Neural-Centric Video Processing Pipeline for Unified Multi-Task Inference

1. Detailed Experimental Settings

1.1. Descriptions of Downstream Task Models, Datasets, and Adapters

In this section, we comprehensively describe the pretrained models, datasets, preprocessing procedures, accuracy evaluation methods, and adapter architectures employed in our evaluations.

1.1.1. ResNet50

- **Pretrained Weight**: Official PyTorch weights pretrained on ImageNet-1K.
- **Dataset**: ImageNet VID 2015, randomly sampled 1,000 video clips from 10 classes (1280×720 resolution). The 10 classes are airplane, bus, car, cattle, dog, domestic cat, elephant, fox, giant panda, and turtle.
- **Input Preprocess**: Resize input frames to 224×224, normalize with ImageNet mean (0.485, 0.456, 0.406) and std (0.229, 0.224, 0.225).
- **Accuracy Evaluation Method**: For each video, we randomly sample 8 frames with uniform intervals. Top-1 accuracy is compared against pseudo-ground-truth labels generated by pretrained ViT-Large predictions with raw frames.
- **Adapter Architecture**: A single Conv2d layer (1×1 kernel), mapping channel dimension from NeRV's intermediate features to the ResNet50's target intermediate layer channels.

1.1.2. CLIP-RN50

- **Pretrained Weight**: Official OpenAI CLIP pretrained weights (RN50 variant trained on Web-scale image-text pairs).
- **Dataset**: ImageNet VID 2015, identical sampling as ResNet50 (1,000 video clips).
- **Input Preprocess**: Resize input frames to 224×224, normalize using CLIP's mean (0.481, 0.458, 0.408) and std (0.269, 0.261, 0.276).
- **Accuracy Evaluation Method**: For each video, we randomly sample 8 frames with uniform intervals. Top-1 accuracy is evaluated directly using ImageNet VID provided labels as ground truth.
- **Adapter Architecture**: A single Conv2d (1×1 kernel), adapting NeRV features to CLIP-RN50 intermediate layer channels.

1.1.3. ViT

- **Pretrained Weight**: Official ViT-B/16 pretrained on ImageNet-21K provided by Google Research.
- **Dataset**: ImageNet VID 2015, identical sampling as ResNet50 (1,000 video clips).
- **Input Preprocess**: Resize frames to 224×224, normalize with ImageNet mean/std, followed by ViT-specific patch embedding (16×16 patches).
- **Accuracy Evaluation Method**: For each video, we randomly sample 8 frames with uniform intervals. Top-1 accuracy is measured using pseudo-ground-truth labels from pretrained ViT-Large.
- **Adapter Architecture**: A single 1×1 convolution layer projecting NeRV intermediate features into the ViT embedding dimension (768). To match ViT's patch embedding structure, features are first resized to a fixed spatial grid (14×14) using AdaptiveAvgPool2d. The projected features are then flattened and normalized, followed by concatenation with a learnable class token and addition of positional embeddings, yielding the standard ViT input shape $(B, 197, 768)$. This ensures compatibility without introducing significant computational overhead.

1.1.4. I3D

- **Pretrained Weight**: Official I3D-ResNet50 pretrained on Kinetics-400 dataset from PyTorchVideo.
- **Dataset**: UCF101 dataset, randomly sampled 500 video clips, identical sampling strategy to SlowFast.
- **Input Preprocess**: 32 consecutive frames temporal sampling, resize frames to 224×224, normalization identical to SlowFast preprocessing.
- **Accuracy Evaluation Method**: For each video, we randomly select a starting point and decode 32 consecutive frames. Top-1 accuracy is evaluated directly against ground-truth labels from UCF101.
- **Adapter Architecture**: A lightweight adapter composed of temporal downsampling (via MaxPool3d if required), spatial upsampling (trilinear interpolation), followed by a single Conv3D (kernel $1 \times 1 \times 1$) to adjust NeRV's 3D intermediate feature channels. This design explicitly preserves the essential spatio-temporal structure required by I3D, efficiently bridging the gap between NeRV represen-

tations and I3D's input expectations.

1.1.5. SlowFast

- **Pretrained Weight**: Official SlowFast-R50 pretrained on Kinetics-400 dataset from PyTorchVideo.
- **Dataset**: UCF101 dataset, randomly sampled 500 video clips from multiple classes.
- **Input Preprocess**: Temporally sample 32 consecutive frames from each clip, resize frames to 224×224, normalize with Kinetics dataset mean/std.
- **Accuracy Evaluation Method**: For each video, we randomly select a starting point and decode 32 consecutive frames. Top-1 accuracy is measured using ground-truth labels provided by the UCF101 dataset.
- **Adapter Architecture**: Same spatio-temporal adapter architecture as I3D.

1.1.6. DETR

- **Pretrained Weight**: Official DETR-R50 weights pretrained on the COCO dataset provided by Facebook Research.
- **Dataset**: ImageNet VID 2015 dataset, randomly sampled 1,000 video clips (same sampling as ResNet50).
- **Input Preprocess**: Resize each frame so shortest side is 800 pixels, maintain aspect ratio, padding to match shape requirements, normalization with COCO mean/std.
- **Accuracy Evaluation Method**: For each video, we randomly sample 8 frames with uniform intervals. Mean Average Precision (mAP) is calculated directly using the ground-truth bounding boxes from the ImageNet VID 2015 dataset annotations.
- **Adapter Architecture**: Same adapter architecture as ViT's patch embedding.

