We present the following items in the Appendix:

- The thin-plate splines warp computation (Section A) and its inverse (Section B)
- The detailed formulation for the semantics-aware refinement step (Section C)
- The layer occlusion model (Section D)
- A stochastic extension of our method (Section E)
- Detailed architectural choices for each of WALDO’s modules (Section F)
- More qualitative samples on nonrigid scenes (Section G)
- The influence of the choice of the pretrained segmentation and optical flow models (Section H)
- An ablation study of our inpainting strategy (Section I)
- Further information about our implementation and the overall training process (Section J)
- A statement about the societal impact of this project (Section K)
- A qualitative study of our approach (Section L)

A. Thin-plate splines warp computation

For clarity, vectors (or points) are denoted by bold face lower case letters, and matrices are denoted by bold face upper case letters throughout this presentation. Let \( \mathbf{p} = (x, y, 1)^T \) be a point in homogeneous coordinates, and \( \mathbf{C}_1 \) and \( \mathbf{C}_2 \) be two sets of such points in \( \mathbb{R}^{3 \times L} \), which we refer to as control points, with \( L \) the (fixed) number of points in each set. The thin-plate splines (TPS) [9] transformation which maps \( \mathbf{p} \) onto \( \mathbf{p}_{12} \) writes:

\[
\mathbf{p}_{12} = \mathbf{Tp} + \mathbf{U} \phi(\mathbf{p}, \mathbf{C}_1), \tag{5}
\]

where \( \phi(\mathbf{p}, \mathbf{C}_1) = [k(\mathbf{p}, \mathbf{p}_i)]_{\mathbf{p}_i \in \mathbf{C}_1} \) is a \( L \)-dimensional vector, and \( (\mathbf{T}, \mathbf{U}) \) are TPS parameters in the form of matrices of dimension \( 3 \times 3 \) and \( 3 \times L \) respectively. The transformation is decomposed into a global affine one (through \( \mathbf{T} \)) and a local non-affine one (through \( \mathbf{U} \) and \( \phi \)). The kernel function \( k \) is defined by \( k: (\mathbf{p}, \mathbf{q}) \rightarrow \|\mathbf{p} - \mathbf{q}\|_2 \log \|\mathbf{p} - \mathbf{q}\|_2 \) and materializes the fact that the TPS transformation minimizes the bending energy. The \( 3(3 + L) \) TPS parameters are found using the constraint that points from \( \mathbf{C}_1 \) should be mapped onto points from \( \mathbf{C}_2 \) using (5). This yields a system of \( 3L \) equations to which Bookstein adds 9 extra ones, as explained in [9], which we formulate as:

\[
\mathbf{C}_1 \mathbf{U}^T = \mathbf{0}_{3 \times 3}. \tag{6}
\]

By applying (5) on pairs of points from \( \mathbf{C}_1 \) and \( \mathbf{C}_2 \), and using extra constraints (6), we obtain the TPS parameters \( (\mathbf{T}, \mathbf{U}) \):

\[
\begin{bmatrix} \mathbf{T}^T \\ \mathbf{U}^T \end{bmatrix} = \Delta(\mathbf{C}_1)^{-1} \begin{bmatrix} \mathbf{C}_2^T \\ \mathbf{0}_{3 \times 3} \end{bmatrix}, \quad \Delta(\mathbf{C}_1) = \begin{bmatrix} \mathbf{C}_1^T & \Phi(\mathbf{C}_1, \mathbf{C}_1) \\ \mathbf{0}_{3 \times 3} & \mathbf{C}_1 \end{bmatrix}, \tag{7}
\]

where \( \Phi(\mathbf{C}_1, \mathbf{C}_1) \) is a \( L \times L \) matrix obtained by stacking \( \phi(\mathbf{p}_1, \mathbf{C}_1) \) for all points \( \mathbf{p}_1 \) in \( \mathbf{C}_1 \).

