Supplementary material - TAPNext: Tracking Any Point as Next Token Prediction

A. Frequently Asked Questions

- Is there a reason for categorization of frame/window/video latency? Can't we just run any model e.g. with frame latency (at timestep t we feed frames $1, 2, \cdots, t$ and obtain track prediction right after the frame t was "fed")
  - This indeed may even turn offline trackers into online ones. However, we would argue that this effectively won't solve the latency problem. The reason is that when such a tracker receives the new frame, it needs to reprocess either all previous frames (for offline trackers) or a window of several recent frames (for window-based trackers). In either case, the time before the prediction is significant effectively disallowing real-time tracking at a high frame rate (see Table 2 for quantification).
- Why does TAPNext require so much more data and time to train?
  - TAPNext is a much more generic model, performing only attention and SSM scanning without any custom components. The generality of TAPNext comes at the cost of much higher compute needed for training (i.e. a large batch and longer optimization).
- TAPNext uses more synthetic and real data to train than previous methods. What is the performance of previous methods given such big datasets remains unclear.
  - Currently there is no standartization of which dataset to use, among previous methods. For exampe, TAPIR uses 100000 videos with camera panning enabled but no motion blur. LocoTrack uses 11000 synthetic videos for training with panning but no motion blur. TAPTR uses also uses 11000 training videos which inlude motion blur but don't include camera panning. All aforementioned methods use 24 frames videos while CoTracker3 trains on synthetic videos of 64 frames and its training dataset includes only 6000 training videos at the resolution of 512×512 (compared to 256×256 for previous methods). This way all previous methods use training datasets of different size (which is varied by two orders of magnitude between CoTracker3 and TAPIR), length, resolution and visual properties (motion blur, camera panning).
- How does TAPNext learn to recover from occlusions?
  - For each point query TAPNext has the corresponding sequence of SSM recurrent hidden states (since SSM is a form of RNN) that contain the information about the tracked point. Even when the visaul occlusion happens, this information is still being processed by recurrent SSM and the corresponding output predicts the coordinate and occlusion flag.
- Why claiming that the method generalizes to 5x longer videos? This seems to be a weaker statement than prior methods can do given that they track potentially infinite videos.
  - TAPNext marks the new stage of point tracking methods with no (tracking) inductive biases that are entirely data driven. On one hand this gives scalability and SOTA tracking quality that comes from the scale of the model and data. On the other hand, due to that it is harder to satisfy guarantees like robustness to long videos. We hope that future research will address this problem of long term tracking of end-to-end trained models.
- Why not putting the TRecViT into the related work?
  - While TRecViT is a strong visual backbone, TAPNext could use other video backbone. Our selection of TRecViT was dictated by its efficiency - not only it requires significantly less time and memory to train (than e.g. a purely attention counterpart), it also processes videos online and needs only one previous recurrent state to perform inference. Due to this reason and since TAPNext's idea is connected to how TRecViT processes tokens, we included the description of the latter.

B. Ablations

Please find ablations in Tables 4 and 5.

C. Scalability of Baselines

To provide evidence for scalability of inductive-bias free architectures, we perform the scalability analysis of TAPIR, which is one of the most popular point trackers. Table 3 shows how the Average Jaccard (AJ) evaluation metric changes as we increase the size and video length of the training data. As the result suggests, TAPIR is not able to benefit from a larger scale data.

TAPIR	Kubric Train	Kubric Val	DAVIS Strided	RoboTAP First	DAVIS First	Kinetics First
Default Kubric 10K / 24f	81.7	81.8	59.5	58.9	54.4	52.1
Panning Kubric 500K / 48f	76.5	76.5	58.0	56.5	52.7	51.0

Table 3. Average Jaccard (%) comparison between TAPIR models trained on different dataset scales and frame lengths.

