# Appendix

## A. Implementation details

**Model architecture.** Our transformer $\Psi$ follows the architecture of CoTracker [6]. We set the feature dimensionality to $d = 128$, use a neighborhood size of $\Delta = 3$, and perform $M = 4$ iterations during training and $M = 6$ during evaluation.

For each query point, we extract feature patches from the initial frame $t^q$ and from the current track estimate at frame $t$: $\phi_{t^q}, \phi_t \in \mathbb{R}^{d \times (2\Delta+1)^2}$.

We then compute the dot product between all pairs of feature vectors, forming a correlation matrix:

$$\phi_{t^q}^\top \phi_t \in \mathbb{R}^{(2\Delta+1)^2 \times (2\Delta+1)^2} \tag{1}$$

which contains $(2\Delta + 1)^4 = 2{,}401$ values for $\Delta = 3$. An MLP maps this correlation matrix to a 256-dimensional representation.

Each token is then formed by concatenating these correlations with additional features—visibility, confidence, and encoded displacements—along the channel dimension. Tokens are arranged into a grid of shape $(B, T, N, C)$, where $B$ is the batch size, $T$ is the number of frames, $N$ is the number of points, and $C$ is the channel dimension.

To process this grid efficiently, we employ factorized attention [1], alternating between temporal and spatial operations. For temporal attention, the grid is reshaped to $(B \cdot N, T, C)$, enabling attention across frames for each point. For spatial (group) attention, it is reshaped to $(B \cdot T, N, C)$, enabling attention across points within each frame.

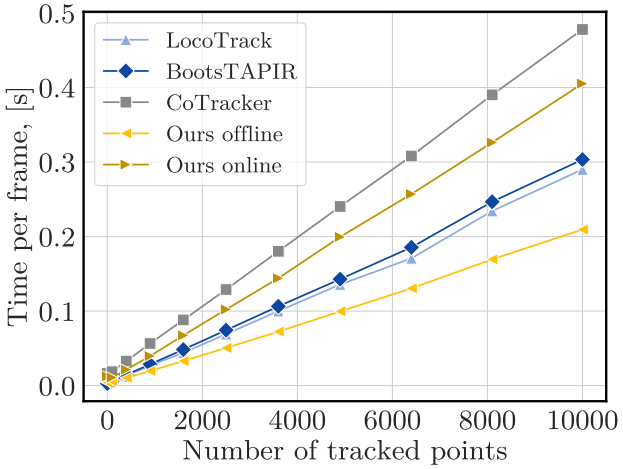


Figure 2. **Efficiency.** We evaluate the speed of different trackers on DAVIS depending on the number of tracks and report the average time each tracker takes to process a frame. Our offline architecture is the fastest among all these models, with LocoTrack being the fastest tracker to date.

**Pre-training.** We pre-train both online and offline model versions on synthetic **TAP-Vid-Kubric** [3, 5] for 50,000 iterations on 32 NVIDIA A100 80GB GPUs with a batch size of 1 video, which takes around 24 hours for the offline version and 35 hours for the online version.

We train CoTracker3 online on videos of length $T = 64$ with a window size of 16 and sample 384 query points per video with a bias towards objects. Since the online version tracks only forward in time, we sample points primarily at the beginning of the video. We train the offline version on videos of length $T \in \{30, 31, \ldots, 60\}$ with time embeddings of size 60. We interpolate time embeddings to the current sequence length both at training and evaluation. We sample 512 query points per video uniformly in time. Both models are trained in bfloat16 with gradient norm clipping using PyTorch Lightning [4] with PyTorch distributed data parallel [7]. The optimizer is AdamW [8] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate $5{\cdot}10^{-4}$, and weight decay $1{\cdot}10^{-5}$. The optimizer adopts a linear warm-up for 1000 steps followed by a cosine learning rate scheduler.