Hence, computing the TPS parameters amounts to inverting a \((L + 3) \times (L + 3)\) matrix, \( \Delta(\mathbf{C}_1) \). Since doing that for each new transformation is impractical, we use a fixed grid of control points associated with each layer for \( \mathbf{C}_1 \), as a proxy to compute the flow between pair of frames. By fixing the value of \( \mathbf{C}_1 \), the corresponding matrix inversion is done only once for the whole training; and we are still able to model different deformations by setting \( \mathbf{C}_2 \) to different values. We note that, once \( \Delta(\mathbf{C}_1)^{-1} \) has been computed, sampling the warp \( \mathbf{w} \) associated with a new \( \mathbf{C}_2 \) is done by finding the corresponding TPS parameterization using (5), and by sampling the deformations for each point \( \mathbf{p} \) in a dense grid \([1, H]\) \times \([1, W]\) using (7), where the spatial resolution \( H \times W \) can be arbitrarily large. Moreover, this process is fully differentiable with respect to \( \mathbf{C}_2 \) and each point \( \mathbf{p} \) as it involves simple algebraic operations. In the main paper, \( \mathbf{C}_1 \) and \( \mathbf{C}_2 \) correspond to the regular grid of control points \( \mathbf{q}_i^t \), associated with layer \( i \), and its deformation \( \mathbf{p}_i^t \) at time step \( t \) respectively. The warp \( \mathbf{w}_i^t \), also in the main paper, corresponds to the inverse transformation to the one presented here, and associates with every point \( \mathbf{p}_{12} \) corresponding to a pixel of the image (right side in Figure A1) the corresponding point \( \mathbf{p} \) in object coordinates (left side). However, computing the inverse transformation cannot be achieved by simply switching the roles of \( \mathbf{C}_1 \) and \( \mathbf{C}_2 \), since \( \mathbf{C}_1 \) have to be kept constant, which is why we resort to warp inversion whose simple formulation is detailed in the next section.
B. Inverse warp computation

Let \( w \) be a warp in \( \mathbb{R}^{2 \times H \times W} \) (where \( H \times W \) is a given spatial resolution), which we could also consider as a mapping from \( \mathbb{R}^2 \) to \( \mathbb{R}^2 \), where \( w(p) \) is the geometric transformation of a point \( p \) sampled on the grid \([1, H] \times [1, W]\). Such a mapping is not surjective with respect to the grid, that is, all the cells in the grid are not necessarily reached by \( w \). As a result, we approximate the inverse warp \( w^{-1} \) by the pixel-accurate inversion in cells for which such a direct mapping exists and use interpolation for filling others, starting from the cardinal neighbours of already inverted cells, and iteratively filling the remaining ones. The warp \( w^{-1} \) may need to point out of the grid for some cells, e.g., when an object moves out of the frame. However, this cannot be extracted from \( w \) which is only defined on the grid. To avoid interpolating wrong values in these cells, we only invert \( w \) in those which are close (as per a given threshold) to the ones initially reached by \( w \), and make others point to an arbitrary position outside of the grid layout. Finally, the same reasons which justify that training errors can backpropagate through a spatial transformer [42] also apply here.

C. Semantics-aware refinement

We now describe in more details the semantics-refinement step introduced in the main body of the paper. Let \( m_t \) be a soft mask in \([0, 1]^{H \times W}\) at a given time step \( t \) in \([1, T]\), and \( c \) be a soft class assignment in \([0, 1]^C\), both of them predicted for the same layer, and let \( s_t \) be an input (soft) semantic map in \([0, 1]^{C \times H \times W}\) also associated with time step \( t \). We denote by \( \bar{c} \), the mean semantic class on the spatio-temporal tube defined by the masks, which, like \( c \), is a vector in \([0, 1]^C\), and writes:

\[
\bar{c} = \sum_{t,h,w} [m_t \odot F(s_t, c)]_{(h,w)} / \sum_{t,h,w} [m_t]_{(h,w)},
\]

where \( \odot \) is the element-wise product, and \( F \) is a class-filtering function parameterized by \( c \) and applied to \( s_t \), that is, at a given spatial location \((h, w)\), the result of kipping dominant classes as per \( c \) in \( s_t \):

\[
[F(s_t, c)]_{(h,w)} = \frac{1}{1 + k_c} \sum_j \langle [s_t]_{(h,w)}, c + k_c \rangle,
\]

with \( k_c \) a constant which defines the degree at which low scoring classes will be filtered out. One can set \( k_c = 0 \) for full effect and greater values for filtering less. In practice, we fix the value of \( k_c \) to 0.1. Each mask \( m_t \) is updated by computing the \( L_1 \) distance between the semantic map \( s_t \) and the mean class \( \bar{c} \) at every location \((h, w)\):

\[
[m_t]_{(h,w)} = (1 - \| [s_t]_{(h,w)} - \bar{c} \|_1) [m_t]_{(h,w)}.
\]

