] has demonstrated that pretraining on image datasets can significantly accelerate video model training. To avoid the high cost of training an image model from scratch, we leverage an open-source solution. Specifically, we adopt Flux [20], a state-of-the-art text-to-image model with 11 billion parameters, which provides sufficient capacity to generate high-quality videos. We initialize our text-to-video model using Flux and empirically find that, despite it being a distilled model, this initialization is effective for further training.

High-Quality Training Data Inspired by the efficient image training strategies from PixArt [8], we hypothesize that high-quality video data can substantially enhance training efficiency. Consequently, we curate a high-quality subset from a large-scale dataset for low-resolution training. For high-resolution fine-tuning, we impose stricter selection criteria to ensure superior video quality.

Learning Motion in Low-Resolution Training a commercial video generation model is computationally expensive, particularly at high resolutions. To mitigate this cost, we first train on 256px resolution videos, allowing the model to learn diverse motion patterns efficiently. However, we observe that while the model captures motion effectively, low-resolution outputs tend to be blurry. Increasing the resolution significantly improves perceptual quality and human evaluative feedback.

Model	#GPUs	GPU Hours	Cost (Single Run)
Movie Gen [31]	6144	1.25M*	$2.5M*
Step-Video-T2V [27]	2992*	500k*	$1M*
Open Sora 2.0	224	100k	$200k

Table 4: **Training Cost Comparison of Different Video Generation Models**. Values marked with * are estimated based on publicly available information. Our Open-Sora 2.0 is trained at 5–10× lower training cost.

As shown in Table 5 and Table 6, training on a 129-frame video at 768px resolution is 40 times slower than at 256px. This performance gap arises due to the quadratic computational complexity of self-attention mechanisms as the number of tokens increases. To optimize efficiency, we allocate the majority of computational resources to low-resolution training, reducing the need for expensive high-resolution computations.

Image-to-Video Models Facilitate Resolution Adaptation We find that adapting a model from 256px to 768px resolution is significantly more efficient using an image-to-video approach compared to text-to-video. We hypothesize that conditioning the model on a static image allows it to focus more on motion generation, a capability that is well-learned during low-resolution training.

Based on this observation, we prioritize training an image-to-video model at high resolution. During inference, we first generate an image from a text prompt and subsequently synthesize a video conditioned on both the image and the text. During training, text/image-to-video training at low resolution followed by brief fine-tuning on high-resolution videos yields high-quality results with minimal additional training.

With our proposed approach, we successfully constrain the one-time training cost to $200K, as detailed in Table 3. We estimate the training costs of Movie Gen [31] and Step-Video-T2V [27] based on publicly available information, as summarized in Table 4. Our model achieves 5–10× lower training costs. We hope that these insights will contribute to reducing the training cost of high-quality video generation models in future research.

4.2 Training Setting

Our training setup is largely based on our previous iteration, Open-Sora 1.2 [43]. We adopt flow matching [23] as our primary training objective and utilize the AdamW [25] optimizer with β values of (0.9, 0.999) and an ϵ value of 1×10^{-15}. No weight decay is applied. The learning rate is set to 5×10^{-5} for the first 40k steps of Stages 1 and 2, followed by a decay to 3×10^{-5} for the final 45k steps. For Stage 3, the learning rate remains at 5×10^{-5}, while in Stage 3, it is reduced to 1×10^{-5}. Gradient norm clipping is applied with a threshold of 1.

Training Objective Our flow matching approach is similar to that used in Stable Diffusion 3 [11]. We denote the video latent as X_0, and Gaussian noise $X_1 \sim \mathcal{N}(0, 1)$. The model f_θ takes as input an interpolated latent $X_t = (1-t)X_0 + tX_1$ and is trained to predict the velocity component $X_0 - X_1$. The corresponding loss function is formulated as:

$$\mathcal{L} = \mathbb{E}_{t, X_0, X_1}[\|f_\theta(X_t, t, y) - (X_0 - X_1)\|],$$

where y represents the conditioning input (text and/or image). The timestep t is first sampled from a logit-normal distribution and then scaled according to the shape of X_0. Given that higher-resolution and longer-duration videos are more susceptible to noise, we apply the following transformation:

$$t \leftarrow \frac{\alpha t}{1 + (\alpha - 1)t},$$

where α is proportional to the product $T \times H \times W$. The same method is applied during inference.

resolution	#frames	max #tokens	batch size	throughput on 8 GPUs
256px	5 to 33	2304	12	12.7 videos/s
	37 to 65	4352	6	6.3 videos/s
	69 to 97	6400	4	4.2 videos/s
	101 to 129	8448	3	3.2 videos/s
256px	1	256	45	47.6 images/s
768px	1	2304	13	13.8 images/s
1024px	1	4096	7	7.4 images/s

Table 5: Batch size and throughput in stage 1 and 2.

resolution	#frames	max #tokens	batch size	throughput on 8 GPUs
768px	5 to 33	20736	6	0.25 videos/s
	37 to 65	39168	4	0.17 videos/s
	69 to 97	57600	3	0.13 videos/s
	101 to 129	76032	2	0.08 videos/s
768px	1	2304	38	1.60 images/s

Table 6: Batch size and throughput in stage 3 with context parallelism 4.

