Live4D: Latent Tracking for Streaming 4D Reconstruction

Minsik Jeon Jay Karhade Deva Ramanan† Shubham Tulsiani†
Carnegie Mellon University



Figure 1. **Live4D tracks any point in 3D from streaming video.** In streaming settings, 3D tracks must be predicted as each frame arrives. However, existing window-based 3D trackers incur high latency in streaming settings. Live4D efficiently predicts dense 3D tracks by decoding queries from the causal memory of a streaming 3D reconstruction model, supporting new queries at any time and tracking through occlusions.

Abstract

We present Live4D, a method for dense 3D point tracking in streaming videos, i.e.*, as each frame arrives, it immediately predicts the current 3D position of every queried pixel, using only the frames observed so far. Our key insight is that the causal memory maintained by streaming 3D reconstruction models can serve as the scene representation for query-based tracking* i.e. *each query attends to this memory to predict its 3D position for every incoming frame. To make dense tracking efficient, we propose hierarchical decoding, where only a coarse set of patch-level 'latent tracks' is computed using the memory, and each pixel query is decoded from its nearby latents. Finally, as queries can be added at any time, we can re-initialize a drifting 3D track by simply adding a new query at the current location of the point, obtained from an off-the-shelf streaming 2D tracker. We find that Live4D, despite not having access to future frames, is competitive with window-based 3D trackers, and improves over lifting streaming 2D tracks while also tracking occluded points. Moreover, hierarchical decoding is 5× faster than directly decoding 10k pixel queries using the memory.*

†Equal advising

1. Introduction

Imagine standing on the sidewalk and watching the street scene in Fig. 1 unfold. As the motorcyclist rides past, we effortlessly perceive where they are in 3D and how they move, and retain a sense of this even when they pass behind the tree, well before they re-emerge. When a car enters the scene, we begin to follow it as well. Our understanding is thus always 'live' *i.e*. it builds on what we have seen so far, and is updated as we continue to watch. In this work, our goal is to build a computational system that can similarly track points in 3D given a streaming video. Specifically, we tackle the task of *dense streaming 3D tracking* – as each frame of a video arrives, we seek to immediately infer, using only the frames observed so far, the current 3D position of every queried pixel (*e.g*. all pixels in a previous frame), while allowing new queries to be added at any time.

We build on recent progress in streaming 3D reconstruction and query-based 3D tracking. Streaming reconstruction methods [3, 33, 42] process a video one frame at a time, and maintain a compact 'memory' of past observations to predict the camera pose and depth for each incoming frame. While these methods allow efficient inference over long videos, they do not track points over time, and combining them with a streaming 2D tracker [13, 41] (by lifting its output to 3D) cannot recover points that are occluded or out of view. In parallel, recent query-based methods [8, 11, 39] have shown impressive results for 3D tracking by encoding a video into a scene representation, and letting each point of interest (a 'query') attend to it to predict its 3D position over time. However, these methods jointly encode a window of frames, which would need to be repeated for every incoming frame. Moreover, as each query attends to the entire representation, dense tracking is expensive.

We present Live4D, a streaming 3D tracker built on a streaming 3D reconstruction model. Our key insight is that the causal memory maintained by streaming reconstruction models can serve as the scene representation for query-based tracking. Specifically, we let each pixel query attend to this memory, as well as its own past states, to predict its 3D position for every incoming frame. This allows Live4D to produce 'pixel tracks' that persist through occlusions, and to add new queries at any time. To make dense tracking efficient, we further propose *hierarchical* decoding *i.e*. we only compute a coarse set of patch-level 'latent tracks' using the memory, and decode each pixel query from its nearby latents. Finally, we note that the predicted 3D tracks are complementary to lifted 2D tracks *i.e*. the former persist through occlusions but can drift over long videos, while the latter are accurate but limited to visible points. We therefore run an off-the-shelf streaming 2D tracker [13] at inference and, as queries can be added at any time, re-initialize a drifting 3D track by simply adding a new query at the current pixel location of the (visible) point.

We evaluate Live4D on 3D tracking for short and long videos using the WorldTrack [7] and a derived LongTrack benchmark. We find that Live4D, despite not having access to future frames, is competitive with window-based 3D trackers [8, 10, 11, 20, 36, 38], while predicting tracks at 10 FPS for videos with over 300 frames. We also show that Live4D improves over lifting streaming 2D tracks, while additionally tracking occluded and out-of-view points, and that the re-initialization is essential for this improvement. Moreover, our hierarchical decoding via latent tracking yields similar tracking performance as directly decoding pixel queries, while being 5x faster for $10k$ pixel queries. More broadly, we believe our work shows that latent tracking over streaming reconstruction models can serve as a basis for dense and low-latency 3D tracking.

2. Related Work

Streaming 2D Point Tracking. Tracking any point (TAP) [9] aims to estimate the 2D trajectory of arbitrary query pixels across a video. Conventionally, 2D trackers are bi-directional and require access to future frames. These trackers refine trajectories jointly over the whole video or over sliding windows of frames [5, 9, 14], and thus are limited to offline inference. For online inference, early online variants of TAPIR and BootsTAPIR [5, 6] are instead retrained with temporally causal attention, while MFT [24] chains optical flow over multiple time spans to track every pixel online. More recently, Track-On2 [1] keeps a memory of past query and context features to localize each point frame by frame, and TAPNext [41] casts tracking as next-token prediction with a causal, recurrent video backbone, a design retained by its successor TAPNext++ [13]. While these trackers are accurate on visible points, they operate in the image plane, so lifting them with depth cannot recover points that are occluded or out of view.

Streaming 3D Reconstruction. SLAM systems [2, 22, 29] estimate camera poses and geometry incrementally from a video stream. However, these systems rely on hand-crafted components and bundle adjustment. Feed-forward reconstruction models such as DUSt3R [34], VGGT [31], and their successors [16, 18, 32] directly regress point maps and cameras from images, but encode all frames jointly and are therefore offline. Recently, several works pair them with SLAM back-ends [4, 21, 23], reintroducing optimization and hand-designed modules. Purely feed-forward streaming methods such as Spann3R [30], CUT3R [33], and StreamVGGT [42] instead process one frame at a time while maintaining a memory of past observations. More recent works also explore diverse forms of memory including test-time training [12], hybrid memory [40], or bounded caches [37]. LingBot-Map [3] keeps the full tokens of a few anchor frames and a local window, together with compact context tokens for all other frames, enabling efficient

