TRACK-ON: TRANSFORMER-BASED ONLINE POINT TRACKING WITH MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we consider the problem of long-term point tracking, which requires consistent identification of points across multiple frames in a video, despite changes in appearance, lighting, perspective, and occlusions. We target online tracking on a frame-by-frame basis, making it suitable for real-world, streaming scenarios. Specifically, we introduce **Track-On**, a simple transformer-based model designed for online long-term point tracking. Unlike prior methods that depend on full temporal modeling, our model processes video frames sequentially without access to future frames, leveraging two memory modules —spatial memory and context memory— to capture temporal information and maintain reliable point tracking over long time horizons. At inference time, it employs patch classification and refinement to identify correspondences and track points with high accuracy. Through extensive experiments, we demonstrate that **Track-On** sets a new state-of-the-art for online models and delivers superior or competitive results compared to offline approaches on TAP-Vid benchmark. Our method offers a robust and scalable solution for real-time tracking in diverse applications.

1 INTRODUCTION

Motion estimation is one of the core challenges in computer vision, with applications spanning video compression (Jasinschi et al., 1998), video stabilization (Battiato et al., 2007; Lee et al., 2009), and augmented reality (Marchand et al., 2015). The objective is to track physical points across video frames accurately. A widely used solution for motion estimation is optical flow, which estimates pixel-level correspondences between adjacent frames. In principle, long-term motion estimation can be achieved by chaining together these frame-by-frame estimations.

Recent advances in optical flow techniques, such as PWC-Net (Sun et al., 2018) and RAFT (Teed & Deng, 2020), have improved accuracy for short-term motion estimation. However, the inherent limitations of chaining flow estimations remain a challenge, namely error accumulation and the difficulty of handling occlusions. To address long-term motion estimation, Sand & Teller (2008) explicitly introduced the concept of pixel tracking, a paradigm shift that focuses on tracking individual points across a video, rather than relying solely on pairwise frame correspondences. This concept, often referred to as "particle video" has been revisited in recent deep learning methods like PIPs (Harley et al., 2022) and TAPIR (Doersch et al., 2023), which leverage dense cost volumes, iterative optimization, and learned appearance updates to track points through time.

Despite the advancements, the existing methods for long-term point tracking face two major limitations. First, they primarily rely on offline processing, where the entire video or a large window of frames is processed at once. This allows models to use both past and future frames to improve predictions but inherently limits their applicability in real-world scenarios (Doersch et al., 2024; 2023; Karaev et al., 2024; Harley et al., 2022; Li et al., 2024). Second, these approaches struggle with scalability, as they often require full attention computation across all frames, leading to significant memory overhead, especially for long videos or large frame windows. These limitations hinder their use in real-world applications, like robotics or augmented reality, where efficient and online processing of streaming video is crucial.

In this paper, we address the challenge of long-term point tracking in an online processing setting (Fig. 1, right), where the model processes video frames sequentially, without access to future frames (Vecerik et al., 2023). We propose a simple transformer-based model, where points of interest are

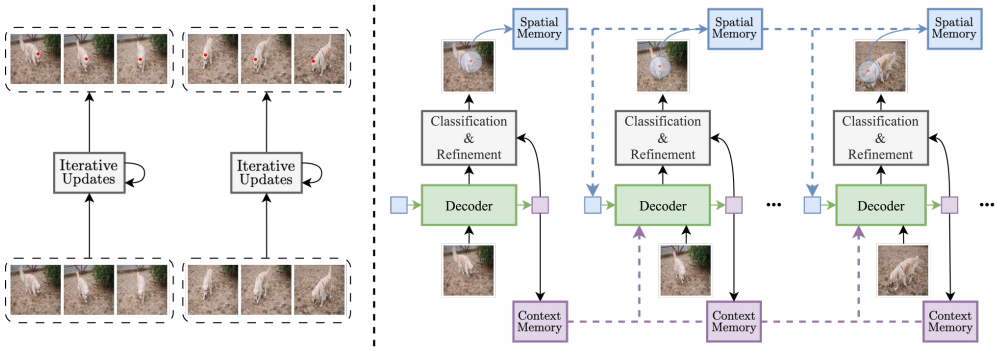


Figure 1: **Offline vs. Online Point Tracking.** We propose an online model, tracking points frame-by-frame (**right**), unlike the dominant offline paradigm where models require access to all frames within a sliding window or the entire video (**left**). In contrast, our approach allows for frame-by-frame tracking in videos of any length. To capture temporal information, we introduce two memory modules: spatial memory, which tracks changes in the target point, and context memory, which stores broader contextual information from previous states of the point.

treated as queries in the transformer decoder, attending the current frame to update their features. Unlike existing methods that aggregate temporal information across video frames, we achieve temporal continuity by updating the query representations with information from two specialized memory modules: spatial memory and context memory. This design enables the model to maintain reliable point tracking over time while avoiding the high computational and memory costs associated with full temporal modeling across entire video sequences.

Specifically, spatial and context memory play distinct but complementary roles. The former aims to reduce tracking drift by updating the query representation with information from the latest frames. This ensures that the query reflects the most recent visual appearance of the tracked point, by storing the content around the model's predictions in previous frames, rather than relying on features of the initial point. On the other hand, context memory provides a broader view of the track's history, storing the point's embeddings from past frames. This allows the model to consider changes to visual content including key information about the point's status, such as whether the point was occluded in previous frames. Overall, spatial memory focuses on positional changes in predictions over time while context memory ensures temporal continuity by providing a full perspective of the track's evolution. Together, these two memory modules aggregate useful temporal information across video.

At training time, the queries from the transformer decoder identify the most likely location by computing embedding similarity with each patch, and are trained using similarity-based classification, akin to contrastive learning. The prediction is then refined by estimating an offset within the local region using a deformable attention block to find the final correspondence. We conduct extensive experiments demonstrating that our simple patch-classification and refinement approach serves as a strong alternative to the dominant iterative update paradigm (Karaev et al., 2024; Harley et al., 2022; Doersch et al., 2023). Our method sets a new state-of-the-art among online models and either matches or surpasses offline approaches on datasets in the TAP-Vid benchmark (Doersch et al., 2022).

In summary, our contributions are as follows: (i) A simple architecture that treats points of interest as queries in the transformer decoder, identifying correspondences through patch classification and refinement; (ii) Memory modules that effectively store past content and address feature drift in an online manner; (iii) Extensive experiments and ablations demonstrating state-of-the-art performance among online models and competitive results compared to offline models on the TAP-Vid benchmark.

2 Methodology

2.1 Problem Scenario

Given an RGB video of T frames, $\mathcal{V} = \{\mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_T\} \in \mathbb{R}^{T\times H\times W\times 3}$, and a set of N predefined queries, $\mathcal{Q} = \{(t^1, \mathbf{p}^1), (t^2, \mathbf{p}^2), \ldots, (t^N, \mathbf{p}^N)\} \in \mathbb{R}^{N\times 3}$, where each query point is specified

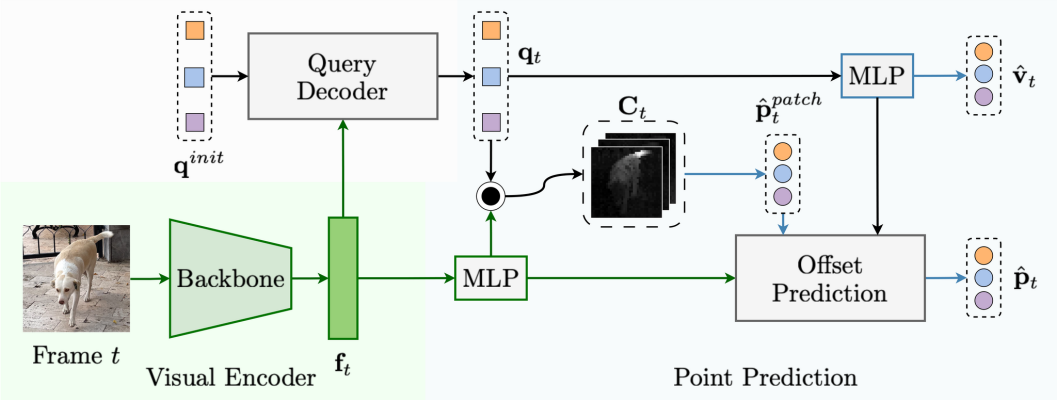


Figure 2: **Overview.** We introduce Track-On, a simple transformer-based method for online, frame-by-frame point tracking. The process involves three steps: (i) **Visual Encoder**, which extracts features from the given frame; (ii) **Query Decoder**, which decodes interest point queries using the frame's features; (iii) **Point Prediction**, where correspondences are estimated in a coarse-to-fine manner, first through patch classification based on similarity, followed by refinement through offset prediction from the patch center. Note that the squares refer to point queries, while the circles represent predictions, either as point coordinates or visibility.

by the start time and pixel's spatial location, our goal is to predict the correspondences $\hat{\mathbf{p}}_t \in \mathbb{R}^{N\times 2}$ and visibility $\hat{\mathbf{v}}_t \in \{0,1\}^N$ for all query points in an online manner, *i.e.* using only frames up to the current target frame t. To address this problem, we propose a transformer-based point tracking model, that tracks points **frame-by-frame**, with dynamic memories $\mathbf{M}$ to propagate temporal information along the video sequence:

$$\hat{\mathbf{p}}_t,\ \hat{\mathbf{v}}_t,\ \mathbf{M}_t = \Phi\left(\mathbf{I}_t,\ \mathcal{Q},\ \mathbf{M}_{t-1};\ \Theta\right) \tag{1}$$

In the following sections, we start by describing the basic transformer architecture for point tracking in Sec. 2.2, then introduce the two memory modules and their update mechanisms in Sec. 2.3.

2.2 Track-On: Point Tracking with a Transformer

Our model is based on transformer, consisting of three components, as illustrated in Fig. 2: Visual Encoder is tasked to extract visual features of the video frame, and initialize the query points; Query Decoder enables the queried points to attend the target frame to update their features; and Point Prediction, to predict the positions of corresponding queried points in a coarse-to-fine (Doersch et al., 2023) manner.

2.2.1 Visual Encoder

We adopt a Vision Transformer (ViT) as visual backbone, specifically, DINOv2 (Oquab et al., 2023), and use ViT-Adapter (Chen et al., 2022b) to upsample the feature. The ViT-Adapter allows us to obtain dense features at a higher resolution than the standard ViT. We then add learnable spatial positional embeddings γ^s to the frame-wise features:

$$\mathbf{f}_t = \Phi_{\text{vis-enc}}\left(\mathbf{I}_t\right) + \gamma^s \in \mathbb{R}^{\frac{H}{S}\times\frac{W}{S}\times D} \tag{2}$$

where D denotes the feature dimension, and S refers to the stride. We use a single-scale feature map from ViT-Adapter for memory efficiency, specifically with a stride of $S = 4$.