Multi-Bucket Training Following the methodology in Open-Sora 1.2 [43], we adopt multi-bucket training to efficiently handle videos of varying frame counts, resolutions, and aspect ratios within the same batch. This strategy optimizes GPU utilization by dynamically assigning batch sizes based on video characteristics. The specific batch sizes used for different training configurations are detailed in Table 5 and Table 6.

To determine the batch sizes, we conduct a search on H200 GPUs. The process is as follows:

1. We first identify the maximum batch size for the configuration with the highest token count, ensuring that it does not cause out-of-memory (OOM) errors.
2. For all other configurations, batch sizes are chosen such that they are the largest possible without exceeding memory constraints, while ensuring that training time does not surpass that of the highest-token configuration.
3. Additionally, we enforce the constraint that the combined execution time for autoencoder encoding and forward pass, as well as the backward pass, does not exceed the reference batch size's execution time. This constraint helps prevent inefficiencies due to synchronous operations in the backward pass.

By employing this strategy, we ensure efficient and scalable training across diverse video data distributions while maximizing hardware efficiency.

4.3 High Compression Autoencoder Adaptation

As discussed earlier, the high computational cost of training video generation models arises from the large number of tokens and the dominance of attention computation. Generating a one-minute video in a single pass would incur an extremely high cost in the future. To further reduce training expenses, we explore training video generation models with high-compression autoencoders (Video DC-AEs).

Advantages. Utilizing a high-compression video autoencoder significantly reduces the number of tokens required for video generation while increasing the number of latent channels incurs minimal additional computational cost (affecting only the input and output layers). We define the token downsample ratio, D_{token}, as the product of the autoencoder's compression ratios ($D_{\text{T|H|W}}$) and the generation model's patch sizes ($P_{\text{T|H|W}}$) in the T, H, W dimensions:

$$D_{\text{token}} = D_{\text{T}} \times D_{\text{H}} \times D_{\text{W}} \times P_{\text{T}} \times P_{\text{H}} \times P_{\text{W}}$$

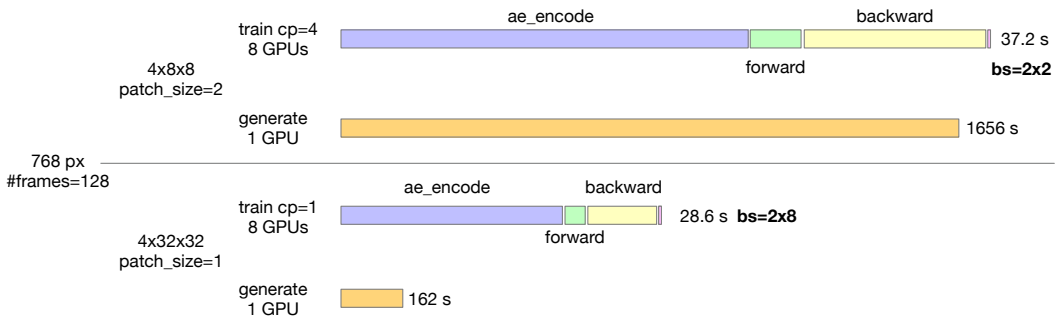


Figure 7: **Training and Inference Speed Comparison at 768px**. This figure compares the training and inference speeds at 768px resolution using HunyuanVideo VAE ($4 \times 8 \times 8$, patch size 2) and Video DC-AE ($4 \times 32 \times 32$, patch size 1). The results demonstrate that Video DC-AE achieves significantly higher efficiency in both training and inference (50 steps) compared to HunyuanVideo VAE, highlighting its advantage in high-resolution video generation.

For instance, in a 5-second, 24fps, 768px video, the HunyuanVideo VAE with a $4 \times 8 \times 8$ compression in combination of generation model's patch size $1 \times 2 \times 2$ ($D_{\text{token}} = 1024$) reduces the token count to 76K, whereas the Video DC-AE with $4 \times 32 \times 32$ compression and patch size of $1 \times 1 \times 1$ has a D_{token} of 4096, effectively reduces the token count to 19K. As shown in Figure 7, this compression yields a $5.2\times$ speedup in training throughput. Additionally, inference speeds improve by over $10\times$, making it a highly efficient approach.

Challenges in Training Video Autoencoders. With the growing interest in video generation models, we observe that despite variations in architecture, downsampling ratios, number of channels, and training losses, video autoencoders trained with similar information downsampling ratios exhibit comparable performance (evaluated at 256px resolution) [9]. We define the information downsampling ratio, D_{info} as follows:

$$D_{\text{info}} = \frac{D_{\text{T}} \times D_{\text{H}} \times D_{\text{W}} \times C_{\text{in}}}{C_{\text{out}}}$$

where C_{in} and C_{out} are the number of input and output channels (note that $C_{\text{in}} = 3$ for videos using RGB channels). We do not consider the actual storage sizes due to possible confounders of different data types (such as float32 or bfloat16) used.

We hypothesize that such a link between D_{info} and the reconstruction performance indicates a fundamental information compression lower bound for a given resolution, thereby constraining the achievable compression ratio unless the number of channels is adjusted. Our two video autoencoders of 256 channels and 128 channels are designed to match the D_{info} of the HunyuanVideo VAE and the StepVideo VAE respectively.