D. Layer occlusion model

Our occlusion model is rather standard, but we include it here for completeness. Ordering scores \( o_i \) are used to filter non-visible parts in a layer \( i \) due to the presence of another layer \( j \) on top of it \((i.e., \quad o_i^j \ll o_i^j)\). The transparency \( m_t^i \) of layer \( i \) at time step \( t \) is updated as follows:

\[
m_t^i = m_t^i \odot \prod_{j \neq i} \left( 1 - \frac{o_j^i}{o_i^j + o_i^j} m_t^j \right),
\]

where \( \odot \) is the element-wise product whose right-hand side component has values between 0 (occluded) and 1 (visible).

E. Stochastic extension of WALDO

In our original description, motions produced by WALDO are purely deterministic and they converge towards the mean of all possible future trajectories given the past. Although it is sufficient for short-term prediction, one could be interested in modeling different behaviours for longer future horizons. For this reason, we propose an extension of WALDO to account for the intrinsically uncertain nature of the future by allowing multiple predictions.

Our approach, illustrated in Figure E1, builds upon generative adversarial networks (GANs) [30]. We consider the future layer prediction module as a generator which computes the future positions of control points given ones from the past. This generator is trained jointly with a discriminator which classifies trajectories as \textit{real or fake}. Both are playing a minimax game, where the discriminator is taught to correctly distinguish real from fake trajectories, while the generator tries to fool
Figure E1. **Stochastic future prediction.** We extend the future layer prediction module of WALDO to allow the prediction of multiple futures. The module itself is not changed, except that it now uses noise (as input and in attention modules). The major difference is that a trajectory discrimination module is used at train time to assist the model in producing a realistic control point trajectory instead of the mean trajectory. At inference, only the future layer prediction module is kept. See text for details.

the discriminator. Through this process we expect synthetic trajectories to gradually improve in realism and to capture multiple modes from the underlying data distribution.

We implement the discriminator with a transformer and use the WGAN loss introduced in [3] for training. In addition to this loss, we find that keeping the initial reconstruction term ($L_p$) is important for the stability of training. Moreover, we normalize gradients from both supervision signals (reconstruction and adversarial) so that they have matching contributions.

F. Detailed architectures

We detail inner operations of each of WALDO’s modules (for the dimensions used on Cityscapes [17]).

F.1. Layered video decomposition

**Input encoding.** Semantic and flow maps ($s$ and $f$) are concatenated and go through a series of $3 \times 3$ convolutional layers (Conv) with padding of 1 and stride of 2, each followed by layer norm (LN) and a GELU activation [35] (Act) to form downscaled features $y$ for a given time step.

**Layer feature extraction.** We combine features encoded from different time steps into ($Y$), and sum them with different embeddings corresponding to their temporal ordering and spatial positioning ($T$ and $P$) to form input (1). Layer features $Z_{obj}$ and $Z_{bg}$ are computed from input (2), which is the concatenation of object and background embeddings ($2a$ and $2b$), themselves expressed as the sum of layer ordering and spatial positioning embeddings ($S$, $L$, $O$ and $B$). The next operations consists in layer norm (LN), self-attention (Att) to update (2) by computing queries, keys and values for (1) but only keys and values for (2), and multi-layer perceptrons (MLP) with GELU activations [35].

**Control point positioning.** We apply similar operations to predict control points ($p_{obj}$, $p_{bg}$) for object and background layers at a given time step from associated features $y$. A key difference with the previous module is that this one is time-independent, and that inputs (1) and (2) play the same role in the transformer blocks. A fully-connected layer (FC) outputs the 3D position of each control point in each of the $16 + 1$ layers.

