TAPNext++: What's *Next* for Tracking Any Point (TAP)?

Sebastian Jung[*,†,3,4] Artem Zholus[*,5,7,8] Martin Sundermeyer[1] Carl Doersch[2] Ross Goroshin[2,5,6]
David Joseph Tan[1] Sarath Chandar[5,7,8,9] Rudolph Triebel[3,4] Federico Tombari[1,10]
[1]Google [2]Google DeepMind [3]German Aerospace Center (DLR) [4]Karlsruhe Institute of Technology (KIT)
[5]Mila - Quebec AI Institute [6]Université de Montréal [7]Chandar Research Lab
[8]Polytechnique Montréal [9]Canada CIFAR AI Chair [10]Technical University Munich (TUM)

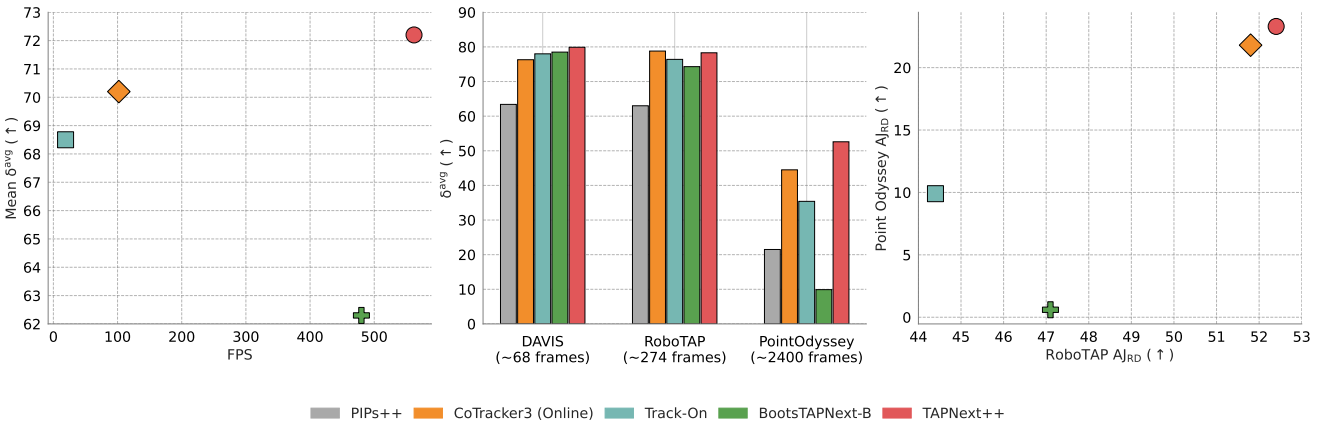

Figure 1. **Overview.** TAPNext++ sets a new state-of-the-art for online point tracking by simultaneously improving accuracy, speed, and long-term robustness. *Left:* Our method reaches the best speed-accuracy trade-off, achieving higher mean δ^{avg} than competing online methods while being substantially faster. *Middle:* Despite running frame-by-frame without explicit memory, TAPNext++ outperforms previous methods on long-sequence benchmarks. *Right:* TAPNext++ also demonstrates superior robustness to occlusion and re-appearences, achieving state-of-the-art AJ_{RD} scores, a new metric evaluating point tracking after re-detection, on both RoboTAP and PointOdyssey.

Abstract

Tracking-Any-Point (TAP) models aim to track any point through a video which is a crucial task in AR/XR and robotics applications. The recently introduced TAPNext approach proposes an end-to-end, recurrent transformer architecture to track points frame-by-frame in a purely online fashion – demonstrating competitive performance at minimal latency. However, we show that TAPNext struggles with longer video sequences and also frequently fails to re-detect query points that reappear after being occluded or leaving the frame. In this work, we present TAPNext++, a model that tracks points in sequences that are orders of magnitude longer while preserving the low memory and compute footprint of the architecture. We train the recurrent video transformer using several data-driven solutions, including training on long 1024-frame sequences enabled by sequence parallelism techniques. We highlight that re-detection performance is a blind spot in the current literature and introduce a new metric, Re-Detection Average Jaccard (AJ_{RD}), to explicitly evaluate tracking on re-appearing points. To improve re-detection of points, we introduce tailored geometric augmentations, such as periodic roll that simulates point re-entries, and supervising occluded points. We demonstrate that recurrent transformers can be substantially improved for point tracking and set a new state-of-the-art on multiple benchmarks. Model and code can be found at https://tap-next-plus-plus.github.io.

1. Introduction

Tracking Any Point (TAP) in a video is a fundamental computer vision problem with many downstream applications in robotics, augmented reality, video editing, and 3D/4D reconstruction. Depending on the application, point tracking is performed online or offline, in 2D or 3D and under varying computational constraints. In this work, we focus on highly efficient, frame-by-frame 2D point tracking which is very relevant for AR applications such as anchoring digital content in the real world. In contrast to optical flow estimation, which predicts dense pixel-level correspondences between adjacent frames, TAP methods attempt to

[*]Equal contribution.
[†]Work done during an internship at Google.

consistently track points over multiple frames under strong appearance changes and occlusions. However, despite recent progress [2, 14, 30], state-of-the-art methods struggle to achieve robust performance in long sequences as shown in PointOdyssey [29]. Common strategies to counteract this issue are: (1) tracking methods that operate over a sliding window of frames instead of one frame at a time [12, 14], (2) hard coding an explicit memory representation of past frames [1, 2], (3) re-detecting points with appearance-based foundation models [2] and (4) keeping the first frame as a persistent input at every time step [12]. However, as our experiments demonstrate, these remedies fail to solve the root issue of temporally deteriorating representations and typically increase runtime and memory usage.

TAPNext [30] reformulates point tracking as next token prediction and deploys linear recurrent SSM layers [23] to track scene dynamics, particularly of query points, over time. Its constant-sized, linear-recurrent memory allows TAPNext to jointly track 1024 points at 348 FPS on an H100 GPU, demonstrating the efficiency required for mobile applications in robotics and augmented reality. However, as also noted in concurrent work [2], the BootsTAPNext model trained on a large number of short videos at a fixed 256×256 resolution [7] fails to generalize to longer sequences [29]. In this work, we present TAPNext++, a model demonstrating that these tracking failures are not inherent to the underlying recurrent video transformer architecture and can be effectively tackled through improved training strategies and datasets.

Tracking points over long time horizons reveals another severe limitation of current TAP methods: when points reappear after being occluded or leaving the view, their re-detection frequently fails. This limits applications, e.g. in AR/VR, where both the camera and the scene are dynamic and reappearances are a frequent occurrence. Recent approaches resort to hard-coded appearance-based re-detection without relying on motion priors required to disambiguate, e.g., multiple instances of visually similar points. Some approaches [14] track occluded points explicitly within the image, which improves re-detection after short-term occlusions, but usually fail for long-term occlusions, particularly when tracked points venture outside the image boundaries. Other methods [27] attempt to build an explicit 3D map using monocular video depth foundation models [18], assuming a mostly static scene and generous computational resources.