However, training a deep video compression network is challenging. Prior work [9] identifies gradient propagation issues in downsample and upsample blocks. To address this, they introduce residual connections in these blocks using pixel shuffle and unshuffle operations. We verify that training the HunyuanVideo VAE architecture with residual connections improves performance. Building on this, we incorporate temporal downsampling into DC-AE to construct Video DC-AE.

Additionally, we find that maintaining similar reconstruction performance requires a 4× increase in channels when doubling the height and width compression ratio, whereas adding a 4× temporal compression on top of the original image DC-AE (no temporal compression) incurs no additional channels possibly due to redundancy in temporal information. Based on these insights, we train a Video DC-AE with a 2x D_{info} and 16x D_{token} as compared to the HunyuanVideo VAE while maintaining comparable performances.

Challenges in Training the Generation Model. While training throughput for the diffusion model improves, we find that high-compression autoencoders—especially those with larger latent channel dimensions—slow down convergence. Prior studies [39, 40] show that when training image diffusion models on AEs with better reconstruction but higher latent dimensionality, the generation quality

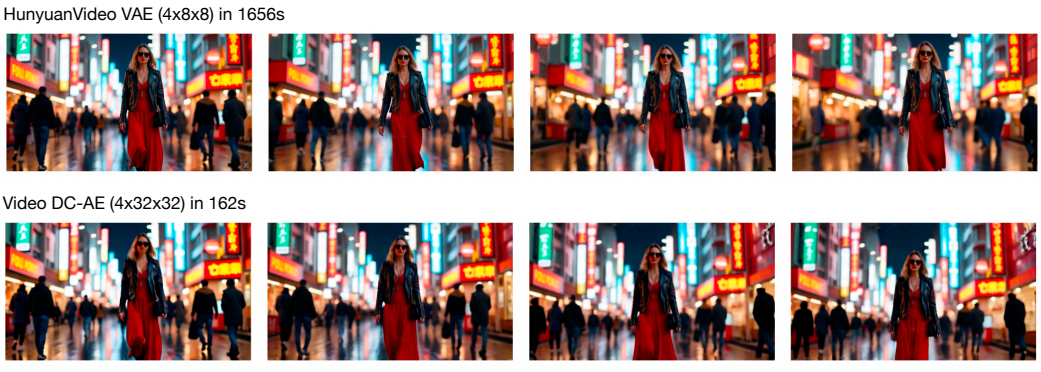


Figure 8: **Comparison of Video Generation with Different Autoencoder Compression Ratios.** The top and bottom rows correspond to lower and higher compression ratios, respectively. While the higher compression ratio AE results in slightly blurrier outputs, it significantly improves generation speed.

often degrades. Additionally, Theorem A.7 in [7] suggests that, under strong assumptions, increasing the channel size by k requires k^5 more data to achieve comparable performance. Meanwhile, Yao and Wang [40] demonstrates that introducing a distillation loss between a pretrained image foundation model and the VAE latents can significantly accelerate diffusion training.

Based on these findings, we hypothesize that current VAE training frameworks struggle to optimize latent space structures for video generation when channel sizes increase. Although reconstruction ability sets an upper bound on generation quality, a well-structured latent space is more critical for effective video synthesis.

Strategies for Using High-Compression Autoencoders. Despite the challenges of training with high-compression autoencoders, we adopt the following strategies to construct a fast video generation model:

1. Latent Space Distillation: After training Video DC-AE, we apply a distillation loss to align the third-layer latents with DINOv2 [30], leveraging their similar latent shapes.
2. Efficient Adaptation for Diffusion Models: Following PixArt, diffusion models can be adapted to different autoencoders. We reinitialize the input and output layers of the model for adaptation. We find that adaptation to high-compression autoencoders behaves a bit differently. Semantic structures adapt quickly, but video outputs appear blurry.
3. Prioritizing Image-to-Video Training: We observe that image-to-video models adapt more efficiently than text-to-video models when switching to high-compression AEs. Thus, we focus on training an image-to-video model for this setting.
4. Tiling in video encoding: High compression AEs trained at low resolutions experience performance degradation when reconstructing high-resolution videos [9]. Although we could have fine-tuned the Video DC-AE at high resolutions, we re-use the tiling code by Kong *et al.* [19] to save training resources.

We first train the model on short videos (up to 33 frames) for 17K iterations using 20M samples across 160 GPUs. Then, we extend training to long videos (up to 128 frames) for 8K iterations on 2M samples. The final model achieves a loss level of 0.5, compared to 0.1 for the initialization model. However, due to computational constraints, training does not fully converge. As shown in Figure 8, while the fast video generation model underperforms the original, it still captures spatial-temporal relationships. We release this model to the research community for further exploration.

Discussion and Future work. We believe high-compression autoencoders are critical for end-to-end video generation, especially for high-resolution videos that contain substantial redundancy. To further reduce training costs, it is essential to: (1) Optimize autoencoder training to yield better latent spaces for diffusion-based learning. (2) Design high-throughput autoencoders, as autoencoder encoding time remains a major computational bottleneck in generation models using high-compression autoencoders.