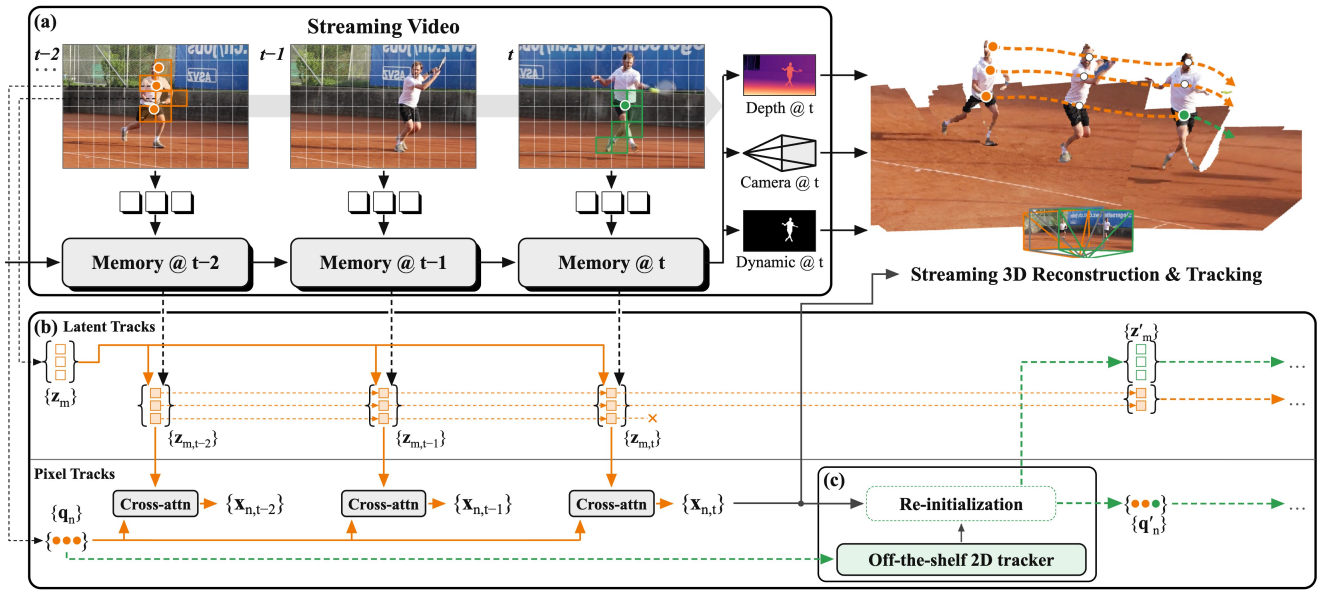


Figure 2. **Overview of Live4D.** Live4D predicts the 3D positions of queried points as each frame arrives, using the causal memory of a streaming 3D reconstruction model. *(a) Streaming reconstruction*: Each incoming frame reads and updates the causal memory while predicting its depth and camera pose. *(b) Hierarchical tracking*: Patch-level latent tracks spawned around queries read the memory to capture 3D motion, while pixel queries are efficiently decoded from their nearby latents without directly accessing the memory. *(c) Track re-initialization*: An off-the-shelf streaming 2D tracker corrects drifting 3D tracks by re-initializing visible points at their current 2D positions, while occluded points continue to be tracked in 3D.

feed-forward reconstruction over long sequences. Our key observation is that their causal memory can directly serve as the scene representation for query-based 3D tracking.

3D Tracking. Early 3D point trackers required per-scene optimization by lifting 2D tracking to 3D using monocular depth [25, 35]. *Sparse* 3D trackers follow a set of query points over a window of frames, typically given depth and camera poses from an external model: SpatialTracker [35] tracks in a triplane representation, TAPIP3D [38] tracks in a persistent world-space feature cloud, and SpatialTrackerV2 [36] jointly estimates camera motion and tracks. These methods refine all tracks iteratively over the window, which makes them slow and inherently offline. A growing line of work extends feed-forward 3D models to jointly predict geometry and motion [7, 15, 20]. TrackEverything [10] achieves long-horizon dense tracking with a de-duplicated 3D scene representation. Closest to ours, query-based methods [8, 11, 19, 20, 39] decouple scene encoding from track decoding. D4RT [39] encodes a video into a scene representation that each spatio-temporal query cross-attends to. Point4D [11] extends this to 3D point queries to chain predictions across hundreds of frames, and ARROW [8] decodes tracks across arbitrary image sets. However, all of these methods jointly encode a window of frames, which must be repeated for every incoming frame in the streaming setting, and every query attends to the full representation, making dense decoding costly. Live4D instead decodes queries from the causal memory of a streaming reconstruction model, and keeps dense decoding cheap with patch-level latent tracks.

3. Method

Given a monocular video stream $\mathcal{I} = \{I_1, I_2, \dots\}$, our goal is to predict the 3D position $\mathbf{x}_{n,t} \in \mathbb{R}^3$ of every pixel query $\mathbf{p}_n = (u_n, v_n, t_n)$, born at pixel (u_n, v_n) of frame I_{t_n}, at each timestep $t \geq t_n$, in the camera coordinate frame of I_1. To this end, we propose **Live4D**, a dense streaming 3D tracker. Live4D builds on the causal memory of the streaming 3D reconstruction model LingBot-Map [3] (Sec. 3.1), where each query reads the memory to predict its current 3D position as each frame arrives (Sec. 3.2). To make dense prediction efficient, pixel queries are decoded from patch-level latent tracks that read the memory on their behalf (Sec. 3.3). Finally, as queries can be added at any time, drifting tracks are re-initialized with new queries from an off-the-shelf streaming 2D tracker (Sec. 3.4).

3.1. Preliminaries

Streaming visual geometry prediction. Streaming 3D reconstruction models process a video one frame at a time, conditioning only on past observations. Each incoming frame I_t is processed by an image encoder $\mathcal{E}$ into patch tokens $F_t = \mathcal{E}(I_t)$, which are concatenated with a small set

of special tokens (1 camera, 1 scale, and 4 register tokens). These tokens attend to a memory $\mathcal{M}_{t-1}$, a key-value cache at every layer that holds the tokens of past frames, to predict the camera pose $\mathbf{P}_t$ and depth map $\mathbf{D}_t$, and are then appended to form $\mathcal{M}_t$:

$$(\mathbf{P}_t,\, \mathbf{D}_t,\, \mathcal{M}_t) = \Phi(I_t;\, \mathcal{M}_{t-1}), \tag{1}$$

where Φ denotes the reconstruction model. We additionally predict a dynamic mask $\mathbf{B}_t$, later used to spawn queries only in dynamic regions if needed.

Since caching all past tokens makes the per-frame cost grow linearly with t, LingBot-Map keeps only part of them in memory: (i) all tokens of the first A frames, the *anchor frames*, which fix the coordinate frame and scale of the reconstruction; (ii) all tokens of the most recent L frames, the *local window*, for local correspondence; and (iii) only the few register, camera and scale tokens of all other frames.

Query-based 4D decoding. Query-based 4D decoding methods [8, 11, 39] decouple scene encoding from track decoding. A video is first encoded with bidirectional attention into a scene representation, and each pixel of interest is then embedded as a query that cross-attends to it to predict the pixel's 3D position at a target timestep. Because queries are decoded independently, any subset of pixels can be tracked on demand without re-encoding the scene. In the streaming setting, however, this design has two limitations: (i) the scene representation depends on future frames, and (ii) dense decoding is costly, since every query cross-attends to the full scene representation.