It is possible to train the online model from scratch on smaller GPUs (e.g., $32 \times 24$GB L4) by sampling 128 points instead of 384 and using 48 frames instead of 64. This reduction does not significantly affect performance, except when tracking several thousand points at a time. Using 8 GPUs instead of 32 increases the training time from 35 to 140 hours.

**Scaling.** We scale CoTracker3 on a dataset of Internet-like videos primarily featuring humans and animals. We visualize the scaling pipeline in Fig. 1. To ensure the quality and relevance of our training data, we use caption-based filtering with specific keywords to select videos containing real-world content while excluding those with computer-generated imagery, animation, or natural phenomena that are challenging to track, such as fire, lights, and water.

When training on real data, we use a similar setup while reducing the learning rate to $5e-5$ with the same cosine scheduler without warm-up. We train both online and offline versions for 10,000 iterations with 384 tracks per video sampled with SIFT on eight randomly selected frames, with frame sampling biased towards the beginning of the video.

**Evaluation.** Following [6], when evaluating CoTracker3 online on TAP-Vid, we add $5 \times 5$ points sampled on a regular grid and $8 \times 8$ points sampled on a local grid around the query point to provide context to the tracker. We do the same for the scaled offline version during inference. The Kubric-trained offline version, however, relies on uniform point sampling during training. For this model, during evaluation on TAP-Vid, we instead sample 1000 additional support points uniformly over time.

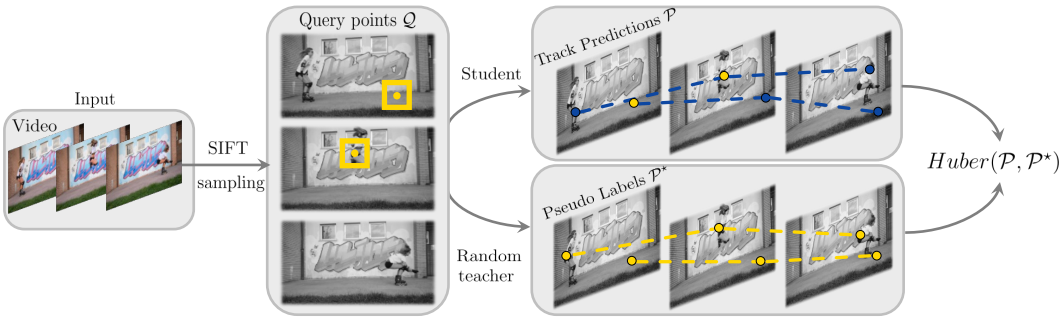


Figure 1. **Scaling pipeline**. Given a video, we randomly choose 8 frames and sample 384 query points across these frames using SIFT [9]. Then, we predict tracks for these query points with the student and randomly selected teacher models. Finally, we compute the difference between the predicted tracks and update the student model.

## B. Performance

In Fig. 2, we compare the speed of CoTracker3 with other point trackers. We measure the average time it takes for the method to process one frame, with the number of tracked points varying between 1 and 10,000. We average this across 20 videos of varying lengths from DAVIS. Even though CoTracker and CoTracker3 apply group attention between tracked points, the time complexity remains linear thanks to the proxy tokens introduced by [6]. While all the trackers exhibit linear time complexity with respect to the number of tracks, CoTracker3 is approximately 30% faster than LocoTrack [2], the fastest point tracker to date.

## C. Additional experiments

**Training with the average of teachers' predictions.** Interestingly, we found that aggregating the predictions of multiple teachers instead of using a random teacher does not improve performance, as shown in Tab. 1, whereas incorporating additional teachers into training consistently enhances the quality of our student model, as demonstrated in Tab. 4 of the main paper.

**Repeated scaling.** We study the effect of iterative scaling to investigate the limits of our multi-teacher scaling pipeline. Specifically, we scale CoTracker3 offline using our pipeline, where one of the teachers is the model itself. We then take this trained student model and attempt to improve it further by re-applying the same scaling pipeline, but with the original student model replaced by the newly trained student model as one of the teachers.

