What If: Understanding Motion Through Sparse Interactions

Supplementary Material

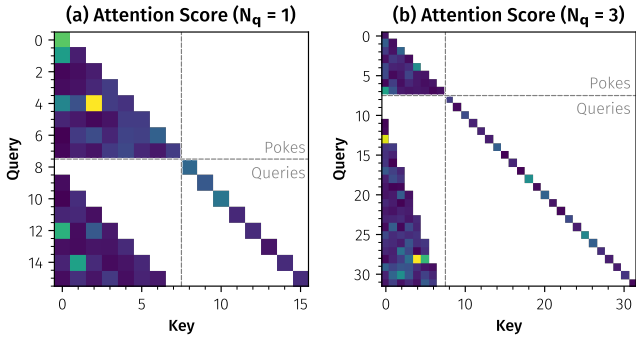


Figure A. **Query-Causal Attention Pattern Visualization**. We show the resulting attention patterns for our query-causal attention for different numbers of queries per poke count. We put poke tokens first, followed by query tokens. **(a)** In the simplest setting, with one query per set of pokes, there is one query token per set of pokes, with each query token attending to one more poke token than its predecessor. **(b)** For $N_q > 1$, the poke attention does not change, but there are multiple query tokens per poke set. Here, even query tokens for the same poke set do not attend to each other to enable parallel evaluation during inference.

A. Implementation Details

The hyperparameters for the base model used for all evaluation and qualitative examples are reported in Tab. B. We train the model for a total of 800k steps with a learning rate of 5.0×10^{-5} using the AdamW [26] optimizer and a linear warmup of 5000 steps. The first 250k steps are trained with a batch size of 32; for the remainder, we set the batch size to 128.

As described in Sec. 4.1, we randomly sample flow pokes and their corresponding positions from the given dense flow grid. We enforce that all flow values are in $[-1, 1]$ by applying a `tanh` and obtain a sinusoidal embedding for the x and y components of the flow. We then find the corresponding image features using the pokes positions and concatenate them with the flow features. Finally, we project them using a residual mapping network consisting of 3 FFN blocks with RMSNorms [53] and GELU [14] activation function. These embeddings are then combined with query tokens. The query tokens represent the locations in the image for which we want to predict a flow distribution and are realized by a learnable token and the corresponding positional encoding.

The flow pokes and query tokens are fed to the transformer, which is 12 blocks deep and has a width of 768. The self-attention uses our *query-causal attention* mask as introduced in Sec. 3.2. We visualize it in Fig. A. In the cross-attention, the pokes and queries attend to the image features, enabling them to learn a global understanding of the scene. In both attention mechanisms, we use 2D Axial RoPE [4, 38] to model the spatial relationships of tokens. The FNNs expand the internal feature dimension by a factor of three, use SwiGLU [34] as an activation function, and are conditioned on whether or not the camera is static using AdaRMSNorms. Whether the camera is static or not is detected using a simple heuristic: we consider it to be static if a significant fraction of the scene's content is static. This information can be directly derived from the training tracks. Specifically, we find that considering a camera static once 40% of the frame move by at most 3px (at our training resolution of 448^2) works well on our training data.

Parameter	Value
Batch size	32→128
Optimizer	AdamW [26]
Learning rate	5.0×10^{-5}
Betas	$(0.9, 0.99)$
Warm-up steps	5000
Steps	800k
Precision	bfloat16
Total Parameters	220M
Flow grid resolution	48×48
Flow scale	$[-1, 1]$
Image size	448×448
Mixtures	4
Covariance	Full
Given pokes	128
Query factor	15
Depth	12
SA width	768
CA width	768
Normalization	RMSNorm [53]
FFN expand factor	3
Activation	SwiGLU [34]
Positional encoding	2D Axial RoPE [4, 38]
Static scene conditioning	AdaRMSNorm

Table B. Hyperparameters for our base model. Ablation models use the same parameters, but only train for 250k steps.

