Supplementary Material

A. Additional model details

Recurrent module In the recurrent module, we compress and contextualize the input data in stages [54], following the design ideas of SEA-RAFT [54], as illustrated in Figure 5. We use parallel 2-layer CNNs (with 3×3 kernels) on the correlation field and the motion field, then concatenate these feature maps, and merge them with a 1×1 convolution. We then concatenate the visibility map, confidence map, and appearance features, and merge them to 256 channels with another 1×1 convolution. We then apply three "space-time" blocks, where the spatial part is a 2D ConvNeXt block (with a 7×7 kernel) and the temporal part is a *pixel-aligned* transformer block (attending the full subsequence span S). By pixel-aligned, we mean that attention only happens along the temporal axis (i.e., with cost quadratic in $S = 16$), and this is done for every pixel in parallel. We note that an important convenience of our design is that all of the tensors in this stage are aligned with the "query" frame; therefore pixel-aligned attention is attention between *corresponding* pixels. After the interleaved spatial and temporal blocks propagate tracking-related information across our window, we emit a new hidden state, and decode this state into explicit revisions for visibility, confidence, and motion. Following SEA-RAFT [54], we only use half of the feature channels for our recurrent module's hidden state, as shown by the "split" step in Figure 5, which saves memory (reducing the number of output channels from 260 to 132) and may also stabilize recurrence.

Model layers The ConvNeXt blocks are standard: layer-scaled kernel-7 grouped convolution → layer norm → expansion linear layer (factor 4) → GELU → reduction linear layer (factor 4) → residual add → linear layer. The pixel-aligned temporal blocks also have a standard form: layer-scaled transformer block with 8 heads and expansion factor 4 → residual add → linear layer. To inform the attention layers on frame ordering, we add 1D sinusoidal position embeddings to the "context" features of a subsequence (see Figure 3 from the main paper), broadcasting these embeddings across the spatial axes. Note that our temporal position embedding is with respect to a subsequence; the model unaware of total video length, and we give no information about the anchor frame's original position in the timeline.

AllTracker-Tiny For AllTracker-Tiny, we use a BasicEncoder backbone with channel dimension 128, making only 2.63M parameters (out of 6.29M total), and leave the rest of the architecture unchanged. We use this model's featuremap as both the "context" features and the RNN hidden state initialization, rather than splitting a 256-channel featuremap into these two parts.

Initialization strategy A close comparison of our model architecture versus SEA-RAFT [54] will reveal that we do not follow SEA-RAFT's choice to directly regress an "initial" optical flow estimate with a secondary CNN. Our main reason for omitting this is to save memory. We also note that when frame gaps are sufficiently large, it may in fact be impossible to estimate optical flow without temporal context (e.g., out-of-bounds motion of 200 pixels vs. 300 pixels appears identical), and therefore the flow regression from SEA-RAFT would not likely be as effective as simply propagating the estimates from the previous window.