We find that the re-detection limitation is a blind spot in current point tracking metrics and evaluations. The closest related metric is the survival rate [29], which measures the average number of frames until point tracking failure. However, the survival rate does not directly measure re-detection performance because it includes many points that are always visible. Therefore, we propose a variation of the Average Jaccard metric specifically targeting the performance of point tracking after reappearance and evaluate recent approaches on it.

Our main contributions are as follows:

- We introduce methods that allow scaling the training of TAPNext architectures to long sequences and multiple resolutions – resulting in a lightweight, fully online model that sets a new state-of-the-art on multiple point tracking benchmarks.
- We introduce Kubric-1024, which extends the original synthetic Kubric dataset [11] to 1024 frames and analyze the effects of long-sequence training on memory.
- We propose novel re-detection metrics that directly measure the accuracy of point tracks after re-appearance and improve on these metrics through tailored geometric augmentations.
- We demonstrate how our contributions effectively enhance the memory of the TAPNext model, outperforming all existing methods in long-context and re-detection scenarios. We highlight that we *do not* require any new model architectures to surpass existing methods while further enhancing the compute and memory efficiency of the original TAPNext.

2. Related Work

Point Tracking. Tracking Any Point (TAP) [7] is a computer vision task where the model is tasked to find the position and visibility of a given *query* point throughout a video. Here, we focus on sequential 2D point tracking, without access to future frames, for its relevance in compute-constrained mobile applications.

Windowed Methods. The majority of prior methods for TAP perform point tracking in two stages. First, they precompute image or video features and then they use these features in an appearance matching and iterative refinement process to track predictions over a window of frames or an entire video. More specifically, TAPNet [7] performs tracking via a simple feature matching on CNN features. TAPIR [8] introduces an iterative refinement procedure within TAPNet. BootsTAPIR [9] extends the method with a large-scale, real world dataset to bootstrap self-supervised training on natural, unlabeled videos. The TAPTR family [16, 17, 24] performs tracking using a DETR-like architecture. LocoTrack [4] incorporates bidirectional 4D correspondences for fast and accurate tracking. CoTracker [13] presents a hybrid CNN-transformer architecture demonstrating the benefit of tracking all point queries jointly. CoTracker3 [14] extends CoTracker by performing semi-supervised finetuning on a real-world pseudo-labeled dataset, showing the importance of training data for point tracking. AllTracker [12] incorporates high-

Figure 2. **Qualitative Comparison on Challenging Re-entry Scenarios.** We compare TAPNext++ with prior methods on two sequences featuring point re-entries at varying locations. *Left:* An object leaves the frame on the right and re-enters from the left. BootsTAPNext and Track-On fail to re-detect the object upon re-entry. In contrast, CoTracker3 and TAPNext++ successfully relocate the points. However, CoTracker3 fails to track static points on the texture-less table, which TAPNext++ tracks correctly. The variant TAPNext++ without Roll also struggles with immediate re-detection, highlighting the importance of roll augmentation for re-entry robustness. *Right:* Three objects are juggled, repeatedly leaving and re-entering the frame. Point trajectories for the last 8 frames are visualized as traces for clarity. TAPNext++ with roll is the only method that robustly tracks all objects through these challenging dynamics, all competing methods fail.

resolution, pixel-dense point tracking by a multi-view optical flow optimization while re-inferring the query frame to avoid temporal degradation. However, on long sequences as present in the PointOdyssey [29] benchmark, points deviate significantly from their initialization, and attending on overlapping sliding windows or query frames is often insufficient, as shown in Fig. 1.

Frame-by-frame Methods. Arguably, defining such task specific context windows could be avoided if temporal information would simply propagate from frame to frame. The TrackOn [1, 2] family performs online, frame-by-frame tracking using patch classification and refinement to identify correspondences and propagates temporal information through explicit memory banks. Notably, TAPNext [30] shows that it is possible to efficiently perform frame-by-frame TAP without explicit memory banks and without point tracking-specific inductive biases. Instead, TAPNext adopts a token-based modeling formulation and casts TAP as temporal masked decoding. TAPNext uses an architecture [23] with alternating ViT and State-Space-Model (SSM) blocks for spatial and temporal feature propagation, respectively. The linear recurrent layers are efficient to train and infer on modern GPU hardware allowing state-of-the-art processing speed for point tracking. However, despite performing well on common metrics and short-term benchmarks, we demonstrate its severe limitations in

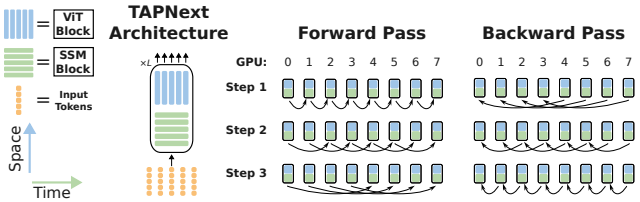

Figure 3. **Distributed Parallel Scan during Training.** On the left, the TAPNext architecture with its SSM and ViT Blocks is shown. Right, example of information flow during multi-gpu training using a distributed parallel scan. Note that information is only passed between the SSM Blocks of the GPUs and not the ViT Blocks. Only inter-GPU communication is shown, intra-GPU parallel scans are not shown for simplicity. For a detailed figure of the TAPNext architecture we refer to [30].

propagating information over long time horizons and in re-detecting points after longer occlusions, two crucial capabilities for real-world applications. In this paper, we overcome the limitations of TAPNext without relying on external appearance-based foundation features [21] that do not model motion.

3. Method

TAPNext++, as TAPNext [30], processes videos in a causal online fashion. It processes videos of T RGB frames, each of size $H \times W$ pixels and Q point queries per video, each

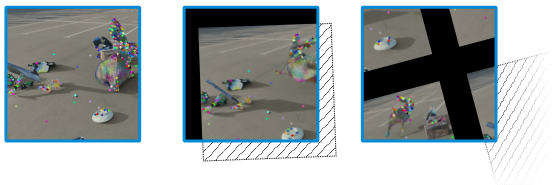

Figure 4. **Roll Augmentation.** Videos are rotated and translated smoothly over time, wrapping around the image with a gap. Ground-truth points are visualized.

of which is an index of the form (t, x, y). For each frame in a video, we apply a standard ViT-style linear projection followed by learned positional encodings that are added to the linear projections. As a result, we get $T \times h \times w$ visual tokens, where h and w are sizes after patchification. We apply spatial positional encoding for each point query. Specifically, the (x, y) component serves as a 2D index in the positional embedding tensor. For each point query, we initialize T *track tokens*, where $T - 1$ are filled with a `[MASK]` token and one, at position t (from the point query), is filled with the position embedding of the given query representing the spatial coordinates x and y. Consequently, we have $T \times Q$ point track tokens. We simply concatenate $T \times Q$ track tokens with $T \times h \times w$ visual tokens after position embedding over the spatial axis resulting in $T \times (h \times w + Q)$ spatiotemporal tokens. The TAPNext architecture performs attention only over the spatial dimension for every time step while causally propagating information for each token sequence independently in the temporal dimension using an SSM layer, see Fig. 3.