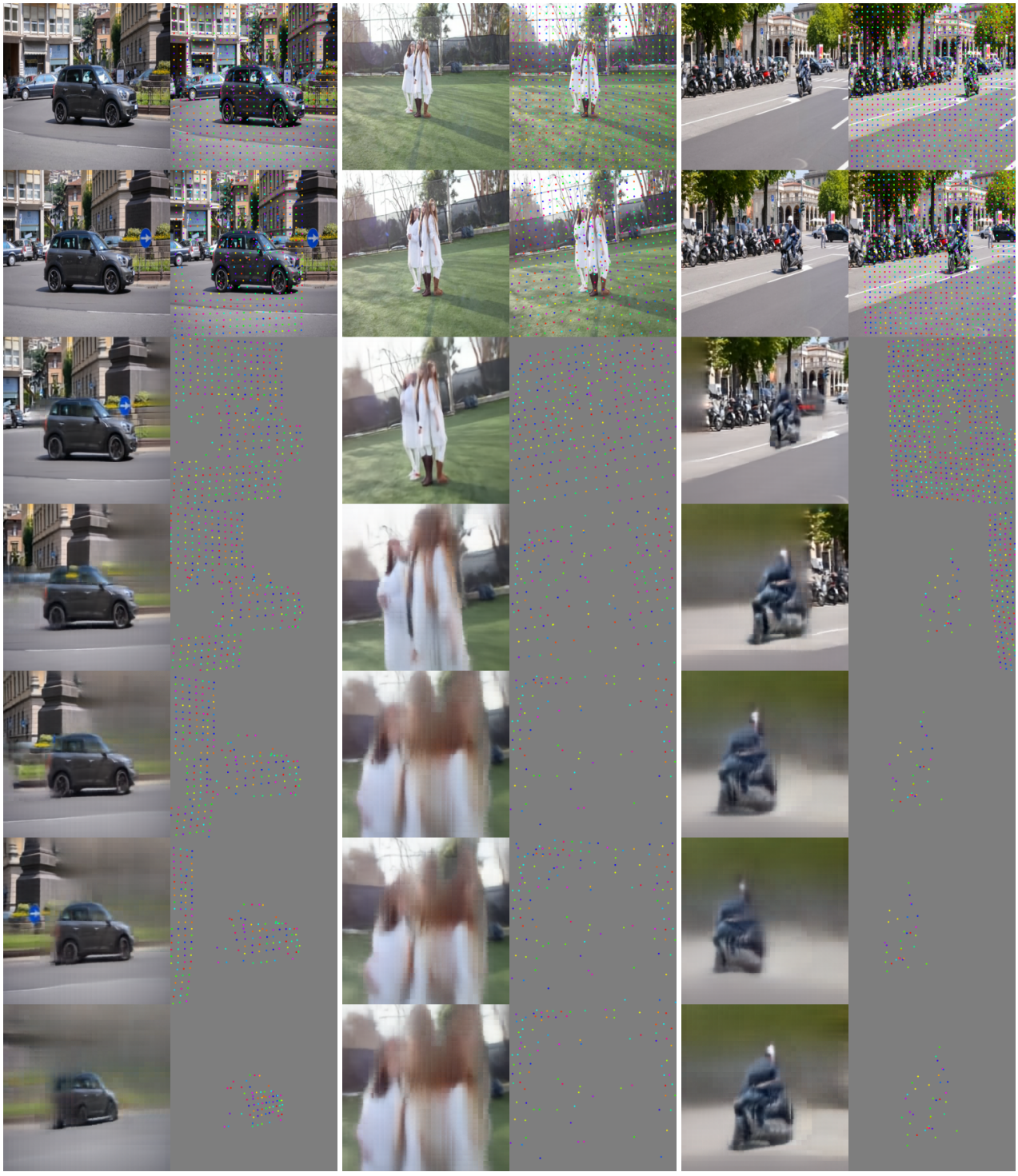


Figure 5. Video Completion by TAPNext variant. **Left:** Outputs of patch-level linear pixel heads. **Right:** Inputs to the model (Visible or masked image and points).

Type of Ablation: default value → ablated value			Average Jaccard
TAPNext-S default (small scale run)			55.0
Classification coordinate head	→	Regression coordinate head	44.7
2× state expansion [10] in SSM	→	No SSM state expansion	53.8
Losses after each ViT Block	→	No intermediate losses	50.5
Regression + Classification coordinate loss	→	Classification coordinate loss	52.7
Regression + Classification coordinate loss	→	Regression coordinate loss	48.1
Image patch 8×8	→	Image patch 16×16	49.7

Table 4. Ablating of the main components of TAPNext-S. We train each model for $150,000$ steps and batch size 128 and on 24 frames (compared to $300,000$ steps, batch 256, and 48 frames for the main TAPNext models in Table 1). We evaluate every ablation on DAVIS query-first. All ablations are independent of each other.

