CoWTracker: Tracking by Warping instead of Correlation

Supplementary Material

6. Implementation Details

6.1. Pseudo-code of CoWTracker

In Figure 8, we provide a minimal PyTorch-style pseudocode of the proposed WarpTracker update loop. First, we extract features for all video frames with a standard backbone and replicate the anchor-frame features for all queries. We then initialize the track field and the hidden state once. After this setup, the algorithm performs a very simple iterative update: at each of the K iterations, we (i) warp the features according to the current tracks, (ii) concatenate the original features, warped features, tracks, and hidden state, (iii) update the hidden state, and (iv) refine the tracks with a small linear head. Importantly, no correlation or cost volume is constructed at any point in this loop; instead, we operate directly on the raw backbone features of each frame. This design makes the method both easy to implement and straightforward to integrate into existing codebases. Please refer to the main paper for a detailed description of each component.

```
# V [T, H, W, C]   - video frames (T = # of frames)
# K                - number of update iterations

# extract features for all frames
F = feature_encoder(V)            # [T, D, H', W']
F_query = F[0:1].repeat(T)        # [T, D, H', W']

# initialize tracks and hidden state
tracks = zeros_like(F[:, :2])     # [T, 2, H', W']
h = init_hidden(F)                # [T, D, H', W']

for _ in range(K):
   # warp features using current tracks
   F_warp = warp(F,tracks)        # [T, D, H', W']

   # combine features and state  [T, 3D+2, H', W']
   x = concat(F, F_warp,tracks, h)

   # update hidden state
   h = update_hidden(x)           # [T, D, H', W']

   # update tracks with a linear head
   tracks += track_head(h)        # [T, 2, H', W']
```

Figure 8. **Pseudocode for CoWTracker algorithm:** our model iteratively refines track prediction across all frames using repeated warping and updates, and as shown, is also simple to implement.

7. Extended Quantitative Results

In this section, we provide expanded quantitative comparisons that complement the results presented in the main paper.

7.1. Extended Evaluation on the AllTracker Benchmark Suite

In addition to the datasets considered in the main paper, we further evaluate our method on the extended benchmark suite introduced by AllTracker [14]. This suite augments standard point-tracking benchmarks with three additional datasets (DriveTrack, EgoPoints, and Horse10) covering challenging driving scenes, egocentric videos, and articulated animal motion, respectively. We follow exactly the same evaluation protocol and dataset splits as AllTracker, and report the δ_{avg} metric averaged over all sequences.

Table 4 compares CoWTracker with recent sparse trackers, dense trackers, and optical-flow models. Our method achieves the best overall performance on this suite, obtaining an average δ_{avg} of 73.6, which improves upon the strongest AllTracker by +2.3 points. CoWTracker is best or tied-best on all individual datasets, including the three newly added benchmarks, demonstrating that CoWTracker generalizes well across diverse motion patterns and scene types.

Method	Data	Dav.	Dri.	Ego.	Hor.	Kin.	Rgb.	Rob.	Avg.
*Optical Flow Models**									
AccFlow [46]	Flow	23.5	26.4	4.0	12.1	38.8	63.2	57.9	32.3
RAFT [37]	Flow	48.5	44.8	41.0	27.8	64.3	82.8	72.2	54.5
SEA-RAFT [44]	Flow	48.7	49.4	44.0	33.1	64.3	85.7	67.6	56.1
Sparse Trackers									
PIPs++ [13]	PO	62.5	51.3	38.5	21.4	64.2	70.4	73.4	54.5
CoTracker2 [18]	Kub	70.9	67.8	43.2	33.9	65.8	73.4	73.0	61.1
LocoTrack [6]	Kub	68.0	66.5	58.4	48.9	70.0	80.3	76.9	67.0
BootsTAPIR [9]	Kub+	67.9	66.9	56.8	48.8	70.6	81.0	78.2	67.2
CoTracker3-Kub [17]	Kub	77.4	69.8	58.0	47.5	70.6	83.4	77.2	69.1
CoTracker3 [17]	Kub+	77.1	69.8	60.4	47.1	71.8	84.2	81.6	70.3
Dense Trackers									
DELTA [29]	Kub	75.3	67.8	40.3	41.8	66.5	83.0	74.8	64.2
AllTracker-Kub [14]	Kub	75.2	66.1	60.3	49.0	71.3	90.1	82.2	70.6
AllTracker [14]	Kub+	76.3	65.8	62.5	49.0	72.3	90.0	**83.4**	71.3
CoWTracker	Kub	**78.0**	**70.7**	**64.4**	**52.7**	**73.1**	**92.8**	**83.4**	**73.6**

Table 4. **Comparison against state-of-the-art point trackers and optical-flow baselines (incl. extra eval datasets used in AllTracker [14]).** Baseline numbers are taken from [14]. *Optical-flow models are assessed *zero-shot* on the tracking task without any task-specific fine-tuning, and understandably yield lower results.