While Zholus et al. [30] showed that this architecture can be applied to standard point tracking benchmarks, our focus lies on identifying its limitations which we elucidate and address. One major flaw of the original TAPNext is its inability to track points for an extended temporal horizon (∼150 frames). This is perhaps unsurprising: the model is trained on 48-frame sequences, meaning that longer sequences are out-of-domain. Although recent works [6, 23] show that the Linear-Recurrent-Unit used in the TAPNext architecture is able to perform well on sequences that are larger than the training sequences, these works train on longer sequences initially in order to learn stable update rules. Motivated by these results, we hypothesise that training on longer sequences also increases long-term tracking stability. Therefore, our first goal is to fine-tune a previously released TAPNext checkpoint on 1024 frames.

Long-Sequence Training. Training TAPNext on long sequences requires fitting lengthy computation graphs and activations into memory, which exceeds single-GPU limits for sequences of 1024 frames. To enable end-to-end training on such sequences we employ sequence parallelism, sharding the input sequence only along the time dimension and assigning each chunk to a different GPU. TAPNext's architecture builds upon Real-Gated Linear Recurrent Units (RG-LRU), a variant of State Space Models (SSMs). Thanks to their linearity, SSM temporal processing can be parallelized using the associative scan primitive. To exploit this property under sequence parallelism, we introduced a distributed parallel scan for both RG-LRU and temporal convolution layers during forward and backward passes. In this setup, each GPU first performs a local scan over its assigned sequence chunk. This is followed by an efficient, logarithmic-time merge operation across GPUs to combine hidden states over chunk boundaries, and a final local update based on the merged states. This distributed scan enables end-to-end training on long sequences by parallelizing temporal computation across multiple devices, overcoming single-device memory limitations while maintaining full temporal context. As can be seen in Fig. 3, distributed parallel scan allows performing forward and backward pass in three instead of seven steps on eight GPUs compared to a naïve sequential communication.

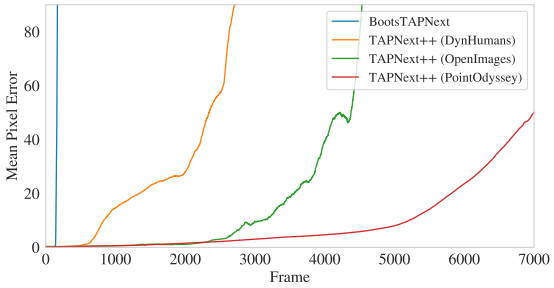

Figure 5. **Long-Sequence Stability.** Mean Pixel Error of 16x16 points after repeatedly inferring a static image through TAPNext variants. Finetuning on 1024-frame videos from PointOdyssey most effectively extends the memory longevity.

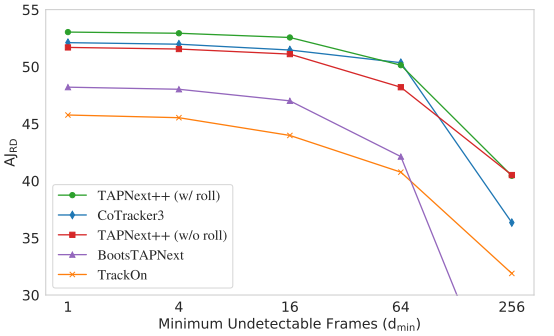

Figure 6. **AJ_{RD} Comparison.** Re-Detection Average Jaccard (AJ_{RD}) for different d_{min} values on RoboTAP. TAPNext++ with roll augmentations preserves AJ after long periods of undetectability. The window-based CoTracker3 exhibits a drop in AJ_{RD} following a sequence of 256 undetectable frames.

Long Sequence Data. Point tracking research requiring long sequence training data with ground truth annotations is currently limited mainly to PointOdyssey [29]. Given that

Figure 7. **Long-Term Dynamic Tracking Example.** We compare state-of-the-art online trackers on tracking clock hands, visualizing a trace of the last 6 frames. While the original BootsTAPNext, Track-On and CoTracker3 fail at tracking both clock hands for the full duration of the video, TAPNext++ tracks both until the end. CoTracker3 and Track-On are only able to track the hour hand due to its fine structure, additionally CoTracker3 loses the hour hand at the end of the video.

the training split of PointOdyssey (v1.2) contains only 131 videos, we identified a need for a new, large-scale, long-sequence tracking dataset. To address this, we generated a synthetic dataset using Kubric [11], consisting of 10,000 videos with 1024 frames. Each scene uses an HDRI from the public Polyhaven [28] library for realistic background and illumination. We populate scenes with 10–20 static and 1–10 dynamic objects sampled from GSO [10], which are randomly placed via an overlap-avoiding algorithm within designated spawn regions. The camera follows a smooth trajectory between a random start and end position, sampled within a spherical shell around the world origin to ensure objects remain in view. To simulate more natural viewing patterns, this base path is augmented with sinusoidal noise, mimicking unsteady camera motion. Furthermore, to create more dynamic pans, the camera's look-at point is animated with sinusoidal noise and projected onto the scene's ground plane, rather than being fixed at the origin. To maintain object motion and interaction throughout the long sequences, we use the PyBullet [5] physics simulator and introduce velocity "bumps": if a dynamic object's velocity falls below a given threshold, a new random linear and angular velocity is applied. This velocity includes a slight bias toward the scene origin, encouraging objects to move within the camera's field of view and preventing scenes from becoming static over time.

Augmentations. Long-sequence tracking demands robustness to points that become invisible for extended periods, either through occlusion or by exiting and re-entering the camera's field of view. These re-entry events are common in real-world applications, such as augmented reality, but pose a significant persistent memory challenge for tracking models. This is especially challenging if points reappear far from their last known position, particularly for trackers that explicitly enforce local search windows. To improve the model's ability to handle such cases, we employ data augmentations that simulate camera jitter and roll during training. We apply temporally-smooth, sinusoidal random translations and rotations to each video sequence and its corresponding point tracks. Crucially, translations are implemented as periodic shifts; pixels and track coordinates shifted off one image boundary wrap around to the opposite side with a given margin. This mechanism explicitly simulates query points leaving the frame on one side and re-entering from another, forcing the model to rely more on appearance matching for re-detection, rather than assuming spatio-temporal proximity to the point's last visible position. To prevent double-appearance of the same tracking point in a single frame, we add a margin of $\sqrt{\left(\frac{H}{2}\right)^2 + \left(\frac{W}{2}\right)^2}$ between the wrapped images. This synthesis of challenging re-entry scenarios improves model robustness against spatial displacements in point trajectories. An example of the roll augmentation can be found in Fig. 4. Additionally we apply aspect ratio augmentations by picking a random aspect ratio between 0.5 and 2.0 during training.

Tracking Occluded Points via Weighted Loss. The standard loss function used in TAPNext only supervises coordinate prediction for points when they are visible. This provides no incentive for the model to predict plausible locations for points that are temporarily occluded, which can hinder their re-detection upon reappearance and negatively impact long-term tracking performance. To improve tracking robustness through occlusions, we modify the training loss to include supervision for points that are occluded but remain within the image boundaries, weighting the position loss of occluded points by 0.2 compared to the position loss on visible points.