**Object masking.** This module predicts an alpha-transparency mask $a$ for an object from its associated features $z$. It is the reverse process compared to the input encoding module, that is, we replace convolutions with transposed ones for progressively upsampling feature maps $z$, and we decrease the size of features at each step instead of increasing it.
Object classification. This module predicts for each object represented by $z$ a soft class assignment $c$, by first averaging $z$ over its spatial dimensions, and then applying layer norm (LN), a fully-connected layer (FC), and a softmax activation (Act) to output $c$ as a categorical distribution over classes.

F.2. Future layer prediction

Past encoding. Past control points corresponding to objects (resp. the background) are transformed into vectors, each of them paired with one layer and one time step, using a fully-connected layer FC1 (resp. FC2). We apply a pooling operation to layer representations ($Z_{obj}$ and $Z_{bg}$) to reduce them to a single vector representing each layer. All these vectors are concatenated, summed with the suitable embeddings (T and P), and passed through two vanilla transformer blocks to produce encoded features $E$.

Future decoding. Future control points are obtained by initializing future representations (2) with some embeddings (T and P), and then alternating between transformer blocks with self-attention on future representations (2), and cross-attention from past to future ones (1 and 2).

F.3. Warping, inpainting and fusion

The final component of our approach is a U-Net [68] composed of 6 downscaling layers and 6 upscaling ones, with as many skip connections between the two branches. Each downscaling (resp. upscaling) layer divides (resp. multiplies) by 2 its input resolution using a $3 \times 3$ convolution (resp. transposed convolution) with a stride of 2, and multiplies (resp. divides) by 2 the size of features so that intermediate features (between the two branches) are of size 512. This module outputs RGB values to update certain regions of an image (filling missing background or object parts, adapting light effects or shadows in other parts), a mask indicating these regions, a score (of confidence) at each pixel location to allow fusing multiple views corresponding to the same image.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Operation</th>
<th>On</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>4×512×8×16</td>
</tr>
<tr>
<td>(T)</td>
<td>Embed.</td>
<td>-</td>
<td>4×512×1×1</td>
</tr>
<tr>
<td>(P)</td>
<td>Embed.</td>
<td>-</td>
<td>1×512×8×16</td>
</tr>
<tr>
<td>1 Sum/Reshape Y, T, P</td>
<td>512×512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S)</td>
<td>Embed.</td>
<td>-</td>
<td>1×512×4×4</td>
</tr>
<tr>
<td>(L)</td>
<td>Embed.</td>
<td>-</td>
<td>1×512×8×16</td>
</tr>
<tr>
<td>(O)</td>
<td>Embed.</td>
<td>-</td>
<td>16×512×1×1</td>
</tr>
<tr>
<td>(B)</td>
<td>Embed.</td>
<td>-</td>
<td>1×512×1×1</td>
</tr>
<tr>
<td>2a Sum/Reshape S, O</td>
<td>256×512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2b Sum/Reshape L, B</td>
<td>128×512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Concat.</td>
<td>2a, 2b</td>
<td>384×512</td>
</tr>
<tr>
<td>LN1</td>
<td>Norm.</td>
<td>1</td>
<td>512×512</td>
</tr>
<tr>
<td>LN2</td>
<td>Norm.</td>
<td>2</td>
<td>384×512</td>
</tr>
<tr>
<td>Att1</td>
<td>[512]×3</td>
<td>2/1</td>
<td>384×512</td>
</tr>
<tr>
<td>LN3</td>
<td>Norm.</td>
<td>2</td>
<td>384×512</td>
</tr>
<tr>
<td>MLP1</td>
<td>[2048, 512]</td>
<td>2</td>
<td>384×512</td>
</tr>
<tr>
<td>LN4</td>
<td>Norm.</td>
<td>2</td>
<td>384×512</td>
</tr>
<tr>
<td>Att2</td>
<td>[512]×3</td>
<td>2/1</td>
<td>384×512</td>
</tr>
<tr>
<td>LN5</td>
<td>Norm.</td>
<td>2</td>
<td>384×512</td>
</tr>
<tr>
<td>MLP2</td>
<td>[2048, 512]</td>
<td>2</td>
<td>384×512</td>
</tr>
<tr>
<td>$Z_{obj}$</td>
<td>Split/Reshape</td>
<td>2</td>
<td>16×512×4×4</td>
</tr>
<tr>
<td>$Z_{bg}$</td>
<td>Split/Reshape</td>
<td>2</td>
<td>1×512×8×16</td>
</tr>
</tbody>
</table>

