SlotODE: Object-Centric Representation Learning via Continuous-Time Neural Dynamical Systems

Om Kathira

May 2026

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the Senior Honors Thesis.

Advised by Dr. Tai-Sing Lee.

This work was supported in-kind by the Google TPU Research Cloud (TRC) program.

Keywords: Deep Learning, Object-Centric Representation Learning, Neural Networks, Slot Attention, Dynamical Systems, Neural Ordinary Differential Equations (Neural ODEs).

Abstract

Learning object-centric representations of complex scenes is a promising step toward enabling efficient abstract reasoning from low-level perceptual features. However, most existing approaches treat object decomposition as a discrete computational procedure, limiting both our understanding of how such structure emerges and our ability to control it at inference time. We introduce SlotODE, a continuous-time formulation of Slot Attention in which iterative refinement is recast as a dynamical system. In our approach, slots evolve under a learned vector field parameterized by a neural ordinary differential equation, enabling analysis of the decomposition process through the lens of dynamical systems. Empirically, SlotODE matches Slot Attention on unsupervised object decomposition at similar parameter counts, outperforming it on unsupervised object segmentation. It also provides an interpretable framework to study the emergence of object-centric structure from a competitive attention mechanism. Lastly, this perspective naturally allows the use of adaptive ODE solvers to dynamically allocate compute based on scene complexity.

Acknowledgments

I'm deeply grateful to Dr. Tai-Sing Lee from the Computer Science Department and Neuroscience Institute for being an invaluable research advisor and personal mentor for my thesis. I'd like to briefly thank my academic advisor, Dr. Carrie Doonan, for her constant support throughout. This thesis wouldn't be the learning experience it was without them.

1 Introduction

The visual world presents the brain with a continuous stream of low-level features - edges, contours, colors, motion, etc - distributed across the retina. Yet, our conscious experience is not of fragmented features but of coherent objects - a red cup resting on a wooden table, a dog moving across grass, etc. How neural populations group local features into discrete object-level representations is known as the *binding problem*, one of the oldest open questions in neuroscience [1, 2]. Decades of behavioral and physiological work suggest that object perception arises from competitive interactions between recurrently connected neural populations where features belonging to one object are selectively enhanced and others suppressed [3]. Certain intermediate visual areas encode object-level information such as border ownership and figure-ground organization [4, 5]. Critically, this information is refined by the *dynamics* of cortical circuits in which populations of neurons evolve continuously in time under coupled differential equations that govern membrane potentials, synaptic currents, and firing rates [6].

In machine learning, the analogous problem of unsupervised scene decomposition can be solved by learning *object-centric representations*. This line of work is motivated by the hypothesis that structured, compositional representations are prerequisites for generalization and reasoning [7, 8, 9]. They can be used to improve machine learning algorithms across various domains like visual reasoning [10], modeling structured environments [11], multi-agent modeling [12, 13, 14], and simulations of physical systems [15, 16, 17]. The dominant architecture in this space is *Slot Attention* [18], which introduces a set of learnable representation vectors, called slots, that compete for ownership of image features through iterative soft attention - allowing them to bind to distinct objects.

The concept of iterative refinement - producing an output z as the limit of a learned update $\mathrm{z} \leftarrow f_\theta(\mathrm{z}, x)$ - recurs across modern machine learning. Energy-based models [19, 20], deep equilibrium models [21], meta-learning algorithms [22, 23, 24], object-centric models [25, 26, 27, 28], and diffusion models [29, 30, 31, 32] all admit some form of iterative refinement. However, in each case, the underlying motivation is dynamical even though the implementation typically uses a discrete map.

In line with this idea, Slot Attention uses a discrete formulation of iterative refinement that is convenient but restrictive - obscuring the structure of the refinement process itself. We cannot easily ask whether slots converge to stable configurations, how quickly they do, or how the shared attention mechanism evolves along the trajectory. These are natural questions in the language of dynamical systems, but they become answerable only when the refinement process is expressed as a continuous-time flow rather than a discrete map.