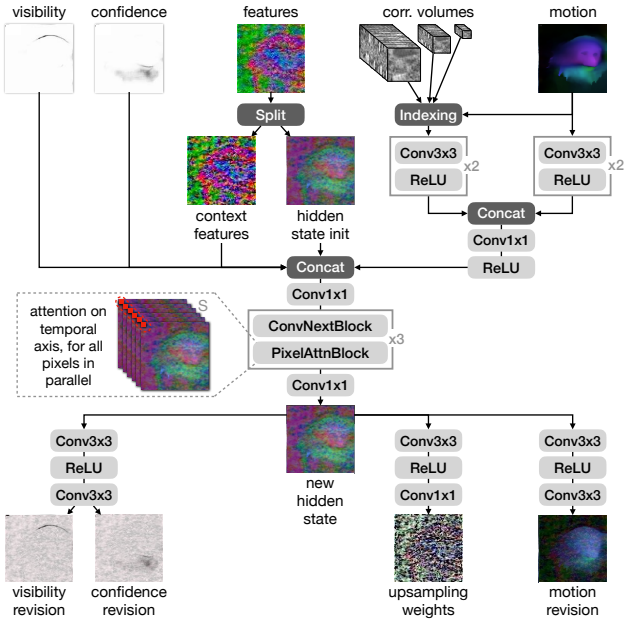

Figure 5. **Detailed view of iterative refinement block.** We consolidate data from visibility, confidence, correlation, motion, and appearance features into a single feature map, then interleave convolutional spatial blocks and pixel-aligned temporal blocks, and output revisions to the features, visibility, confidence, and motion. This refinement process is iterated 4 times (with shared weights).

B. Additional training details

We train with mixed precision in PyTorch (bfloat16).

From point tracking datasets, we use samples which have anywhere upwards of 256 valid annotated tracks (after augmentations), and trim to a maximum of 6144 tracks to keep memory usage predictable. From optical flow datasets we use dense supervision, but note that this data does not include visibility labels.

The major variables in speed and memory consumption are batch size, video length, input resolution, and number of refinement steps. To match inference speed across inputs of different length and keep memory consumption within our budget (8x A100 40G), we use the following settings: on optical flow data (where video length is 2), we use batch size

Table 8. Comparison against CoTracker3 with different evaluation protocols. We evaluate δ_{avg} (higher is better), using an input resolution of 384×512. The benchmarks are BADJA [3], CroHD [46], TAPVid-DAVIS [11], DriveTrack [1], EgoPoints [10], Horse10 [33], TAPVid-Kinetics [11], RGB-Stacking [28], and RoboTAP [52]. "CoTracker3*" indicates values reported in the CoTracker3 paper under a more expensive evaluation protocol (and on fewer datasets). Parameter counts are in millions.

Method	Params.	Training	Bad.	Cro.	Dav.	Dri.	Ego.	Hor.	Kin.	Rgb.	Rob.	Avg.
CoTracker3-Kub [24]	25.39	Kubric	47.5	**48.9**	**77.4**	**69.8**	58.0	47.5	70.6	83.4	77.2	64.5
CoTracker3*-Kub [24]	25.39	Kubric	-	-	76.7	-	-	-	66.6	81.9	73.7	-
CoTracker3 [24]	25.39	Kubric+15k	48.3	44.5	77.1	**69.8**	60.4	47.1	71.8	84.2	81.6	65.0
CoTracker3* [24]	25.39	Kubric+15k	-	-	76.3	-	-	-	68.5	83.6	78.8	-
AllTracker	16.48	Kubric+mix	**51.5**	44.0	76.3	65.8	**62.5**	**49.0**	**72.3**	**90.0**	**83.4**	**66.1**