Table F3. Control point positioning.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Operation</th>
<th>On</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>-</td>
<td>-</td>
<td>512×8×16</td>
</tr>
<tr>
<td>(P)</td>
<td>Embed.</td>
<td>-</td>
<td>512×8×16</td>
</tr>
<tr>
<td>1 Sum/Reshape y, P</td>
<td>128×512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Z_{obj}$</td>
<td>-</td>
<td>-</td>
<td>16×512×4×4</td>
</tr>
<tr>
<td>$Z_{bg}$</td>
<td>-</td>
<td>-</td>
<td>1×512×8×16</td>
</tr>
<tr>
<td>(S)</td>
<td>Embed.</td>
<td>-</td>
<td>1×512×4×4</td>
</tr>
<tr>
<td>(L)</td>
<td>Embed.</td>
<td>-</td>
<td>1×512×8×16</td>
</tr>
<tr>
<td>(O)</td>
<td>Embed.</td>
<td>-</td>
<td>16×512×1×1</td>
</tr>
<tr>
<td>(B)</td>
<td>Embed.</td>
<td>-</td>
<td>1×512×1×1</td>
</tr>
<tr>
<td>2a Sum/Reshape $Z_{obj}$, S, O</td>
<td>16×512×4×4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2b Sum/Reshape $Z_{bg}$, L, B</td>
<td>1×512×8×16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Concat.</td>
<td>2a, 2b</td>
<td>384×512</td>
</tr>
<tr>
<td>3</td>
<td>Concat.</td>
<td>1, 2</td>
<td>512×512</td>
</tr>
<tr>
<td>LN1</td>
<td>Norm.</td>
<td>3</td>
<td>512×512</td>
</tr>
<tr>
<td>Att1</td>
<td>[512]×3</td>
<td>3</td>
<td>512×512</td>
</tr>
<tr>
<td>LN2</td>
<td>Norm.</td>
<td>3</td>
<td>512×512</td>
</tr>
<tr>
<td>MLP1</td>
<td>[2048, 512]</td>
<td>3</td>
<td>512×512</td>
</tr>
<tr>
<td>LN3</td>
<td>Norm.</td>
<td>3</td>
<td>512×512</td>
</tr>
<tr>
<td>Att2</td>
<td>[512]×3</td>
<td>3</td>
<td>512×512</td>
</tr>
<tr>
<td>LN4</td>
<td>Norm.</td>
<td>3</td>
<td>512×512</td>
</tr>
<tr>
<td>MLP2</td>
<td>[2048, 512]</td>
<td>3</td>
<td>512×512</td>
</tr>
<tr>
<td>2</td>
<td>Split</td>
<td>3</td>
<td>384×512</td>
</tr>
<tr>
<td>FC</td>
<td>[3]</td>
<td>2</td>
<td>384×3</td>
</tr>
<tr>
<td>$p_{obj}$</td>
<td>Split/Reshape</td>
<td>2</td>
<td>16×3×4×4</td>
</tr>
<tr>
<td>$p_{bg}$</td>
<td>Split/Reshape</td>
<td>2</td>
<td>1×3×8×16</td>
</tr>
</tbody>
</table>
G. Additional visual results on nonrigid scenes

Figure G2. Future prediction from \( T = 4 \) frames on Taichi-HD (256 x 256). Nonrigid motions can be visualized by the associated warps, predicted from the control points between \( T \) and \( T + 10 \) (colors represent different directions).

We further illustrate (Figure G2) that the motion representation proposed in WALDO allows us to represent nonrigid object deformations by training on Taichi-HD [73] dataset. Our approach can handle complex human motions such as leaning forward / backward, moving one leg while keeping the other on the ground, raising individual arms...