3.2. From Streaming Reconstruction to Tracking

We adapt query-based 4D decoding to streaming video by using the causal memory $\mathcal{M}_t$ of a streaming reconstruction model as the scene representation. Since this memory is updated as each frame arrives, queries can predict their current 3D positions using only observations available up to timestep t. We first describe direct query decoding from the causal memory, before introducing its more efficient hierarchical implementation in Sec. 3.3.

Query construction. For each pixel query $\mathbf{p}_n = (u_n, v_n, t_n)$, we construct an initial pixel query token $\mathbf{q}_n$ that encodes its birth pixel location and appearance. Specifically, we combine the Fourier encoding γ of its pixel coordinates with an appearance feature bilinearly sampled from the patch tokens of its birth frame:

$$\mathbf{q}_n = \gamma(u_n, v_n) + F_{t_n}(u_n, v_n). \tag{2}$$

Query Tracking with Causal Memory. At each timestep $t \geq t_n$, our goal is to obtain a query state $\mathbf{q}_{n,t}$ from which the current 3D position of the queried point can be decoded. We obtain this state by letting the initial query $\mathbf{q}_n$ attend to the causal memory $\mathcal{M}_t$ and its own previous states:

$$\mathbf{q}_{n,t} = \Phi'\left(\mathbf{q}_n; \mathcal{M}_t, \{\mathbf{q}_{n,t'}\}_{t'<t}\right), \tag{3}$$

where Φ' denotes the query-processing transformer layers. The 3D position is then decoded from $\mathbf{q}_{n,t}$ through a decoder Ψ:

$$\mathbf{x}_{n,t} = \Psi(\mathbf{q}_{n,t}). \tag{4}$$

We implement Φ' using Mixture-of-Transformers (MoT) layers [17], initialized from the reconstruction layers Φ with separate attention projections and feed-forward networks for query tokens, and key/value projections for memory. At each layer, query tokens attend to memory, but not vice versa, keeping reconstruction independent of the queries. Specifically, each query attends to (i) patch tokens from the local window of L frames, including the current frame, (ii) special tokens from all earlier frames, and (iii) its own states at birth and the most recent L states, rather than its full history. During attention, we use 3D RoPE [28] to encode spatiotemporal coordinates (u_n, v_n, t), where t is the current timestep. Queries do not attend to one another and are thus processed independently.

Loss. We supervise $\mathbf{x}_{n,t}$ at every timestep from birth, with a confidence-weighted L1 loss:

$$\mathcal{L}_{\text{point}} = \sum_n \sum_{t \geq t_n} c_{n,t} \|\mathbf{x}_{n,t} - \mathbf{x}^*_{n,t}\|_1 - \lambda_c \log c_{n,t}, \tag{5}$$

where $\mathbf{x}^*_{n,t}$ denotes the ground truth and $c_{n,t}$ is predicted confidence. We additionally supervise auxiliary predictions of the 2D position ($\mathcal{L}_{\text{2D}}$), visibility ($\mathcal{L}_{\text{vis}}$), and 3D displacement from birth ($\mathcal{L}_{\text{disp}}$), and apply a reprojection loss ($\mathcal{L}_{\text{reproj}}$) that aligns each query's 3D position at birth with its pixel under the predicted camera. Since the reconstruction layers are also trained, we retain the depth and camera losses of the base model and a dynamic mask loss ($\mathcal{L}_{\text{recon}}$), yielding the total loss as follows:

$$\mathcal{L} = \mathcal{L}_{\text{point}} + \mathcal{L}_{\text{2D}} + \mathcal{L}_{\text{vis}} + \mathcal{L}_{\text{disp}} + \mathcal{L}_{\text{reproj}} + \mathcal{L}_{\text{recon}}. \tag{6}$$

3.3. Hierarchical Decoding with Latent Tracks

In Sec. 3.2, each pixel query attends to the causal memory through the query layers Φ' at every timestep, making the per-frame cost grow linearly with the number of pixel queries. To address this, we propose hierarchical decoding: instead of attending to the memory directly, each pixel query obtains its 3D position from nearby patch-level *latent tracks*. Each latent track attends to the memory on behalf of the pixel queries around it, maintaining a temporally evolving representation of the local 3D motion around its patch.

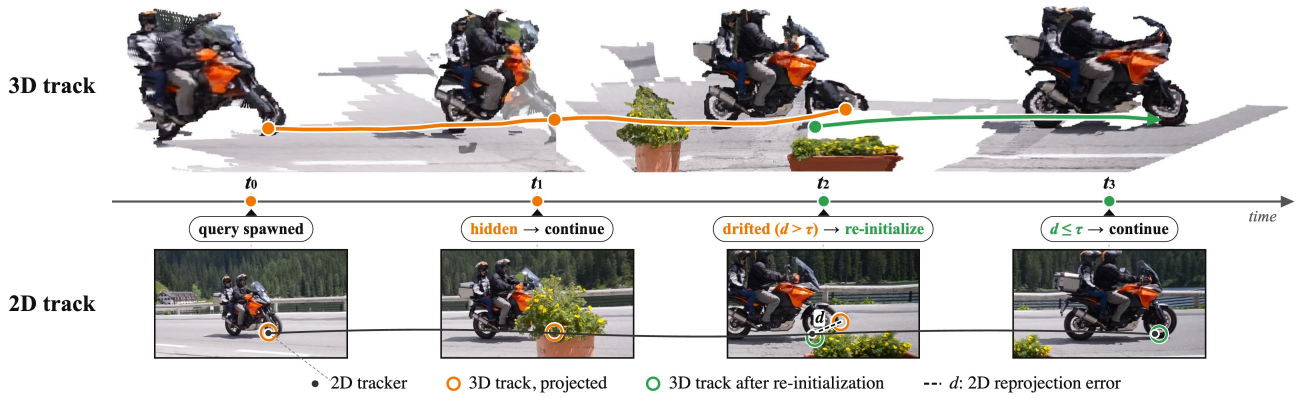


Figure 3. **Track re-initialization.** Alongside 3D tracking, we run an off-the-shelf streaming 2D tracker to obtain the 2D position and visibility of each query. When a point is occluded, we continue tracking it in 3D. When it becomes visible and the projected 3D track deviates from the 2D track by more than a threshold (τ), we re-initialize the query at its current 2D position; otherwise, it continues.