Query Initialization: To initialize the query features ($\mathbf{q}^{init}$), we apply bilinear sampling to the feature map at the query location $\left(\mathbf{p}^i\right)$:

$$\mathbf{q}^{init} = \left\{\text{sample}(\mathbf{f}_{t^i},\ \mathbf{p}^i)\right\}_{i=1}^{N} \in \mathbb{R}^{N\times D}$$

In practice, we initialize the query based on the features of the start frame t^i for i-th query and propagate them to the subsequent frames.

2.2.2 QUERY DECODER

After extracting the visual features for the frame and query points, we adopt a variant of transformer decoder (Vaswani et al., 2017), with 3 layers, *i.e.* cross-attention followed by a self-attention, with an additional feed forward network between attentions:

$$\mathbf{q}_t = \Phi_{\text{q-dec}}\left(\mathbf{q}^{init},\ \mathbf{f}_t\right) \in \mathbb{R}^{N\times D} \tag{3}$$

The points of interest are treated as queries, which update their features by iteratively attending the visual features of current frame with cross attention. These updated queries can thus be used to search for its best match within the current frame, as explained in the following section.

2.2.3 POINT PREDICTION

Unlike previous work that regresses the exact location of the points, we formulate the tracking as a matching problem to one of the patches, as required by ViT encoding, that provides a coarse estimate of the correspondence. To further find the exact correspondence with higher precision, we predict offsets to the patch center. Additionally, we also infer the visibility $\hat{\mathbf{v}}_t \in [0,1]^N$ and the uncertainty $\hat{\mathbf{u}}_t \in [0,1]^N$ for the point of interest.

Patch Classification: We first pass the visual features into 4-layer MLP, and downsample the resulting features to obtain multiscale maps, *i.e.*, $\mathbf{f}_t \rightarrow \mathbf{h}_t \in \mathbb{R}^{\frac{H}{S}\times\frac{W}{S}\times D} \rightarrow \mathbf{h}_t^l \in \mathbb{R}^{\frac{H}{2^l\cdot S}\times\frac{W}{2^l\cdot S}\times D}$. We then compute multi-scale similarity maps across the 4 levels, *i.e.*, cosine similarity between the decoded queries and patch embeddings:

$$\mathbf{C}_t^l = \frac{\mathbf{q}_t \cdot \mathbf{h}_t^l}{\|\mathbf{q}_t\| \cdot \|\mathbf{h}_t^l\|} \in \mathbb{R}^{N\times\frac{H}{2^l\cdot S}\times\frac{W}{2^l\cdot S}}, \quad l \in \{0,\ldots,3\} \tag{4}$$

The final similarity map $\mathbf{C}_t$ is computed as the weighted average of these multi-scale similarity maps using learned coefficients at the highest resolution. We apply a temperature to scale the similarity map, and take softmax spatially over the patches within the current frame. We train the model with a classification objective, the patch with points of interest is treated as the ground-truth class. Specifically, we perform P-class classification, where P is the total number of patches in the frame. We select the center of the patch with the highest similarity ($\hat{\mathbf{p}}^{patch} \in \mathbb{R}^{N\times 2}$) as our coarse prediction.

Offset Prediction: For the exact correspondence ($\hat{\mathbf{p}}_t \in \mathbb{R}^{N\times 2}$), we further predict an offset $\hat{\mathbf{o}}_t \in \mathbb{R}^{N\times 2}$ to the patch center by incorporating features from the local region around the inferred patch, as shown in Fig. 3:

$$\hat{\mathbf{o}}_t = \Phi_{\text{off}}(\mathbf{q}_t,\ \mathbf{h}_t,\ \hat{\mathbf{p}}_t^{patch}), \qquad \hat{\mathbf{p}}_t = \hat{\mathbf{p}}_t^{patch} + \hat{\mathbf{o}}_t \tag{5}$$

Here, Φ_{off} is a deformable transformer decoder (Zhu et al., 2021) block with 3 layers, excluding self-attention. In this decoder, the query $\mathbf{q}_t$ is processed using the key-value pairs $\mathbf{h}_t$, with the reference point set to $\hat{\mathbf{p}}_t^{\text{patch}}$. To limit the refinement to the local region, the offsets are constrained by the S (stride) and mapped to the range $[-S, S]$ using a tanh activation.

In addition, we predict the visibility $\hat{\mathbf{v}}_t$ and uncertainty $\hat{\mathbf{u}}_t$ with two separate 2-layer MLPs, followed by a sigmoid. At training time, we define a prediction to be uncertain if the prediction error exceeds a threshold ($\delta_u = 8$ pixels) or if the point is occluded. During inference, we classify a point as visible if its probability exceeds a threshold δ_v. Although we do not directly utilize uncertainty in our predictions during inference, we found predicting uncertainty to be beneficial for training.

Training: We train our model using the ground-truth trajectories $\mathbf{p}_t \in \mathbb{R}^{N\times 2}$ and visibility information $\mathbf{v}_t \in \{0,1\}^N$. For patch classification, we apply cross-entropy loss based on the ground-truth class, patch $\mathbf{c}^{patch}$. For offset prediction $\hat{\mathbf{o}}_t$, we minimize the ℓ_1 distance between the predicted offset and the actual offset. We supervise the visibility $\hat{\mathbf{v}}_t$ and uncertainty $\hat{\mathbf{u}}_t$ using binary cross-entropy loss. The total loss is a weighted combination of these four:

$$\mathcal{L} = \lambda\, \mathcal{L}_{\text{CE}}\left(\mathbf{C}_t,\ \mathbf{c}^{patch}\right)\ .\ \mathbf{v}_t + \mathcal{L}_{\ell_1}\left(\hat{\mathbf{o}}_t,\ \mathbf{o}_t\right)\ .\ \mathbf{v}_t + \mathcal{L}_{\text{CE}}(\hat{\mathbf{v}}_t, \mathbf{v}_t) + \mathcal{L}_{\text{CE}}(\hat{\mathbf{u}}_t, \mathbf{u}_t) \tag{6}$$

Discussion: Till this point, our model has exclusively considered relocating the queried points within the current frame. However, as the appearance of points consequently changes over time, the embedding similarity between the initial query point and future correspondences tends to decrease gradually (Fig. 4). This problem, known as feature drift, leads to inaccurate predictions (Li et al., 2024) when solely relying on the feature similarity with the initial point.

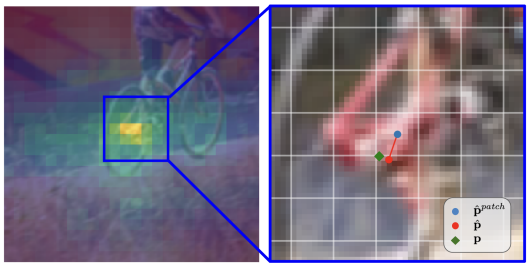


Figure 3: **Offset Head**. Starting with a rough estimation from patch classification (**left**), where lighter colors indicate higher correlation, we refine the prediction using the offset head (**right**). The selected patch center and the final prediction are marked by a blue dot and a red dot, respectively, with the ground-truth represented by a diamond.

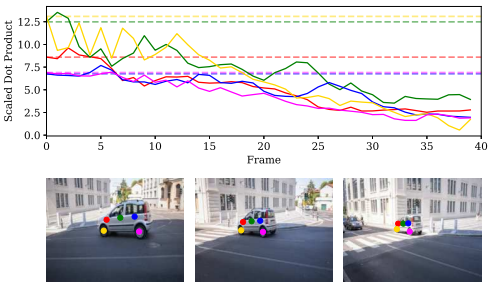


Figure 4: **Feature Drift.** For the tracks shown below (start, middle, and final frames), the plot above illustrates the decreasing similarity between the features of the initial query and its correspondences over time, with the initial similarity indicated by horizontal dashed lines.

2.3 Track-On with Memory

Here, we introduce two types of memories: **spatial memory** and **context memory**. Spatial memory stores information around the predicted locations, allowing us to update the initial queries based on the latest predictions. Context memory preserves the past states of the track by storing previously decoded queries, ensuring continuity over time and preventing inconsistencies. This design enables our model to effectively capture temporal relations in long-term videos while also adapting to changes in the target's features to address feature drift.

We store the past features for each of the N queries independently, with up to K embeddings of dimension D per query in each memory module. Once fully filled, the earliest entry from the memory will be obsoleted as a new entry arrives, operating as a First-In First-Out (FIFO) queue.

2.3.1 Spatial Memory

Here, we introduce the spatial memory module that stores fine-grained local information from previous frames, enabling continuous updates to the initial query points. This adaptation to appearance changes helps mitigate feature drift.

Memory Construction: We zero-initialize the memory, $\mathbf{M}^s_0$, update its content with each frame. For the first frame, we make a prediction using initial query $\mathbf{q}^{init}$ without memory.

Memory Write ($\Phi_{\text{q-wr}}$): To update the memory with the new prediction, $\mathbf{M}^s_{t-1} \rightarrow \mathbf{M}^s_t$, we extract a feature vector around the predicted point $\hat{\mathbf{p}}_t$ on the feature map $\mathbf{f}_t$ of the current frame and add it to the memory:

$$\mathbf{M}^s_t = [\mathbf{M}^s_{t-1},\ \Phi_{\text{q-wr}}\left([\mathbf{q}^{init}, \mathbf{q}_t],\ \mathbf{f}_t,\ \hat{\mathbf{p}}_t\right)] \tag{7}$$

Here, $\Phi_{\text{q-wr}}$ is a 3-layer deformable transformer decoder without self-attention, using the concatenated $\mathbf{q}^{init}$ and $\mathbf{q}_t$ over the channel dimension as the query vector, attending a local neighborhood of predicted point for update. Utilizing deformable attention for the local summarization process helps prevent error propagation over time, as the query can flexibly select relevant features from any range, ideally refining itself in cases of inaccurate predictions.