TAPNext Variants	Average Jaccard 24 Frames	Average Jaccard Full length
Temporal Attention	68.4	17.3
Temporal SSM	70.0	55.0

Table 5. Temporal attention ablation of TAPNext-S (under the same protocol as Table 4). We change the temporal SSM block to the temporal attention block [48] that uses rotary positional embeddings [45]. SSM blocks enable much better temporal generalization in TAPNext beyond the 24 frame training sequences.

D. Attention visualization

For convenience, larger versions of Figures 3 and 4 are reproduced in the appendix as Figures 6 and 7. We also show the raw spatial attention maps of the attention heads from the same layer in Figure 8. Full videos of attention visualization can be found in the supplementary files. Notably in the point-to-point attention video there is strong connection between clusters on different objects, but as the video progresses and the objects (colored in red and blue) move independently these connections quickly disappear. This hints at the model's emergent motion segmentation ability.

E. Additional Evaluations of TAPNext

To further assess the temporal extrapolation capabilities of TAPNext, we evaluate the model on RoboTAP [53], which we group by video length. Table 6 show the result of these evaluations. The result suggests that TAPNext is capable of reliably tracking videos of length up to around 300 frames, aligning with our previous claim on the $5\times$ extrapolation capabilities of TAPNext.

Table 7 shows evaluation of TAPNext on two extra datasets – RoboTAP and RGB-Stacking. The non-bootstrapped version of the model shows SOTA performance of RGB-Stacking. The bootstrapped version shows the second best performance after CoTracker3.

#Frames	**<100**	**100 -200**	**200 -300**	**300 -500**	**>500**
BootsTAPIR	79.7	67.6	61.0	**50.0**	**53.2**
TAPNext	82.7	70.3	59.5	38.4	18.4
BootsTAPNext	**85.3**	**74.2**	**64.3**	46.3	21.6

Table 6. Average Jaccard (%) on RoboTAP under varying `Max #Frames` bins.

Model	**RoboTAP First**	**RGB-S First**
CoTracker3	**66.4**	71.7
TAPNext	60.1	**77.2**
BootsTAPNext	64.6	66.2

Table 7. Average Jaccard (%) on RoboTAP and RGB-Stacking.

F. Joint-Tracking and Support Points

	AJ	PTS (δ^{avg})	OA
Individual Points + 4x4 global + 9x9 local grid	62.2	76.1	90.9
Joint Points	62.4	76.6	90.5

Table 8. Comparison of joint query point tracking v.s. individual queries and support points on DAVIS first evaluation of TAPNext-B

Because TAPNext processes query points jointly there is a concern that semantic correlation between query points can give it an unfair advantage compared to methods that process queries individually. Therefore we use the methodology from Cotracker [26, 27] to evaluate tracking of one point at a time with additional query points sampled from local and global regular grids. Similar to Cotracker we found that TAPNext benefits from mainly from a local support grid of points (Figure 9). We found no major difference in performance between the one point evaluation (with the best support point scheme) compared to evaluating on all query points jointly, see Table 8.

G. Coordinate Prediction via Classification

The coordinate prediction head of TAPNext predicts $(x, y) \in [0, H] \times [0, W]$ coordinates. The x and y coordinates are