Latent tracks. Instead of processing pixel queries through Φ', we process a coarser set of latent queries $\mathbf{z}_m$ placed at patch centers. For each pixel query, we instantiate latent queries at its k nearest patch centers in its birth frame, where pixel queries with overlapping neighborhoods share them. Analogously to pixel queries in Sec. 3.2, each latent query is constructed from its patch-center coordinates and appearance feature, and attends to the causal memory $\mathcal{M}_t$ and its own previous states to produce a latent track:

$$\mathbf{z}_{m,t} = \Phi'\left(\mathbf{z}_m; \mathcal{M}_t, \{\mathbf{z}_{m,t'}\}_{t'<t}\right). \tag{7}$$

Each latent track $\mathbf{z}_{m,t}$ serves as a local motion feature around its patch at timestep t shared by nearby pixel queries.

Decoding pixel queries. At each timestep t, the token $\mathbf{q}_n$ of each pixel query (Eq. (2)) cross-attends to its latent tracks $\mathbf{z}_{m,t}$ through a lightweight decoder Ψ to predict its current 3D position:

$$\mathbf{x}_{n,t} = \Psi\left(\mathbf{q}_n; \{\mathbf{z}_{m,t}\}_{m\in\mathcal{N}_k(n)}\right), \tag{8}$$

where $\mathcal{N}_k(n)$ denotes the indices of the k nearest latent tracks to pixel query n. Unlike in Sec. 3.2, pixel queries neither attend to the memory nor maintain their own past states; both are handled by the latent tracks, of which there is at most one per patch. Decoding a pixel query thus reduces to a cheap readout from k latent tracks, enabling efficient dense tracking. Training follows Sec. 3.2, with the same losses applied to pixel-level predictions.

3.4. Track Re-initialization

Our 3D tracks persist through occlusion but can drift over long sequences, whereas online 2D trackers [13, 41] provide reliable locations for visible points, but lifting their predictions to 3D becomes unreliable during occlusion. Since pixel queries can be added cheaply at any frame (Sec. 3.3), we combine the two at inference: whenever a point is visible and its track has drifted, the track is re-initialized from the 2D track to cut off accumulated drift, while 3D tracking continues for the remaining points.

Re-initialization. We use TAPNext++ [13] as the online 2D tracker, which runs causally alongside Live4D. At every K-th frame t, Live4D predicts the 3D position $\mathbf{x}_{n,t}$ of each query, whose projection is $\hat{\mathbf{p}}_{n,t} = \pi(\mathbf{P}_t, \mathbf{x}_{n,t})$ with π denoting perspective projection, while the 2D tracker provides its pixel position $\mathbf{p}^{\text{2D}}_{n,t}$ and visibility. We replace the query only when the point is visible and its track has drifted:

$$\mathbf{p}_n \leftarrow \begin{cases} (\mathbf{p}^{\text{2D}}_{n,t},\, t), & \text{if visible } \&\ \|\hat{\mathbf{p}}_{n,t} - \mathbf{p}^{\text{2D}}_{n,t}\| > \tau, \\ \mathbf{p}_n, & \text{otherwise,} \end{cases} \tag{9}$$

where τ is a drift threshold in pixels. Re-initialization creates new pixel and latent queries at the current frame while preserving the identity of the tracked point. We retain the past 3D predictions and use the re-initialized query to predict subsequent positions of the same point. Occluded tracks keep their current query without re-initialization, so that 3D tracking continues where the 2D tracker cannot localize the point.

3.5. Implementation Details

We train Live4D on a mixture of dynamic and static datasets, treating points unprojected from depth in static scenes as stationary trajectories. The reconstruction layers are initialized from LingBot-Map [3], with $A = 8$ anchor frames and a local window of $L = 16$ frames at inference, sampled from 16 to 64 during training. Training clips contain 24 to 80 frames at a width of 518 pixels. Each pixel

Table 1. **Evaluation on LongTrack.** Live4D outperforms all streaming methods across all mean metrics and achieves the lowest EPE among all methods, including window-based methods with access to future frames. We report APD↑ and EPE↓ (m) over all points (All) and dynamic points (Dyn.) on full-length sequences of up to 300 frames, with queries at the first frame. †Run on overlapping 48-frame windows with stride 32, each prepended with the first frame, as the full sequence exceeds the memory budget. Within each block, best in **bold** and second underlined.

	Method	DR				PO				PStudio				ADT				Mean			
		APD↑		EPE↓		APD↑		EPE↓		APD↑		EPE↓		APD↑		EPE↓		APD↑		EPE↓	
		All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.
Window-based	TAPIP3D [38] + LingBot-Map	0.797	0.654	0.207	0.366	0.511	0.475	0.953	0.985	0.646	0.646	0.301	0.301	0.647	0.637	**0.513**	0.564	0.650	0.603	0.493	0.554
	SpatialTrackerV2 [36]	0.762	0.667	0.253	0.387	0.532	0.531	0.727	0.681	0.727	0.727	0.230	0.230	0.460	0.527	1.638	1.270	0.620	0.613	0.712	0.642
	TrackEverything [10]	0.871	0.703	0.151	0.328	**0.631**	0.589	0.850	0.855	0.700	0.700	0.263	0.263	**0.709**	0.618	0.553	0.666	**0.728**	0.652	0.454	0.528
	Point4D [11]	0.851	0.752	0.163	0.258	0.532	0.568	0.746	0.660	0.712	0.712	0.253	0.253	0.526	**0.650**	1.146	**0.495**	0.655	0.670	0.577	**0.416**
	Any4D† [15]	0.857	0.672	0.166	0.356	0.477	0.474	0.690	0.727	0.608	0.608	0.330	0.330	0.401	0.445	0.885	0.988	0.586	0.550	0.518	0.600
	4RC† [20]	0.839	0.686	0.178	0.351	0.570	0.566	0.751	0.794	0.636	0.636	0.314	0.314	0.435	0.421	0.967	1.090	0.620	0.577	0.552	0.637
	OpenD4RT† [27]	0.810	0.740	0.197	0.289	0.419	0.441	0.824	0.808	0.745	0.745	0.214	0.214	0.365	0.406	1.124	1.152	0.585	0.583	0.590	0.616
	SM4RT† [19]	0.827	0.663	0.186	0.363	0.560	0.550	0.727	0.766	0.720	0.720	0.238	0.238	0.446	0.430	0.922	1.025	0.638	0.591	0.518	0.598
	ARROW† [8]	**0.883**	**0.884**	**0.134**	**0.149**	0.622	**0.639**	**0.463**	**0.486**	**0.811**	**0.811**	**0.163**	**0.163**	0.475	0.438	0.865	1.062	0.698	**0.693**	**0.406**	0.465
Streaming	TAPNext++ + LingBot-Map	0.811	0.697	0.207	0.309	0.564	0.593	0.608	0.585	0.628	0.628	0.323	0.323	**0.790**	**0.721**	**0.253**	**0.426**	0.698	0.660	0.348	0.411
	CoTracker3 + LingBot-Map	0.801	0.643	0.220	0.382	0.511	0.498	0.869	0.944	0.606	0.606	0.364	0.364	0.745	0.671	0.332	0.528	0.666	0.605	0.446	0.554
	Live4D (w/o re-init.)	0.814	0.718	0.193	0.298	0.559	0.577	0.574	0.555	0.616	0.616	0.314	0.314	0.682	0.584	0.572	0.803	0.668	0.624	0.413	0.493
	Live4D	**0.839**	**0.765**	**0.169**	**0.249**	**0.614**	**0.639**	**0.474**	**0.437**	**0.643**	**0.643**	**0.290**	**0.290**	0.767	0.708	0.274	0.430	**0.716**	**0.689**	**0.302**	**0.351**

query is decoded from its $k = 9$ nearest latents, and re-initialization uses $K = 10$ and $\tau = 4$ px. Further details are provided in the supplementary material.