Input Frames	2*	10	20	40	60	80	100	200
All	0.07	0.27	0.57	1.37	2.42	3.75	5.35	10.9
- Backbone *(VGGT)*	0.02	0.10	0.25	0.76	1.51	2.54	3.85	7.89
- Tracker	0.05	0.17	0.32	0.61	0.91	1.21	1.50	3.01
Point/sec ($\times 10^6$)	5.37	6.96	6.60	5.49	4.66	4.01	3.51	3.45
Frame/sec	28.6	37.0	35.1	29.2	24.8	21.3	18.7	18.4

Table 5. **CoWTracker runtime vs. video length.** We report *Runtime (in seconds, all and by components, ↓ better)* and *Point / Frame per Second (↑ better)*. *equivalent to optical flow runtime.

7.2. Runtime Analysis

We benchmark the runtime of CoWTracker on a single NVIDIA H100 GPU with FlashAttention-3 [33]. For all results we use $K{=}5$ refinement steps, and input resolution 336×560.

Table 5 reports end-to-end runtime as a function of video length, together with a breakdown into backbone and tracker cost, as well as the resulting point and frame throughput. The tracker itself scales approximately linearly with the number of frames and contributes a smaller fraction of the total runtime for longer sequences. For shorter clips (≤ 40 frames) CoWTracker tracks around 6–7 million points per second and runs at over 30 frames per second. For longer videos throughput gradually decreases but remains around 3.5 million points per second and nearly 20 frames per second.

The main runtime growth for long sequences stems from the VGGT backbone, whose complexity scales quadratically with video length. We note that (i) the backbone is essentially orthogonal to our tracker, meaning that any future advances in backbone architecture—including more efficient designs—will directly improve end-to-end speed, and (ii) because the backbone is a general-purpose 3D representation model, WarpTracker effectively acts as a lightweight add-on that can be plugged into existing architectures used for other tasks, delivering the new functionality with very little extra runtime. In practice, one can also process long videos in shorter chunks to avoid the quadratic cost of VGGT, trading only about 1% in δ accuracy for a substantial runtime reduction.

7.3. Memory Analysis

Figure 9 reports the memory required to store either a cost volume or a pair of feature maps (the alternative used by our cost-volume-free warping based head). Cost volumes grow roughly $16\times$ in memory whenever the image resolution doubles, because both the number of spatial locations and the search window increase. As a result, using stride $1/2$ features becomes prohibitively expensive (hundreds of gigabytes in our setting), and even stride $1/4$ is costly. This is why most prior dense trackers operate at stride $1/8$, trading away high-frequency details and often degrading tracking accuracy, especially around thin structures and object boundaries.

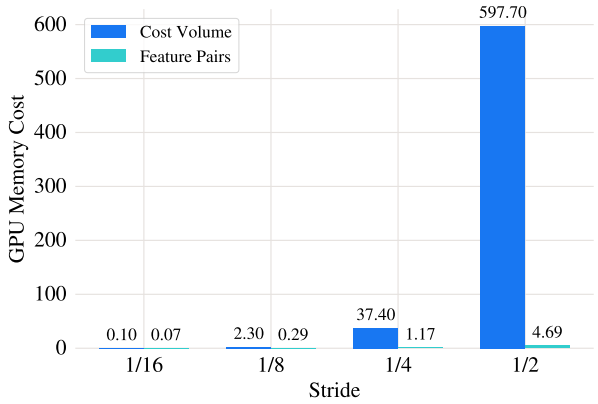

Figure 9. **Cost volumes are prohibitively expensive at high resolution.** Memory grows by $\sim 16\times$ whenever the image size doubles, forcing most methods to use stride 1/8 and sacrifice accuracy. Compared to cost volumes, feature pairs are far more memory efficient. Numbers computed using 100-frame video length, 384×512 resolution, and 16-bit precision.

Our method avoids cost volumes entirely and operates directly on feature maps using a warping-based head. This design removes the quadratic dependence on the spatial search window and yields much more favorable memory scaling with resolution. As shown in Fig. 9, our head can run comfortably on stride $1/2$ features, which only takes a few gigabytes of memory to store, enabling high-resolution tracking without sacrificing practicality.

8. Extended Qualitative Results

In this section, we provide additional qualitative visualizations to further demonstrate the robustness and generalization capabilities of our proposed method.

8.1. Comparison to Existing Methods

We first evaluate our model on highly challenging sequences characterized by non-rigid motion and severe occlusions and compare with existing methods. As illustrated in Fig. 10, the top row ("Dive-in") presents a scenario where a person enters the water, causing significant non-rigid deformation and drastic appearance changes. While baseline methods like DELTA and AllTracker lose the target or drift significantly, our method maintains a consistent track.