High-Resolution Fine-Tuning. Following prior work [13],

Table 1. **Online Tracking Comparison.** We report results on multiple point tracking benchmarks and compare against current state-of-the-art online trackers.

#	Method	Res.	PointOdyssey				DAVIS			RGB-Stacking			RoboTAP				Kinetics			Mean
			$\delta^{avg}\uparrow$	Survival ↑	MTE ↓	$AJ_{RD}\uparrow$	AJ ↑	$\delta^{avg}\uparrow$	OA ↑	AJ ↑	$\delta^{avg}\uparrow$	OA ↑	AJ ↑	$\delta^{avg}\uparrow$	OA ↑	$AJ_{RD}\uparrow$	AJ ↑	$\delta^{avg}\uparrow$	OA ↑	$\delta^{avg}\uparrow$
1	PIPs++	256 × 256	21.5	38.1	46.1	–	–	63.4	–	–	58.5	–	–	63.0	–	–	–	–	–	51.6
2	CoTracker2	384 × 512	30.2	55.2	215.0	3.9	62.2	75.7	89.3	67.4	78.9	85.2	58.6	70.6	87.0	33.3	48.8	64.5	85.8	64.0
3	CoTracker3 (Online)	384 × 512	44.5	56.3	20.7	21.8	63.8	76.3	90.2	71.7	83.6	90.2	**66.4**	**78.8**	**90.8**	51.8	55.8	68.5	88.3	70.2
4	Track-On	384 × 512	35.4	47.5	33.5	9.9	65.0	78.0	90.8	71.4	**85.2**	91.7	63.5	76.4	89.4	44.4	53.9	67.3	87.8	68.5
5	BootsTAPNext-B	256 × 256	9.9	13.0	92.1	0.6	65.2	78.5	91.2	66.2	78.3	86.8	62.6	74.3	88.4	47.1	**57.3**	**70.6**	87.4	62.3
6	**TAPNext++**	256 × 256	**52.6**	**67.9**	**13.4**	**23.3**	**66.6**	**79.9**	**92.1**	**73.4**	84.8	**95.1**	61.1	75.2	89.6	**52.4**	54.4	68.7	**89.0**	**72.2**

Table 2. **Inference Speed.** Speed comparison of TAPNext++ to TAPNext, LocoTrack-B (256 × 256), online CoTracker3 and Track-On running on Nvidia H100 GPUs. The latency metric is defined as the maximum (worst case) time between passing a frame to the model and receiving predicted points and it includes the time it takes to fill and process the initial frame buffer. *(frame)* indicates per frame inference, *(window)* is when we track with non-overlapping chunks of 32 frames. All models are implemented in PyTorch with TAPNext++ relying on FlashAttention3 [25].

Query Points	Model	Average FPS ↑	Latency (ms) ↓
256	LocoTrack-B (window)	452	2210
	CoTracker3 (online)	102	80
	Track-On (frame)	28	36
	TAPNext (frame)	189	5.29
	TAPNext++ (frame)	193	**5.18**
	TAPNext (window)	480	66
	TAPNext++ (window)	**562**	57
1024	LocoTrack-B (window)	124	8000
	CoTracker3 (online)	45	177
	Track-On (frame)	23	44
	TAPNext (frame)	182	5.47
	TAPNext++ (frame)	191	**5.23**
	TAPNext (window)	300	106
	TAPNext++ (window)	**348**	57

which shows high-resolution inputs improve point tracking, we fine-tune our best TAPNext++ checkpoint at 512 × 512. Since the TAPNext encoder uses a fixed 8 × 8 patch size, the larger input produces an expanded spatial token grid. Consequently, we resize the learned 2D positional embeddings from the base 32 × 32 grid to 64 × 64 using bicubic interpolation. The output layer size (256 + 256 bins) remains unchanged. The model is then fine-tuned on 256-frame sequences for 120k steps.

4. Experiments

4.1. Metrics

We evaluate performance using the standard metrics from TAP-Vid [7]: Occlusion Accuracy (OA), positional error (δ^{avg}), and Average Jaccard (AJ) which evaluates both occlusion and position accuracy. Additionally, for the PointOdyssey dataset, we report Survival Rate and Median Translation Error (MTE) as defined by Zheng et al. [29].

While informative, these metrics average performance over entire tracks and do not specifically quantify a tracker's ability to re-detect points that reappear after being undetectable (i.e. occluded or out-of-frame) for many frames.

Re-Detection Average Jaccard AJ_{RD}

To address this, we propose the *Re-Detection Average Jaccard* (AJ_{RD}), a new metric that measures tracking quality after a point reappears, conditioned on how long it was undetectable. We start with definitions:

Undetectable Point: A point is undetectable in frame t if its ground-truth visibility v_t is 0, either because it is occluded by another object or itself, or it is outside the image bounds. Otherwise, the point is detectable ($v_t = 1$).

Reappearance Event: A reappearance event occurs for a track at frame t_r if the point is detectable at frame t_r but was undetectable for the d preceding frames, i.e., $v_{t_r} = 1$ and $v_{t_{r-1}} = \cdots = v_{t_{r-d}} = 0$.

Duration of Undetectability: The value $d \geq 1$ is the duration of undetectability.

Eligible Reappearance Event: To focus on how well trackers handle different lengths of time intervals for point disappearance, we only consider a reappearance event i if its duration d_i is *strictly greater than the maximum disappearance duration of all previous reappearance events* for that same track. We call such events eligible. This ensures that for any given track, only events that set a new record for invisibility duration are counted. For example, if for a particular track the sequence of $d_i = \mathbf{1}, 1, \mathbf{4}, 3, 2$ then the *eligible* reappearance events are highlighted in bold.

A tracker might locate a point in frame t_r but fail to track it in frames $t_r + 1, t_r + 2, \ldots$, even if the point remains visible. AJ_{RD} is designed to measure this post-reappearance tracking quality, particularly for points that were undetectable for a long time. We define $AJ_{RD}^{d_{min}}$ as follows: First, identify all *eligible* reappearance events where the duration of undetectability d is greater than or equal to d_{min}. For each such event occurring at a frame t_r, consider the track segment consisting of all frames from t_r to the end of the video sequence. $AJ_{RD}^{d_{min}}$ is the Average Jaccard (AJ) [7] calculated only over these specific track segments (i.e., for all frames $t \geq t_r$). Therefore this metric measures tracking quality on segments that immediately follow a reappearance after an undetectability period of at least d_{min} frames. It can be plotted against

d_{min} to show how post-reappearance tracking quality behaves as the duration of undetectability increases. To report a summary score across different intervals, we define AJ_{RD} as the mean of $\text{AJ}_{\text{RD}}^{d_{\text{min}}}$ over several d_{min} values $d_{\text{min}} \in \{1, 4, 16, 64, 256\}$. This value is calculated *per sample* and averaged over all samples in a dataset. We calculate this metric for PointOdyssey and RoboTAP as both contain long sequences and many reappearance events.