In this work, we introduce *SlotODE*, a continuous-time reformulation of Slot Attention in which iterative refinement is recast as integration of a learned vector field. Slots evolve under an autonomous *neural ordinary differential equation* [33] and the competitive attention mechanism becomes a term in this vector field rather than a step in a discrete loop. The reformulation does not change what the architecture computes - rather, it changes the structure of computation. We show that SlotODE matches Slot Attention on unsupervised

object decomposition - reaching near-identical ARI-FG[1] on CLEVR[2] at a comparable parameter count and outperforms Slot Attention on unsupervised object segmentation with higher mIoU[3]. The continuous-time integration naturally allows the use of adaptive ODE solvers that allocate compute based on local error estimates. Instead of committing to a fixed number of iterations, we observe that per-scene compute budget tracks scene complexity without any explicit design for this behavior. Beyond adaptive computation, the autonomous formulation makes tools from dynamical systems available to study the trained model as Jacobian spectra and contraction measures become well-defined empirical quantities of the learned vector field.

A summary of our main contributions.

- We reformulate Slot Attention as an autonomous neural ODE, replacing the discrete GRU-based update with continuous-time integration of a learned vector field.
- We show that SlotODE matches Slot Attention on unsupervised object decomposition (using metric ARI-FG) at a comparable parameter count on CLEVR.
- We show that SlotODE outperforms Slot Attention on unsupervised object segmentation (using metric mIoU) at a comparable parameter count on CLEVR.
- We demonstrate that adaptive ODE solvers applied to the learned vector field produce per-scene compute budgets that track scene complexity without any explicit design for it.
- We develop and apply an analytical framework for studying the learned slot dynamics directly - through trajectory geometry, Jacobian spectra, and contraction measures - none of which have clean analogues under the discrete formulation.

[1] Adjusted Rand Index - Foreground (ARI-FG) is a clustering-based pixel assignment metric commonly used to evaluate object-centric representation learning models like Slot Attention.

[2] See Appendix A.

[3] Mean Intersection-over-Union (mIoU) is a standard metric used to evaluate segmentation performance by measuring overlap between predicted and ground-truth masks.

2 Background

2.1 Slot Attention

Slot Attention introduces a set of learnable representation vectors called *slots* that compete for explanatory ownership of image features through iterative soft attention. Given a visual scene encoded by a CNN backbone as a set of spatial feature vectors $F = \{f_1, \ldots, f_M\} \subset \mathbb{R}^{D_{\text{enc}}}$, the mechanism maintains N slot vectors $s_1, \ldots, s_N \in \mathbb{R}^{D_{\text{slot}}}$ and refines them over T discrete iterations. At each iteration t, slots attend to encoded features via scaled dot-product attention [34] as follows:

$$q_i^{(t)} = \text{LayerNorm}\big(s_i^{(t)}\big)\, W_Q, \qquad k_j = f_j\, W_K, \qquad v_j = f_j\, W_V \tag{1}$$

$$a_{ij}^{(t)} = \frac{\exp\big(\frac{\langle q_i^{(t)},\, k_j\rangle}{\sqrt{D_{\text{slot}}}}\big)}{\sum_{i'=1}^{N} \exp\big(\frac{\langle q_{i'}^{(t)},\, k_j\rangle}{\sqrt{D_{\text{slot}}}}\big)} \tag{2}$$

Note that the softmax in Equation 2 normalizes over slots (index i) instead of spatial positions (index j). This normalization axis forces slots to compete with one another. Each feature position distributes its information across slots in proportion to their queries, so a slot must "win" a region of the image to receive its features. After renormalization over spatial positions, the weighted readout $\hat{a}_i^{(t)} = \sum_j \bar{a}_{ij}^{(t)} v_j$ is fed through a Gated Recurrent Unit (GRU) [35] to update the slot's state:

$$s_i^{(t+1)} = \text{GRU}(\hat{a}_i^{(t)},\ s_i^{(t)}) \tag{3}$$

After T iterations, the final slots $s_i^{(T)}$ are decoded independently through a spatial broadcast decoder [36]. Each slot produces a reconstruction and a mask. The masks compete via per-pixel softmax to form the final composite image. The entire architecture is trained end-to-end with pixel-level mean squared error (MSE), and importantly, no ground-truth segmentation labels are used. Slot Attention has been widely successful, achieving strong unsupervised object decomposition on synthetic benchmarks and inspiring a family of extensions to video [25], language [37], and real-world scenes [38].