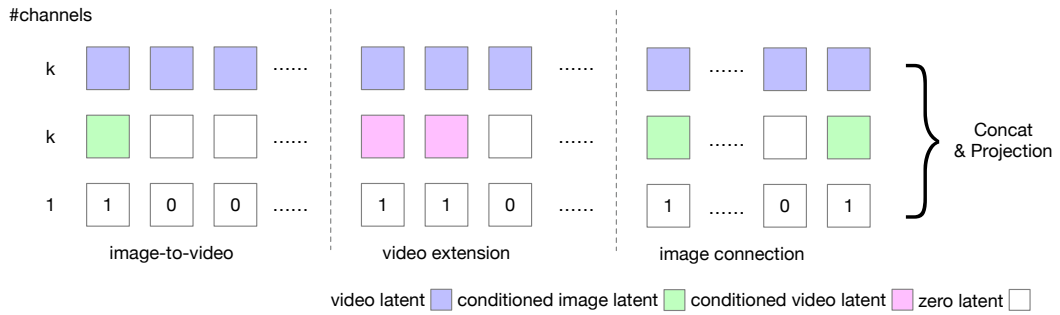


Figure 9: **Image and Video Condition Framework** Condition information is injected into the model via an additional channel dimension. A mask mechanism is introduced to distinguish different input types. This design enables support for a wide range of image-to-video (I2V) and video-to-video (V2V) tasks.

4.4 AE Training

We train Video DC-AE with two different channel sizes, namely, 256 channels and 128 channels from scratch. Following DC-AE, our training loss has no KL loss component. We first train the Video DC-AE with reconstruction loss L_1 and perceptual loss L_{LPIPS} for 250k steps:

$$\mathcal{L} = \mathcal{L}_1 + 0.5\mathcal{L}_{\text{LPIPS}},$$

then add an adversarial loss component, L_{adv}, for another 200k steps:

$$\mathcal{L} = \mathcal{L}_1 + 0.5\mathcal{L}_{\text{LPIPS}} + 0.05\mathcal{L}_{\text{adv}}.$$

We train each AE model on 8 GPUs with a local batch size of 1. Our training data consists of 32 frames of 256px videos at an aspect ratio of 1:1. We use a fixed learning rate of 5e-5 for the AE using the AdamW optimizer with β values of (0.9, 0.999) and an ϵ value of 1×10^{-15} without weight decay. The learning rate is adjusted to 1e-4 for the discriminator without changing any other optimizer parameters. We apply gradient norm clipping at a threshold of 1.

5 Conditioning

5.1 Image-to-Video Training

Open-Sora 1.2 introduces a universal conditioning framework, allowing both image and video conditions to be applied at any frame. However, its original method replaces noisy inputs with the condition, leading to inconsistent timesteps across different inputs. To address this issue, we modify the framework by concatenating the condition as additional channels, ensuring that the velocity prediction task remains unchanged.

As illustrated in Figure 9, our approach first encodes image or video conditions using an autoencoder, then concatenates the encoded features with the original video latent representation. An extra channel is introduced to indicate the task type, increasing the number of channels from k to $2k + 1$.

To improve generalization, we introduce image condition dropout similar to text condition dropout. During training, dropping the image condition reduces the problem to a text-to-video setting, where zero tensors are concatenated to the video latent. For T/I2V training, we set the dropout ratio to 12.5%, ensuring robustness across various image-to-video (I2V) and text-to-video (T2V) tasks.

5.2 Image-to-Video Inference

We use classifier-free guidance for inference [13]. As our model is conditioned on both image and text, a straightforward approach is to use a single guidance scale:

$$v_t = v_\theta(x_t, t, \varnothing, \varnothing) + g \cdot (v_\theta(x_t, t, \textit{txt}, \textit{img}) - v_\theta(x_t, t, \varnothing, \varnothing)),$$

where v_θ represents the predicted velocity, and g is the guidance scale. However, we find that this approach is suboptimal. Image guidance requires only a small guidance scale, as large values make

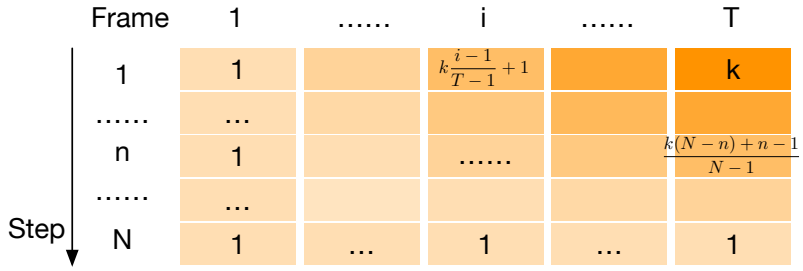


Figure 10: **Heatmap of Image Guidance Scale Across Denoising Steps and Latent Frames** Darker regions indicate higher image guidance values, emphasizing stronger influence on later frames and earlier denoising steps.

the entire video static, while text guidance benefits from a higher scale, improving semantic alignment. To address this, we decouple the guidance terms as follows:

$$\begin{aligned} v_t = & \, v_\theta(x_t, t, \varnothing, \varnothing) \\ & + g_{img} \cdot \big(v_\theta(x_t, t, \varnothing, img) - v_\theta(x_t, t, \varnothing, \varnothing)\big) \\ & + g_{txt} \cdot \big(v_\theta(x_t, t, txt, img) - v_\theta(x_t, t, \varnothing, img)\big). \end{aligned}$$

A further issue arises when using high image guidance, as it sometimes introduces flickering in generated videos. To mitigate this, we introduce a guidance oscillation technique [14] for image guidance. Taking 50-step sampling as an example, after the first 10 steps, we alternate the image guidance scale: during odd-numbered steps, we apply the original g_{img}, while for even-numbered steps, we reduce it to 1. This helps balance stability and motion consistency.