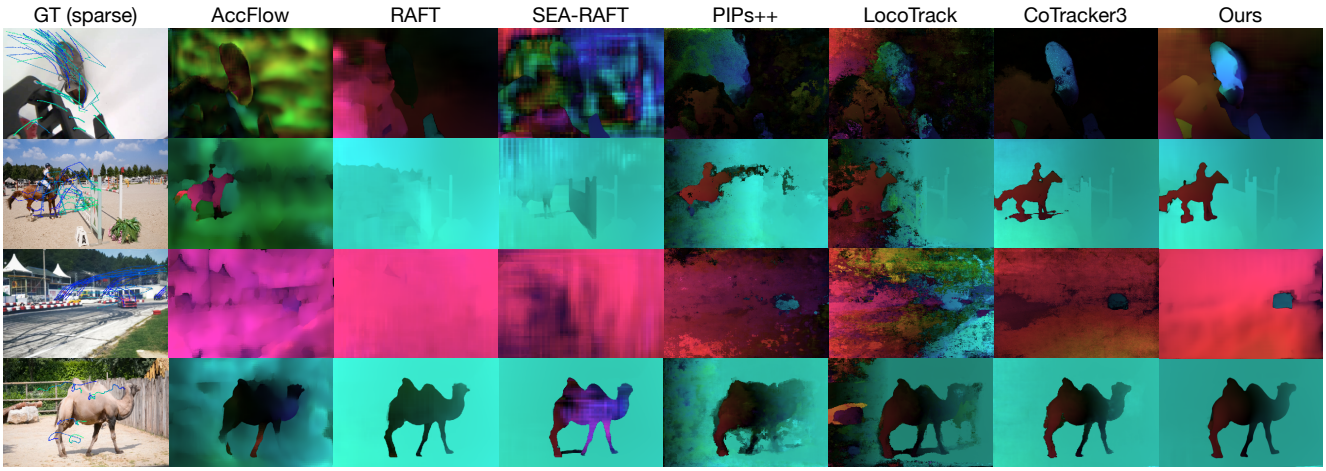

Figure 6. **Visualization of dense correspondence maps produced by all models.** On the far left column we show the ground truth trajectories overlaid on the first frame of the input video, with blue-to-green colormap. (Note that a ground truth flow map does not exist in this data.) The flow maps in the other columns show the estimated correspondence field from the first frame of a video to the last frame of the video. Note that RAFT and SEA-RAFT only make use of the first and last frames, while other methods use the intermediate frames as well.

8, resolution 384×768, and 4 refinement steps; on videos of length 24, we use batch size 1, resolution 384×512, and 4 refinement steps; on videos of length 56, we use batch size 1, resolution 256×384, and 3 refinement steps. In the first stage of training (on Kubric alone), we use only videos of length 24 and 56, and split our 8 GPUs by using 4 for the 24-frame videos and 4 for 56-frame videos. In the second stage of training (on the wider mix of data), we split our 8 GPUs by using 1 for optical flow, 3 for videos of length 24, and 4 for videos of length 56. We sync gradients across GPUs after each backward pass.

For our BCE loss, we apply the sigmoid first and then use the direct BCE loss, which we (counter-intuitively) found to be more numerically stable than BCE with logits.

We note that *no architecture modifications are required* to train jointly for optical flow estimation and point tracking. In optical flow, the temporal attention is a redundant operation, but we do not disable the temporal transformer, as there are still MLP layers within it which participate in the processing.

C. Additional baseline details

As mentioned in the main paper, the performance of CoTracker-style models depends how the query points are grouped. Intuitively, if multiple queries lie on the same object, they will be tracked more accurately. When these methods are tasked with tracking all of the benchmark queries at once, they tend to exploit a bias in the data and perform better than they would perform on random queries. The authors of these methods suggest a strategy for mitigating this effect, which consists of running each query in a separate pass, while also adding "support" points around the query and a sparse grid covering the image. In our high-resolution multi-benchmark evaluation, these steps would be prohibitively expensive (e.g., weeks). We therefore simply give the models the advantage of the data bias: we give all queries at once, and supplement them with a sparse grid of points around the image. We compare this evaluation protocol to author-reported results in Table 8. We find that our cheaper protocol over-estimates the accuracy of CoTracker3, but our own model is still more accurate on average.

Table 9. Optical flow end-point error ("EPE-All") in the offical SINTEL test benchmark.

Model	Clean	Final
SEA-RAFT [54]	**1.309**	**2.601**
RAFT [47]	1.609	2.855
GMFlow [57]	1.736	2.902
PWC-Net [42]	4.386	5.042
AllTracker	1.673	3.244

Table 10. Optical flow end-point error in the CVO "Final" (T=7) and "Extended" (T=48) test sets, for visible/occluded pixels.