For reference, the architecture is concisely described as follows.

Algorithm 1 Slot Attention [18]

Require: Image $x \in \mathbb{R}^{3\times H\times W}$, number of slots N, number of iterations T
Ensure: Reconstruction $\hat{x}$, slot masks M

$F \leftarrow \text{Encoder}(x)$ ▷ $F \in \mathbb{R}^{P\times D_{\text{enc}}}$, $P = H \times W$
$K \leftarrow FW_K, \quad V \leftarrow FW_V$ ▷ Key and value projections
$s_i^{(0)} \sim \mathcal{N}(\mu, \sigma^2)$ for $i = 1, \ldots, N$ ▷ Sample initial slots
for $t = 0, \ldots, T-1$ **do**
  $Q^{(t)} \leftarrow \text{LayerNorm}(S^{(t)})\, W_Q$ ▷ Query projection
  $A_{ij}^{(t)} \leftarrow \text{Softmax}_i\big(q_i^{(t)} \cdot k_j \,/\, \sqrt{D}\big)$ ▷ Softmax over slots (competition)
  $\bar{A}_{ij}^{(t)} \leftarrow A_{ij}^{(t)} \,/\, \sum_{j'=1}^{P} A_{ij'}^{(t)}$ ▷ Normalize over spatial positions
  $\hat{a}_i^{(t)} \leftarrow \sum_{j=1}^{P} \bar{A}_{ij}^{(t)}\, v_j$ ▷ Weighted value aggregation
  $s_i^{(t+1)} \leftarrow \text{GRU}\big(\hat{a}_i^{(t)},\ s_i^{(t)}\big)$ ▷ Recurrent slot update
  $s_i^{(t+1)} \leftarrow s_i^{(t+1)} + \text{MLP}\big(\text{LayerNorm}(s_i^{(t+1)})\big)$ ▷ Residual MLP
end for
$\hat{x}_i,\ m_i \leftarrow \text{Decoder}(s_i^{(T)})$ for $i = 1, \ldots, N$ ▷ Per-slot reconstruction + mask logit
$M_i \leftarrow \text{Softmax}(m_1, \ldots, m_N)_i$ ▷ Pixel-wise mask competition
$\hat{x} \leftarrow \sum_{i=1}^{N} M_i \odot \hat{x}_i$ ▷ Composite reconstruction

2.2 Neural Ordinary Differential Equations

A parallel line of work has shown that many discrete iterative architectures naturally admit a continuous-time formulation. Observe that the residual update $\mathrm{z}^{(t+1)} = \mathrm{z}^{(t)} + f_\theta(\mathrm{z}^{(t)})$ of a residual network [39] is an Euler discretization of an ordinary differential equation (ODE) [33]:

$$\frac{d\mathrm{z}}{dt} = f_\theta\big(\mathrm{z}(t), t\big), \qquad \mathrm{z}(t_1) = \mathrm{z}(t_0) + \int_{t_0}^{t_1} f_\theta\big(\mathrm{z}(\tau), \tau\big)\, d\tau \tag{4}$$

The forward pass of a deep network can be viewed as integrating a dynamical system [40, 41]. When the vector field f_θ does not depend explicitly on t, the system is considered *autonomous*:

$$\frac{d\mathrm{z}}{dt} = f_\theta(\mathrm{z}) \tag{5}$$

Autonomous systems provide a natural setting for analyzing dynamical systems. A *fixed point* z^* satisfying $f_\theta(\mathrm{z}^*) = 0$ corresponds to an equilibrium where the state no longer evolves. The local behavior near z^* is characterized by the Jacobian $J = \frac{\partial f_\theta}{\partial \mathrm{z}}\big|_{\mathrm{z}^*}$. If every eigenvalue λ of J satisfies $\mathrm{Re}(\lambda) < 0$, the fixed point is a *stable attractor*, and nearby trajectories converge to it exponentially at a rate governed by the spectral gap. If some eigenvalues have a positive real part, the fixed point is unstable and perturbations grow. The trace $\mathrm{tr}(J) = \sum_i \mathrm{Re}(\lambda_i)$ determines whether the flow is locally *volume-contracting* ($\mathrm{tr}(J) < 0$, the system is dissipative) or locally *volume-expanding* ($\mathrm{tr}(J) > 0$).