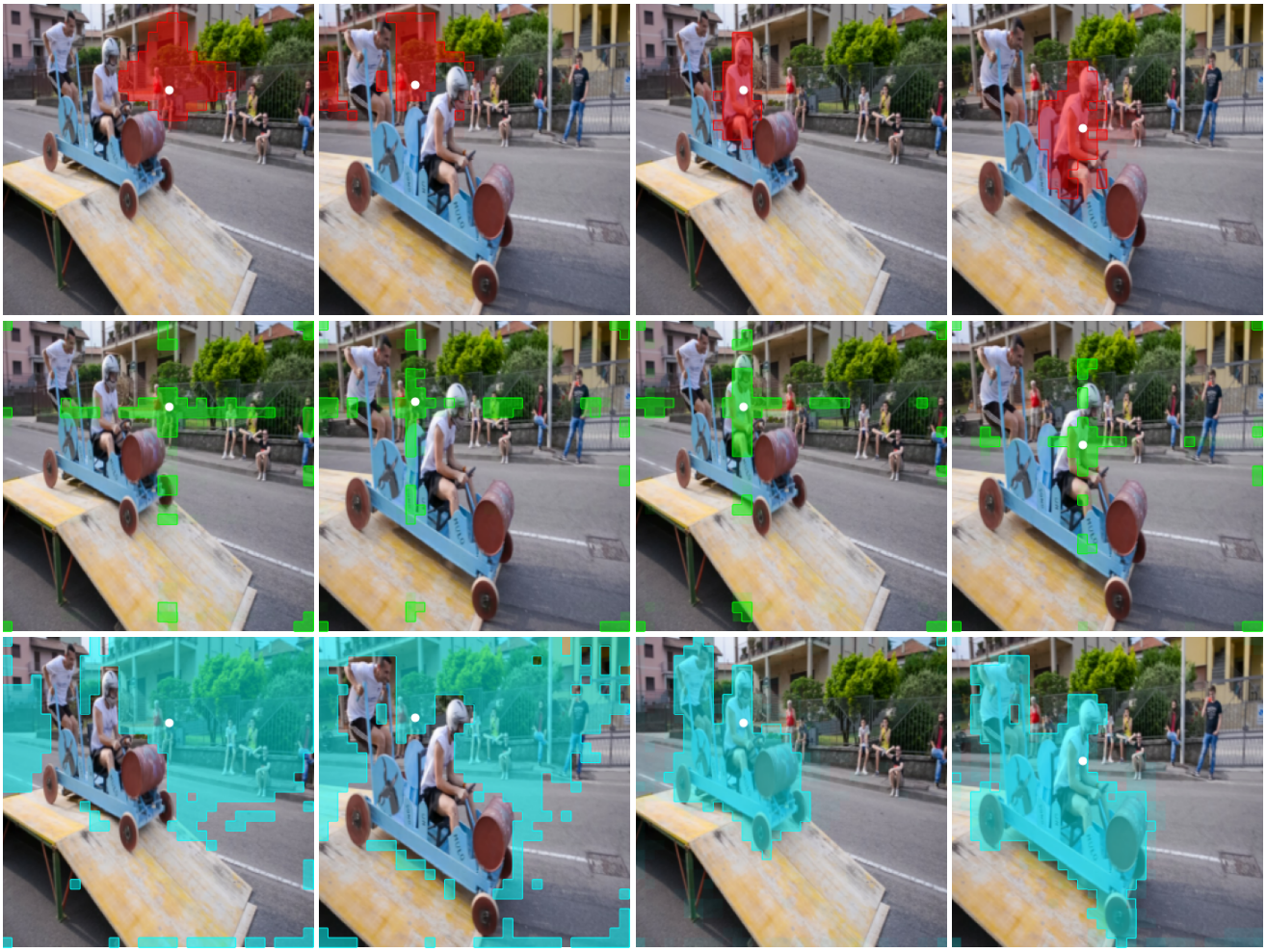


Figure 6. Three attention patterns learned by TAPNext. We visualize attention maps where the attention queries are the point track tokens and the keys are image tokens, which correspond to 8×8 patches. Each row is a certain (layer, head) pair. We observe patterns: **(top)** Cost-volume-like attention head; **(middle)** Coordinate-based readout head; **(bottom)** motion-cluster-based readout head. Note that these are just intermediate heads in the backbone.

Figure 7. Point-to-point attention map visualizations. Tracked points are nodes and (scaled) attention weights are edges, the thicker the edge the higher the weight between points. Two frames from a video are used to visualize two attention layers. Note that in all images we see strong attention between points on objects that are moving together.