Furthermore, the bottom row of Fig. 10 highlights a scenario with rapid camera motion where the target object frequently exits the camera frame. This sequence also necessitates tracking a small object (a drone). Our method successfully handles these re-entry scenarios and accurately tracks

Figure 10. **Qualitative comparison on challenging sequences involving non-rigid water motion and small object tracking.** Columns compare DELTA, AllTracker, and Ours. The top sequence exhibits large non-rigid motion and drastic appearance changes as the person dives into the water. The bottom sequence features large camera motion where objects repeatedly go out of frame, requiring the tracking of a small object. Our method maintains a consistent track in these difficult conditions, whereas DELTA and AllTracker fail to recover or exhibit drift. Notably, our approach successfully tracks the small object in the bottom sequence, thanks to our high-resolution feature maps that preserve fine details. Numbers in lower-right boxes indicate frame numbers.

the small target, a capability we attribute to our utilization of high-resolution feature maps which preserve fine-grained spatial details.

8.2. Generalization Across Domains

To assess the universality of our tracking model, we test CoWTracker on videos from three distinct domains without fine-tuning. Figure 11 displays qualitative results on:

- **Ego-centric videos** (top row), where camera motion is erratic and hand-object interaction is frequent.
- **Self-driving scenes** (middle row), involving dynamic street environments.
- **Robotics manipulation** (bottom row), requiring precise tracking of manipulated objects.

Notably, our method exhibits strong temporal stability, successfully maintaining accurate tracks over very long sequences (up to 600 frames), as demonstrated in the ego-centric and robotics examples.

8.3. Zero-Shot Optical Flow Estimation

Although our architecture is designed for tracking videos, it naturally produces optical flow by treating an image pair as a 2-frame "video". In the main paper, we presented results on the Sintel dataset. Here, in Fig. 12, we extend this analysis to the Spring [26] (top two rows) and KITTI [12] (bottom two rows) datasets.

Qualitatively, our predicted flow exhibits sharp motion boundaries and accurate alignment with the ground truth. Crucially, we emphasize that our model is **not** trained on any optical flow datasets (including Sintel, Spring, or

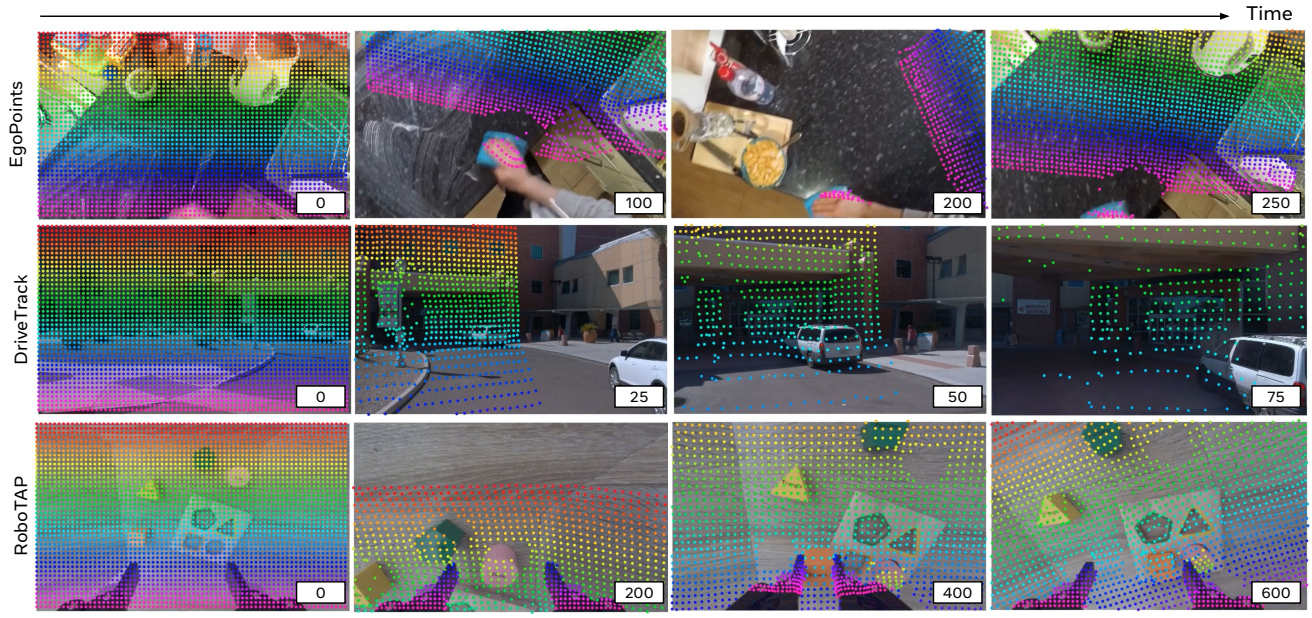

Figure 11. **Generalization to diverse video domains.** Our model demonstrates high tracking accuracy across a variety of domains, including ego-centric videos (top), self-driving scenes (middle), and robotics manipulation tasks (bottom). Notably, our approach is robust over long temporal extents, successfully maintaining tracks in sequences as long as 600 frames (see bottom row).