Query Update ($\Phi_{\text{q-up}}$): With the memory module, the initial query points first read the spatial memory $\mathbf{M}^s_{t-1}$, which is then passed into the query decoder to estimate the correspondence. Specifically, the update is performed by predicting a residual change to the initial query based on the memory content:

$$\mathbf{q}^{init}_t = \Phi_{\text{q-up}}\left(\mathbf{q}^{init},\ \mathbf{M}^s_{t-1}\right) = \mathbf{q}^{init} + \phi_{\text{qqm}}\left(\mathbf{q}^{init},\ \phi_{\text{mm}}(\mathbf{M}^s_{t-1} + \gamma^s)\right) \tag{8}$$

Here, ϕ_{mm} is a transformer encoder layer that captures dependencies across different time steps, by sharing information in memory. ϕ_{qqm} is a transformer decoder layer without initial self-attention,

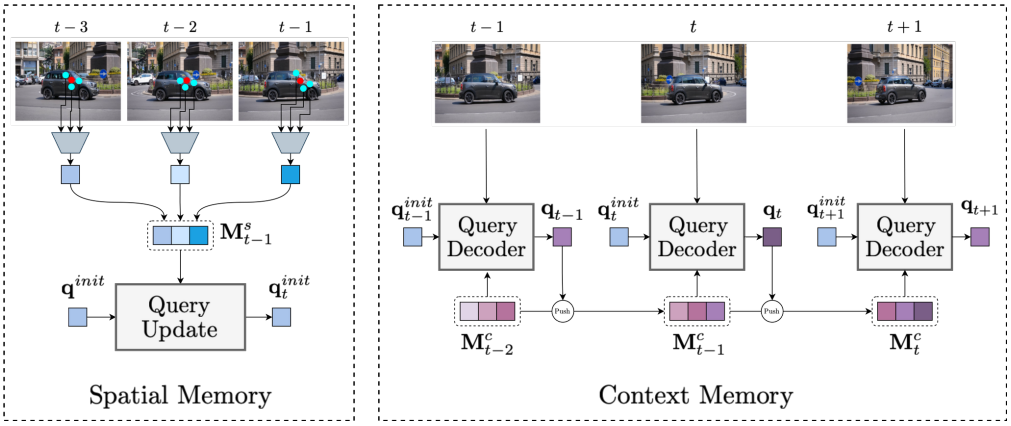


Figure 5: **Memory Modules.** Spatial memory $\mathbf{M}^s_{t-1}$ (**left**) is used to update the initial query $\mathbf{q}^{init}$ from the first frame to $\mathbf{q}^{init}_t$ on the current frame. The goal is to resolve feature drift by storing the content around the model's predictions in previous frames. Context memory $\mathbf{M}^c_{t-1}$ (**right**) is input to the query decoder which updates $\mathbf{q}^{init}_t$ to $\mathbf{q}_t$. It provides a broader view of the track's history with appearance changes and occlusion status by storing the point's embeddings from past frames.

where $\mathbf{q}^{init}$ attends to updated memory, followed by a linear layer, and $\gamma^s \in \mathbb{R}^{K\times D}$ is learnable, temporal positional embeddings.

Instead of sequentially updating the query embeddings at each time step, *e.g.* extracting $\mathbf{q}^{init}_t$ using $\mathbf{q}^{init}_{t-1}$, we update them with respect to the initial query $\mathbf{q}^{\text{init}}$, conditioned on all previous predictions stored in the memory. This approach prevents error propagation by taking into account the entire history of predictions.

2.3.2 Context Memory

In addition to spatial memory, we introduce a context memory that incorporates historical information of the queried points from a broader context, enabling the model to capture past occlusions and visual changes. Specifically, we store the decoded query features from previous time steps in context memory, $\mathbf{M}^c_{t-1}$. We then integrate it by extending the query decoder with an additional transformer decoder layer without self-attention, where queries attend to memory with added learnable temporal embeddings $\gamma^c \in \mathbb{R}^{K\times D}$:

$$\mathbf{q}_t = \Phi_{\text{q-dec}}\left(\mathbf{q}^{init}_t,\ \mathbf{f}_t,\ \mathbf{M}^c_{t-1} + \gamma^c\right) \tag{9}$$

Changes to the query decoder with memory are shown in red. For the writing operation, we add the most recent $\mathbf{q}_t$ to $\mathbf{M}^c_{t-1}$ and remove the oldest item, following the same procedure as in the spatial memory. Our experiments demonstrate that incorporating past content temporally with context memory enables more consistent tracking with additional benefits over spatial memory, especially in visibility prediction, since spatial memory focuses only on the regional content where the point is currently visible.

2.3.3 Inference-Time Memory Extension

Although the memory size K is fixed at training time, the number of video frames at inference can be different from the training frame limit, impacting the optimal number of frames stored in memory for effective content capture. To address this, we extend the memory size during inference by linearly interpolating the temporal positional embeddings, γ^s and γ^c, to sizes K_s and K_c for spatial memory and context memory, respectively. In particular, we train our memories with $K = 12$, and extend them to $K_s = K_c = 48$ at inference time.

Table 1: **Quantitative Results on TAP-Vid Benchmark.** This table shows results in comparison to the previous work on TAP-Vid under queried first setting, in terms of AJ, δ^x_{avg}, and OA. † Release model. ‡ Reproduction using the official checkpoints.

Model	DAVIS			RGB-Stacking			Kinetics		
	AJ ↑	δ^x_{avg}↑	OA ↑	AJ ↑	δ^x_{avg}↑	OA ↑	AJ ↑	δ^x_{avg}↑	OA ↑
Offline									
TAPIR†	58.3	71.1	87.7	-	-	-	52.5	65.3	85.5
CoTracker (All) ‡	60.8	74.8	88.4	60.5	73.3	83.5	48.4	62.2	83.2
CoTracker (Single) ‡	61.0	75.4	88.3	57.2	73.8	81.1	48.8	63.5	83.2
SpatialTracker	61.1	76.3	89.5	63.5	77.6	88.2	50.1	65.9	86.9
BootsTAPIR	61.4	74.0	88.4	-	-	-	54.7	68.5	86.3
BootsTAPIR †	62.4	74.6	89.6	-	-	-	55.8	68.8	86.6
TAPTR	63.0	76.1	91.1	60.8	76.2	87.0	49.0	64.4	85.2
Online									
DynOMo	45.8	63.1	81.1	-	-	-	-	-	-
TAPIR †	56.7	70.2	85.7	67.7	-	-	51.5	64.4	85.2
CoTracker (All) ‡	55.9	68.7	83.7	59.5	71.9	82.4	-	-	-
SpatialTracker ‡	57.3	70.6	85.0	52.9	69.4	80.3	-	-	-
BootsTAPIR †	59.7	72.3	86.9	-	-	-	**55.1**	**67.5**	86.3
Ours	**62.9**	**75.9**	**89.7**	**72.1**	**84.1**	**93.0**	53.2	66.6	**87.1**

3 Experiments

3.1 Experimental Setup

Datasets: We use TAP-Vid (Doersch et al., 2022) for both training and evaluation, consistent with previous work. Specifically, we train our model on TAP-Vid Kubric, a synthetic dataset of 11,000 video sequences, each with a fixed length of 24 frames. For evaluation, we use three other datasets from the TAP-Vid benchmark: TAP-Vid DAVIS, which includes 30 real-world videos from the DAVIS video dataset; TAP-Vid RGB-Stacking, a synthetic dataset of 50 videos focused on robotic manipulation tasks, mainly involving textureless objects; TAP-Vid Kinetics, a collection of over 1,000 real-world online videos.

Metrics: We evaluate tracking performance with the following metrics of TAP-Vid benchmark: Occlusion Accuracy (OA), which measures the accuracy of visibility prediction; δ^x_{avg}, the average proportion of visible points tracked within 1, 2, 4, 8, and 16 pixels; Average Jaccard (AJ), which jointly assesses visibility and localization precision.

Evaluation Protocol: We follow the standard protocol of TAP-Vid benchmark by first downsampling the videos to 256×256. We evaluate models in the queried first protocol, which is the natural setting for causal tracking. In this mode, the first visible point in each trajectory serves as the query, and the goal is to track that point in subsequent frames.

As most of the models are offline, we perform comparisons to these models using their official checkpoints under the online setting. In particular, for a video with T frames, we treat each target frame as the last frame of the video and run the model for that frame. While processing frame t, we provide the model with all prior frames from the beginning of the video up to frame t. This process is repeated for each frame sequentially, ensuring predictions are based only on past and present frames, without access to future frames. This gives an *upper bound*, for how well the model could work in an online setting.

3.2 Results

We compare Track-On to previous works in Table 1. We report both the online and offline results of SpatialTracker (Xiao et al., 2024) and CoTracker (Karaev et al., 2024), with their offline models evaluated in an online setting, as explained above. The reported results for TAPIR (Doersch et al.,

Table 2: **Offset Head.** The effect of removing the offset head (Φ_{off}) on models with varying strides.

Φ_{off}	Stride	$\delta^{2px}\uparrow$	$\delta^{4px}\uparrow$	$\delta^{8px}\uparrow$	AJ $\uparrow$	$\delta^{x}_{avg}\uparrow$	OA $\uparrow$
✗	8	16.2	54.0	88.1	41.0	51.5	89.4
✓		63.7	81.9	89.9	60.9	73.9	89.4
✗	4	47.6	79.9	90.1	52.9	65.7	89.4
✓		67.0	83.3	90.8	62.9	75.9	89.7

Table 3: **Memory Components.** The effect of spatial memory ($\mathbf{M}^s$), context memory ($\mathbf{M}^c$), and inference-time memory extension (IME).

Model	$\mathbf{M}^s$	$\mathbf{M}^c$	IME	AJ $\uparrow$	$\delta^{x}_{avg}\uparrow$	OA $\uparrow$
A	✗	✗	✗	52.4	67.8	79.9
B	✓	✗	✗	58.7	73.4	86.5
C	✗	✓	✗	59.3	73.4	88.8
D	✓	✓	✗	62.5	75.7	89.8
E	✓	✓	✓	62.9	75.9	89.7

2023) and BootsTAPIR (Doersch et al., 2024) are also included for both modes, along with their public release models, which demonstrate better performance than originally reported in the papers.

Comparison on DAVIS. Our model outperforms all online models across all evaluation metrics, demonstrating a 3.2 AJ improvement over the closest competitor, BootsTAPIR. Furthermore, it performs comparably to leading offline models, closing the gap with the state-of-the-art TAPTR (Li et al., 2024). It's important to note that most offline models exhibit substantial performance declines when evaluated in an online setting. Specifically, CoTracker, SpatialTracker, and BootsTAPIR show reductions of 4.9, 3.8, and 2.7 AJ, respectively.

Comparison on RGB-Stacking. The dataset consists of long video sequences, with lengths of up to 250 frames, making it ideal for evaluating models' long-term processing capabilities. Our model significantly surpasses both online and offline competitors, achieving an AJ improvement of 8.6 in offline mode and 4.4 in online mode. These results underscore a key limitation of the windowed inference approach used by CoTracker, SpatialTracker, and TAPTR, which struggles with long video sequences due to limited temporal coverage. By efficiently extending the temporal span through our memory mechanism, our model achieves substantial performance gains on long videos.

Comparison on Kinetics. The dataset comprises a variety of internet videos. For TAPIR and BootsTAPIR, the only distinction lies in the use of millions of real-world videos for fine-tuning, emphasizing the crucial role of training data. All other methods, including ours, are trained on TAP-Vid Kubric, without access to large-scale data and computational resources. Despite this difference, our model ranks second in both AJ and δ^{x}_{avg}, closely trailing BootsTAPIR in both online and offline settings, while achieving superior OA. This highlights the potential benefits of training on large-scale real-world data, which appears to be more advantageous for datasets like Kinetics compared to others.