H. Influence of the choice of the pretrained segmentation and optical flow models

The use of pretrained networks in prior works vary widely according to the level of information required by each method, and depending on what was available at the time to extract these information. We summarize the differences in Table H1.

Table H1. Pretrained models used by different methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Optical flow estimation</th>
<th>Semantic segmentation</th>
<th>Instance segmentation</th>
<th>Depth estimation</th>
<th>Object tracking</th>
<th>Video Frame Interpolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPVFI [100]</td>
<td>RAFT [77]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>RIFE [39]</td>
</tr>
<tr>
<td>VPCL [29]</td>
<td>RAFT [77]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Vid2vid [90]</td>
<td>FlowNet2 [40]</td>
<td>DeepLabV3 [14]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SADM [7]</td>
<td>PWCNet [76]</td>
<td>DeepLabV3 [14]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WALDO</td>
<td>RAFT [77]</td>
<td>DeepLabV3 [14]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

We thus evaluate the influence of the choice of the pretrained segmentation and optical flow models to WALDO’s performance. Results on Cityscapes and KITTI test set, obtained by substituting segmentation model DeepLabV3 [14] with MobileNetV2 [70] or ViT-Adapter [15], and optical flow model RAFT [77] with PWCNet [76], are presented in Table H2.

Table H2. Ablation studies of optical flow estimation and semantic segmentation methods on the Cityscapes and KITTI test sets. Like in the main paper, we compute multi-scale SSIM (\( \times 10^3 \)) and LPIPS (\( \times 10^3 \)) for the \( k \)th frame and report the average for \( k \) in \( [1, K] \).

<table>
<thead>
<tr>
<th>Flow</th>
<th>Segmentation</th>
<th>Cityscapes</th>
<th> </th>
<th> </th>
<th> </th>
<th> </th>
<th> </th>
<th> </th>
<th> </th>
</tr>
</thead>
<tbody>
<tr>
<td>[76]</td>
<td>[70]</td>
<td>[77]</td>
<td>[14]</td>
<td>[15]</td>
<td>( K = 1 )</td>
<td>( K = 5 )</td>
<td>( K = 10 )</td>
<td>( K = 1 )</td>
<td>( K = 3 )</td>
</tr>
<tr>
<td>✓ ✓</td>
<td>✓</td>
<td>947 062 836 122</td>
<td>753 175</td>
<td></td>
<td>856 116 760 171</td>
<td>697 214</td>
<td></td>
<td>859 112 756 166</td>
<td>692 209</td>
</tr>
<tr>
<td>✓ ✓</td>
<td>✓</td>
<td>954 055 849 111</td>
<td>768 165</td>
<td></td>
<td>867 108 766 163</td>
<td>702 206</td>
<td></td>
<td>866 109 767 163</td>
<td>703 205</td>
</tr>
</tbody>
</table>

Despite PWCNet [76] having approximately twice the average end-point-error of RAFT [77], using it instead of RAFT only results in a small performance drop for video prediction with WALDO. This is in line with the conclusions of Geng et al. in [29] who conducted a similar comparison. Moreover, replacing DeepLabV3 [14] with MobileNetV2 [70], with respective segmentation performance of 81 and 75 in terms of test set mIoU on Cityscapes, yields only little loss of video prediction quality. Conversely, using a more advanced segmentation model like ViT-Adapter [15], with 85 test set mIoU on Cityscapes, does not change the results substantially. Our interpretation is that the segmentation quality is not the limiting factor for WALDO’s performance once it has reached a sufficient level. To conclude, WALDO is quite robust to the choice of the pretrained segmentation and optical flow models.
I. Ablation study of the inpainting strategy

<table>
<thead>
<tr>
<th>Adversarial inpainting [51]</th>
<th>Temporal consistency</th>
<th>$K = 1$</th>
<th>$K = 5$</th>
<th>$K = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSIM ↑ LPIPS ↓</td>
<td>SSIM ↑ LPIPS ↓</td>
<td>SSIM ↑ LPIPS ↓</td>
<td>FVD ↓</td>
</tr>
<tr>
<td>✓</td>
<td>957 049 853 105 770 158 061</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>957 049 854 105 771 158 057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓ ✓</td>
<td>957 049 854 105 771 158 055</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure I3. Visual ablation of our inpainting strategy. We start off with a method that fills in empty regions with the sole objective to minimize the reconstruction error (✓), we then propose to leverage an off-the-shelf adversarial inpainting (adv. inp.) method [51] to improve realism, and we finally illustrate our full strategy where we use the predicted flow to ensure the temporal consistency (temp. const.) of inpainted regions. For clarity, these regions are indicated in the last row. Please zoom in for details.