4.2. Experimental Setup

We conduct extensive experiments to evaluate the effectiveness of our proposed training adaptations. All models were fine-tuned based on the publicly available BootsTAPNext-B checkpoint. We performed fine-tuning for 20,000 steps per dataset utilizing the AdamW optimizer [20] with a cosine annealing learning rate schedule [19]. The peak learning rate was set to 2×10^{-5}, preceded by a 100-step linear warmup. For coordinate prediction, we implemented a loss re-weighting strategy where contributions from points that are occluded but remain within the frame are down-weighted by a factor of 0.2. For fine-tuning, we use 8 NVIDIA H100 GPUs with an effective batch size of 1 while the evaluations were performed on a single H100.

Training Datasets. In this work we used a combination of video and image datasets. We utilized three primary video datasets providing ground-truth point tracks for both dynamic objects and static background regions: 1) The training split of PointOdyssey v1.2 [29], which comprises 131 videos featuring diverse object types and complex, long-term camera trajectories. 2) A newly generated synthetic long-sequence Kubric dataset (Kubric-1024), consisting of 10,000 videos, each 1024 frames in length. This dataset models dynamic scenarios, including falling and randomly jumping objects under various camera motions. 3) A novel synthetic dataset, also newly generated, containing 10,000 videos of 240 frames each, depicting dynamic indoor scenes with human motion and camera movement (DynHumans). For our initial long-sequence experiments, we also employed 10,000 images from OpenImages v7 [15]. These images were used to artificially synthesize long-sequence videos of 1024 frames by applying smooth in-plane homography transformations and the aforementioned roll augmentation to a single static image. Ground-truth point trajectories were derived from randomly sampled points on the image.

Evaluation Datasets. We evaluate our method on both real-world and synthetic video datasets. We use three *real* benchmarks: TAP-Vid-DAVIS [7], which contains 30 videos (34-104 frames) from the DAVIS 2017 validation set [22]; TAP-Vid-Kinetics [7], consisting of 1,189 videos (250 frames each) from the Kinetics-700 validation set [3]; and RoboTAP [26], with 265 videos (average 271.9 frames). To quantify long-term tracking performance, we use the *synthetic* PointOdyssey v1.2 dataset [29], featuring 13 videos with 1,149-4,325 frames in its test split. We also report results on TAP-Vid-RGB-Stacking [7], a second synthetic benchmark consisting of 50 videos of 250 frames each. All images are resized to 256×256 for evaluation unless otherwise stated.

4.3. Evaluations

The efficacy of recurrent architectures in point tracking is limited by state degradation over long sequences, hindering the application in long-term tracking scenarios. To quantify this limitation, we first analyze the memory capability of BootsTAPNext. We conducted a controlled experiment by tracking points on a static image sequence: query points are initialized on the first frame, and since the scene is static, ground-truth points remain stationary. As illustrated in Fig. 5, BootsTAPNext's baseline performance degrades significantly after approximately 150 frames even when the same static frame is fed as input at every time step, indicating a loss of information about initial query locations in its recurrent state. We hypothesized that training on longer video sequences could mitigate this effect. Indeed, fine-tuning on DynHumans, which consists of 240-frame videos, extended the model's effective memory to 600 frames in the same static-image experiment.

This initial result motivates us to train on even longer sequences of 1024 frames. We first explore synthetically generated long videos from OpenImages, using homography and roll augmentations to simulate motion. Although this approach further improved memory retention, as shown in Fig. 5, training exclusively on this synthetic data proved detrimental to the model's overall tracking performance across standard benchmarks, achieving low Average-Jaccards of 25.00 (DAVIS), 41.00 (RoboTAP) and 30.76 (Kinetics). We attribute this to the model overfitting to homographic transformations, compromising its ability to generalize to more complex, real-world motion.

Consequently, we shifted to fine-tuning using the PointOdyssey dataset, picking a random window of 1024 frames for each sample. Fine-tuning TAPNext on PointOdyssey extended its memory capacity significantly further than either DynHumans or synthetic OpenImages data (Fig. 5). Furthermore, fine-tuning on PointOdyssey using random crops with a standard 1:1 aspect ratio yielded substantial improvements on the PointOdyssey test set, leading to state-of-the-art results including an enhanced survival rate of 68.31 (Tab. 3, row 2), confirming the potential of the TAPNext architecture for long-term tracking. However, this specialized training led to reduced performance on other benchmarks like DAVIS, indicating overfitting to the PointOdyssey training set.

Table 3. **Impact of Augmentations.** Data augmentation while finetuning TAPNext on PointOdyssey (PO) impacts generalizability. While less augmentations lead to higher results on PO, using roll and aspect ratio augmentations lead to overall better results across other datasets.

#	Aug.		PointOdyssey				DAVIS			RoboTAP				Kinetics		
	Roll	Aspect	δ^{avg} ↑	Survival ↑	MTE ↓	AJ_{RD} ↑	AJ ↑	δ^{avg} ↑	OA ↑	AJ ↑	δ^{avg} ↑	OA ↑	AJ_{RD} ↑	AJ ↑	δ^{avg} ↑	OA ↑
1	✓	✓	49.3	63.5	17.4	20.4	**66.0**	**79.5**	**91.9**	59.8	74.2	88.4	51.1	53.6	68.3	88.5
2			51.4	68.3	**15.6**	**22.8**	64.7	78.4	91.6	59.3	73.7	88.0	50.7	53.0	68.0	88.2
3		✓	**52.0**	**68.6**	16.4	22.3	65.3	79.1	91.4	59.2	73.9	88.0	50.0	52.6	67.8	87.8
4	✓		49.7	65.0	15.8	20.7	64.7	78.8	91.4	**60.4**	**74.7**	**88.7**	**52.0**	**53.7**	**68.4**	**88.6**

Table 4. **Multi-Dataset Finetuning.** Incorporating multiple datasets in the finetuning helps generalizability while also improving the results on PointOdyssey (PO) compared to training on PO alone.

#	Dataset			PointOdyssey				DAVIS			RoboTAP				Kinetics		
	PO	Kub1024	DynHumans	δ^{avg} ↑	Survival ↑	MTE ↓	AJ_{RD} ↑	AJ ↑	δ^{avg} ↑	OA ↑	AJ ↑	δ^{avg} ↑	OA ↑	AJ_{RD} ↑	AJ ↑	δ^{avg} ↑	OA ↑
1	✓	✓	✓	**52.6**	67.9	**13.4**	**23.3**	**66.6**	**79.9**	**92.1**	**61.1**	75.2	**89.6**	52.4	**54.4**	**68.7**	**89.0**
2	✓	✓		51.8	**71.0**	14.4	**23.3**	65.6	79.0	92.0	**61.1**	**75.4**	89.0	**52.6**	53.9	68.4	88.7
3	✓			49.3	63.5	17.4	20.4	66.0	79.5	91.9	59.8	74.2	88.4	51.1	53.6	68.3	88.5

To improve generalization, we incorporate aggressive roll augmentations and variable aspect ratio crops during fine-tuning. As shown in Tab. 3 (row 1 vs. row 2), these augmentations improve performance across multiple benchmarks (e.g., boosting DAVIS AJ from 64.7% to 66.0%) while maintaining a high survival rate of 63.5% on PO. Additionally, roll augmentations significantly improve re-detection capabilities, as shown in Fig. 6.