3.3 Ablation Study

Offset Head and Stride: The offset head is essential for refining patch classification outputs, enabling more precise localization. Specifically, the offset head allows for precision beyond the patch size S (stride). In Table 3, we examine the impact of removing the offset head (Φ_{off}) for two stride values, $S = 4$ and $S = 8$. For both values, the addition of the offset head significantly enhances AJ and $\delta^{x}_{avg}$by refining predictions within the local region. With stride 4, the offset head notably improves δ^{2px}, while for stride 8, it improves both δ^{2px} and δ^{4px}. This demonstrates that while patch classification offers coarse localization, the offset head provides further refinement, achieving pixel-level precision.

Larger stride values improve memory efficiency by reducing feature resolution, but they also risk losing important details necessary for accurate tracking. For instance, increasing the stride from 4 to 8 results in performance drops of 12% for TAPIR and 16% for CoTracker, as reported in their ablation studies. However, our coarse-to-fine approach mitigates the negative effects of stride 8, leading to only a minimal decline of 3%, highlighting the robustness of our model to larger stride values.

Memory Modules: To demonstrate the importance of the proposed memory mechanism, we conduct an ablation study, as shown in Table 3. We start by evaluating the model without memory (Model-A), which corresponds to the vanilla model described in Sec. 2.2. As expected, due to the model's lack of temporal processing, Model-A performs poorly, particularly in OA. Introducing temporal information through either spatial memory (Model-B) or context memory (Model-C) leads

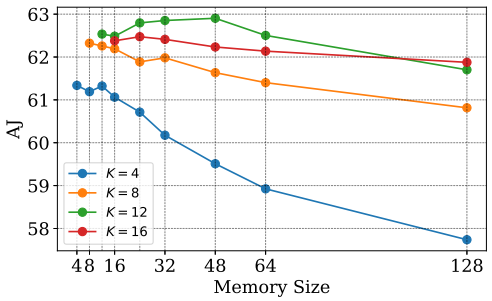


Figure 6: **Memory Size.** The effect of varying memory sizes set for training and extended during inference.

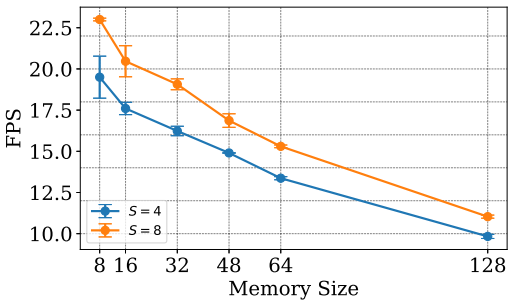


Figure 7: **Efficiency.** Inference speed for different memory sizes and strides.

to significant performance improvements. Model-C, in particular, achieves higher OA by providing a more comprehensive view of the track's history, including occlusions. Combining both memory types (Model-D) further boosts performance, highlighting the complementary strengths of the two memory modules. Lastly, incorporating the memory extension at inference time yields slight improvements in AJ and δ^x_{avg}, leading to an overall enhancement in performance.

Memory Size: We experimented with varying memory sizes, $K = \{4, 8, 12, 16\}$, during training and extended them further during inference, as shown in Fig. 6. When the same memory size is used for both training and inference, a memory size of 12 (green) yields the best performance, followed by 16 (red), 8 (orange), and 4 (blue). The lower performance of smaller memory sizes (4 and 8) suggests that memory needs to be sufficiently large to capture meaningful temporal dependencies. However, increasing memory size does not always lead to better results, as seen in the comparison between 12 and 16. This is because the memory also serves as a bottleneck for summarizing relevant information, and with a memory size of 16, this summarization function appears to be less effective compared to size 12, particularly for 24-frame-long training clips. The results indicate that extending memory during inference can improve performance up to a certain threshold. For example, with $K = 12$, performance improves up to a memory size of 48, after which it consistently declines. This suggests that while memory extension is beneficial, there is an optimal range beyond which performance begins to degrade.

Efficiency: We plot the inference speed (frames per second, FPS) as a function of memory size in Fig. 7, comparing stride values of 4 and 8. Unlike offline methods, our approach does not leverage temporal parallelization in the visual encoder, processing frames sequentially in the online setting. We report FPS on the TAP-Vid DAVIS dataset, averaging results from tracking 400 points over three runs on a single NVIDIA V100 GPU. As memory size increases, the model becomes slower due to the higher computational cost of temporal attention in memory operations. For example, with stride 4, the FPS drops from 20 with $K = 8$ to 15 with $K = 48$, and down to 10 with $K = 128$. In contrast, using a stride of 8 consistently offers faster inference speeds, ranging from approximately 22 FPS to 12 FPS across all memory sizes, mainly because there are fewer visual tokens to process with this larger stride. Our framework allows flexibility in adjusting this hyperparameter based on real-time application needs. Furthermore, our model is highly memory-efficient, requiring only 1.24 GB and 1.20 GB of GPU memory to process a frame with 400 tracks for strides of 4 and 8, respectively, highlighting the efficiency of our frame-by-frame tracking approach.

4 Related Work

Tracking Any Point: Point tracking, presents significant challenges, particularly for long-term tracking where maintaining consistent tracking through occlusions is difficult. PIPs (Harley et al., 2022) was one of the first approaches to address this by predicting motion through iterative updates within temporal windows. TAP-Vid (Doersch et al., 2022) initiated a benchmark for evaluation. TAPIR (Doersch et al., 2023) improved upon PIPs by refining initialization and incorporating depthwise convolutions to enhance temporal accuracy. BootsTAPIR (Doersch et al., 2024) further advanced TAPIR by utilizing student-teacher distillation on a large corpus of real-world videos. In

contrast, CoTracker (Karaev et al., 2024) introduced a novel approach by jointly tracking multiple points, exploiting spatial correlations between points via factorized transformers. More recently, TAPTR (Li et al., 2024) adopted a design inspired by DETR (Carion et al., 2020; Zhu et al., 2021), drawing parallels between object detection and point tracking. DINO-Tracker (Tumanyan et al., 2024) took a different route, using DINO as a foundation for test-time optimization.

However, all these models are designed for offline tracking, assuming access to all frames within a sliding window (Karaev et al., 2024) or the entire video (Doersch et al., 2023; 2024). Models with online variants (Doersch et al., 2024; 2023) are re-trained with a temporally causal mask to process frames sequentially on a frame-by-frame basis, despite being originally designed for offline tracking. In contrast, we explicitly focus on online point tracking by design, enabled by novel memory modules to capture temporal information. Additionally, many of these models use a regression objective with iterative updates, originally developed for optical flow (Teed & Deng, 2020), while we introduce a new paradigm based on patch classification and refinement.

Another line of research, orthogonal to ours, explores leveraging scene geometry for point tracking. SpatialTracker (Xiao et al., 2024) extends CoTracker to the 3D domain by tracking points in three-dimensional space, while OmniMotion (Wang et al., 2023) employs test-time optimization to learn a canonical representation of the scene. Concurrent work DynOMO (Seidenschwarz et al., 2024) also uses test-time optimization, utilizing Gaussian splats for online point tracking.

Causal Processing in Videos: Online, or temporally causal models rely solely on current and past frames without assuming access to future frames. This is in contrast to current practice in point tracking with clip-based models, processing frames together. Causal models are particularly advantageous for streaming video understanding (Yang et al., 2022a; Zhou et al., 2024), embodied perception (Yao et al., 2019), and processing long videos (Zhang et al., 2024; Xu et al., 2021), as they process frames sequentially, making them well-suited for activation caching. Due to its potential, online processing has been studied across various tasks in computer vision, such as pose estimation (Fan et al., 2021; Nie et al., 2019), action detection (Xu et al., 2019; De Geest et al., 2016; Kondratyuk et al., 2021; Eun et al., 2020; Yang et al., 2022b; Wang et al., 2021; Zhao & Krähenbühl, 2022; Xu et al., 2021; Chen et al., 2022a), temporal action localization (Buch et al., 2017; Singh et al., 2017), object tracking (He et al., 2018; Wang et al., 2020), video captioning (Zhou et al., 2024), and video object segmentation (Cheng & Schwing, 2022; Liang et al., 2020).

In causal models, information from past context is commonly propagated using either sequential models (De Geest et al., 2016), which are inherently causal, or transformers with causal attention masks (Wang et al., 2021). However, these models often struggle to retain information over long contexts or face expanded memory requirements when handling extended past contexts. To address this, some approaches introduce memory modules for more effective and efficient handling of complex tasks. For example, LSTR (Xu et al., 2021) separates past context as long-term and short-term memories for action detection, while XMem (Cheng & Schwing, 2022) incorporates a sensory memory module for fine-grained information in video object segmentation. Long-term memory-based modeling is also applied beyond video understanding (Balazevic et al., 2024), including tasks like long-sequence text processing and video question answering (Zhang et al., 2021). We also employ an attention-based memory mechanism, which is specialized in point tracking with two types of memory; one focusing on spatial local regions around points, and another on broader context.

5 Conclusion & Limitation

In this work, we presented **Track-On**, a simple yet effective transformer-based model for online point tracking. To establish correspondences, our model employs patch classification, followed by further refinement with offset prediction. We proposed two memory modules that enable temporal continuity efficiently while processing long videos. Our model significantly advances the state-of-the-art in online point tracking with fast inference and narrows the performance gap between online and offline models across a variety of public datasets.

Despite the strengths of our proposed model, there remain certain limitations. Specifically, the model may suffer from precision loss on thin surfaces and struggle to distinguish between instances with similar appearances, as observed in our failure cases (see Appendix). Future work could address these challenges by exploring learnable upsampling techniques to achieve higher-resolution feature maps and improve feature sampling accuracy.

REFERENCES

Ivana Balazevic, Yuge Shi, Pinelopi Papalampidi, Rahma Chaabouni, Skanda Koppula, and Olivier J Henaff. Memory consolidation enables long-context video understanding. In *Proc. of the International Conf. on Machine learning (ICML)*, 2024.

Sebastiano Battiato, Giovanni Gallo, Giovanni Puglisi, and Salvatore Scellato. Sift features tracking for video stabilization. In *14th international conference on image analysis and processing (ICIAP 2007)*, 2007.

Benjamin Biggs, Thomas Roddick, Andrew Fitzgibbon, and Roberto Cipolla. Creatures great and smal: Recovering the shape and motion of animals from video. In *Proc. of the Asian Conf. on Computer Vision (ACCV)*, 2019.

Shyamal Buch, Victor Escorcia, Chuanqi Shen, Bernard Ghanem, and Juan Carlos Niebles. Sst: Single-stream temporal action proposals. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2020.

Junwen Chen, Gaurav Mittal, Ye Yu, Yu Kong, and Mei Chen. Gatehub: Gated history unit with background suppression for online action detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022a.

Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2022b.

Ho Kei Cheng and Alexander G Schwing. Xmem: Long-term video object segmentation with an atkinson-shiffrin memory model. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2022.

Roeland De Geest, Efstratios Gavves, Amir Ghodrati, Zhenyang Li, Cees Snoek, and Tinne Tuytelaars. Online action detection. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2016.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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 *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023.

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. *arXiv preprint arXiv:2402.00847*, 2024.

Hyunjun Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Learning to discriminate information for online action detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Zhipeng Fan, Jun Liu, and Yao Wang. Motion adaptive pose estimation from compressed videos. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021.