Beyond richer analysis, the ODE perspective offers practical benefits. *Adaptive solvers* like the Dormand-Prince method adjust their step size based on local error estimates. They take larger steps when the vector field is smooth and smaller steps when it changes rapidly. This provides a principled mechanism for dynamically allocating computation. Intuitively, regions of state space where dynamics are "easy" (e.g. slot states are barely changing), the field can be integrated through quickly, while regions where dynamics are "hard" (e.g. slots are actively competing), can be integrated through more carefully. This allows computation that can adapt to scene complexity without any explicit design for it.

2.3 Neural Networks as Dynamical Systems

The idea that depth in a neural network can be understood as time in a dynamical system represents a shift in how we think about what deep networks do. A standard feedforward network applies a sequence of transformations $z^{(\ell+1)} = g_\ell(z^{(\ell)})$, and our analysis of its behavior is largely empirical. We can probe activations, visualize features, or measure gradients. When we instead view the same computation as integration of a continuous vector field, tools used to analyze dynamical systems become useful.

This perspective has been thoroughly developed for transformers. [42] first recognized that the residual connections in transformer blocks can be interpreted as a Lie-Trotter splitting scheme, a classical numerical method for operator splitting in differential equations. In this view, self-attention and the feed-forward network (FFN) are not two arbitrary sequential operations but two parts of a vector field, each contributing a term to the overall dynamics of the hidden state.

In time, [43, 44] developed a rigorous mathematical framework treating transformer layers as a particle system. Each token is a particle and self-attention can be thought of as an interaction kernel between particles. Under simplifying assumptions (shared weights across layers and identity KQV matrices), they show that the dynamics converge to clusters where tokens collapse toward shared representations at infinite depth. Their analysis uses tools from optimal transport and mean-field theory, connecting the softmax attention kernel to interaction energies studied in statistical mechanics.

Recently, [45] took this further by directly parameterizing transformer weights as functions of a continuous layer index - a non-autonomous neural ODE where the vector field varies smoothly with depth. Their spectral analysis of the attention mechanism reveals that trained models exhibit *increasing* eigenvalue magnitudes over depth, contradicting the weight-sharing assumption in the theoretical analyses of [43]. They also leverage Lyapunov exponents to study token-level sensitivity - how much a perturbation to one input token affects the output at another position. This connection between dynamical stability and model interpretability is powerful - the Lyapunov exponent is a single number that quantifies whether a particular input-output relationship is stable or chaotic, something that attention maps alone cannot capture.

More practically, [46] apply a related dynamical systems view to Slot Attention itself, reformulating its iterative refinement as a *fixed-point iteration* rather than an unrolled computation graph. They observe that the slot update in Equations 1–3 has the structure $s^{(t+1)} = F_\theta(s^{(t)}, x)$ for features x, and that well-trained

Slot Attention models produce iterates with small forward residuals $\|F_\theta(\mathrm{s}^{(t)}, \mathrm{x}) - \mathrm{s}^{(t)}\|$, consistent with convergence to an approximate fixed point. They exploit this by training with implicit differentiation [21], computing gradients through the fixed point directly via the implicit function theorem rather than backpropagating through unrolled iterations. Importantly, their analysis remains within the discrete-map framework where slots evolve through a sequence of applications of F_θ. Our work formalizes this by treating slot refinement as integration of a learned vector field rather than repeated application of an update rule.