4. Experiments

We first verify the tracking accuracy and latency of Live4D on long and short video sequences, comparing against lifted 2D trackers and window-based 3D trackers (Sec. 4.2). We then ablate our key design choices, including hierarchical decoding with latent tracks, fine-tuning the reconstruction layers, and track re-initialization (Sec. 4.3).

4.1. Experimental Setting

Benchmarks. We evaluate on two benchmarks built on four datasets: Dynamic Replica (DR), PointOdyssey (PO), Panoptic Studio (PStudio), and Aria Digital Twin (ADT). *WorldTrack* [7] consists of 64-frame clips with queries at the first frame. To evaluate longer-term tracking, we construct *LongTrack* using the full-length WorldTrack sequences of PStudio (150 frames) and ADT (300 frames), and 300-frame clips from the original test splits of DR and PO.

Baselines. We compare Live4D against two categories: **(1)** ***Lifted 2D streaming trackers*** combine TAPNext++ [13] or CoTracker3 [14] with per-frame depth and camera poses from LingBot-Map [3], keeping invisible points at their last visible 3D position. These are the only streaming baselines and thus provide the most direct comparison to Live4D. **(2)** ***Window-based 3D trackers*** process frames jointly with bidirectional attention. Among these, TAPIP3D [38], TrackEverything [10], SpatialTrackerV2 [36] and Point4D [11] support long-sequence inference via 3D chaining. The remaining methods—

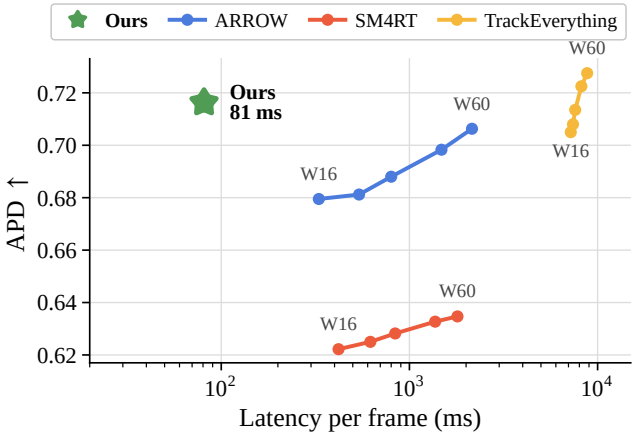


Figure 4. **Live4D outperforms window-based methods with shorter windows, while achieving substantially lower latency.** We compare APD against per-frame latency on the LongTrack benchmark, with an average of approximately 2K queries per sequence. Window-based methods are evaluated with varying window sizes $W \in \{16, 24, 32, 48, 60\}$.

Any4D [15], 4RC [20], OpenD4RT [27], SM4RT [19], and ARROW [8]— cannot process entire LongTrack sequences due to memory constraints. Following OpenD4RT [27], we evaluate these methods on long sequences by chaining predictions across windows of size W and stride S, each prepended with the first frame. We find this strategy more accurate than reprojection-based chaining [11] (details in supplementary material).

Metrics. Following prior work [7, 11, 15], we report the average percentage of points within distance thresholds of $\{0.1, 0.3, 0.5, 1.0\}$ m (APD) and the mean 3D end-point er-

Table 2. **Evaluation on WorldTrack.** Live4D outperforms all streaming methods and performs comparably to most window-based methods on dynamic points, despite having no access to future frames. We report APD↑ and EPE↓ (m) over all points (All) and dynamic points (Dyn.) on 64-frame clips with queries at the first frame. Within each block, best in **bold** and second underlined.

	Method	DR				PO				PStudio				ADT				Mean			
		APD↑		EPE↓		APD↑		EPE↓		APD↑		EPE↓		APD↑		EPE↓		APD↑		EPE↓	
		All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.	All	Dyn.
Window-based	TAPIP3D [38] + LingBot-Map	0.693	0.635	0.307	0.405	0.617	0.629	0.445	0.486	0.636	0.636	0.307	0.307	0.913	0.889	0.112	0.121	0.715	0.697	0.293	0.330
	SpatialTrackerV2 [36]	0.743	0.740	0.257	0.263	0.670	0.658	0.415	0.450	0.759	0.759	0.201	0.201	0.870	0.830	0.156	0.184	0.760	0.747	0.257	0.275
	TrackEverything [10]	0.764	0.704	0.252	0.342	0.738	0.730	0.412	0.432	0.708	0.708	0.243	0.243	0.924	0.873	0.099	0.125	0.783	0.754	0.252	0.285
	Point4D [11]	0.787	0.720	0.225	0.289	0.709	0.715	0.310	0.332	0.736	0.736	0.228	0.228	0.903	0.882	0.140	0.112	0.784	0.763	0.226	0.240
	Any4D [15]	0.790	0.693	0.218	0.313	0.668	0.680	0.350	0.372	0.642	0.642	0.296	0.296	0.579	0.696	0.415	0.278	0.670	0.678	0.320	0.315
	4RC [20]	0.813	0.712	0.206	0.318	0.771	0.731	0.324	0.441	0.694	0.694	0.260	0.260	0.836	0.718	0.183	0.279	0.778	0.714	0.243	0.325
	OpenD4RT [27]	0.768	0.818	0.243	0.187	0.681	0.747	0.314	0.252	0.796	0.796	0.176	0.176	0.722	0.732	0.276	0.320	0.742	0.773	0.252	0.234
	SM4RT [19]	0.826	0.728	0.187	0.283	0.810	0.759	0.236	0.320	0.793	0.793	0.172	0.172	0.867	0.803	0.153	0.188	0.824	0.771	0.187	0.241
	ARROW [8]	**0.879**	**0.915**	**0.143**	**0.093**	**0.855**	**0.871**	**0.152**	**0.137**	**0.857**	**0.857**	**0.135**	**0.135**	**0.936**	**0.907**	**0.082**	**0.094**	**0.882**	**0.887**	**0.128**	**0.115**
Streaming	TAPNext++ + LingBot-Map	0.702	0.682	0.302	0.356	0.626	0.644	0.423	0.426	0.623	0.623	0.323	0.323	**0.904**	0.882	0.121	0.117	0.714	0.708	0.292	0.305
	CoTracker3 + LingBot-Map	0.690	0.644	0.318	0.408	0.591	0.598	0.523	0.574	0.611	0.611	0.346	0.346	0.896	0.881	0.126	0.118	0.697	0.683	0.328	0.361
	Live4D (w/o re-init.)	0.716	0.705	0.280	0.305	0.617	0.674	0.388	0.297	0.620	0.620	0.309	0.309	0.896	0.859	0.124	0.132	0.712	0.715	0.275	0.261
	Live4D	**0.728**	**0.735**	**0.270**	**0.274**	**0.628**	**0.696**	**0.374**	**0.276**	**0.631**	**0.631**	**0.298**	**0.298**	0.903	**0.897**	**0.119**	**0.103**	**0.723**	**0.740**	**0.265**	**0.238**