Given PointOdyssey's limited size (131 training samples), we further enhance robustness by fine-tuning on a mixture of datasets: PointOdyssey, Kubric-1024, and DynHumans. As shown in Tab. 4, incorporating multiple datasets improves generalization and boosts PointOdyssey performance beyond training on PointOdyssey alone. While mixing PointOdyssey and Kubric-1024 yields the highest PO survival rate (71.0%), mixing all three datasets (Tab. 4, row 1) provides the best overall performance across all benchmarks and constitutes our final model, TAPNext++. Our fine-tuning strategy on TAPNext achieves a new state of the art in long-term tracking. As shown in Tab. 1, TAPNext++ outperforms previous methods on PointOdyssey (δ^{avg} 52.6, Survival 67.9%, AJ_{RD} 23.3), DAVIS (AJ 66.6%, δ^{avg} 79.9, OA 92.1%), and RGB-Stacking (AJ 73.4%, OA 95.1%). Notably, it achieves the highest mean δ^{avg} (72.2) across all benchmarks, despite using a lower input resolution (256×256) than competitors and having the lowest latency. This contrasts sharply with BootsTAPNext-B, which struggles on PointOdyssey (Survival 13.0%), demonstrating our method successfully addresses its long-sequence limitations. Our model also achieves the best AJ_{RD} on PointOdyssey and RoboTAP, highlighting its superior re-detection capabilities. Finally, we evaluate a high-resolution version of our model. The input images are directly resized to 512×512 resolution without prior resizing to 256×256 and the TAP metrics are calculated on the 256×256 scale. The high-resolution model achieves δ^{avg} 52.2, AJ 33.5%, Survival 69.1% on PointOdyssey and δ^{avg} 80.0, AJ 66.9%, OA 92.5% on Davis. In addition to the advantages discussed above, we compare inference speed among different online trackers in Tab. 2, demonstrating the low latency and high FPS of TAPNext++. Qualitative comparisons are shown in Fig. 2 and Fig. 7, highlighting that our method is the only one successfully tracking the challenging re-detection and long-term scenarios.

5. Conclusion

This work addressed key limitations of current Tracking-Any-Point (TAP) models: their degraded performance on long sequences and their inability to re-detect points following occlusion or points re-entering the frame. We presented TAPNext++, a model that re-purposes the efficient TAPNext architecture to show that these failures are not architectural but are a result of insufficient training strategies and datasets. We successfully scaled the model to long sequences by fine-tuning on 1024-frame videos, which we enabled through multi-GPU sequence parallelism. To specifically target re-detection failures, we introduced tailored geometric augmentations and evaluated their effectiveness. Furthermore, we identified re-detection as a blind spot in current evaluations and proposed a new metric called Re-Detection Average Jaccard (AJ_{RD}), to rigorously measure post-reappearance tracking quality. We encourage the community to further work on remaining limitations of TAPNext++ and other point tracking approaches, such as solving hour-long point tracking with constant memory, tracking occluders and modeling motion ambiguities. Our resulting model, TAPNext++, achieves state-of-the-art performance on multiple point tracking benchmarks while retaining the low memory and compute footprint required for real-world applications.

References

[1] Görkay Aydemir, Xiongyi Cai, Weidi Xie, and Fatma Guney. Track-on: Transformer-based online point tracking with memory. In *The Thirteenth International Conference on Learning Representations*, 2025. 2, 3

[2] Görkay Aydemir, Weidi Xie, and Fatma Güney. Track-on2: Enhancing online point tracking with memory. *arXiv preprint arXiv:2509.19115*, 2025. 2, 3

[3] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017. 7

[4] Seokju Cho, Jiahui Huang, Jisu Nam, Honggyu An, Seungryong Kim, and Joon-Young Lee. Local all-pair correspondence for point tracking. In *European conference on computer vision*, pages 306–325. Springer, 2024. 2

[5] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021. 5

[6] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024. 4

[7] Carl Doersch, Ankush Gupta, Larisa Markeeva, Adria Recasens, Lucas Smaira, Yusuf Aytar, Joao Carreira, Andrew Zisserman, and Yi Yang. Tap-vid: A benchmark for tracking any point in a video. *Advances in Neural Information Processing Systems*, 35:13610–13626, 2022. 2, 6, 7

[8] Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. Tapir: Tracking any point with per-frame initialization and temporal refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10061–10072, 2023. 2

[9] Carl Doersch, Pauline Luc, Yi Yang, Dilara Gokay, Skanda Koppula, Ankush Gupta, Joseph Heyward, Ignacio Rocco, Ross Goroshin, João Carreira, and Andrew Zisserman. Bootstap: Bootstrapped training for tracking-any-point. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 3257–3274, 2024. 2

[10] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2553–2560. IEEE, 2022. 5

[11] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, et al. Kubric: A scalable dataset generator. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3749–3761, 2022. 2, 5

[12] Adam W Harley, Yang You, Xinglong Sun, Yang Zheng, Nikhil Raghuraman, Yunqi Gu, Sheldon Liang, Wen-Hsuan Chu, Achal Dave, Suya You, et al. Alltracker: Efficient dense point tracking at high resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5253–5262, 2025. 2

[13] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker: It is better to track together. In *European conference on computer vision*, pages 18–35. Springer, 2024. 2, 5

[14] Nikita Karaev, Yuri Makarov, Jianyuan Wang, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker3: Simpler and better point tracking by pseudo-labelling real videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6013–6022, 2025. 2

[15] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *International journal of computer vision*, 128(7):1956–1981, 2020. 7

[16] Hongyang Li, Hao Zhang, Shilong Liu, Zhaoyang Zeng, Feng Li, Bohan Li, Tianhe Ren, and Lei Zhang. Taptrv2: Attention-based position update improves tracking any point. *Advances in Neural Information Processing Systems*, 37: 101074–101095, 2024. 2

[17] Hongyang Li, Hao Zhang, Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, and Lei Zhang. Taptr: Tracking any point with transformers as detection. In *European Conference on Computer Vision*, pages 57–75. Springer, 2024. 2

[18] Zhengqi Li, Richard Tucker, Forrester Cole, Qianqian Wang, Linyi Jin, Vickie Ye, Angjoo Kanazawa, Aleksander Holynski, and Noah Snavely. Megasam: Accurate, fast and robust structure and motion from casual dynamic videos. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 10486–10496, 2025. 2

[19] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 7

[20] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 7

[21] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. 3

[22] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alex Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*, 2017. 7

[23] Viorica Pătrăucean, Xu Owen He, Joseph Heyward, Chuhan Zhang, Mehdi S. M. Sajjadi, George-Cristian Muraru, Artem Zholus, Mahdi Karami, Ross Goroshin, Yutian Chen, Simon Osindero, João Carreira, and Razvan Pascanu. Trecvit: A recurrent video transformer, 2024. 2, 3, 4

[24] Jinyuan Qu, Hongyang Li, Shilong Liu, Tianhe Ren, Zhaoyang Zeng, and Lei Zhang. Taptrv3: Spatial and temporal context foster robust tracking of any point in long video. *arXiv preprint arXiv:2411.18671*, 2024. 2