Method	Trained On	EPE@1 ↓	EPE@2 ↓	EPE@5 ↓	EPE@10 ↓	EPE@100 ↓
InstantDrag [36]	Faces	9.24	9.12	8.82	8.39	7.29
Motion-I2V [35]	**Generic (Zero-Shot)**	29.08	27.40	24.22	20.90	n/a
Ours (full training)	**Generic (Zero-Shot)**	**7.64**	**6.87**	**5.32**	**4.20**	**2.51**
Vision Feature Extractor Initialization						
Jointly Trained Pre-Trained (**Ours**)	**Generic (Zero-Shot)**	**8.08**	**6.96**	**5.38**	3.99	2.33
Frozen Pre-Trained	**Generic (Zero-Shot)**	8.30	7.22	5.44	**3.78**	**2.22**
Trained from Scratch	**Generic (Zero-Shot)**	8.15	7.51	6.14	4.73	2.57
GMM Component Count						
1 Component	**Generic (Zero-Shot)**	**7.60**	**6.87**	5.42	4.18	2.59
2 Components	**Generic (Zero-Shot)**	8.98	7.87	5.83	4.08	2.34
4 Components (**Ours**)	**Generic (Zero-Shot)**	8.08	6.96	**5.38**	3.99	2.33
8 Components	**Generic (Zero-Shot)**	8.23	7.19	5.57	4.01	**2.29**
16 Components	**Generic (Zero-Shot)**	8.41	7.26	5.44	**3.90**	2.34
GMM Covariance Parametrization						
Full Covariance, 4 Components (**Ours**)	**Generic (Zero-Shot)**	**8.08**	**6.96**	**5.38**	3.99	2.33
Diagonal, 4 Components	**Generic (Zero-Shot)**	8.13	7.09	5.40	**3.98**	**2.24**

Table A. Extension of Tab. 1 including our ablations. The experiment is identical to the original one. The ablation models have been trained for 250k steps compared to 800k for the full training. EPE@N refers to the endpoint error given N pokes.

We find that directly using the output of the transformer for GIVT [41] produces unreasonable distributions and thus bad performance. Therefore, we use a simple MLP to project the transformer's output which alleviates that problem. Similarly, the input MLP consists of a Fourier embedding for the flow (embedding horizontal and vertical flow separately and then concatenating their features), followed by a linear layer that concatenates the Fourier embedding with the local image features extracted by the image feature extractor $\mathcal{E}(\mathcal{I})$, and 3 FFN blocks.

B. Ablations

We show an extended version of Tab. 1 that includes quantitative results for hyperparameter ablations, which were used to motivate the choices mentioned in the main paper, in Tab. A. All comparison models follow the original training recipe but are only trained until 250k steps due to compute constraints.

First, we ablate whether to initialize the vision encoder with pre-trained weights and whether to continue training them. We find that initializing with pre-trained weights gives a performance boost in this setting. We hypothesize that, for very long trainings, both versions might end up performing similarly well. As noted in Sec. 4.1, freezing the feature extractor when initializing with DINOv2 [29] empirically results in a model with reduced instance segmentation capabilities compared to the unlocked version. We show a qualitative example of this in Fig. B. This behavior can also be observed in the quantitative evaluations, where, for low poke counts, the jointly trained model performs better than the one with a frozen feature extractor. At high poke counts, instance segmentation capabilities likely become less relevant, as the movement of all instances is likely already given explicitly via the conditioning.

Figure B. Jointly training the vision encoder is important. We train a model with a frozen pretrained vision encoder. The model struggles with instance-specificity, predicting the same movement for the woman's hand as it does for the man's hand. When jointly training the vision encoder with the flow poke transformer, the man's hand's movement does not directly influence the woman's hand.

When ablating the number of GMM components, we find that adding too many components reduces the model's performance when conditioned on low numbers of given pokes. Only predicting a single component results in better quantitative performance at low given poke counts, but, obviously prevents the model from predicting multimodal distributions (see, e.g., Figs. 1, 3 and E), omitting a central property of our model. Parametrizing the covariance matrices as a pure diagonal matrix as done in GIVT [41] results in slightly reduced performance on average. Qualitatively, it also prevents angled distributions that our model successfully uses to express directional uncertainty (see, e.g., Fig. 1).