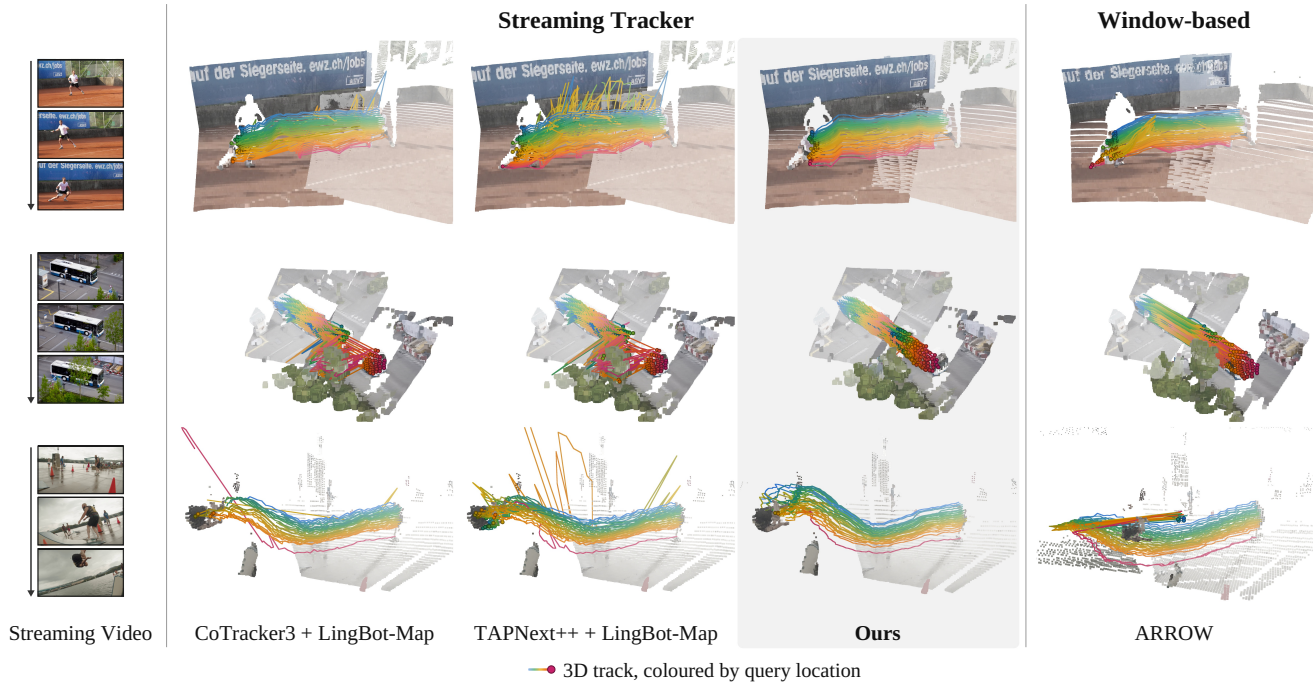


Figure 5. **Qualitative comparison of 3D tracking.** We compare Live4D against streaming 2D trackers lifted to 3D (CoTracker3 and TAPNext++ with LingBot-Map) and the window-based method ARROW. Live4D produces coherent dense 3D trajectories, even through occlusions where lifted 2D trackers fail, achieving tracking quality comparable to ARROW without accessing future frames.

ror (EPE), after aligning the ground truth and predicted trajectories with a single median scale. We measure latency on a single H200 GPU as the inference time for one frame of Live4D or one window of window-based methods, since the latter must process a full window to predict the current frame's tracks.

4.2. Streaming 3D Tracking

Tracking over long sequences. Table 1 compares tracking accuracy on *LongTrack* sequences of up to 300 frames (150 for PStudio). Live4D outperforms all streaming baselines across all mean metrics and achieves the lowest mean EPE among all compared methods, including window-based trackers. For methods requiring chunked inference, we use window size $W = 48$ and stride $S = 32$.

To investigate whether window-based methods can

Table 3. **Efficiency of latent tracks.** With latent tracks (w/ latent), the per-frame latency scales slowly with the number of queries, while accuracy matches that of decoding pixel queries directly from the memory (w/o latent). Latency is measured on a single H200 (141 GB), where *Full* decodes every pixel of a 294×518 frame (152,292 queries) and OOM denotes out of memory.

	APD↑		Latency (ms)↓ / #queries					
	All	Dyn.	256	1K	4K	10K	40K	Full
w/o latent	0.634	0.590	33.0	39.8	69.7	143.5	OOM	OOM
w/ latent	0.627	0.593	27.7	28.0	28.8	30.9	56.3	78.4

achieve comparable latency by using shorter windows, we examine their accuracy–latency trade-off in Fig. 4 by varying their window size $W \in \{16, 24, 32, 48, 60\}$ with stride $S = W$ (except for TrackEverything's geometry estimation, where $S = W/2$). While some window-based methods achieve higher accuracy with larger windows, reducing the window size to improve latency degrades their accuracy. Even with smaller windows, these methods are substantially slower than Live4D while achieving lower APD.

WorldTrack benchmark. Table 2 evaluates tracking accuracy on WorldTrack, whose 64-frame clips fit within a single window. Live4D outperforms all streaming baselines across all mean metrics and achieves competitive accuracy with window-based methods on dynamic points, despite not accessing future frames.

Qualitative Results. Fig. 5 shows qualitative results on DAVIS [26] and in-the-wild videos. While lifted 2D trackers lose track of occluded points and produce jittery trajectories near object boundaries, Live4D maintains coherent dense 3D tracks through motion and occlusion, achieving visually comparable results to the window-based method ARROW without accessing future frames.

4.3. Ablations

We study whether hierarchical decoding enables efficient dense tracking without sacrificing accuracy, and how our key design choices—fine-tuning the reconstruction layers, latent track density, and track re-initialization—affect tracking accuracy and latency. For all ablations, we use a shorter training schedule than the final model.