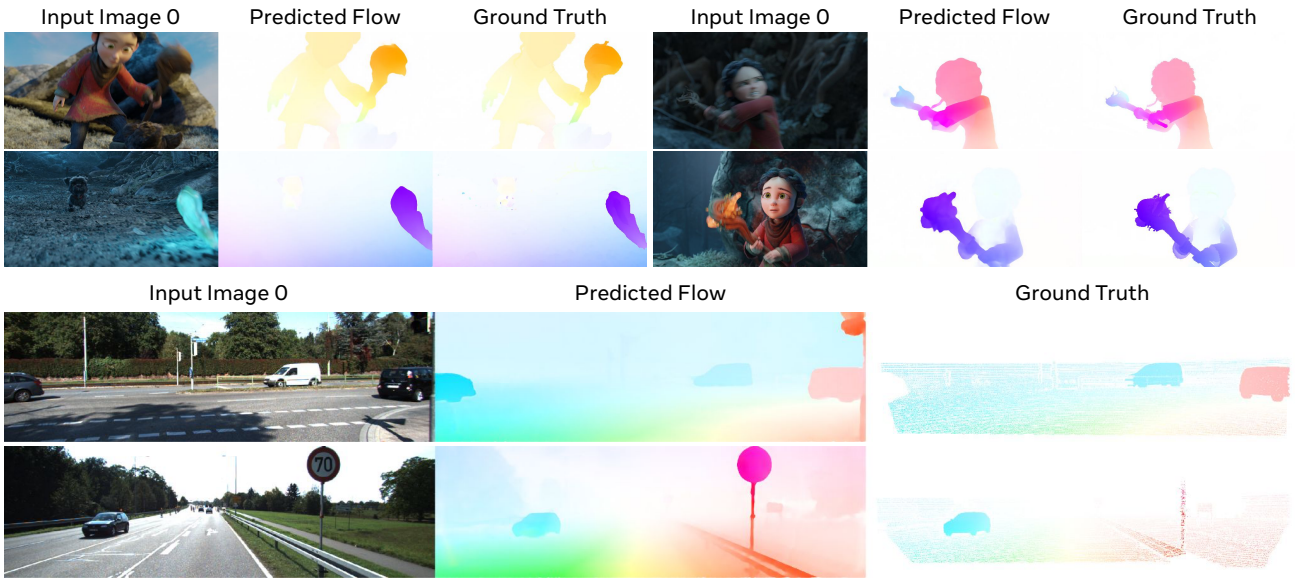

Figure 12. **Additional qualitative optical flow results.** Complementing the Sintel results in the main paper, we visualize performance on the Spring dataset (top two rows) and the KITTI dataset (bottom two rows). Our model predicts high-quality optical flow with sharp motion boundaries and accurate dense correspondence that closely matches the ground truth. It is important to note that we employ the exact same tracking model used throughout the paper; it was *not* trained on any optical flow datasets, including Spring or KITTI, demonstrating its strong generalization capability.

KITTI). These results are achieved using the exact same tracking weights, highlighting the generality of motion understanding ability embedded in our model.

8.4. Supplementary Video

To provide a better visualization of the tracking performance, we include a supplementary video (https://youtu.be/QQP8TZPMZMw). This video demonstrates

the accuracy and robustness of our method in dynamic scenarios that are difficult to fully appreciate through static images.

9. Limitations

CoWTracker has several limitations that suggest future research directions. Our method may fail under extreme viewpoint changes, long-range full occlusions, or severe specularities. While the iterative warping head is lightweight, overall throughput is heavily dependent on the chosen backbone, such as VGGT. Nonetheless, the marginal overhead of CoWTracker remains small, and the benefits of higher-resolution feature indexing persist even when using more efficient backbones. Another constraint is the quadratic complexity of the VGGT backbone with respect to video length, which requires processing longer clips in chunks. Furthermore, the current iterative refinement process tends to saturate after five to six steps, indicating a ceiling on the model's self-correction capability. Finally, the model is currently trained exclusively on synthetic Kubric data. Diverse real-world data remains significantly underleveraged; incorporating natural videos could improve robustness to lighting and noise patterns that are difficult to simulate.