Adam W Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. Particle video revisited: Tracking through occlusions using point trajectories. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2022.

Anfeng He, Chong Luo, Xinmei Tian, and Wenjun Zeng. A twofold siamese network for real-time object tracking. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Radu S Jasinschi, T Na Veen, et al. Motion estimation methods for video compression—a review. *Journal of the Franklin Institute*, 1998.

Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Dynamicstereo: Consistent dynamic depth from stereo videos. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. CoTracker: It is better to track together. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2024.

Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. Movinets: Mobile video networks for efficient video recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Ken-Yi Lee, Yung-Yu Chuang, Bing-Yu Chen, and Ming Ouhyoung. Video stabilization using robust feature trajectories. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2009.

Hongyang Li, Hao Zhang, Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, and Lei Zhang. Taptr: Tracking any point with transformers as detection. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2024.

Yongqing Liang, Xin Li, Navid Jafari, and Jim Chen. Video object segmentation with adaptive feature bank and uncertain-region refinement. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2019.

Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. Pose estimation for augmented reality: a hands-on survey. In *IEEE Trans. on Visualization and Computer Graphics (VCG)*, 2015.

Xuecheng Nie, Yuncheng Li, Linjie Luo, Ning Zhang, and Jiashi Feng. Dynamic kernel distillation for efficient pose estimation in videos. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019.

Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Peter Sand and Seth Teller. Particle video: Long-range motion estimation using point trajectories. In *International Journal of Computer Vision (IJCV)*, 2008.

Jenny Seidenschwarz, Qunjie Zhou, Bardienus Duisterhof, Deva Ramanan, and Laura Leal-Taixé. Dynomo: Online point tracking by dynamic online monocular gaussian reconstruction. *arXiv preprint arXiv:2409.02104*, 2024.

Gurkirt Singh, Suman Saha, Michael Sapienza, Philip HS Torr, and Fabio Cuzzolin. Online real-time multiple spatiotemporal action localisation and prediction. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2017.

Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2020.

Narek Tumanyan, Assaf Singer, Shai Bagon, and Tali Dekel. Dino-tracker: Taming dino for self-supervised point tracking in a single video. *arXiv preprint arXiv:2403.14548*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

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 *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2023.

Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. Tracking everything everywhere all at once. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023.

Xiang Wang, Shiwei Zhang, Zhiwu Qing, Yuanjie Shao, Zhengrong Zuo, Changxin Gao, and Nong Sang. Oadtr: Online action detection with transformers. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021.

Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. Towards real-time multi-object tracking. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2020.

Yuxi Xiao, Qianqian Wang, Shangzhan Zhang, Nan Xue, Sida Peng, Yujun Shen, and Xiaowei Zhou. Spatialtracker: Tracking any 2d pixels in 3d space. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Mingze Xu, Mingfei Gao, Yi-Ting Chen, Larry S Davis, and David J Crandall. Temporal recurrent networks for online action detection. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019.

Mingze Xu, Yuanjun Xiong, Hao Chen, Xinyu Li, Wei Xia, Zhuowen Tu, and Stefano Soatto. Long short-term transformer for online action detection. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Jinrong Yang, Songtao Liu, Zeming Li, Xiaoping Li, and Jian Sun. Real-time object detection for streaming perception. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022a.

Le Yang, Junwei Han, and Dingwen Zhang. Colar: Effective and efficient online action detection by consulting exemplars. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022b.

Yu Yao, Mingze Xu, Yuchen Wang, David J Crandall, and Ella M Atkins. Unsupervised traffic accident detection in first-person videos. In *Proc. IEEE International Conf. on Intelligent Robots and Systems (IROS)*, 2019.

Haoji Zhang, Yiqin Wang, Yansong Tang, Yong Liu, Jiashi Feng, Jifeng Dai, and Xiaojie Jin. Flash-vstream: Memory-based real-time understanding for long video streams. *arXiv preprint arXiv:2406.08085*, 2024.

Zhu Zhang, Chang Zhou, Jianxin Ma, Zhijie Lin, Jingren Zhou, Hongxia Yang, and Zhou Zhao. Learning to rehearse in long sequence memorization. In *Proc. of the International Conf. on Machine learning (ICML)*, 2021.

Yue Zhao and Philipp Krähenbühl. Real-time online video detection with temporal smoothing transformers. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2022.

Xingyi Zhou, Anurag Arnab, Shyamal Buch, Shen Yan, Austin Myers, Xuehan Xiong, Arsha Nagrani, and Cordelia Schmid. Streaming dense video captioning. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2021.

Appendices

A Experiment Details

A.1 Training Details

We train our model for 75 epochs, equivalent to approximately 50K iterations, using a batch size of 4 and a gradient accumulation step of 4, resulting in an effective batch size of 16. The model is optimized using the AdamW optimizer (Loshchilov & Hutter, 2019) on 4 $\times$ L40S GPUs, with mixed precision. The learning rate is set to a maximum of 5×10^{-4}, following a cosine decay schedule with a linear warmup period covering 1% of the total training time. A weight decay of 1×10^{-5} is applied, and gradient norms are clipped at 1.0 to ensure stable training. Input frames are resized to 384×512 using bilinear interpolation before processing.

For training, we utilize entire clips of length 24 from TAP-Vid Kubric. We adopt the data augmentation techniques from CoTracker (Karaev et al., 2024), including random cropping to a size of 384×512 from the original 512×512 frames, followed by random Color Jitter and Gaussian Blur. Each training sample includes up to $N = 480$ points. We apply random key masking with a 0.1 ratio during attention calculations for memory read operations throughout training.

For the training loss coefficients, we set λ to 5. During training, we clip the offset loss to the stride S to prevent large errors from incorrect patch classifications and stabilize the loss. Deep supervision is applied to offset head (Φ_{off}), and the average loss across layers is used. We set the softmax temperature τ to 0.05 in patch classification.

A.2 Implementation Details

All of our modules consist of either a Self-Attention Block, Cross-Attention Block, or Deformable Cross-Attention Block. Each block includes multi-head self-attention, multi-head cross-attention, or multi-head deformable cross-attention, followed by a 2-layer feed-forward network with a hidden dimension expansion ratio of 4. Each multi-head attention uses 8 heads, while deformable multi-head attention operates with 4 levels, sampling 4 points per head. We extract multi-level feature maps by downsampling the input feature map and set the feature dimension D to 256. Following CoTracker (Karaev et al., 2024), we add a global support grid of size 20×20 during inference.

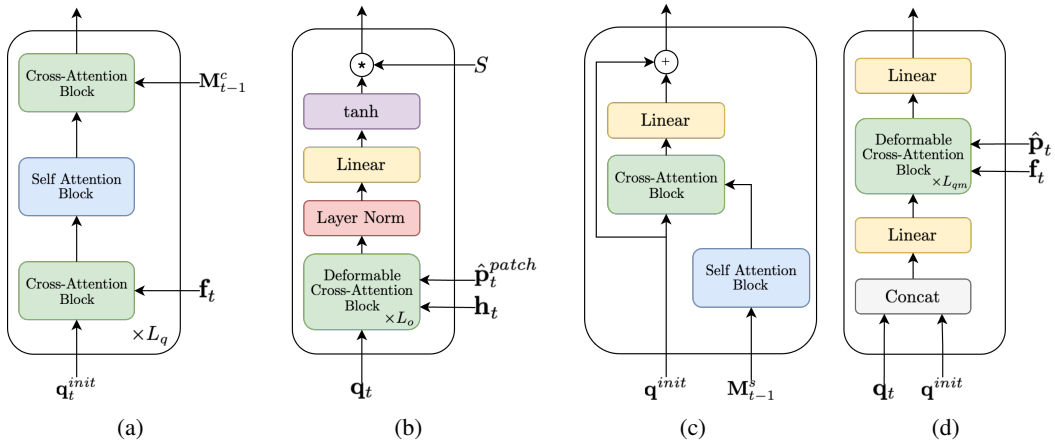


Figure 8: **Modules**. This figure describes the detailed architecture of modules in our approach: (a) Query Encoder, (b) Offset Head, (c) Query Updater, (d) Query Memory Writer.

Visual Encoder ($\Phi_{\text{vis-enc}}$): We use the ViT-Adapter (Chen et al., 2022b) with DINOv2 ViT-S/14 (Oquab et al., 2023) as the backbone. The DINOv2 inputs are resized to 378×504, as the default input size of 384×512 is not divisible by the patch size of 14. The backbone outputs,

with a dimension of 384, are projected to D using a single linear layer, implemented as a 1×1 convolution.

Query Decoder ($\Phi_{\text{q-dec}}$): Query Decoder is shown in Fig. 8a. We set the number of layers L_q to 3. Positional embedding γ^c from the context memory is applied only to the keys, not the values, ensuring time-invariance in the queries while enabling the model to differentiate between time steps during attention score calculation.

Offset Prediction (Φ_{off}): Fig. 8b shows the architecture of the offset prediction head. We set the number of layers L_o to 3. Following DETR (Carion et al., 2020), we normalize the decoded queries before projecting them through a linear layer, as the per-layer loss is calculated for the offset head.

Query Update ($\Phi_{\text{q-up}}$): The query update is detailed in Fig. 8c. In both self-attention and cross-attention blocks, we mask items corresponding to frames where points are predicted as occluded.

Spatial Memory Write ($\Phi_{\text{q-wr}}$): The spatial memory writer module is depicted in Fig. 8d. We set the number of layers L_{qm} to 3.

B Additional Comparisons

Table 4: **Quantitative Results on Dynamic Replica.**

Model	Online	δ^{vis}_{avg} ↑
TAP-Net	✓	53.3
PIPs	✗	47.1
TAPIR	✗	66.1
CoTracker (Single)	✗	68.9
BootsTAPIR	✗	69.0
TAPTR	✗	69.5
Ours	✓	**72.7**

Table 5: **Quantitative Results on RoboTAP.**

Model	Online	AJ ↑	δ^{x}_{avg} ↑	OA ↑
TAP-Net	✓	45.1	62.1	82.9
TAPIR	✗	59.6	73.4	87.0
CoTracker (Single)	✗	52.0	65.5	78.8
BootsTAPIR	✗	**64.9**	**80.1**	86.3
TAPTR	✗	60.1	75.3	86.9
Ours	✓	62.2	75.8	**88.8**

Dynamic Replica: We compare our model to previous work on the Dynamic Replica dataset (Karaev et al., 2023), a benchmark designed for 3D reconstruction using 20 sequences, each consisting of 300 frames, as shown in Table 4. Following prior work (Karaev et al., 2024), we evaluate models using $\delta^{\text{vis}}_{\text{avg}}$, similar to the evaluation protocol in the TAP-Vid benchmark. Unlike previous work, we do not report $\delta^{\text{vis}}_{\text{occ}}$, as our model is not supervised for occluded points. Despite being an online model, our approach outperforms other offline competitors. In particular, it surpasses TAPTR by 3.2 points and BootsTAPIR, which was trained on 15M real-world videos, by 3.7 points in $\delta^{\text{vis}}_{\text{avg}}$. This demonstrates the robustness of our model, especially in handling longer video sequences.