Efficiency of latent tracks. Table 3 compares hierarchical decoding with latent tracks against directly decoding each pixel query from the memory. For latency evaluation, we sample up to 10k queries from dynamic regions, then sample additional queries throughout the image. The two variants achieve comparable accuracy on LongTrack (within 0.01 APD), but differ significantly in efficiency. Without latent tracks, latency increases quickly for up to 10K queries and runs out of memory at 40K queries. With latent tracks, latency remains below 31ms up to 10K queries and reaches only 78.4ms when decoding every pixel (152K queries). This is because only the latent tracks perform expensive attention over the memory, while pixel queries are cheaply decoded from nearby latents. As a result, the cost of memory attention depends on the spatial coverage of queries rather than their number, enabling efficient dense tracking without sacrificing accuracy.

Table 4. **Ablation of design choices.** Fine-tuning the reconstruction layers and using one latent track per patch both improve accuracy, and re-initialization brings the largest gain at the cost of latency. Results are averaged over the four LongTrack datasets, and latency is measured per frame with 10K queries on a single H200, including TAPNext++ for (D). Best in **bold**.

				APD↑		EPE↓		Latency
	Latent	Recon.	Re-init.	All	Dyn.	All	Dyn.	(ms)↓
(A)	1×1	Frozen		0.600	0.571	0.485	0.539	30.7
(B)	2×2	Tuned		0.614	0.585	0.482	0.539	28.2
(C)	1×1	Tuned		0.627	0.593	0.446	0.498	30.9
(D)	1×1	Tuned	✓	**0.663**	**0.645**	**0.362**	**0.387**	86.6

Design choices. Table 4 evaluates our key design choices: fine-tuning the reconstruction layers, latent track density, and track re-initialization. Latency is measured with 10K queries sampled from dynamic regions. Fine-tuning the reconstruction layers jointly with the query layers improves tracking accuracy (A vs. C). Using one latent track per 2×2 patch region instead of one per patch reduces accuracy while saving only 2.7 ms (B vs C). Finally, track re-initialization provides the largest accuracy improvement, at the cost of increased latency from running TAPNext++ alongside the 3D tracker (C vs. D).

5. Conclusion

We presented Live4D, a method for dense 3D point tracking from streaming video. Our key insight is that the causal memory of a streaming 3D reconstruction model can serve as a scene representation for query-based tracking, enabling 3D tracks to be predicted as each frame arrives. To make dense tracking efficient, we introduced hierarchical decoding with patch-level latent tracks. We further combined 3D tracking with an off-the-shelf streaming 2D tracker to correct drift while preserving tracks through occlusions. Experiments show that Live4D outperforms streaming baselines and is competitive with window-based methods at substantially lower latency. More broadly, we demonstrate the potential of streaming reconstruction models as a foundation for efficient, long-horizon 4D perception.

Limitations. While Live4D can track points through occlusions, its 3D predictions may accumulate drift over long sequences. Our re-initialization strategy mitigates this by using an off-the-shelf streaming 2D tracker, but corrections are only possible when points are visible and reliably localized in 2D. Consequently, points that remain occluded or out of view for extended periods may accumulate errors without being corrected. Moreover, re-initialization introduces additional computational overhead from running a separate 2D tracker, increasing per-frame latency.

LLM Usage Statement

We used Codex (OpenAI) and Claude (Anthropic) as collaborative assistants during the development of this work, primarily for code development, debugging, and manuscript writing, including improvements to clarity, grammar, and presentation. All research ideas, methodological decisions, experiments, analyses, and conclusions were directed and verified by the authors. All AI-assisted outputs were reviewed and validated by the authors, who take full responsibility for the content and correctness of this work.

References

[1] Görkay Aydemir, Weidi Xie, and Fatma Güney. Track-on2: Enhancing online point tracking with memory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026. 2

[2] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE transactions on robotics*, 37(6):1874–1890, 2021. 2

[3] Lin-Zhuo Chen, Jian Gao, Shangzhan Zhang, Yihang Chen, Nan Xue, Jianyuan Wang, Christian Rupprecht, Xun Cao, Xing Zhu, Yujun Shen, et al. Geometric context transformer for streaming 3d reconstruction. In *European Conference on Computer Vision*, pages 293–314. Springer, 2026. 2, 3, 5, 6

[4] Kai Deng, Zexin Ti, Jiawei Xu, Jian Yang, and Jin Xie. Vggt-long: Chunk it, loop it, align it–pushing vggt's limits on kilometer-scale long rgb sequences. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5446–5453. IEEE, 2026. 2

[5] 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 *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10027–10038. IEEE, 2023. 2

[6] Carl Doersch, Pauline Luc, Yi Yang, Dilara Gokay, Skanda Koppula, Ankush Gupta, Joseph Heyward, Ignacio Rocco, Ross Goroshin, Joao Carreira, et al. Bootstap: Bootstrapped training for tracking-any-point. In *Asian Conference on Computer Vision*, pages 483–500. Springer, 2024. 2

[7] Haiwen Feng, Junyi Zhang, Qianqian Wang, Yufei Ye, Pengcheng Yu, Michael J Black, Trevor Darrell, and Angjoo Kanazawa. St4rtrack: Simultaneous 4d reconstruction and tracking in the world. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8503–8513. IEEE, 2025. 2, 3, 6

[8] Ilya Fradlin, Christian Schmidt, Jens Piekenbrinck, Karim Knaebel, Gonzalo Martin Garcia, and Bastian Leibe. ARROW: Arbitrary reconstruction and tracking of 4d observations in the wild. *arXiv preprint arXiv:2610.01314*, 2026. 2, 3, 4, 6, 7

[9] Adam W Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. Particle video revisited: Tracking through occlusions using point trajectories. In *European Conference on Computer Vision*, pages 59–75. Springer, 2022. 2

[10] Ayush Jain, Sreeharsha Paruchuri, Ishita Gupta, Fan Zhang, Tanner Schmidt, Jakob Engel, Katerina Fragkiadaki, and Adam W Harley. Trackeverything: Long horizon dense tracking via de-duplicating 3d scene representations. *arXiv preprint arXiv:2609.30222*, 2026. 2, 3, 6, 7

[11] Minsik Jeon, Jay Karhade, Deva Ramanan, and Shubham Tulsiani. Point4d: Long-range 4d motion reconstruction. *arXiv preprint arXiv:2609.09145*, 2026. 2, 3, 4, 6, 7

[12] Haian Jin, Rundi Wu, Tianyuan Zhang, Ruiqi Gao, Jonathan T Barron, Noah Snavely, and Aleksander Hołyński. Zipmap: Linear-time stateful 3d reconstruction via test-time training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21748–21759, 2026. 2

[13] Sebastian Jung, Artem Zholus, Martin Sundermeyer, Carl Doersch, Ross Goroshin, David Joseph Tan, Sarath Chandar, Rudolph Triebel, and Federico Tombari. Tapnext++: What's next for tracking any point (tap)? *arXiv preprint arXiv:2604.10582*, 2026. 2, 5, 6