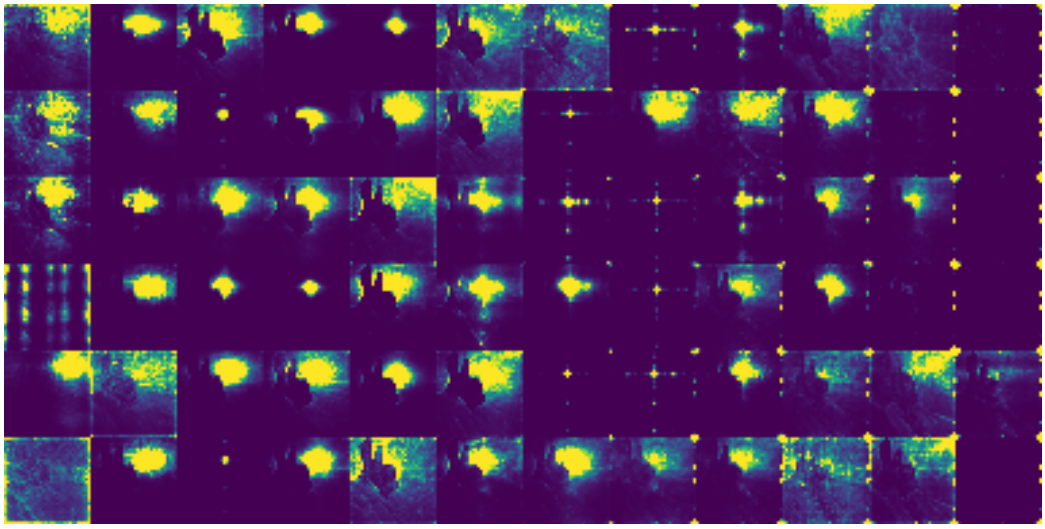


Figure 8. **X-axis:** layers; **Y-axis:** attention heads. We visualize attention maps for the TAPNext-S model which has 12 layers and 6 attention heads. We visualize the same video as in Figure 6, specifically the timestep corresponding to the leftmost column. Also, we use the same qeury point as in Figure 6. The attention maps visualize the attention where query is the track tokens and key is image patch, we visualize it for every head in every layer. The patterns found in Figure 6 moslty are repeated accross the model.

discretized into $n = 256$ values, corresponding to pixel locations, and the head outputs the discrete distribution (p_i where $0 < i < n$, $p \in \mathbb{R}^n$) for both coordinates independently. In particular, each point token is passed to the MLP where the last layer is softmax. To obtain the final, sub-pixel, coordinate we use the truncated soft $argmax$ operation (the same as TAPNet [11]). First we truncate the probability distribution around argmax: we set all probability bins that are more than Δ steps away from argmax to zero ($\Delta = 20$ in our experiments). After that we renormalize the probability distribution since after setting some probability bins to zero, the result is no longer a probability distribution. After we compute the new truncated probability distribution, we simply compute the expected probability bin:

$$\hat{x} = \frac{H}{n} \sum_{j=1}^{n} \left(j \cdot \frac{p_j \cdot \mathbb{1}[|j - \arg\max p| \leqslant \Delta]}{\sum_{k=1}^{n} p_k \cdot \mathbb{1}[|k - \arg\max p| \leqslant \Delta]} \right)$$

The above equation defines the continuous coordinate prediction $\hat{x}$ for the x coordinate (valued in the range $[0, H]$). We use the same formula to compute the prediction for the y coordinate denoted $\hat{y}$.

This parameterization can be very easily implemented in Jax:

```
import jax.numpy as jnp
def trunc_softargmax(p, delta=20):
  n = p.shape[0] # p is vector
  js = jnp.arange(n)
  j = jnp.argmax(p)
  m = jnp.abs(js - j) <= delta
  p *= m
  p /= p.sum()
  return (p * js).sum()
```

As we mention earlier, we use two loss functions for coordinate prediction: Huber loss on the continuous prediction ($\hat{x}$ and $\hat{y}$) and softmax cross entropy for discrete prediction (p_x and p_y):

$$\begin{aligned} L(x, y, p_x, p_y) = w_1 L_H(x, \hat{x}) + w_2 L_H(y, \hat{y}) \\ + w_3 L_C(\text{one\_hot}(x), p_x) \\ + w_4 L_C(\text{one\_hot}(y), p_y) \end{aligned}$$

L_H is the huber loss, L_C is the softmax with crossentropy loss. Coefficients w_1, w_2, w_3, w_4 are weights applied to each loss component. In our experiments, we use $w_1 = w_2 = 0.1$ and $w_3 = w_4 = 1.0$. This combined loss is applied to every layer. This means that the $T \times Q$ point track tokens after every ViT block are fed to the coordinate heads (shared MLPs with softmax output) and the the aforementioned loss is applied to the outputs of the coordinate heads. Similarly, the visibility head which is also an MLP with the sigmoid as the final activation. The loss for this head is binary cross entropy. Like the coordinate head, the visibility head is applied to every layer.