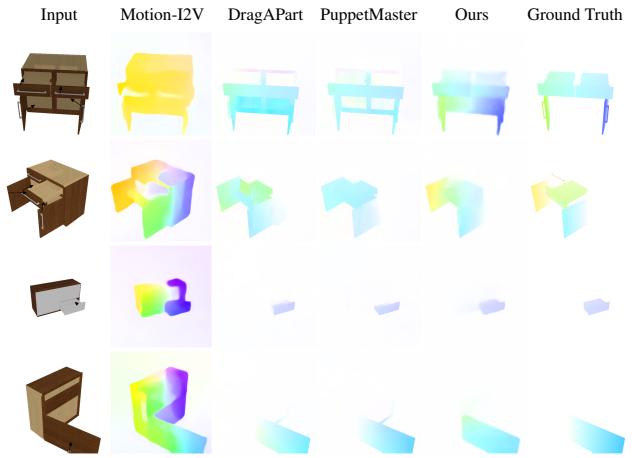


Figure C. **Qualitative Results for Articulated Object Motion Estimation**. We compare on Drag-A-Move [21] with Motion-I2V [35], DragAPart [21], and PuppetMaster [20]. Our model is qualitatively more capable of capturing complex conditioning with multiple different pokes than DAP and PM in this setup. Motion-I2V often fails to accurately follow the conditioning locally.

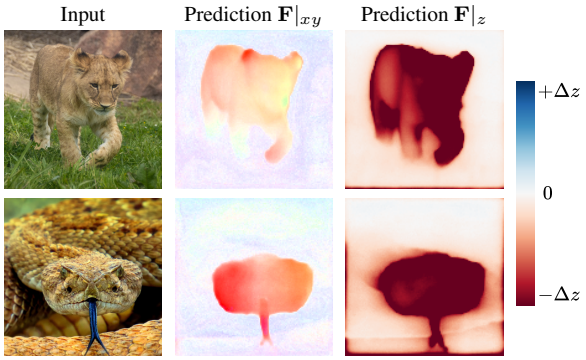


Figure D. **3D Motion Estimation**. We show unconditional 3D motion estimation samples from an FPT variant fine-tuned on 3D track data. The in-plane motion prediction $\mathbf{F}|_{xy}$ resembles that of a 2D FPT model, while this version can also successfully predict plausible out-of-plane motion $\mathbf{F}|_{z}$.

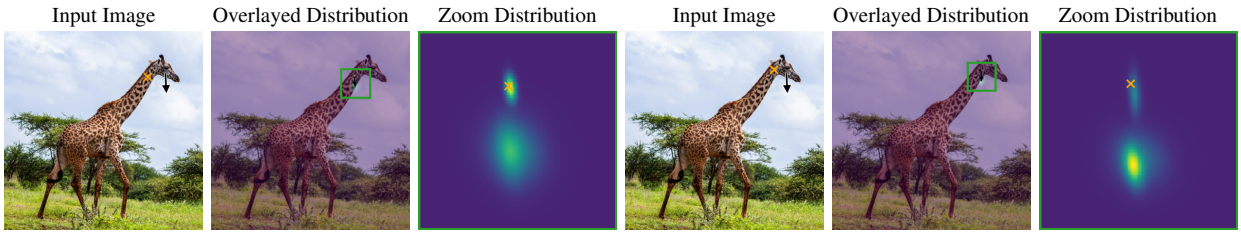
C. Additional Qualitative Examples

For additional context for our quantitative results in Tab. 2, we show visualizations of some samples from that experiment in Fig. C.

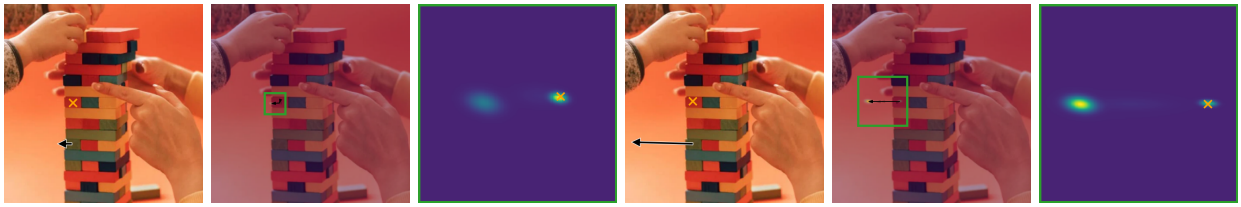
We also show additional qualitative examples for and pointwise motion predictions in Fig. E and samples for dense motion estimation in Fig. F.