[14] Nikita Karaev, Yuri Makarov, Jianyuan Wang, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker3: Simpler and better point tracking by pseudo-labeling real videos. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1–10. IEEE, 2025. 2, 6

[15] Jay Karhade, Nikhil Keetha, Yuchen Zhang, Tanisha Gupta, Akash Sharma, Sebastian Scherer, and Deva Ramanan. Any4d: Unified feed-forward metric 4d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14578–14589, 2026. 3, 6, 7

[16] Nikhil Keetha, Norman Müller, Johannes Schönberger, Lorenzo Porzi, Yuchen Zhang, Tobias Fischer, Arno Knapitsch, Duncan Zauss, Ethan Weber, Nelson Antunes, et al. Mapanything: Universal feed-forward metric 3d reconstruction. In *2026 International Conference on 3D Vision (3DV)*, pages 499–509. IEEE, 2026. 2

[17] Weixin Liang, Lili Yu, Liang Luo, Srinivasan Iyer, Ning Dong, Chunting Zhou, Gargi Ghosh, Mike Lewis, Wen-tau Yih, Luke Zettlemoyer, et al. Mixture-of-transformers: A sparse and scalable architecture for multi-modal foundation models. *arXiv preprint arXiv:2411.04996*, 2024. 4

[18] Haotong Lin, Sili Chen, Junhao Liew, Donny Y Chen, Zhenyu Li, Guang Shi, Jiashi Feng, and Bingyi Kang. Depth

anything 3: Recovering the visual space from any views. *arXiv preprint arXiv:2511.10647*, 2025. 2

[19] Shing Ho J Lin, Wenzhao Zheng, Dong Zhuo, Yuqi Wu, Jie Zhou, and Jiwen Lu. Sm4rt: Learning structured motion geometry for 4d reconstruction. *arXiv preprint arXiv:2607.22534*, 2026. 3, 6, 7

[20] Yihang Luo, Shangchen Zhou, Yushi Lan, Xingang Pan, and Chen Change Loy. 4rc: 4d reconstruction via conditional querying anytime and anywhere. *arXiv preprint arXiv:2602.10094*, 2026. 2, 3, 6, 7

[21] Dominic Maggio, Hyungtae Lim, and Luca Carlone. Vggt-slam: Dense rgb slam optimized on the sl (4) manifold. *Advances in Neural Information Processing Systems*, 38: 129839–129867, 2026. 2

[22] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 2

[23] Riku Murai, Eric Dexheimer, and Andrew J Davison. Mast3r-slam: Real-time dense slam with 3d reconstruction priors. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16695–16705. IEEE, 2025. 2

[24] Michal Neoral, Jonáš Šerỳch, and Jiří Matas. Mft: Long-term tracking of every pixel. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 6823–6833. IEEE, 2024. 2

[25] Tuan Duc Ngo, Peiye Zhuang, Chuang Gan, Evangelos Kalogerakis, Sergey Tulyakov, Hsin-Ying Lee, and Chaoyang Wang. Delta: Dense efficient long-range 3d tracking for any video. *arXiv preprint arXiv:2410.24211*, 2024. 3

[26] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alex Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*, 2017. 8

[27] RHOS Team. OpenD4RT: An unofficial open-source PyTorch implementation of D4RT, 2026. GitHub repository, Lijiaxin0111/Open-d4rt. 6, 7

[28] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. 4

[29] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in neural information processing systems*, 34:16558–16569, 2021. 2

[30] Hengyi Wang and Lourdes Agapito. 3d reconstruction with spatial memory. In *2025 International Conference on 3D Vision (3DV)*, pages 78–89. IEEE, 2025. 2

[31] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5294–5306. IEEE, 2025. 2

[32] Jianyuan Wang, Minghao Chen, Shangzhan Zhang, Nikita Karaev, Johannes Schönberger, Patrick Labatut, Piotr Bojanowski, David Novotny, Andrea Vedaldi, and Christian Rupprecht. Vggt-ω. *arXiv preprint arXiv:2605.15195*, 2026. 2

[33] Qianqian Wang*, Yifei Zhang*, Aleksander Holynski, Alexei A. Efros, and Angjoo Kanazawa. Continuous 3d perception model with persistent state. In *CVPR*, 2025. 2

[34] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Geometric 3d vision made easy. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20697–20709. IEEE, 2024. 2

[35] Yuxi Xiao, Qianqian Wang, Shangzhan Zhang, Nan Xue, Sida Peng, Yujun Shen, and Xiaowei Zhou. Spatialtracker: Tracking any 2d pixels in 3d space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20406–20417, 2024. 3

[36] Yuxi Xiao, Jianyuan Wang, Nan Xue, Nikita Karaev, Iurii Makarov, Bingyi Kang, Xin Zhu, Hujun Bao, Yujun Shen, and Xiaowei Zhou. Spatialtrackerv2: 3d point tracking made easy. In *ICCV*, 2025. 2, 3, 6, 7

[37] Shuai Yuan, Yantai Yang, Xiaotian Yang, Xupeng Zhang, Zhonghao Zhao, Lingming Zhang, and Zhipeng Zhang. Infinitevggt: Visual geometry grounded transformer for endless streams. *arXiv preprint arXiv:2601.02281*, 2026. 2

[38] Bowei Zhang, Lei Ke, Adam Harley, and Katerina Fragkiadaki. Tapip3d: Tracking any point in persistent 3d geometry. *Advances in Neural Information Processing Systems*, 38: 135284–135303, 2026. 2, 3, 6, 7

[39] Chuhan Zhang, Guillaume Le Moing, Skanda Koppula, Ignacio Rocco, Liliane Momeni, Junyu Xie, Shuyang Sun, Rahul Sukthankar, Joëlle K Barral, Raia Hadsell, et al. Efficiently reconstructing dynamic scenes one d4rt at a time. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7382–7392, 2026. 2, 3, 4

[40] Junyi Zhang, Charles Herrmann, Junhwa Hur, Chen Sun, Ming-Hsuan Yang, Forrester Cole, Trevor Darrell, and Deqing Sun. Loger: Long-context geometric reconstruction with hybrid memory. In *European Conference on Computer Vision*, pages 263–281. Springer, 2026. 2

[41] Artem Zholus, Carl Doersch, Yi Yang, Skanda Koppula, Viorica Patraucean, Xu Owen He, Ignacio Rocco, Mehdi SM Sajjadi, Sarath Chandar, and Ross Goroshin. Tapnext: Tracking any point (tap) as next token prediction. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9693–9703. IEEE, 2025. 2, 5

[42] Dong Zhuo, Wenzhao Zheng, Jiahe Guo, Yuqi Wu, Jie Zhou, and Jiwen Lu. Streaming visual geometry transformer. In *The Fourteenth International Conference on Learning Representations*, 2026. 2