RoboTAP: We evaluate our model on the RoboTAP dataset (Vecerik et al., 2023), which contains 265 real-world robotic sequences with an average length of over 250 frames, in Table 5. We adopt the same metrics used in the TAP-Vid benchmark: AJ, δ^{x}_{avg}, and OA. Our model ranks second in AJ and δ^{x}_{avg}, trailing BootsTAPIR that works in an offline version, while achieving superior OA. This dataset, which includes textureless objects, represents a key challenge where fine-tuning on real-world videos yields significant improvements, as learning to track points on textureless objects is notably difficult, as noted in BootsTAPIR. Among other models trained on the same dataset (TAP-Vid Kubric), our model surpasses the closest competitor by 2.1 in AJ, 0.5 in δ^{x}_{avg}(TAPTR), and 1.8 in OA (TAPIR).

BADJA: We compare our model to previous work on the BADJA challenge (Biggs et al., 2019) in Table 6, a dataset for animal joint tracking, consisting of 7 sequences. We use two metrics for evaluation: δ^{seg}, which measures the proportion of points within a threshold relative to the size of the segmentation mask (specifically, points within a threshold of $0.2\sqrt{A}$, where A is the area of the segmentation mask); and δ^{3px}, the ratio of points tracked within a 3-pixel range.

Among online models, our model achieves state-of-the-art results, outperforming CoTracker (Karaev et al., 2024) by a margin of 4.0 in δ^{seg} and also achieving the best performance in δ^{3px}. Among offline models, our model surpasses several works in δ^{seg}, including PIPs (Harley et al., 2022), Omnimotion (Wang et al., 2023), both modes of CoTracker -single and all-, and TAPTR (Li et al.,

Table 6: **Quantitative Results on BADJA.** ‡ Reproduction using official checkpoints.

Model	δ^{seg} ↑	δ^{3px} ↑
Offline		
PIPs	61.9	13.5
TAPIR	66.9	15.2
OmniMotion	57.2	13.2
CoTracker (All) ‡	59.7	16.2
CoTracker (Single) ‡	66.1	19.1
SpatialTracker	69.2	17.1
TAPTR	64.0	18.2
Online		
TAP-Net	54.4	6.3
CoTracker (All) ‡	56.5	15.0
CoTracker (Single) ‡	<u>62.7</u>	17.0
SpatialTracker ‡	62.5	<u>17.6</u>
Ours	**66.7**	**17.8**

Table 7: **Quantitative Results for Two-View Matching on TAP-Vid DAVIS.** ‡ Reproduction using official checkpoints.

Model	AJ ↑	δ^x_{avg} ↑	OA ↑
CoTracker (Single) ‡	36.3	52.6	74.3
SpatialTracker ‡	28.7	44.1	74.4
Ours	**52.4**	**67.8**	**79.9**

2024), while also outperforming SpatialTracker (Xiao et al., 2024) and TAPIR (Doersch et al., 2023) in δ^{3px}. Overall, no offline model surpasses our model's performance in both metrics.

Two-View Correspondence: We evaluated the two-view correspondence estimation abilities of state-of-the-art models with iterative updates, *i.e.* CoTracker and SpatialTracker on TAP-Vid DAVIS. Without any temporal cues, we provide the models only with the query and target frames for matching, rather than the full video. The results, shown in Table 7, clearly demonstrate that our model performs significantly better in estimating correspondences, improving AJ by 16.1 and 23.7 compared to CoTracker and SpatialTracker, respectively. This improvement can be attributed to patch classification in our approach, which can operate without requiring simultaneous optimization across time steps, underscoring the advantages of frame-by-frame matching.

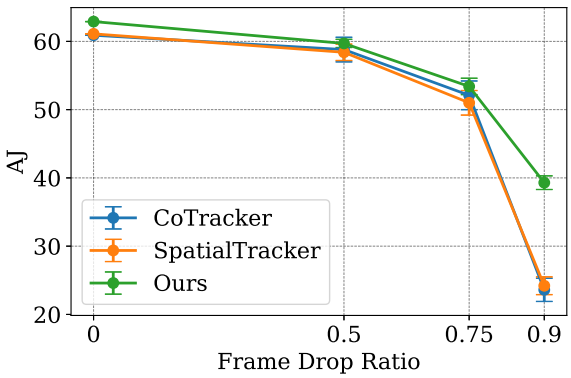


Figure 9: **Temporal Robustness.** The effect of randomly dropping frames on AJ on TAP-Vid DAVIS.

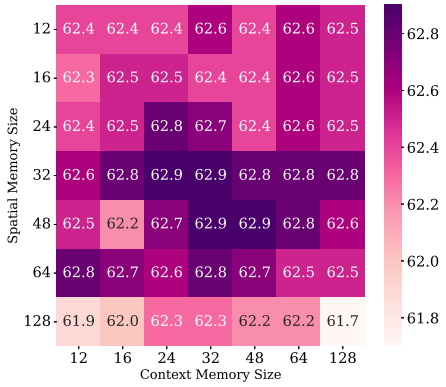


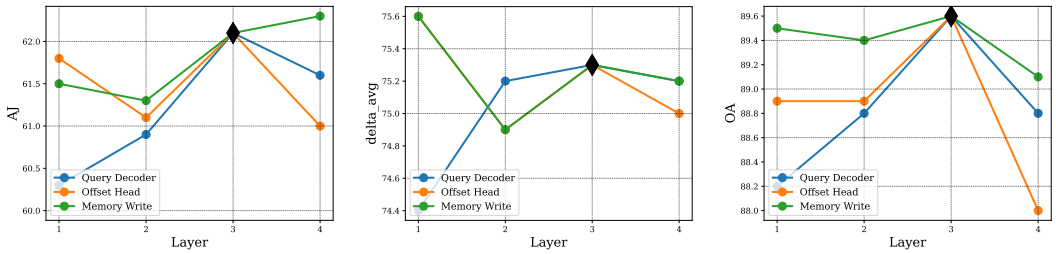
Figure 10: **Inference-Time Memory Extension.** The effect of jointly varying the sizes of context and spatial memory during inference on AJ on TAP-Vid DAVIS.

Temporal Robustness: We evaluate the temporal robustness of prior methods, CoTracker (Karaev et al., 2024) and SpatialTracker (Xiao et al., 2024), which rely on windowed iterative updates, against our frame-by-frame approach by randomly dropping frames at varying ratios on TAP-Vid DAVIS, as shown in Fig. 9. The experiments were conducted using 6 different seeds with drop ratios of 0.5, 0.75, and 0.9. All methods perform similarly at a 0.5 drop ratio, demonstrating robustness to moderate frame loss. However, at a 0.9 drop ratio, the performance of prior methods drops to around 40% of their clean performance in AJ, while our method maintains approximately 65% of its clean

performance. This highlights that our approach is less dependent on temporal consistency, unlike previous methods that heavily rely on continuity in a temporal window for iterative updates.

C Additional Ablations

Inference-Time Memory Extension: To explore the behavior of inference-time memory extension with respect to varying memory sizes and independently determine the optimal range, we extended the ablation study from Sec. 3.3. We conducted a more detailed search, as shown in Fig. 10, using our default model trained with a memory size $K = 12$. We report the AJ score on TAP-Vid DAVIS with varying context memory size $\mathbf{K}_c$ and spatial memory size $\mathbf{K}_s$. The results show that larger values for $\mathbf{K}_s$ degrade performance, likely due to spatial representations becoming obsolete while increasing $\mathbf{K}_c$ impacts the performance less (bottom row vs. rightmost column). Optimal performance is achieved when both memories are within the range of 24 to 48, corresponding to the middle of the matrix.



(a) Change in AJ (b) Change in δ^x_{avg} (c) Change in OA

Figure 11: **Ablating the Number of Layers.** The impact of varying the number of layers in the query decoder $\Phi_{\text{q-dec}}$ (blue), offset head Φ_{off} (orange), and the spatial memory writer $\Phi_{\text{q-wr}}$ (green). Our default setting, with 3 layers, is marked with a black diamond.

Number of Layers: We experimented with varying the number of layers for the query decoder $\Phi_{\text{q-dec}}$ (Sec. 2.2.2), offset head Φ_{off} (Sec. 2.2.3), and memory writer $\Phi_{\text{q-wr}}$ (Sec. 2.3) one at a time, as shown in Fig. 11, training each configuration for 50 epochs, and without inference-time memory extension. During these experiments, only the tested hyperparameter was changed, while all other settings were kept the same as the default model. For the query decoder (blue), increasing the number of layers from 1 to 3 consistently improves scores across all metrics. However, increasing it to 4 results in either slight drops (AJ and δ^x_{avg}) or a significant decrease (OA). Conversely, the layer numbers for the offset head (orange) and spatial memory writer (green) do not show consistent trends, instead exhibiting slight fluctuations with changes in layer count. Overall, 3 layers provide the best performance when considering all metrics together. Although different modules achieve their optimal performance at varying layer counts, the performance differences between these hyperparameter settings are not substantial, indicating a degree of robustness in our model.

Table 8: **Number of Scales.** The effect of changing number of scales for similarity calculation.

Scale Set (2^l)	AJ ↑	δ^x_{avg} ↑	OA ↑
{0}	61.4	74.9	88.6
{0, 1}	61.3	74.7	88.8
{0, 1, 2}	61.0	74.7	88.7
{0, 1, 2, 3}	**62.1**	**75.3**	**89.6**

Table 9: **Backbone and Number of Learnable Parameters**. The backbone, number of learnable parameters, and the AJ score of previous work on TAP-Vid DAVIS. All models are online.

Model	Backbone	#L.P.	AJ ↑
TAPIR	ResNet-18	29M	56.7
CoTracker (All)	Residual CNN	45M	55.9
BootsTAPIR	ResNet-18 + 4 Conv	78M	59.7
Ours	Residual CNN	41M	61.6
	ViT-Adapter	23M	**62.9**

Number of Scales: We experimented with different numbers of scales in the similarity map $\mathbf{C}_t$ (Eq (4)) by varying the set of scales, *i.e.* downsample ratios, to {0}, {0, 1}, {0, 1, 2}, and {0, 1, 2, 3}, as shown in Table 8. The number of scales corresponds to the size of the scale set. Similar to

the layer number ablation, we trained our model for 50 epochs for each configuration. Our results indicate that the model is robust to the choice of scale numbers, except when using 4 scales, which provides the best overall performance. We observed an increase of 0.7 in AJ and 0.4 in δ^x_{avg} when comparing $\{0\}$ to $\{0, 1, 2, 3\}$, as well as an improvement of 0.8 in OA when comparing $\{0, 1\}$ to $\{0, 1, 2, 3\}$. These findings demonstrate that using 4 scales, with downsample ratios of 1, 2, 4, and 8, yields optimal performance.