D. Extension to 3D Motion

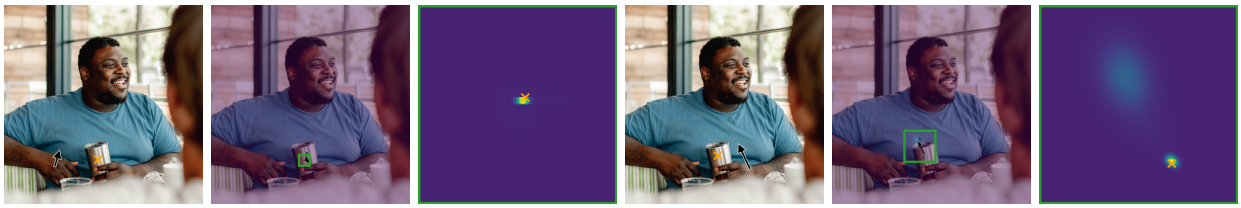
In this paper, we evaluated the Flow Poke Transformer in the two-dimensional setting, meaning that the model only reasons in the image plane. However, the architecture itself is not limited to this setting and can trivially be extended to higher dimensions if desired. We show qualitative motion prediction results from such a version in Fig. D, where the model also successfully predicts reasonable out-of-plane motion in full 3D. This model was obtained by continued training from a 2D FPT checkpoint with 3D trackers obtained using SpatialTrackerV2 [47, 48] on a subset of OpenVid1M [27]. The model is capable of predicting movement towards and away from the camera without any pokes, as shown by the near-static background and the clearly segmented motion of the animals in all three dimensions. When combining the predicted flow in Z-direction with a depth estimation of the input image, it is possible to tell which parts in the image will be occluded in the future.



(a) When the head of the giraffe is moving down, we get different flow distributions depending on how close the query is to the head. Since the head can also move down without the neck following, we get distributions with more emphasis on no movement when the query is further away from the head (first example). When the query gets really close to the head (second example), the likelihood of movement at the query also increases which can be seen in the stronger bottom mode.

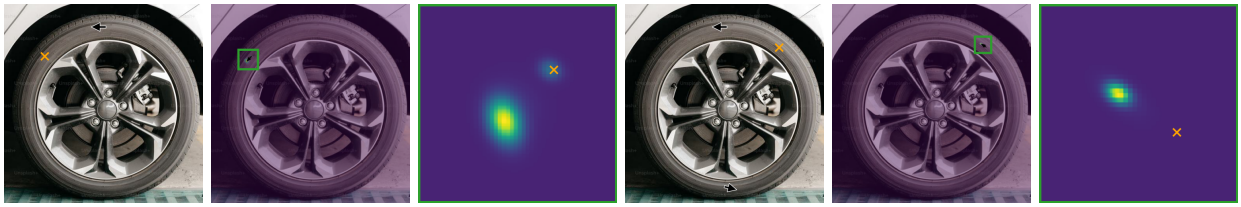


(b) The model accounts both for the possibility of the tower falling over with the brick's movement and with it staying stationary. The likelihood of the tower falling over depends on the velocity with which the brick is removed.



(c) Depending on which hand moves, the cup is predicted to be either stationary or *potentially* moving together with the hand holding it. Note that the case of the cup not moving with the hand holding it is very improbable, as visualized by the arrow pointing to that mode having substantially less opacity.

(d) Depending on the height of the position queried on the tree, the magnitude of the predicted movement changes, reflecting typical intuition as to how a tree moves.



(e) The model is capable of understanding the effect of rotational movements.

Figure E. Visualization of flow distribution for different pokes on the same image. The overlayed distribution visualizes the potential movement in the overall images, with the opacity of arrows denoting how likely each mode is (the more likely a mode, the less transparent the arrow).