We find that this second round of scaling leads to slight improvements in performance metrics. This suggests that the student model has already distilled most of the knowledge from the other teachers during the initial training phase. We report the results in Tab. 2.

**Convergence behavior during scaling.** We examine the convergence behavior of our scaling pipeline by fixing the dataset and all hyperparameters, varying only the number of iterations over the dataset. We show in Tab. 3 that increasing the number of iterations leads to improved performance on TAP-Vid but with diminishing returns. Specifically, we

| Teacher selection strategy | Kinetics | DAVIS | RoboTAP | RGB-S |
|---|---|---|---|---|
| **Random** | **68.2** | **77.0** | **78.8** | **83.3** |
| Averaging | 67.4 | 76.5 | 77.9 | 82.4 |
| Median | 67.3 | 76.3 | 77.3 | 81.1 |

Table 1. **Supervision.** Random sampling of teachers consistently leads to better $\delta_{\text{avg}}$ on TAP-Vid compared to supervision with either the mean or the median of all teachers' predictions.

| Model | Average on TAP-Vid | | |
|---|---|---|---|
| | AJ ↑ | $\delta_{\text{avg}}$ ↑ | OA ↑ |
| Kub+15k | 64.0 | 76.8 | **90.2** |
| Kub+15k+15k | **64.2** | **76.9** | 89.7 |

Table 2. **Repeated scaling.** We scale CoTracker3 offline, then start from a scaled model, and scale it again with the scaled model as one of the teachers. Repeated scaling slightly improves tracking accuracy.

| Num. of iterations | Average on TAP-Vid | | |
|---|---|---|---|
| | AJ ↑ | $\delta_{\text{avg}}$ ↑ | OA ↑ |
| 1k | 63.3 | 75.6 | 87.5 |
| **15k** | 64.0 | 76.8 | **90.2** |
| 30k | 64.4 | 76.8 | 89.7 |
| 60k | **64.4** | **77.0** | 89.5 |

Table 3. **Longer training on 15k videos.** We train CoTracker3 offline for longer to determine the optimal number of iterations for a given number of videos. As a trade-off between training costs and the results obtained, we use the same number of iterations as the number of videos.