Backbone: We replaced our ViT-Adapter backbone with the Residual CNN backbone from CoTracker and compared the performance with other models, considering their respective feature backbones, total number of learnable parameters, and online AJ performance on TAP-Vid DAVIS in Table 9. Our default model achieves the best performance while using only half the parameters of CoTracker (23M vs. 45M) and one-third of BootsTAPIR (23M vs. 78M), thanks to the efficiency of the ViT-Adapter architecture, which leverages DINOv2 features. Even when our model uses the Residual CNN backbone of CoTracker, it still outperforms other models. However, it lags behind our default model by 1.3 AJ, despite having 18M additional parameters.

D Analysis of Spatial Memory

To evaluate the impact of spatial memory (Sec. 2.3) in reducing feature drift, we conducted an additional analysis comparing the tracking performance of the initial feature sampled from the query frame, $\mathbf{q}^{init}$, with the query feature updated using spatial memory at frame t, denoted as $\mathbf{q}^{init}_t$. For this evaluation, we introduced two new metrics: (i) the similarity ratio score (s_{sr}), which measures how well the updated query features align with the feature at the target point compared to the initial query, and (ii) the matching score (s_m), which quantifies the improvement in matching performance achieved by the updated queries, based on their similarity to all features in the target frame.

Similarity Ratio Score: Ideally, $\mathbf{q}^{init}_t$ should provide a better starting point for detecting correspondences compared to $\mathbf{q}^{init}$, particularly when the object's appearance changes significantly. To assess whether $\mathbf{q}^{init}_t$ is more similar to the feature at the ground-truth correspondence location than $\mathbf{q}^{init}$, we calculate the ratio of their similarity to ground-truth, as a way of quantifying the increase in the similarity after the update:

$$s_{sr}(t) = \frac{\mathbf{q}^{init}_t \cdot \text{sample}(\mathbf{f}_t,\ \mathbf{p}_t)}{\mathbf{q}^{init} \cdot \text{sample}(\mathbf{f}_t,\ \mathbf{p}_t)} \tag{10}$$

Here, $\mathbf{p}_t$ represents the location of the ground-truth correspondence point, and $\mathbf{f}_t$ is the feature map of the target frame. On the DAVIS dataset, we calculated s_{sr} for visible points, achieving a score of 1.20, indicating that spatial memory introduces a 20% increase in similarity compared to the initial feature. In Fig. 12, we visualize the similarity scores for different tracks over time for a sample video from the DAVIS dataset. The plot highlights that the similarity increases more significantly toward

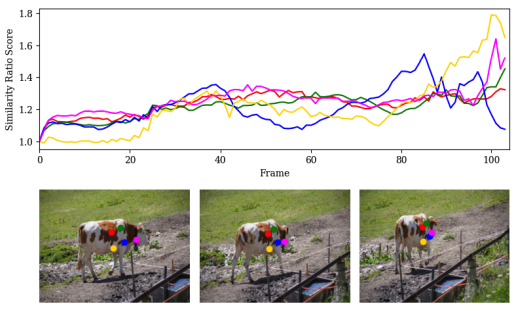


Figure 12: **Similarity Ratio Score.** The similarity ratio score $s_{sr} > 1$ over frames for different tracks, demonstrates increased similarity with ground-truth location on the target frame when utilizing spatial memory.

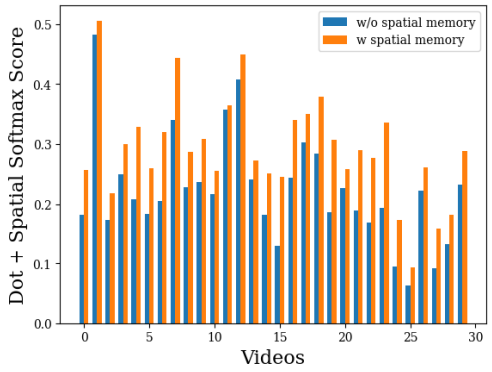


Figure 13: **Spatial Memory Effect on Matching Score.** The effect of using spatial memory to update query feature on matching score s_m.

the end of the video, where appearance changes are more severe. Moreover, the score is consistently greater than 1, showing that $\mathbf{q}_t^{init}$ always provides better initialization than $\mathbf{q}^{init}$ in this video.

Matching Score: While the similarity ratio score indicates how the initial query becomes more similar to the target frame, it does not account for the similarity between other tokens in the target frame. To address this, we extend our analysis to track performance based on similarity without any decoding, using only the initial or updated queries. We define the matching score $s_m(t,\ \mathbf{q}^*)$ as:

$$s_m(t,\ \mathbf{q}^*) = \text{sample}(\text{softmax}(\frac{\mathbf{f}_t \cdot \mathbf{q}^*}{\tau}),\ \mathbf{p}_t) \tag{11}$$

This metric computes the similarity between the any query feature $\mathbf{q}^*$ and frame features $\mathbf{f}_t$ using the dot product, followed by spatial softmax to obtain the probability distribution across all feature tokens. We then sample the probability at the ground-truth location $\mathbf{p}_t$. A higher s_m score indicates greater correlation between $\mathbf{q}^*$ while also accounting for the similarity with all other tokens. We calculated s_m for each visible point on the DAVIS dataset with $\tau = 0.1$, both with and without spatial memory, *i.e.* $s_m(t,\ \mathbf{q}_t^{init})$ vs. $s_m(t,\ \mathbf{q}^{init})$. The mean per-video results are shown in Fig. 13. Our analysis reveals a consistent increase in the matching score across all videos, with the average s_m increasing from 0.22 to 0.29, a 32% improvement in performance.

In Fig. 14, we visualize the matching scores with and without spatial memory, highlighting how spatial memory enhances similarity to the target feature. For example, in Fig. 14a, spatial memory successfully keeps the search query $\mathbf{q}_t^{init}$ up-to-date, while the initial query $\mathbf{q}^{init}$ yields nearly zero matching scores during frames 10–20 and 30–50. This clearly demonstrates that spatial memory provides a significantly better search query compared to the initial features. In another example (Fig. 14b), the matching score for $\mathbf{q}^{init}$ remains near zero after 15 frames due to changes in the object's scale and appearance. In contrast, spatial memory maintains a higher level of similarity, adapting to these changes effectively. However, spatial memory is not always able to recover from drift. For instance, in Fig. 14d, the matching score drops to zero after 10 frames, resulting in a complete tracking failure.

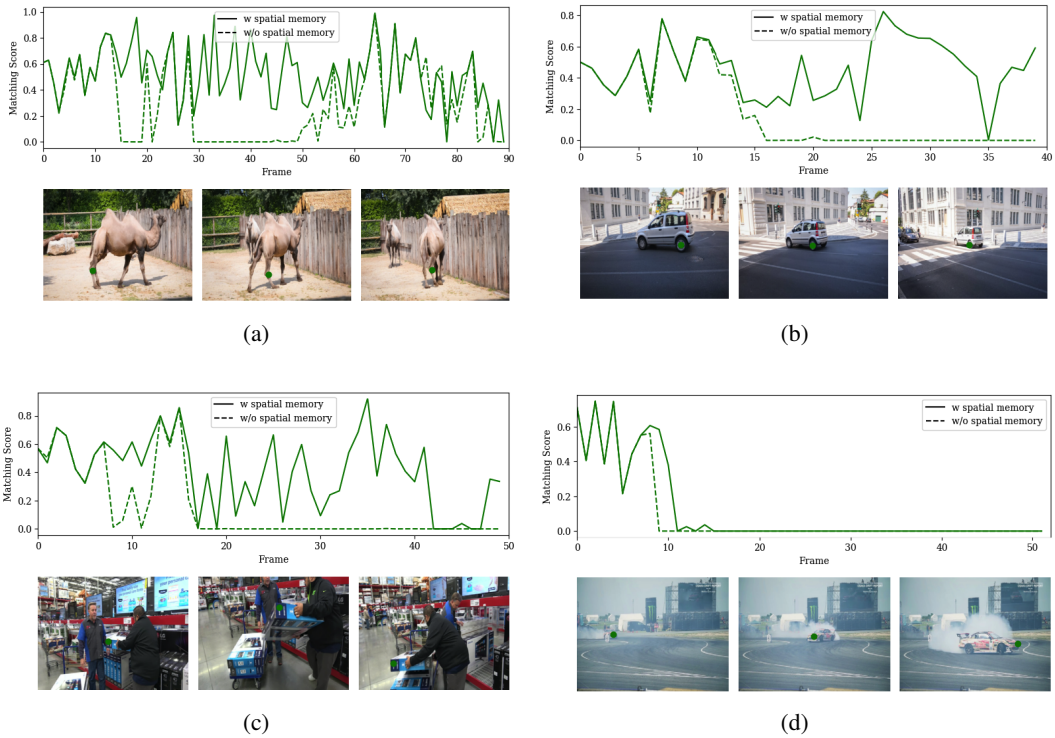


Figure 14: **Matching Score.** The effect of using the updated query from spatial memory, $\mathbf{q}_t^{init}$, compared to the initial query sample, $\mathbf{q}^{init}$, on improving matching based on similarity. The plot in the upper row shows the matching score s_m over time for the point marked in the lower row.

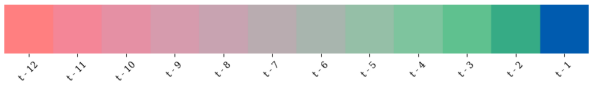
E Analysis of Temporal Embeddings

We initialize the temporal positional embeddings of memories, γ^c and γ^s, as learnable parameters to enable the model to differentiate between memory timestamps. To better understand what these learned embeddings represent, we visualize the context memory embeddings, γ^c, trained with memory size $K_c = 12$, in Fig. 15. The embeddings are reduced from $D = 256$ dimensions to 3 using Principal Component Analysis (PCA), and scaled between 0 and 1, with each dimension mapped to the Red, Green, and Blue color channels.

The visualization reveals three distinct features corresponding to timestamps: $t-1$ (blue), $t-2$ (green), and $t-12$ (red). The intermediate timestamps between $t-2$ and $t-12$ appear as smooth interpolations of these colors, with the midpoint, $t-7$, combining blue and red. This indicates that the model differentiates time steps smoothly, from the most recent to the oldest.

During inference, we apply linear interpolation, where new embeddings are represented as a weighted average of the learned features. In Fig. 15b, we extend the embeddings to $K_c = 48$ and apply PCA in the same manner. The visualization confirms that the temporal order of frames is preserved.

Additionally, we conducted an experiment where we trained the model without temporal positional embeddings, as shown in Table 10, and reported the results on the TAP-Vid DAVIS dataset without inference-time memory extension. Including temporal positional embeddings resulted in an increase of 1.8 AJ, 1.1 δ^x_{avg}, and 2.1 OA, highlighting the importance of temporal embeddings in the memory for the model's performance.



(a) Visualization of γ^c features with the original size $K_c = 12$.

(b) Visualization of γ^c after extension to $K_c = 48$.