Figure I3. Visual ablation of our inpainting strategy. We start off with a method that fills in empty regions with the sole objective to minimize the reconstruction error (✓), we then propose to leverage an off-the-shelf adversarial inpainting (adv. inp.) method [51] to improve realism, and we finally illustrate our full strategy where we use the predicted flow to ensure the temporal consistency (temp. const.) of inpainted regions. For clarity, these regions are indicated in the last row. Please zoom in for details.

Given that we use an off-the-shelf inpainting method [51] trained on external data [111], we assess the impact of the use of such a model in our approach on quantitative measurements.

Results are presented in Table I1 and show that although gains in SSIM and FVD are possible, these gains remain small. In addition, no perceptible change is observed on LPIPS. Our temporal consistency strategy, which consists in filling frames one by one and using the predicted flow to propagate new contents into subsequent frames, allow small extra gains on the FVD metric.

The benefits from our inpainting strategy are more visible in qualitative samples presented in Figure I3. Without adversarial inpainting (✓), empty image regions are filled using a model trained to minimize the reconstruction error. We observe that this results in blurry image parts with important artefacts. Using adversarial inpainting produces much more realistic images when considering frames individually, but filling each of them independently is still not very natural (best viewed in the videos included in the project webpage). Our approach for producing temporally-consistent outputs is able to solve this
issue. For example, in the left-most sample sequence of the third row of Figure I3, we see that inpainted image parts match between different time steps although the camera is moving.

J. Further implementation details

We follow [99] and group semantic classes which form a consistent entity together, e.g., riders with their bicycle, traffic lights/signs with the poles, which allows us to represent those within a single object layer. We use horizontal flips, cropping, and color jittering as data augmentation. Although we encounter signs of over-fitting in some experiments, validation curves do not increase during training, nor after convergence. So best model selection is not necessary, and we always save the last checkpoint. We find that a good initialization of object regions helps the layer decomposition module to reach a better optimum and to converge faster. Hence, we add a warmup period during which the module is trained without flow reconstruction ($\lambda_f = 0$), and then progressively increase the associated parameter during training ($\lambda_f > 0$) once we start having good object proposals. Without doing so, the module tends to rely on the background layer alone to reconstruct the scene motion, which, as a result, leads to an under-use of the object layers. We also find that removing data-augmentation in the last few epochs when training the warping, inpainting and fusing module slightly improves the performance at inference time.

K. Societal impact

The total cost for this project, including architecture and hyperparameter search, training, testing and comparisons with baselines has been around 25K GPU hours, with an associated environmental cost of course. On the other hand, we strive to minimize this cost by decomposing video prediction into efficient lightweight modules, and our approach will hopefully contribute to eventually improve the safety of autonomous vehicles, by, say, predicting the motion of nearby agents.

L. Qualitative study of WALDO

In this section, we provide qualitative samples for each of the three modules which compose WALDO.

Layered video decomposition. We illustrate in Figure L1 our strategy to decompose videos into layers as a way to build inter-frame connections using a compact representation of motion from which we recover the dense scene flow.

We observe that objects are predicted in regions which match our pseudo ground truth, constructed from input segmentation and flow maps, even in difficult regions like the poles. Although they share the same semantic class, the three cars in the left-most example in Figure L1 are correctly segmented into different objects. Still, it may happen that multiple objects are merged into the same layer, or that an object is over-segmented into multiple layers (like the ego vehicle in these examples). This is due, in part, to the limitations of our approach which indicates regions of interest for the objects, but does not impose how they should be split among the different layers. When computing video decompositions, we also position a set of control points associated with each layer. The delta of control points between pairs of time steps produces sparse motion vectors for the background and the objects. We show that, although we use a small number of points, we are able to accurately recover the dense scene flow using TPS transformations, and that motion discontinuities occur at layer boundaries as expected.