Model	T=7	T=48
AccFlow [55]	1.15 / 4.63	28.1 / 52.9
DOT [27]	**0.84 / 4.05**	3.71 / **7.58**
AllTracker	1.03 / 4.10	**3.41** / 7.93

D. Additional optical flow results

We ran AllTracker on the official SINTEL test benchmark, yielding the scores shown in Table 9, with other official scores included for comparison. We note that it is common in optical flow literature to produce a different model for each benchmark (finetuned with a particular data mix and resolution), but we simply use our original checkpoint. AllTracker's optical flow is not as accurate as SEA-RAFT, but comparable to RAFT or GMFlow. Qualitatively, AllTracker's flow maps appear to be coarser than the ones from SEA-RAFT [54], suggesting that the model is underfitting; better models might be obtained with greater compute.

We additionally evaluate in CVO, the multi-frame optical flow dataset used by AccFlow and DOT, showing results in Table 10. We find that on short sequences DOT (combining CoTracker2 and RAFT) performs best; on long sequences AllTracker performs best. We also find that AllTracker is 3x the speed of DOT.

In sum, these results suggest that AllTracker is not state-of-the-art for optical flow estimation, even though it includes optical flow data in its training. Attaining top performance optical flow and point tracking with a single model remains an open challenge.

E. Additional ablation details

Validation dataset As mentioned in the main paper, we construct a validation dataset for our ablation studies, using BADJA [3], CroHD [46], TAPVid-Davis [11], DriveTrack [1], Horse10 [33], and RoboTAP [52]. The purpose of these studies is to obtain a quick (but reliable) look at performance, and therefore we do not use these datasets in their entirety, and note we also exclude some of our available datasets. We subsample from these datasets by (1) selecting the first frame with any annotations and tracking only the queries on that frame, (2) trimming all videos to a maximum length of 300 frames. These choices, along with our truncated training regime (training only 100,000 steps and only on Kubric) allows for most ablation experiments to (individually) start and finish within 24 hours.

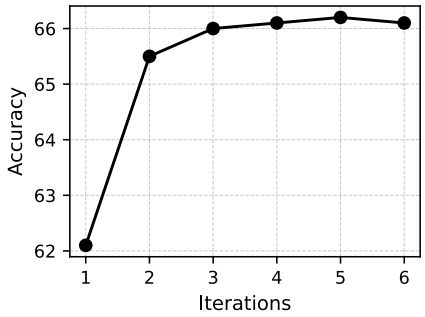

Figure 7. Accuracy over inference steps. Accuracy rises quickly then plateaus. In the main evaluation we use 4 iterations.

Inference steps In the main paper we apply the recurrent refinement module 4 times. Figure 7 shows performance at different iterations, evaluating δ_{avg} over all datasets and averaging, using an input resolution of 384×512. On the first step the model achieves 62.1 accuracy, which already outperforms most state-of-the-art models. Accuracy rises to its peak at 5 iterations, then begins to drop. In the main paper we report results at 4 iterations, because we find that step 4 and step 5 produce similar accuracy at higher resolutions.

F. Additional qualitative results

To obtain long-range flow estimates from our point tracker baselines, we query them to track every pixel of the first frame of the video. We perform these queries in "batches" of 10,000, which is the maximum that fits on our GPU. Note that CoTracker3 benefits from processing these jointly, whereas PIPs++ and LocoTrack do not, due to the design of these models. To obtain long-range flow estimates from the optical flow baselines, we pair the first frame with every other frame, creating $T-1$ frame pairs for flow estimation, where T is the length of the video.

We show additional visualizations of multiple models' dense outputs in Figure 6. We notice striking dissimilarity across the outputs of the methods, attesting to the difficulty of the task, and to the usefulness of visualizing point tracks as flow maps. We find that when the foreground displacements are large, the flow models often "give up" on the dynamic foreground and produce a motion field that only describes the background. We also find that PIPs++ and LocoTrack often struggle with spatial smoothness, while the flow models do not. CoTracker3 occasionally fails on smoothness too (see row 3 with the car), but less so. Our model appears to produce results that are smooth and accurate, which matches intuitions for a model that blends the 2D processing of flow models with the temporal coherence of point trackers.