Note that, despite using a parameterization that produces bounded ranges for coordinate values ($(x, y) \in [0, H] \times [0, W]$), it is still possible to predict (of course, not with the same trained model) the coodinates out of the view, similarly to what other models (e.g. CoTracker [26]) do. For that, we could simply map e.g. the x coordinate to some range $[-d, H + d]$ instead of $[0, H]$ for some positive d. This way, some probability bins would be responsible for out-of-view prediction while some other will do in-view prediction. Since out-of-view prediction is not a part of the TAP-Vid benchmark, we do not use it.

H. TAPNext Hyperparameters

We sample batches containing query points of shape $[B, Q, 3]$, where $B = 256$ is the batch size, $Q = 256$ number of query points per video. The last dimension represents the t, x, y coordinate of each query point. Importantly, we train our model on a mixture that simulates query point being in the beginning of the video ($t = 0$) and at the intermediate timestep ($t > 0$). The weights of this mixture are $[0.8, 0.2]$, respectively. The $t = 0$ mixture component contains query points queried at the 0^{th} frame in the video. The $t > 0$ mixture component contains the ones queried at any timestep, not necessarily starting at the 0^{th} frame. Since our model is causal, it cannot track points before point query is given. Therefore, for the second component we mask the coordinate losses for frames preceding the known query and set the visibility label to zero. We use a weight of 1.0 for both visibility and coordinate classification losses, and 0.1 for coordinate regression. We use the AdamW optimizer with weight decay of 0.01 for $300,000$ steps using a cosine learning rate schedule with 2500 warm-up steps, peak and final learning rate values of 0.001 (for -S model) and 0, respectively. We clip gradient norm to 1.0. We found it important to use 8×8 patches, smaller than the 16×16 used in ViT for classification, aligning with the intuition that smaller patches work well for spatially fine-grained tasks.

	Name	TAPNext-S	TAPNext-B
	layers	12	
	parameters	56M	194M
ViT Block	attention heads	12	12
	width	384	768
	dropout	0.0	0.0
SSM Block	width	384	768
	LRU width	768	768
	heads	12	12

Table 9. TAPNext hyperparameters specific to each size of the model.

The full list of hyperparameters is available in Tables 9 and 10. We implement the model in Jax and use TPUv6e for training. Specifically, we use 16×16 TPU slice to train both TAPNext-S and TAPNext-B. Since we are using a large batch ($B \times T = 256 \times 48 = 12288$ images in our case), we use activation checkpointing (implemented in the same way as in the original ViT[4]). This setup is roughly equivalent to 50 H100 GPUs in both compute and memory and our training takes 4 days for TAPNext-S and 5 days for TAPNext-B. Note that despite a large compute and memory requirement for training, TAPNext inference runs quickly on a single GPU (see Table 2).

I. Strided Evaluation with TAPNext as Causal Tracker

Recall that in training when TAPNext is trained on queries at $t > 0$ (i.e. after the 0^{th} frame), the coordinate losses corresponding to frames preceding the query is masked and visible target is set to 0.0. For complete and comparable evaluation, however, we require track predictions for every frame in the sequence. To obtain these predictions with our causal, per-frame tracker, we run the tracker both forwards and backwards in time starting at the frame corresponding to the query. These two sequences of predictions corresponding to normal and reverse time processing are then concatenated together to obtain the full sequence of prediction. For strided evaluation this process is repeated for every query point in the sequence.

J. Generation Pipeline for a Large Scale Synthetic Dataset

The original MOVi-F is similar to MOVi-E but includes random motion blur and is rendered at 512×512 resolution, with a 256×256 downscaled option. To enhance model performance on real-world videos with panning, we modify the MOVi-E dataset to adjust the camera's "look at" point to follow a random linear trajectory by sampling a start point a within a medium-sized sphere (4 unit radius), a travel-through point b near the center of the workspace (1 unit radius), and an end point c extrapolated along the line from a to b by up to 50% of their distance. The "look at" path randomly switches between $a \to c$ and $c \to a$. This modification initially resulted in a 100K video Panning Kubric MOVi-E dataset. To further exploit the scalability of TAPNext and improve its generalization capability to longer real world videos, we developed a new Panning Kubric MOVi-F data generation pipeline, combining data generation pipelines from both Kubric [18] and TAPIR [12]. Building on this, we created a new large-scale Panning Kubric MOVi-F dataset with 500,000 videos. The rendered videos combine both panning effect and motion blur. We then increase each video