[25] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37: 68658–68685, 2024. 6

[26] Mel Vecerik, Carl Doersch, Yi Yang, Todor Davchev, Yusuf Aytar, Guangyao Zhou, Raia Hadsell, Lourdes Agapito, and Jon Scholz. Robotap: Tracking arbitrary points for few-shot visual imitation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5397–5403. IEEE, 2024. 7

[27] Yuxi Xiao, Jianyuan Wang, Nan Xue, Nikita Karaev, Yuri Makarov, Bingyi Kang, Xing Zhu, Hujun Bao, Yujun Shen, and Xiaowei Zhou. Spatialtrackerv2: Advancing 3d point tracking with explicit camera motion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6726–6737, 2025. 2

[28] Greg Zaal, Rob Tuytel, Rico Cilliers, James Ray Cock, Andreas Mischok, Sergej Majboroda, Dimitrios Savva, and Jurita Burger. Polyhaven: a curated public asset library for visual effects artists and game designers, 2021. 5

[29] Yang Zheng, Adam W. Harley, Bokui Shen, Gordon Wetzstein, and Leonidas J. Guibas. Pointodyssey: A large-scale synthetic dataset for long-term point tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19855–19865, 2023. 2, 3, 4, 6, 7

[30] Artem Zholus, Carl Doersch, Yi Yang, Skanda Koppula, Viorica Patraucean, Xu Owen He, Ignacio Rocco, Mehdi S. M. Sajjadi, Sarath Chandar, and Ross Goroshin. Tapnext: Tracking any point (tap) as next token prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9693–9703, 2025. 2, 3, 4

TAPNext++: What's *Next* for Tracking Any Point (TAP)?

Supplementary Material

A. Frequently Asked Questions

Q: Is the improved performance on PointOdyssey simply due to training on the PointOdyssey dataset?

To verify generalization, we trained a variant of TAPNext++ exclusively on Kubric-1024 and DynHumans, excluding PointOdyssey from training entirely. As shown in Tab. 5, this zero-shot variant still outperforms BootsTAPNext-B by 38.3 percentage points in survival rate and exceeds Track-On by 3.8 points, demonstrating that the long-sequence training recipe generalizes to out-of-domain videos beyond the PointOdyssey distribution.

Table 5. **Training without PointOdyssey.** Evaluation results on PointOdyssey (PO) with and without including PO in the training mix.

Method	δ^{avg} ↑	Survival ↑	MTE ↓
CoTracker3	44.5	56.3	20.7
Track-On	35.4	47.5	33.5
BootsTAPNext-B (Baseline)	9.9	13.0	92.1
Ours (w/ PO)	52.6	67.9	13.4
Ours (w/o PO)	37.3	51.3	26.4

Q: Does the roll augmentation give TAPNext++ an unfair advantage on AJ_{RD}?

We isolate the effect of roll augmentation in Tab. 3. While it benefits RoboTAP and Kinetics, it slightly reduces scores on PointOdyssey and DAVIS. Crucially, even *without* roll augmentation, our model remains state-of-the-art on DAVIS (AJ $=$ 65.3) and PointOdyssey (δ^{avg} $=$ 52.0, Survival$=$ 68.6, MTE$=$ 16.4, AJ_{RD} $=$ 22.3). The roll augmentation is a principled, task-specific training technique that any method can equally adopt; it is not a post-hoc advantage restricted to TAPNext++.

Q: Does TAPNext++ claim to solve infinite-length point tracking?

No. We explicitly target long but finite sequences and do not claim that performance degradation is fully eliminated at arbitrary sequence lengths. Our contribution is to demonstrate that such degradation is *not* an inherent architectural limitation of *linear* recurrent units—in contrast to standard non-linear recurrent architectures that suffer from vanishing gradients. By extending the effective tracking range from ∼150 frames to over 4000 frames without any architectural change, we provide new insights into the practical scalability of linear recurrent units for point tracking.

Q: Is TAPNext++ merely a "bag of tricks" with limited technical novelty?

We argue that system-algorithmic co-design is a fundamental research contribution, particularly in the context of scalable training. Concretely: (1) the distributed parallel scan for SSM blocks is not a standard engineering detail but the technical enabler that makes end-to-end training on 1024-frame sequences feasible on multiple GPUs—without it, naïve sequence parallelism would require $\mathcal{O}(N)$ sequential communication rounds instead of $\mathcal{O}(\log N)$; (2) the roll augmentation is a principled, task-specific design directly addressing the well-known failure mode of points exiting and re-entering the frame; (3) AJ_{RD} fills a concrete gap in the evaluation literature by directly measuring post-reappearance tracking quality, which no prior metric captures.

Q: How does Kubric-1024 differ from prior Kubric datasets?

Kubric-1024 required a fundamental redesign to sustain scene dynamics over 1024 frames. In prior Kubric variants, object motion dissipates quickly due to passive physics. We address this by introducing "velocity bumps"—random linear and angular impulses applied whenever an object's speed drops below a threshold, with a slight bias toward the scene origin to keep objects within the camera's field of view throughout the full sequence. We additionally apply sinusoidal noise to both the camera position and look-at point to simulate natural camera jitter, preventing the model from overfitting to the unnaturally smooth trajectories present in prior synthetic datasets.

Q: Why does the high-resolution (512×512) variant sometimes underperform the 256×256 model in Tab. 1?

To ensure a fair comparison, the high-resolution variant was fine-tuned using the same training recipe as the 256×256 model rather than a separately tuned configuration. The optimal hyperparameters for higher-resolution fine-tuning—including learning rate, schedule length, and batch composition—likely differ and were not optimized in this work. The results in Tab. 1 should therefore be treated as a conservative lower bound on the potential of the 512×512 model.

B. Long-Sequence Training with Sequence Parallelism

Training TAPNext on long sequences ($T = 1024$ frames or more) requires fitting large computation graphs and intermediate activations into memory, which exceeds single-device limits. To enable end-to-end training on such sequences, we adopt Sequence Parallelism (SP). The input sequence is partitioned into N chunks along the tempo-

ral dimension, with each chunk processed by a dedicated GPU. This strategy requires communicating $\mathcal{O}(T^2)$ bits for attention-based models via context parallelism. At the same time the linear recurrent nature of the State Space Models (SSMs) allows performing forward and backward propagation with only $\mathcal{O}(\log T)$ sequential and $\mathcal{O}(T \log T)$ total communication bits as we show below.