1.1.7. BLIP

- **Pretrained Weight**: Official BLIP pretrained weights (ViT-B/16 backbone trained on large-scale image-text pairs) from Salesforce.
- **Dataset**: MSR-VTT dataset, randomly sampled 100 videos.
- **Input Preprocess**: Resize frame to 224×224, normalization following ImageNet mean/std, text tokenization using BLIP's default tokenizer.
- **Accuracy Evaluation Method**: For each video, we randomly sample 8 frames with uniform intervals. Captioning accuracy is evaluated by CLIP score comparison between generated captions and BLIP's original captions from full RGB frames.
- **Adapter Architecture**: Same adapter architecture as ViT's patch embedding.

1.2. Architecture Selection Strategy

To ensure reconstruction quality while maintaining computational efficiency, we assign architectures based on video resolution and frame count, targeting PSNR $\geq$ 33 dB. Table 1 summarizes the architecture configurations used across different video characteristics.

Frame Count	Stem Dim	Lower Width
≤ 40	24	124
$40 <$ # frame ≤ 100	24	148
$100 <$ # frame ≤ 150	32	148
$150 <$ # frame ≤ 200	48	148
$200 <$ # frame ≤ 300	64	196
# frame > 300	128	196

Table 1. NeRV architecture configuration based on frame count.

1.3. Dataset-Specific Configurations

ImageNet VID 2015 (1280×720 resolution): Videos are encoded using a 5-block hierarchical architecture with upsampling factors of [5, 2, 2, 2, 2]. This configuration progressively increases spatial resolution from the initial latent representation to full 1280×720 output through five decoding stages. The stem dimension and channel counts are selected according to Table 1 based on the computed lower width.

UCF101 (320×240 resolution): Due to varying temporal lengths, we scale model capacity based on frame count. Videos are categorized into six model size configurations: 0.1M, 0.3M, 0.5M, 1.0M, 1.5M, and 2.0M parameters. Each configuration adjusts the stem dimension and channel progression to match temporal complexity while maintaining the target reconstruction quality (PSNR $\geq$ 33 dB). The 4-block architecture with upsampling factors of [5, 2, 2, 2, 2] is used to reach the 320×240 resolution.

All architectures employ instance normalization and ReLU activation after each convolutional layer within the decoding blocks, following the standard NeRV design principles.

We further analyze computational complexity by measuring the FLOPs and BPP (bits per pixel) required for each decoding block. The detailed values are provided in Table 2, computed for a representative video with 115 frames using both NeRV and HNeRV architectures.

For FLOPs, computational complexity increases dramatically in later blocks due to up-convolution operations that process progressively larger spatial resolutions. As shown in the table, the final blocks consume over 70% of total FLOPs, making them the primary target for reduction. Therefore, selectively omitting these later decoding blocks, as enabled by our neural-centric pipeline, significantly reduces both computational cost and latency without compromising downstream task performance.

For BPP, the parameter distribution follows a different pattern. The initial MLP layers that generate frame-specific

embeddings from temporal indices consume the majority of parameters. While the reduction is not as dramatic as with FLOPs, eliminating later blocks still achieves substantial BPP savings. This observation enables task-dependent optimization strategies: to minimize BPP, one can reduce the embedding dimension for frame index encoding; to minimize FLOPs, one can focus on simplifying MLP layers and reducing channel counts in later CNN blocks. This flexibility allows our approach to adapt to different deployment constraints—whether prioritizing storage efficiency, computational efficiency, or a balance of both.

Block	GFLOPs	Parameters
Initial Feature	0.0001	0.0006
Block 1	0.009	0.0447
Block 2	0.129	0.0439
Block 3	2.389	0.0865
Block 4	9.556	0.0595
Block 5	38.22	0.0397
Head	0.265	0.0005

Table 2. FLOPs and BPP (bits per pixel) of NeRV's intermediate blocks.

2. More Visualizations

2.1. Feature's Output

We visualize the intermediate feature maps from ResNet50's layer - layer2.2.conv2, alongside the corresponding output features produced by our trained adapter from NeRV block 3 in Figure 1. The goal of this visualization is to verify whether the adapter output aligns with the abstraction level of the target backbone features.

Although exact channel-wise correspondence is not expected due to independent training, the visualization clearly demonstrates that both representations share highly similar abstraction characteristics. This confirms that our adapter effectively translates NeRV's intermediate neural representations into features suitable for direct injection into pretrained backbones, enabling efficient inference without sacrificing representational quality.

3. Future Work

Image generation models, initially dominated by CNN-based architectures such as GANs, VAEs, and diffusion models, have recently transitioned toward Transformer-based designs, demonstrating superior scalability and flexibility. Inspired by this trend, a compelling avenue for future work involves shifting the underlying representation of video from CNN-based architectures, such as NeRV, toward Transformer-centric implicit neural representations.

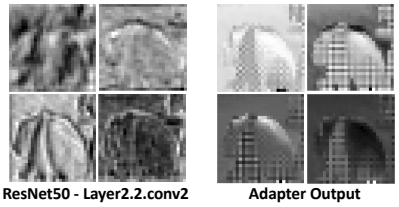

Figure 1. Visualization of intermediate features

Currently, Transformer-based models—including ViT, DETR, and BLIP—primarily reduce computations at the patch embedding stage but still rely on CNN-derived features for video encoding. Developing a fully Transformer-based implicit video representation encoder would align more naturally with these architectures, enabling direct interaction with Transformer backbones and potentially enhancing multi-task performance and computational efficiency.

Recent efforts in video Transformer architectures (e.g., TimeSformer, Video Swin Transformer) suggest the feasibility of such representations, paving the way for an efficient multitask inference framework, which we term Neural Video Pipeline–Transformer (NVP-Transformer). We believe this direction holds significant promise for creating scalable, robust, and task-optimized video representation methods, thereby opening new horizons for neural-centric video processing.