Beyond oscillation, we introduce a dynamic image guidance scaling strategy. Since the image condition is primarily applied to the first frame, frames toward the end of the video require stronger image guidance to maintain coherence. At the same time, as denoising progresses, the video scene is mostly formed, making image guidance less critical at later diffusion steps. To optimize for both effects, we dynamically adjust g_{img} based on both the frame index and the denoising step. As shown in Figure 10, we test both linear and quadratic scaling, finding that linear scaling across both dimensions provides the best performance. In practice, we use default guidance values of $g_{img} = 3$, $g_{txt} = 7.5$, achieving a balance between motion fidelity and semantic accuracy.

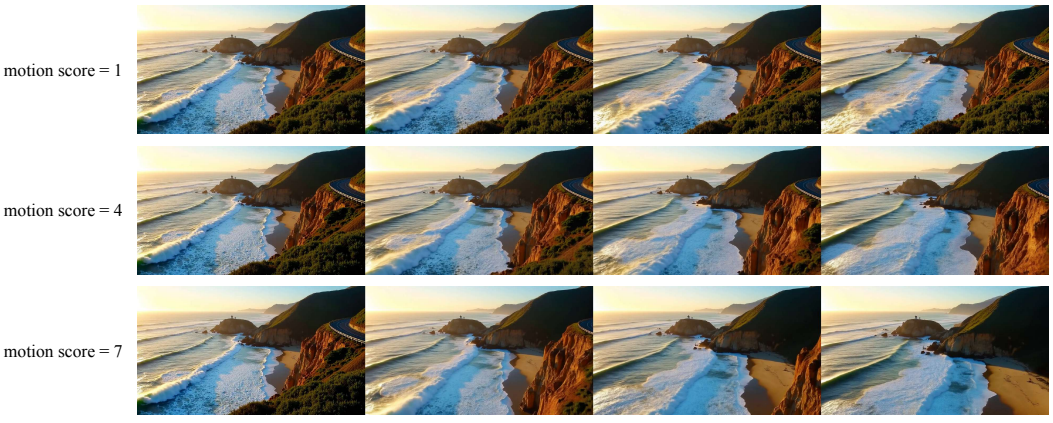


Figure 11: **Effect of Motion Score on Video Generation.** This figure illustrates the impact of different motion scores on the generated video. As the motion score increases, the camera movement becomes more pronounced and the overall dynamic movement increases within the scene.

5.3 Motion Score

Controlling the level of motion strength is a crucial feature in video generation. Depending on the scene, users may prefer high-fidelity videos with minimal motion or highly dynamic videos with significant movement. While techniques such as classifier-free guidance scaling and shifting denoising steps can influence motion intensity, we find that these approaches are often entangled with other factors, such as visual quality and text alignment, making precise control challenging. To address this, we explicitly model the motion dynamics as a separate controllable parameter.

We leverage the motion score obtained from data pre-processing, which quantifies the level of dynamics in a video. This motion score is appended to the caption as an additional conditioning signal. During inference, adjusting the motion score allows for effective and independent control over the dynamic level of the generated video, as demonstrated in Figure 11.

6 System Optimization

We train our models using ColossalAI [21], an efficient parallel training system. Our hardware setup includes H200 GPUs, whose 141GB memory enables more effective data parallelism (DP) and allows for more aggressive selective activation checkpointing, significantly optimizing resource utilization. Additionally, we leverage PyTorch compile [1] and Triton kernels [29] to accelerate training efficiency.

6.1 Parallelism Strategy

To efficiently handle high-resolution video training, we employ multiple parallelization techniques. For video autoencoders, we adapt tensor parallelism (TP) [36] to convolution layers by partitioning weights either the input or output channel dimensions, reducing memory consumption while preventing out-of-bound indexing for high-resolution training.

For MMDiT training, we combine Zero Redundancy Optimizer (ZeroDP) [33] with Context Parallelism (CP) [24]. Here, video and text sequences are partitioned across GPUs along the sequence dimension, enabling each GPU to compute attention independently. This approach effectively mitigates the memory bottleneck and enhances the efficiency of attention computation, which is particularly beneficial for high-resolution videos, where attention complexity grows quadratically.

Empirical evaluations on H200 GPUs (141GB memory) indicate that employing CP alone yields an optimal trade-off between memory efficiency and computational performance. During Stage 1 and Stage 2 training, we exclusively utilize data parallelism (DP) in combination with ZeRO-2, achieving a maximum FLOPs utilization (MFU) of 38.19%. In Stage 3, we integrate ZeRO-2 with CP=4, resulting in to an MFU of 35.75%.