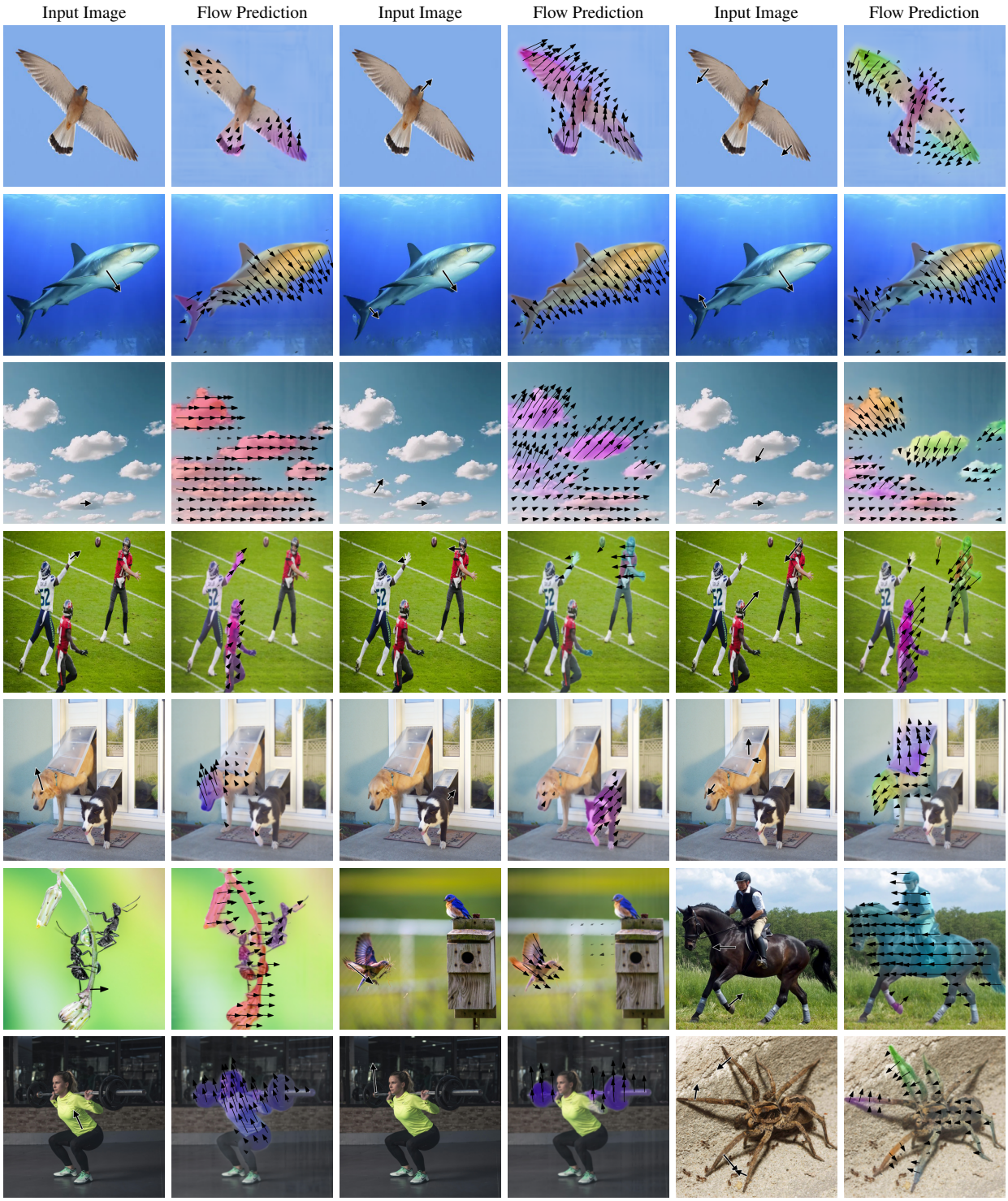


Figure F. Qualitative samples visualizing motion predictions inferred from a single image and (optionally) pokes.