Future layer prediction. We compare in Figure L2 the scene dynamics extracted from our decomposition strategy to the one inferred via the future layer prediction module.

We accurately reconstruct complex motions under various scenarios: when the background is static, moves towards the camera due to the ego motion, or sideways when the car is turning; in the presence of different kinds of objects such as trucks, cars, bikes or various road elements; and whether these objects move in the same direction or not.

Warping, fusion and inpainting. We illustrate in Figure L3 how future frames are finally synthesized.

Warping past frames to obtain future ones is not enough, as some regions may not be recovered from the past. In particular, we see that shadows may not always be consistent with the new position of objects, e.g., for the vehicle in the bottom-right example in Figure L3. Our fusion and inpainting strategy is able to fill empty regions with realistic and temporal-consistent content (see also Sec I), and handles shadows reasonably well. Finally, we show that by fusing multiple views from the past context, WALDO is able to reduce disocclusions significantly (from the grey + black to only black regions in the last row of the figure).
Figure L1. Visualization of the layered video decomposition. Semantic segmentation maps and optical flow maps are extracted from RGB frames using off-the-shelf methods [14, 77]. We combine both to construct pseudo ground truths for object discovery: in **white**, moving foreground regions towards which objects are attracted; in **grey**, static foreground ones which remain neutral; and in **black**, the background which repulses objects. We then show predicted object regions and their decomposition into layers. Each layer is tracked over time using a small set of control points. We compute motion vectors between points in pairs of frames (here, between consecutive time steps), and reconstruct from these and the layer masks the complex scene flow. Please zoom in for details.
Figure L2. Visualization of future layer prediction. We use control points from the layered video decomposition as supervision. We compare motion vectors reconstructed from these points to the ones predicted for up to time step $T + 10$ from a context of $T=4$ past frames. The motion vectors are computed between time step $T$ and time step $t$ in $\{T + 1, T + 10\}$. Different colors correspond to different layers. Please zoom in for details.
Figure L3. Visualization of the warping, fusion and inpainting module. We use the layer decomposition computed on a whole video to reconstruct the last 10 frames using the $T = 4$ first ones as context. This is done by warping, inpainting and fusing different views from the context. We compare real frames, warped ones, and fused/inpainted ones. We also illustrate the effect of disocclusion, by showing in the last row, in **black**, regions which are not visible in any frame of the context, and, in **grey**, those which are not visible in some frames but visible in others. Please zoom in for details.
References

[17] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes dataset for semantic urban scene understanding. In CVPR, 2016. 2, 5, 16
References

[69] Masaki Saito, Shunta Saito, Masanori Koyama, and Susuke Kobayashi. Train sparsely, generate densely: Memory-efficient unsupervised training of high-resolution temporal GAN. *IJCV*, 2020. 2


[72] Baoguang Shi, Xinggang Wang, Pengyuan Lyu, Cong Yao, and Xiang Bai. Robust scene text recognition with automatic rectification. In *CVPR*, 2016. 2


[76] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, 2018. 9, 19

[77] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *ECCV*, 2020. 2, 3, 5, 9, 19, 22


[96] Bohan Wu, Suraj Nair, Roberto Martin-Martin, Li Fei-Fei, and Chelsea Finn. Greedy hierarchical variational autoencoders for large-scale video prediction. In *CVPR*, 2021. 1


[99] Yue Wu, Rongrong Gao, Jaesik Park, and Qifeng Chen. Future video synthesis with object motion prediction. In *CVPR*, 2020. 1, 2, 6, 7, 19, 21

[100] Yue Wu, Qiang Wen, and Qifeng Chen. Optimizing video prediction via video frame interpolation. In *CVPR*, 2022. 1, 2, 6, 7, 19


[110] Yunzhi Zhang and Jiajun Wu. Video extrapolation in space and time. In ECCV, 2022. 6