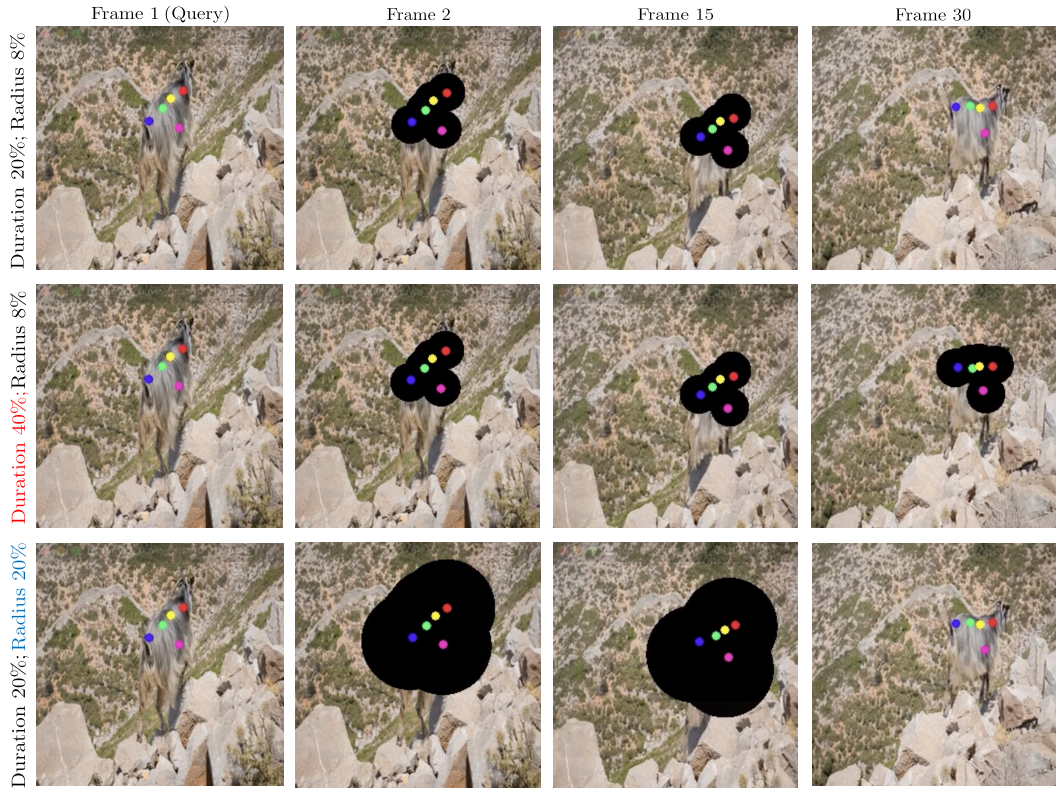


Figure 3. **Occlusions.** We occlude all tracked points with black circles of different sizes for several consecutive frames. We experiment with different scenarios, discussed in the text. For each, we also visualize the predicted positions of the tracked points.

| Radius (% of img. width) $\delta_{avg}$ ↑ | |
|---|---|
| 0 | 76.9 |
| 4 | 48.8 |
| 8 | 42.2 |
| 12 | 39.8 |
| 20 | 36.4 |
| 40 | 30.2 |
| 80 | 23.3 |
| 100 | 19.9 |

Table 4. **Varying size of occlusions.** We report tracking accuracy on DAVIS depending on the radius of artificially added occluding circles, which cover all tracked points for half of the video (30 frames on average, see Fig. 3).

| Duration (% of vid. len.) $\delta_{avg}$ ↑ | |
|---|---|
| 0 | 76.9 |
| 20 | 61.3 |
| 40 | 48.2 |
| 60 | 37.5 |
| 80 | 29.9 |
| 100 | 21.1 |

Table 5. **Varying duration of occlusions.** We report tracking accuracy on DAVIS depending on the duration of artificially added occluding circles with radius of 8% of the image width. Occluders cover all the tracked points, see Fig. 3.

observe a saturation point beyond which further increases in the number of training iterations do not yield significant improvements in model quality. We thus use the same number of iterations as the number of training videos with a batch size of 32, iterating over each video 32 times.

**Occlusions.** We investigate the effect of occlusions of different sizes and lengths on the tracking accuracy on TAP-Vid DAVIS. Specifically, we occlude all the tracked points with black circles of different sizes for several consecutive frames (see Fig. 3). We then measure how this affects the tracking accuracy using offline tracking in various scenarios, which is discussed next.

First, we show that CoTracker3 can successfully utilize temporal context to track points through occlusions. To do so, we occlude all tracked points in each video for half of the video length, starting right after the query frame, but let the points be visible in the second half. The occluding circle is centered on the ground truth track. We vary the radius of the occluding circle and increase it from 0% to 100% of the video width. As Tab. 4 shows, the tracking accuracy is still 19.9%, even with a radius of 100%; this is because the model sees the second half of the video and can track points there. If we occlude all frames rather than half with a radius of 100%, the accuracy drops to 2%.