[4] https://github.com/google-research/big_vision/blob/main/big_vision/models/vit.py

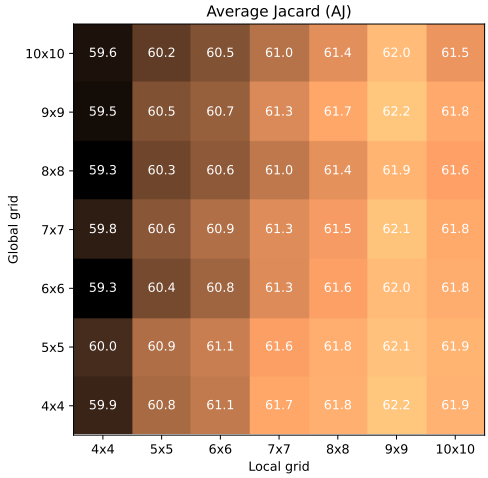
Name	Value
Optimization	
Optimizer	AdamW
Global batch size	256
Number of queries per video	256
Max gradient norm	1.0
Weight decay	0.01
Number of optimization steps	$300,000$
Warmup	linear
Number of warmup steps	2500
LR before warmup	0
LR schedule	cosine
Peak LR (TAPNext-S)	0.001
Peak LR (TAPNext-B)	0.0005
Final LR	0
Precision	`float32`
Regression loss weight(s)	0.1
Classification loss weight(s)	1.0
Data	
Dataset size (videos)	500.000
Dataset resolution	256×256
Number of frames per video	48
Camera panning	Enabled
Motion blur	Enabled
Prob. of sampling query with $t = 0$	0.8
Prob. of sampling query with $t > 0$	0.2
Model	
Patch size	8×8
Image position embedding	learned
Point position embedding	sincos2d
Point position embedding resolution	256×256
MLP head number of layers	3
MLP head hidden size	256
MLP head activation	GELU
Softargmax threshold (Δ)	20
Coord. softmax temperature	2
Huber loss weight	0.1
Coordinate CE loss weight	1.0
Visibility CE loss weight	1.0

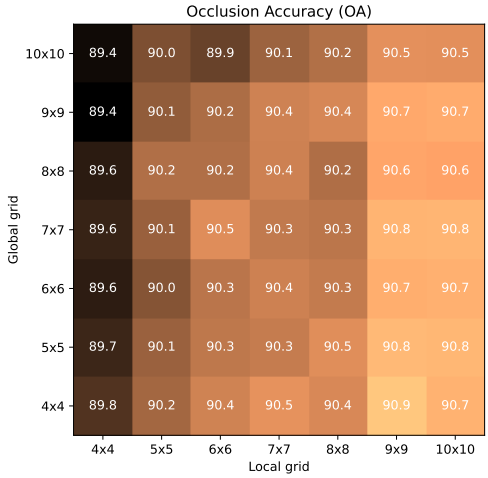
Table 10. TAPNext hyperparameters

duration from the default 24 frames to 48 frames. These enhancements allow more robust training and long term inference stability for TAPNext, addressing the challenges posed by real-world long term point tracking. Figure 10 shows the comparison between the new dataset and existing ones.

K. Prediction Visualization on DAVIS

We present additional examples of point track predictions from TAPNext on the TAPVid-DAVIS dataset in Figure 11. All DAVIS track videos can be found in the supplementary files. The query points are all initialized in the first frame, and TAPNext performs tracking in a causal manner. While the model occasionally makes errors over longer time spans due to the inherent challenges of maintaining long-term precision in causal tracking, TAPNext demonstrates robust performance by reliably tracks points across a wide range of scenarios, including foreground and background elements, as well as small and large motions.