Figure 15: **Learned Temporal Embeddings.** We visualize the temporal embeddings by applying Principal Component Analysis (PCA) and reducing dimension to 3 and scale to 0 and 1, where each dimension corresponds to each channel of RGB. Upper row (15a) shows the original embeddings, and lower row (15b) shows the memory after extension.

Table 10: **Temporal Positional Embeddings.** The effect of removing learnable temporal positional embeddings in the memory modules, *i.e.* γ^s and γ^c, during training.

$\gamma^{\{s,c\}}$	AJ ↑	δ^x_{avg} ↑	OA ↑
✗	60.7	74.6	87.7
✓	**62.5**	**75.7**	**89.8**

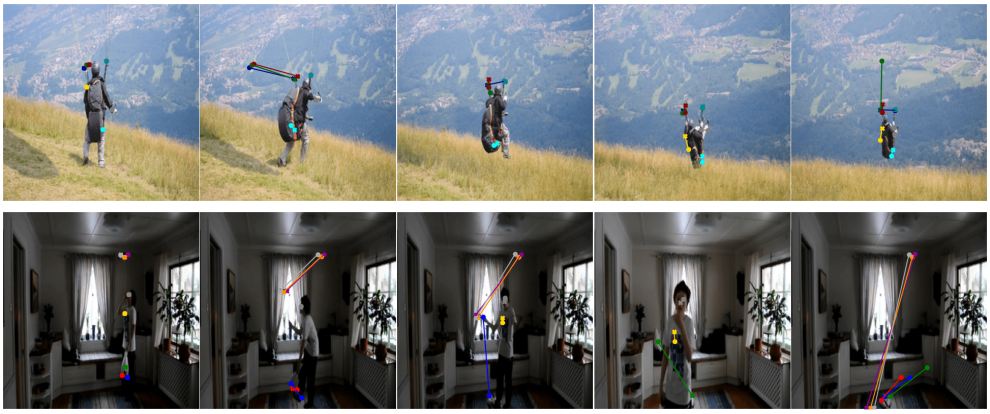
F Failure Analysis

We identify two common failure cases: (i) tracking points on thin surfaces, and (ii) localization on uniform areas. We visualize examples of these failure cases, *i.e.* predictions with $\delta^{8px} < 0.9$, in Fig. 16. In the visualizations, our predictions are represented as dots, while ground-truth correspondences are marked with crosses. The line connecting the ground-truth and prediction indicates the error.

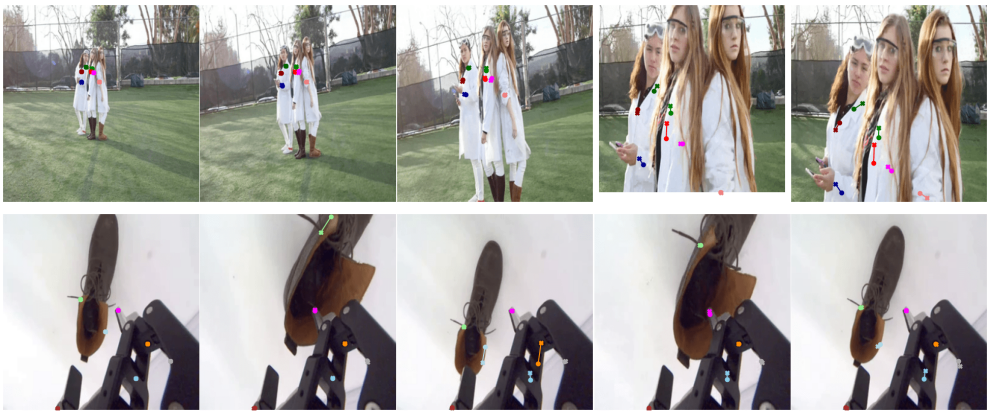
Thin Surfaces: Points of interest on thin surfaces and edges may not be well-represented in feature-level resolution due to the lack of pixel-level granularity. This limitation causes the model to track incorrect points by missing the actual point of interest. For instance, in the upper row of Fig. 16a, which shows an example from the DAVIS dataset, the model fails to track a rope and instead tracks the background, as the precision is insufficient to accurately represent the thin structure. Similarly, in the bottom row, from the Kinetics dataset, points of interest on a thin surface (*e.g.* a drone) are mislocalized, with the model tracking the background instead of the object.

Uniform Areas: Localization on uniform areas is more challenging, likely because most objects in the training dataset have descriptive textures (Doersch et al., 2024). While our model can roughly

localize points, it lacks precision in such cases. This issue is illustrated in Fig. 16b, where the upper row shows an example from the DAVIS dataset, and the bottom row shows one from RoboTAP.



(a) Failure Cases due to Thin Surfaces.



(b) Failure Cases due to Mislocalization on Uniform Areas.

Figure 16: **Common Failure Cases.** We identify two common failure cases: tracking points on thin surfaces (a), and localization on uniform areas (b). We visualize predictions with $\delta^{8px} < 0.9$, where predictions are shown as dots and ground-truth correspondences are marked with crosses. Different tracks are depicted in distinct colors.

G Qualitative Results

We provide some qualitative results from TAP-Vid DAVIS videos in Fig. 17. The top row shows equally sampled frames from the video, with ground truth correspondences indicated by green diamonds. The local regions around the ground truth points, outlined by green rectangles, are displayed in the bottom row, along with our model's patch predictions (blue) and the refined final points obtained using the offset head (red). The middle row shows the correlation scores for patch classification, as explained in Sec. 2.2.3, where lighter colors correspond to higher correlation. For visualization purposes, we do not apply temperature scaling to the correlation scores. We displayed the frame as grayscale if the point is occluded.

These visualizations demonstrate that our model can accurately predict dynamic points (first video) and points in the background, moving according to camera motion (second video). Moreover, we observe that the offset head refines the predictions correctly toward the ground truth locations (third video). Nevertheless, our model may fail to establish correct correspondences in some cases, as shown in the last row. For example, in the second column, our model misses the point because the visual similarity between the wrongly predicted patch and the query point is high.

Additionally, we provide a visual comparison with previous works, CoTracker (Karaev et al., 2024) and SpatialTracker (Xiao et al., 2024), on videos from TAP-Vid DAVIS, in Fig. 18. In this figure, the results are arranged from top to bottom in the following order: ground truth, CoTracker, SpatialTracker, and ours. In the first video, our model tracks background objects more accurately than CoTracker (middle column) and localizes points on the moving object's surface with greater precision (last column). In the second video, our model accurately tracks points on the moving object (the upper fish) compared to the other models.

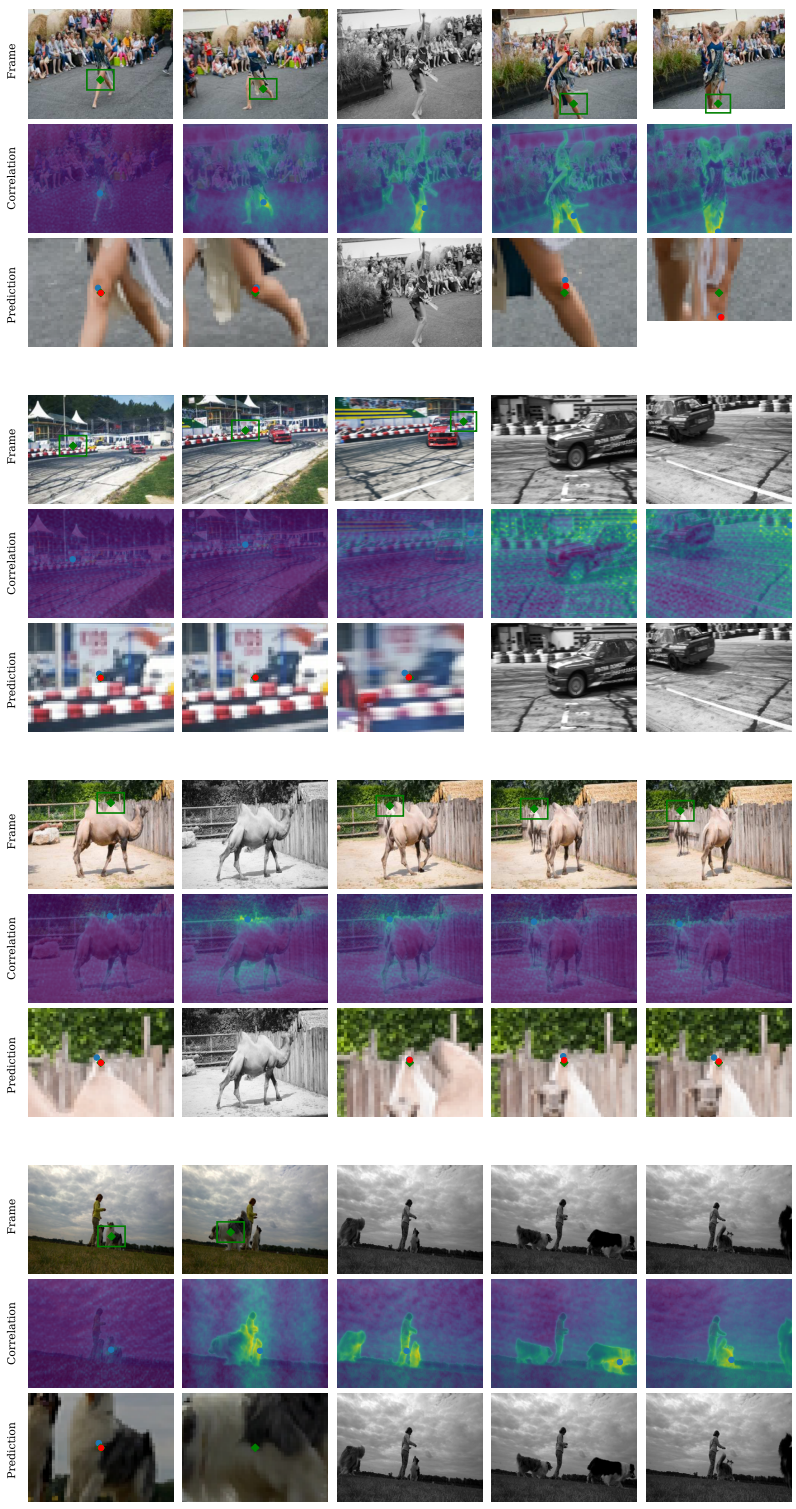


Figure 17: **Qualitative Results on TAP-Vid DAVIS.** The top row shows the frames and ground-truth correspondences, with frames in grayscale indicating occlusion. The middle row shows the correlation map; and the bottom row, the zoomed-in local region around the ground-truth point (marked by a rectangle in the top row). In this region, the blue dot marks the selected patch center; the red dot, the refined prediction; and the green diamond, the ground-truth.

Figure 18: **Qualitative Comparison on TAP-Vid DAVIS.** We compare to previous work CoTracker (Karaev et al., 2024) and SpatialTracker (Xiao et al., 2024) by marking predictions for different tracks in different colors.