Second, we show that CoTracker3 can also successfully utilize the spatial context given by other tracked points. To do so, we fix the radius of the occlusion to 8% of the image width and vary the duration of the occlusion from 0 to 100% of the video length, with the average video length being 60

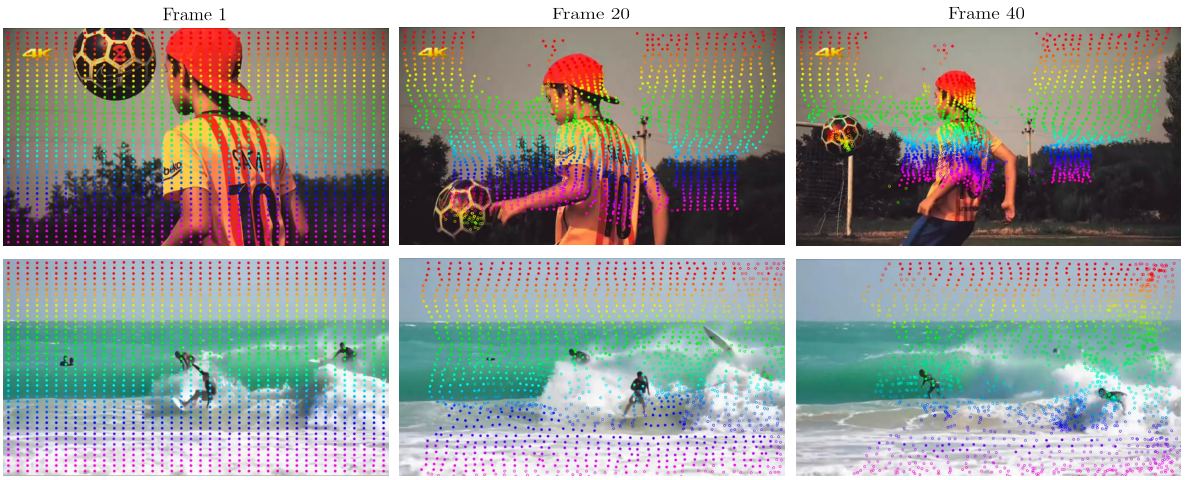


Figure 4. **Failure cases.** Featureless surfaces is a common mode of failure: the model cannot track points sampled in the sky or on the surface of water.

frames. In Tab. 5, short occlusions of 20% of video length (12 frames on average) affect performance, but not significantly: accuracy drops from 76.9% to 61.3%. When occluding points for the whole duration of the video (100%), the model can still approximate the location of these points due to the presence of unoccluded support points (21.1% accuracy vs 2% when all points are occluded).

## D. Failure cases

In Fig. 4, we show examples of failure cases. We track a grid of $40 \times 40$ points from the first frame and demonstrate that the model cannot reliably track points sampled in the sky or on the surface of water, partly because the task is ambiguous in these cases: it is unclear whether the tracked point in the sky should remain static or move with the camera. Other common sources of failure are tracking shadows of objects and tracking through long occlusions.

## E. Limitations

A key limitation of our pseudo-labeling pipeline is its reliance on the quality and diversity of teacher models. The observed saturation in performance on TAP-Vid during scaling suggests that the student model absorbs knowledge from all the teachers and, after a certain point, struggles to improve further. Thus, we need stronger or more diverse teacher models to achieve additional gains for the student model.

## References

[1] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *Icml*, page 4, 2021. 1

[2] Seokju Cho, Jiahui Huang, Jisu Nam, Honggyu An, Seungryong Kim, and Joon-Young Lee. Local all-pair correspondence for point tracking. *Proc. ECCV*, 2024. 2

[3] Carl Doersch, Ankush Gupta, Larisa Markeeva, Adrià Recasens, Lucas Smaira, Yusuf Aytar, João Carreira, Andrew Zisserman, and Yi Yang. Tap-vid: A benchmark for tracking any point in a video. *arXiv*, 2022. 1

[4] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 2019. 1

[5] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, et al. Kubric: A scalable dataset generator. In *Proc. CVPR*, 2022. 1

[6] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker: It is better to track together. *Proc. ECCV*, 2024. 1, 2

[7] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020. 1

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 1

[9] David G Lowe. Object recognition from local scale-invariant features. In *Proc. ICCV*, 1999. 2