6.2 Activation Checkpointing

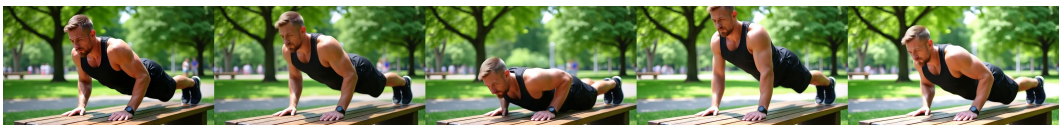
Activation checkpointing is applied selectively to reduce memory consumption without significantly increasing computational overhead. Instead of storing intermediate activations, only block inputs are retained, and the forward pass is recomputed during backpropagation. To minimize slowdowns, we avoid enabling checkpointing for every layer. In Stages 1 and 2, it is applied only to 8 layers of dual blocks and all single blocks, whereas in Stage 3 it is enabled for all blocks, supplemented by activation offloading to the CPU. The offloading mechanism further reduces memory footprint by asynchronously transferring activation tensors, leveraging pinned memory and asynchronous data movement to minimize training slowdowns.

6.3 Auto Recovery

To ensure continuous training in large-scale distributed environments, we implement an auto-recovery system to handle unexpected failures such as InfiniBand failures, storage system crashes, and NCCL errors. The system continuously monitors training status, checking for unresponsiveness, significant slowdowns, or loss spikes. Upon detecting issues, all processes are halted, and a faulty node checker diagnoses faulty nodes. If necessary, backup machines are deployed, and training resumes

(a) Prompt: A scene from a disaster movie.

(b) Prompt: A panda bear with distinct black patches climbs and rests on a wooden log platform amid lush, natural foliage.



(c) Prompt: A man performs push-ups on a wooden bench in a sunny park, captured from a side angle in a medium shot. The focus is on his upper body and technique, with natural sunlight accentuating the scene. Lush greenery and distant park-goers contribute to the energetic, realistic setting.

(d) Prompt: A playful dog in a pink coat with a red leash dashes across a muddy field with sparse crops. The camera tracks its energetic movement from right to left against a backdrop of trees and distant power lines under an overcast sky. The realistic, medium shot captures a candid, lively moment in soft, diffused light.

(e) Prompt: A drone camera circles a historic church on a rocky outcrop along the Amalfi Coast, highlighting its stunning architecture, tiered patios, and the dramatic coastal views with waves crashing below and people enjoying the scene in the warm afternoon light.

Figure 12: High-quality videos generated by Open-Sora 2.0

automatically from the last checkpoint without encountering a loss spike. With this strategy, our GPU utilization rate exceeds 99%, minimizing downtime and optimizing hardware efficiency.

6.4 Dataloader

To accelerate data movement between the host (CPU) and devices (GPU), we optimize PyTorch's dataloader. Instead of relying on PyTorch's default pinned memory allocation, which invokes cudaMallocHost and may block CUDA kernel execution, we employ a pre-allocated pinned memory buffer to prevent dynamic memory allocations and reduce overhead, particularly for large video inputs. Furthermore, we overlap data transfers with computation, ensuring that data for the next step is prefetched while the current batch is being processed.

Additionally, we mitigate Python's garbage collection (GC) overhead, which can unpredictably pause execution and cause severe inefficiencies in multi-process distributed training. To address this, we disable global GC and implement manual memory management, preventing unnecessary synchronization delays.

6.5 Checkpoint Optimization

Efficient model checkpointing is essential for minimizing recovery latency in distributed training. The checkpoint saving process involves three key steps: first, if the model is sharded, complete weights must be reconstructed via inter-GPU communication; second, the model weights are transferred from CUDA memory to CPU memory; and finally, the weights are written from the CPU to disk.

To optimize the second and third steps, we employ pre-allocated pinned memory to significantly accelerate weight transfer and implement asynchronous disk writing via C++, thereby ensuring that file I/O does not block the main training process. These optimizations reduce model-saving overhead to the order of seconds, which greatly accelerates the training.

For checkpoint loading, the process involves reading model weights from disk and transferring them from CPU to CUDA. We improve efficiency by using asynchronous pinned memory copying with multi-threaded allocation, and also implementing pipelined execution between shard reading and weight transfer. Furthermore, for large models stored in multiple shards, we overlap these phases across shards to maximize parallelism and accelerate the loading process. These optimizations reduce the overhead of training resumption.

7 Performance

Our model supports both text-to-video and image-to-video generation at 256×256px and 768×768px resolutions (hereafter referred to as 256px and 768px), generating videos up to 128 frames long. At 24 FPS, this corresponds to a 5-second duration. Since our model is optimized for image-to-video generation, the default text-to-image-to-video pipeline first generates an image using the FLUX model, which is then used as the starting frame for video generation. The generated results are presented in Figure 12.

To benchmark our model against other approaches, we generate videos using Open-Sora 2.0, as well as several closed-source APIs and open-source models, using a set of 100 text prompts. To ensure fairness, we only perform a single inference per model, avoiding any cherry-picking. For all models, we use their default settings, with video lengths ranging from 5 to 6 seconds and resolutions between 768px and 720p, depending on their generation constraints.

A blinded evaluation was conducted by 10 professional evaluators, assessing videos based on three key criteria:

- **Visual Quality**: Which video exhibits higher visual fidelity and is aesthetically more pleasing?
- **Prompt Adherence**: Which video aligns more accurately with the given text prompt?
- **Motion Quality**: Which video maintains more consistent motion and better adheres to physical laws?