The Real-Gated Linear Recurrent Unit (RG-LRU) used in TAPNext, like other SSMs, is defined by a linear recurrence relation of the form:

$$h_t = a_t \odot h_{t-1} + x_t, \tag{1}$$

where $h_t, a_t, x_t \in \mathbb{R}^D$ are the hidden state, recurrence parameters, and input at time t, respectively, and $\odot$ denotes element-wise multiplication. This recurrence can be expressed as an associative binary operator. First, if $h_1 = a_1 \odot h_0 + x_1$ and $h_2 = a_2 \odot h_1 + x_2$, then $h_2 = (a_2 \odot a_1) \odot h_0 + (a_2 \odot x_1 + x_2)$. This allows us to define an associative binary operator $\oplus$:

$$(a_2, x_2) \oplus (a_1, x_1) = (a_2 \odot a_1, a_2 \odot x_1 + x_2). \tag{2}$$

Because this operator is associative, we can compute the recurrence using a parallel scan algorithm, rather than a sequential loop. We leverage this property to implement a *distributed parallel scan* for RG-LRU during the forward and backward passes, enabling efficient training on distributed sequence chunks. Our distributed scan operates in three phases:

1. **Local Scan:** Each GPU $j \in \{0, \ldots, N-1\}$, holding sequence chunk $\{(a_t^j, x_t^j)\}_{t=0}^k$, computes a local parallel scan assuming its initial hidden state h_{in}^j is zero. This produces a preliminary local output y'^j and a chunk summary $S_j = (\alpha_j, \chi_j)$, where $\alpha_j = a_k^j \odot \cdots \odot a_0^j$ and $\chi_j = y'^j_k$. This summary S_j represents the affine transformation of the chunk: $h_{\text{out}}^j = \alpha_j \odot h_{\text{in}}^j + \chi_j$. This phase is executed in parallel across all N GPUs.
2. **Cross-GPU Prefix Scan:** To find the correct input state h_{in}^j for each chunk $j > 0$, we must compute $h_{\text{in}}^j = h_{\text{out}}^{j-1}$. This requires composing the transformations of all preceding chunks: $h_{\text{in}}^j = (\alpha_{P_{j-1}} \odot h_{\text{start}}) + \chi_{P_{j-1}}$, where $P_{j-1} = S_{j-1} \oplus S_{j-2} \oplus \cdots \oplus S_0 = (\alpha_{P_{j-1}}, \chi_{P_{j-1}})$ and h_{start} is the initial state of the entire sequence (typically zero). Computing all prefix compositions $\{P_0, P_1, \ldots, P_{N-1}\}$ is a standard prefix scan problem over the set of summaries $\{S_0, S_1, \ldots, S_{N-1}\}$ using the operator $\oplus$. This is solved efficiently across N GPUs using tree-based algorithms (e.g., recursive doubling) with $O(\log N)$ communication rounds, rather than $O(N)$ rounds required by sequential propagation.
3. **Local Correction:** Once each GPU j receives its correct initial state h_{in}^j from Phase 2, it corrects its preliminary output y'^j by adding the contribution from h_{in}^j: $y_t^j = y_t'^j + c_t^j \odot h_{\text{in}}^j$, where $c_t^j = a_t^j \odot a_{t-1}^j \odot \cdots \odot a_0^j$. This phase is also performed in parallel across all GPUs.

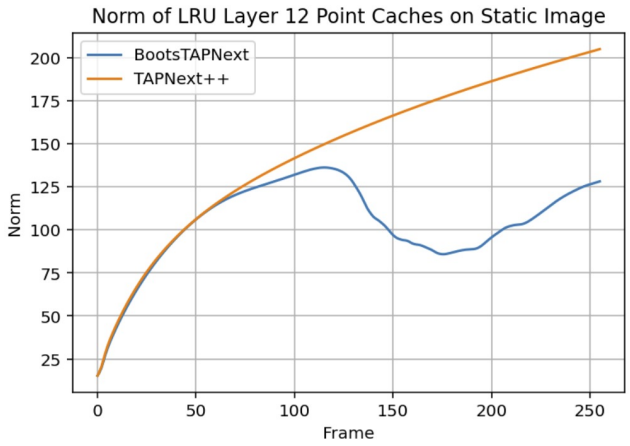

Figure 8. **LRU State Norm Growth on re-feeding a Single Image.** BootsTAPNext shows unstable LRU state after 100 frames. TAPNext++ has a monotonically growing state.

The backward pass follows a similar distributed scan pattern to propagate gradients efficiently across GPUs.

In addition to that, we also implement sequence parallelism for temporal causal convolution layers. Unlike the linear recurrence, each step of the sequence may only depend on a constant number of past steps in forward pass for temporal convolution (or future steps during backward pass). Therefore, we only need to do steps 1 and 2 and for the latter we can do one linear chain of communications in parallel.

This method allows TAPNext to be trained end-to-end on sequences significantly longer than those trainable on a single device, with only a logarithmic increase in communication overhead with respect to the number of GPUs.

C. Dynamic Model Depth

Prior work[30] has shown that TAPNext-like architectures can be trained using a layer-wise supervision strategy, meaning that each layer is trained using its own loss. As a result, the model supports dynamic inference depth: it can be executed with fewer layers at test time without requiring retraining. Although the original architecture uses twelve layers, we observed that the predictions produced after only eight layers are already comparable in quality to those obtained after all twelve. This allows one to reduce memory consumption and further increase the achievable inference speed (FPS). Note that in this work, all experiments were executed using the full twelve layers.

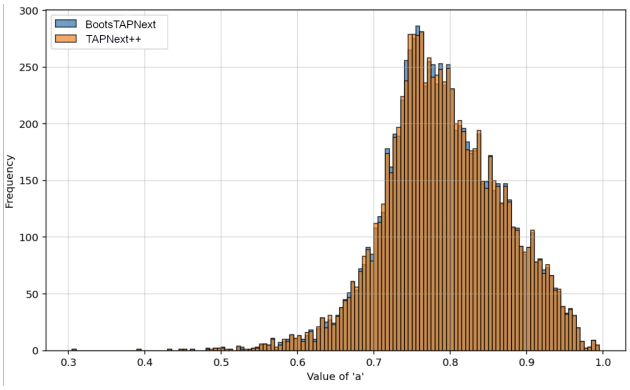

Figure 9. **Eigenvalue Distribution.** Distribution of eigenvalues a. Almost no change can be observed between BootsTAPNext and TAPNext++.

D. LRU State analysis

We analyze the temporal behavior of the LRU state associated with the point-token stream in the twelfth layer. To isolate the intrinsic LRU dynamics from changes caused by varying inputs, we repeatedly feed the same static image into the network and track the norm of the resulting cache state over time. As shown in Fig. 8, the original BootsTAPNext exhibits irregular and unstable cache growth after approximately 100 frames, whereas TAPNext++ produces a smooth and nearly monotonic evolution of the cache norm. Also note that both network show equal state growth in the beginning of the video. Importantly, even with a constant input image, changes in the LRU state are expected: the LRU update rule operates directly on the current activation and does not explicitly enforce invariance over repeated inputs.

To better understand this behavior, we further inspect the distribution of the learned eigenvalues a that parametrize the LRU update (cf. Fig. 9). Surprisingly, we do not observe a pronounced shift toward eigenvalues near one in TAPNext++, which would correspond to stronger long-term memory retention. This suggests that the improved stability of TAPNext++ does not arise from extending the effective memory horizon of the LRU block. Instead, the model may have learned to regulate or compensate for long-sequence LRU dynamics elsewhere in the architecture—potentially through more stable interactions between point- and patch-token pathways or through modifications in downstream attention layers.