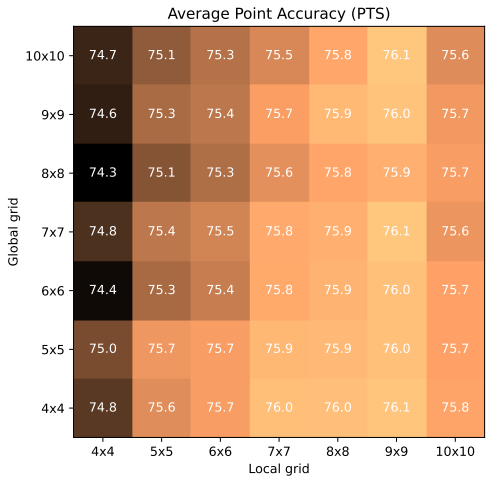
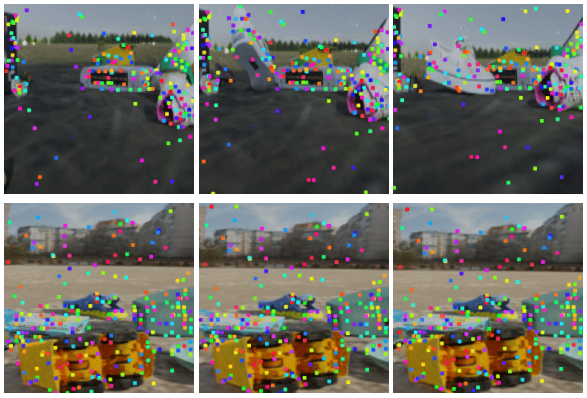
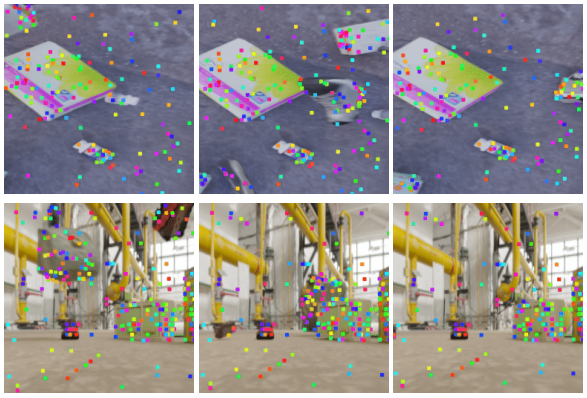


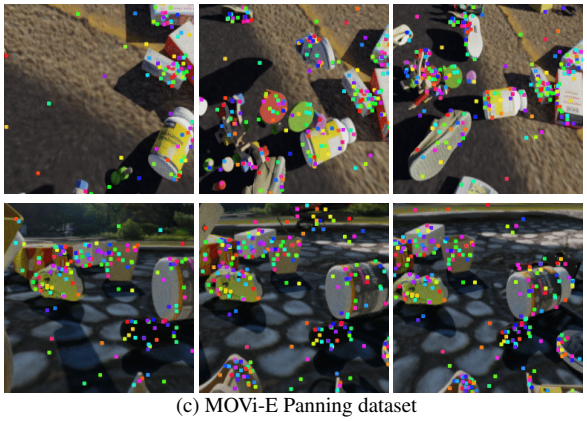
Figure 9. One point at a time tracking performance with various support points grid configurations.



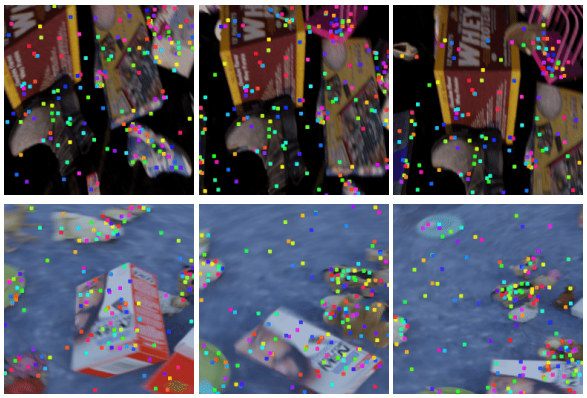
(a) Kubric MOVi-E dataset



(b) Kubric MOVi-F dataset



(c) MOVi-E Panning dataset



(d) MOVi-F Panning dataset (Ours)

Figure 10. **Kubric dataset comparison** – We show two exampler videos for each dataset with first frame, middle frame and last frame, with groundtruth point tracks.

Figure 11. **Prediction on TAPVid-DAVIS** – We show the tail visualization of semi-dense point track predictions for 5 example videos on DAVIS dataset. For each video, we show the first query frame, 20th frame and 40th frame.