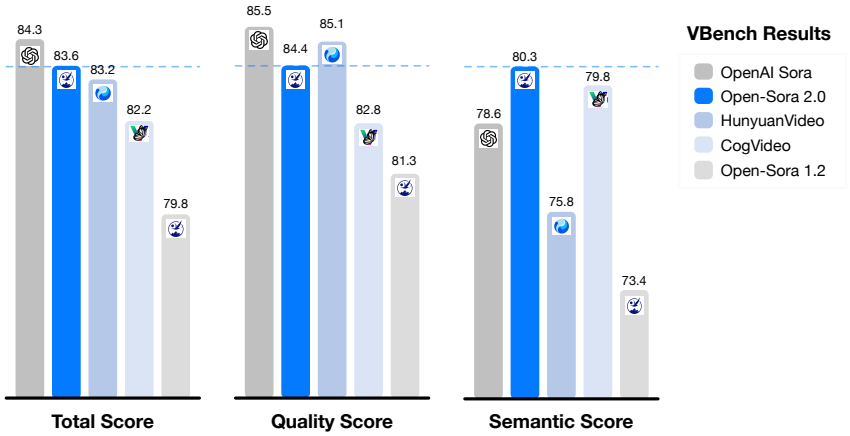


Figure 13: **VBench Score Comparison.** Our model outperforms open-source text-to-video models by leveraging a text-to-image-to-video generation approach. Additionally, our latest version significantly narrows the performance gap between Open-Sora and OpenAI's Sora, demonstrating substantial improvements in video generation quality and coherence.

The evaluation results, shown in Figure 1, indicate that our model outperforms existing models in several dimensions and achieves competitive performance across all categories.

We further evaluate our model's performance using VBench [17] in Table 13, demonstrating significant improvements from Open-Sora 1.2 to 2.0. The performance gap between Open-Sora and OpenAI's Sora has been reduced from 4.52% to 0.69%, highlighting substantial advancements in video generation quality. Additionally, our model achieves a higher VBench score compared to CogVideoX1.5-5B and HunyuanVideo, further establishing its superiority among current open-source text-to-video models.

8 Conclusion

This report introduces Open-Sora 2.0, a commercial-level video generation model that was trained for only $200k, which is 5-10 times more cost-efficient than comparable models like MovieGen and Step-Video-T2V. This achievement highlights that high-quality video generation models can be developed with highly controlled costs through careful optimization of data curation, model architecture, training strategy, and system optimization. Despite its significantly lower training cost, Open-Sora 2.0 performs comparably to leading video generation models including HunyuanVideo and Runway Gen-3 Alpha. The model supports both text-to-video and image-to-video generation at resolutions up to 768×768 pixels for videos up to 5 seconds in length.

Looking ahead, several challenges remain in video generation. First, deep compression video VAE technology is still underexplored. While we aggressively increase the compression ratio to reduce latent tokens, this approach introduce reconstruction quality loss and adaptation difficulties. Additionally, diffusion models often produce unpredictable artifacts such as object distortion and unnatural physics, with users having limited control over these details. The field needs further research on artifact prevention and enhanced control over generated content. We hope that by open-sourcing Open-Sora 2.0, we can provide the community with tools to collectively tackle these challenges, fostering innovation and advancements in the field of video generation.

Contributors

Core contributors are those who have been involved in the development of Open-Sora 2.0 throughout its entire process, while contributors are those who contributed part-time. All contributors are listed in **alphabetical order by first name**.

- **Project Leaders:** Xiangyu Peng, Zangwei Zheng.
- **Core Contributors:**
  - **Model & Training:** Chenhui Shen, Tom Young, Xinying Guo.
  - **Infrastructure:** Binluo Wang, Hang Xu, Hongxin Liu, Mingyan Jiang, Wenjun Li, Yuhui Wang.
  - **Data & Evaluation:** Anbang Ye, Gang Ren, Qianran Ma, Wanying Liang, Xiang Lian, Xiwen Wu, Yuting Zhong, Zhuangyan Li.
- **Contributors:** Chaoyu Gong, Guojun Lei, Leijun Cheng, Limin Zhang, Minghao Li, Ruijie Zhang, Silan Hu, Shijie Huang, Xiaokang Wang, Yuanheng Zhao, Yuqi Wang, Ziang Wei.
- **Corresponding Authors:** Yang You (youy@comp.nus.edu.sg).

References

[1] J. Ansel *et al.*, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf.

[2] P. Authors, *Paddleocr: A practical ultra lightweight ocr system*, https://github.com/PaddlePaddle/PaddleOCR, 2020.

[3] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] T. Brooks *et al.*, "Video generation models as world simulators," 2024. [Online]. Available: https://openai.com/research/video-generation-models-as-world-simulators.

[5] H. Cai, J. Li, M. Hu, C. Gan, and S. Han, "Efficientvit: Lightweight multi-scale attention for high-resolution dense prediction," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 17 302–17 313.

[6] B. Castellano, *Pyscenedetect: Video scene cut detection and analysis tool*, version X.Y.Z, 2023. [Online]. Available: https://github.com/Breakthrough/PySceneDetect.