What emerges from this body of work is a general research direction: *reformulating a neural architecture as a dynamical system does not require changing what the architecture computes, but changes what we can say about how it computes*. The discrete-to-continuous shift trades a sequence of opaque layer-wise transformations for a vector field whose properties - fixed points, stability, sensitivity, attractor structure, etc - are grounded by well-developed mathematical theory.

This is the lens through which we approach Slot Attention. The iterative refinement in Equations 1–3 already *looks* like a dynamical system - slots evolve, attention sharpens, and masks converge - but the discrete GRU-based formulation hides the underlying dynamics. Several specific questions highlight this:

- **Convergence.** Do slots converge to a stable terminal configuration? If they do, how fast? The answer determines whether additional iterations improve or degrade object decomposition.
- **Stability.** Is the linearization of the learned vector field at the reached state consistent with a stable attractor, and does this hold consistently across scenes?
- **Adaptive computation.** Can we let learned dynamics determine how many refinement steps are needed rather than fixing T? Some scenes are simple and some complex. The ODE framework provides this via adaptive solvers at no architectural cost.

None of these questions can be addressed as rigorously within the standard Slot Attention framework. The GRU update is a discrete map with internal gates, not a continuous vector field. There is no well-defined velocity or notion of integration time that can be continuously varied. Reformulating the dynamics as a neural ODE is what makes these questions answerable.

3 Methods

SlotODE follows the encoder-bottleneck-decoder structure of Slot Attention [18], replacing the discrete GRU-based refinement loop with a continuous-time dynamical system. We denote images by $x \in \mathbb{R}^{3\times H\times W}$ and the encoded features by $F \in \mathbb{R}^{P\times D_{\text{enc}}}$, where $P = H \times W$ is the number of spatial positions and D_{enc} is the encoder feature dimension. The model maintains N slot vectors of dimension D_{slot}, collectively written as $S \in \mathbb{R}^{N\times D_{\text{slot}}}$, with individual slots s_n for $n = 1, \ldots, N$. Keys and values derived from scene features are $K, V \in \mathbb{R}^{P\times D_{\text{slot}}}$, queries from slots are $Q \in \mathbb{R}^{N\times D_{\text{slot}}}$, and attention weights are $A \in \mathbb{R}^{N\times P}$. The MLP hidden dimension is D_{mlp}. We use θ to denote all learnable parameters and t for integration time.

3.1 Encoder

The encoder maps an input image x to a set of feature vectors. A four-layer convolutional network with 5×5 kernels and ReLU activations produces a feature map $\hat{F} \in \mathbb{R}^{D_{\text{enc}}\times H\times W}$. We add a soft positional embedding - at each spatial position (i, j), we construct a vector of normalized distances to the image borders,

$$p_{ij} = \left(\tfrac{j}{W},\ \tfrac{i}{H},\ 1 - \tfrac{j}{W},\ 1 - \tfrac{i}{H}\right), \tag{6}$$

project it through a learned linear layer $p_{ij} \mapsto W_{\text{pos}}\, p_{ij} \in \mathbb{R}^{D_{\text{enc}}}$, and add the result to the corresponding feature column. The spatially-embedded features are then flattened to $F \in \mathbb{R}^{P\times D_{\text{enc}}}$ and passed through a residual MLP with layer normalization followed by two linear layers and a ReLU nonlinearity:

$$F \leftarrow F + W_1\, \text{ReLU}(W_0\, \text{LayerNorm}(F)). \tag{7}$$

This encoder is architecturally identical to the one used in standard Slot Attention, ensuring any differences in decomposition behavior arise solely from slot dynamics.

3.2 Slot Initialization

We initialize N slot vectors $S_0 = \{s_0^1, \ldots, s_0^N\}$ by sampling from a learned Gaussian distribution,

$$s_0^n \sim \mathcal{N}(\mu_\theta,\, \text{diag}(\sigma_\theta^2)), \quad n = 1, \ldots, N, \tag{8}$$

where $\mu_\theta \in \mathbb{R}^{D_{\text{slot}}}$ and $\log \sigma_\theta \in \mathbb{R}^{D_{\text{slot}}}$ are learned parameters shared across all slots. This random initialization breaks permutation symmetry among slots and is helpful in preventing slot collapse.