[7] H. Chen *et al.*, "Masked autoencoders are effective tokenizers for diffusion models," *arXiv preprint arXiv:2502.03444*, 2025.

[8] J. Chen *et al.*, "Pixart-alpha: Fast training of diffusion transformer for photorealistic text-to-image synthesis," *arXiv preprint arXiv:2310.00426*, 2023.

[9] J. Chen *et al.*, "Deep compression autoencoder for efficient high-resolution diffusion models," *arXiv preprint arXiv:2410.10733*, 2024.

[10] H. W. Chung *et al.*, "Scaling instruction-finetuned language models," *Journal of Machine Learning Research*, vol. 25, no. 70, pp. 1–53, 2024.

[11] P. Esser *et al.*, "Scaling rectified flow transformers for high-resolution image synthesis," in *Forty-first International Conference on Machine Learning*, 2024.

[12] FFmpeg Developers, *Ffmpeg*, Available from https://ffmpeg.org/, 2023. [Online]. Available: https://ffmpeg.org/.

[13] J. Ho and T. Salimans, "Classifier-free diffusion guidance," *arXiv preprint arXiv:2207.12598*, 2022.

[14] J. Ho *et al.*, *Imagen video: High definition video generation with diffusion models*, 2022. arXiv: 2210.02303 [cs.CV].

[15] J. Hoffmann *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[16] W. Hong, M. Ding, W. Zheng, X. Liu, and J. Tang, "Cogvideo: Large-scale pretraining for text-to-video generation via transformers," *arXiv preprint arXiv:2205.15868*, 2022.

[17] Z. Huang *et al.*, "Vbench: Comprehensive benchmark suite for video generative models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 21 807–21 818.

[18] J. Kaplan *et al.*, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[19] W. Kong *et al.*, "Hunyuanvideo: A systematic framework for large video generative models," *arXiv preprint arXiv:2412.03603*, 2024.

[20] B. F. Labs, *Flux*, https://github.com/black-forest-labs/flux, 2024.

[21] S. Li *et al.*, "Colossal-ai: A unified deep learning system for large-scale parallel training," in *Proceedings of the 52nd International Conference on Parallel Processing*, ser. ICPP '23, Salt Lake City, UT, USA: Association for Computing Machinery, 2023, pp. 766–775, ISBN: 9798400708435. DOI: 10.1145/3605573.3605613. [Online]. Available: https://doi.org/10.1145/3605573.3605613.

[22] B. Lin *et al.*, "Open-sora plan: Open-source large video generation model," *arXiv preprint arXiv:2412.00131*, 2024.

[23] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," *arXiv preprint arXiv:2210.02747*, 2022.

[24] H. Liu, M. Zaharia, and P. Abbeel, *Ring attention with blockwise transformers for near-infinite context*, 2023. arXiv: 2310.01889 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.01889.

[25] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[26] Luma AI. "Dream machine," Luma AI. (2025), [Online]. Available: https://lumalabs.ai/dream-machine.

[27] G. Ma *et al.*, "Step-video-t2v technical report: The practice, challenges, and future of video foundation model," *arXiv preprint arXiv:2502.10248*, 2025.

[28] X. Ma *et al.*, "Latte: Latent diffusion transformer for video generation," *arXiv preprint arXiv:2401.03048*, 2024.

[29] OpenAI, *Triton: An open-source programming language for writing highly efficient gpu code*, Accessed: 2025-03-12, 2019. [Online]. Available: https://github.com/triton-lang/triton.

[30] M. Oquab *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv preprint arXiv:2304.07193*, 2023.

[31] A. Polyak *et al.*, "Movie gen: A cast of media foundation models," *arXiv preprint arxiv:2410.13720*, 2024.

[32] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *ICML*, 2021.

[33] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, *Zero: Memory optimizations toward training trillion parameter models*, 2020. arXiv: 1910.02054 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1910.02054.

[34] RunwayML. "Introducing Gen-3 Alpha," RunwayML. (2025), [Online]. Available: https://runwayml.com/research/introducing-gen-3-alpha.

[35] C. Schuhmann. "Improved aesthetic predictor," GitHub. (2021), [Online]. Available: https://github.com/christophschuhmann/improved-aesthetic-predictor (visited on 03/10/2025).

[36] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, *Megatron-lm: Training multi-billion parameter language models using model parallelism*, 2020. arXiv: 1909.08053 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1909.08053.

[37] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *Neurocomputing*, vol. 568, p. 127 063, 2024.

[38] Q. Team, "Qwen2.5 technical report," *arXiv preprint arXiv:2412.15115*, 2024.

[39] E. Xie *et al.*, "Sana: Efficient high-resolution image synthesis with linear diffusion transformers," *arXiv preprint arXiv:2410.10629*, 2024.

[40] J. Yao and X. Wang, "Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models," *arXiv preprint arXiv:2501.01423*, 2025.

[41] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.

[42] Y. Zhang *et al.*, "Video instruction tuning with synthetic data," *arXiv preprint arXiv:2410.02713*, 2024.

[43] Z. Zheng *et al.*, "Open-sora: Democratizing efficient video production for all," *arXiv preprint arXiv:2412.20404*, 2024.