3.3 Slot Dynamics as an Autonomous ODE

In standard Slot Attention, slots are updated through T iterations of a recurrence:

$$s_{t+1}^n = \text{GRU}(s_t^n,\, u_t^n), \tag{9}$$

where u_t^n is a slot-specific aggregation of scene features F via competitive attention. We replace this discrete iteration with continuous dynamics governed by a neural ODE.

3.3.1 Scene Representation

The encoded features F are first projected into slot-dimensional space via a learned linear layer, then mapped to keys and values:

$$\tilde{F} = W_{\text{in}}\,\text{LayerNorm}(F), \qquad K = W_K\,\tilde{F}, \qquad V = W_V\,\tilde{F}, \tag{10}$$

where $W_{\text{in}} \in \mathbb{R}^{D_{\text{slot}} \times D_{\text{enc}}}$ and $W_K, W_V \in \mathbb{R}^{D_{\text{slot}} \times D_{\text{slot}}}$. Crucially, K and V are computed once and held fixed throughout integration. The scene representation does not change - only the slots evolve. This is what makes the system autonomous. The vector field depends on the current slot state and fixed constants, not external time-varying input.

3.3.2 Vector Field

Given a slot state $S \in \mathbb{R}^{N \times D_{\text{slot}}}$, we define the velocity $\dot{S} = f_\theta(S)$ as follows.

Competitive attention. Queries are computed from the current slot state, and attention weights are normalized to enforce competition among slots for each spatial feature:

$$Q = \text{LayerNorm}(S)\,W_Q, \tag{11}$$

$$A_{np} = \frac{\exp(\frac{q_n \cdot k_p}{\sqrt{D_{\text{slot}}}})}{\sum_{n'=1}^{N} \exp(\frac{q_{n'} \cdot k_p}{\sqrt{D_{\text{slot}}}})}, \tag{12}$$

where the softmax is taken over the slot axis (index n), not the feature axis. This slot-normalized softmax forces slots to compete as each spatial feature is softly assigned to the slot whose query most closely matches the feature's key. The resulting attention weights are then normalized over features to form a proper weighted average:

$$\hat{A}_{np} = \frac{A_{np}}{\sum_{p'=1}^{P} A_{np'}}, \qquad u_n = \sum_{p=1}^{P} \hat{A}_{np}\,v_p. \tag{13}$$

The vector $u_n \in \mathbb{R}^{D_{\text{slot}}}$ is the attention readout for slot n - a weighted mixture of scene values according to the slot's current region of responsibility.

Attention gate. An element-wise gate modulates how the attention readout influences each slot dimension:

$$g_n = \sigma\big(W_g\,[s_n\,;\,u_n]\big), \tag{14}$$

where $[\cdot\,;\,\cdot]$ denotes concatenation, $W_g \in \mathbb{R}^{D_{\text{slot}} \times 2D_{\text{slot}}}$, and σ is the sigmoid function. At initialization, with random weights, the gate outputs are centered near 0.5, providing a balanced starting point. The gate allows the system to selectively suppress or amplify the influence of attention on each slot, giving the dynamics control over how information is incorporated.

Feedforward term. An FFN provides a nonlinear transformation mediated by attention:

$$h_n = W_1 \,\mathrm{ReLU}\big(W_0\,[\mathrm{LayerNorm}(s_n)\,;\,u_n]\big), \tag{15}$$

with weight matrices $W_0 \in \mathbb{R}^{D_{\mathrm{mlp}} \times 2D_{\mathrm{slot}}}$ and $W_1 \in \mathbb{R}^{D_{\mathrm{slot}} \times D_{\mathrm{mlp}}}$.

Velocity. The vector field combines gated attention and the feedforward term:

$$f_\theta(S)_n = g_n \odot u_n + h_n. \tag{16}$$

3.3.3 Integration

The slot state evolves under the initial value problem

$$\frac{dS}{dt} = f_\theta(S), \qquad S(0) = S_0, \tag{17}$$

integrated from $t = 0$ to $t = T$. During training, we use the Euler method with a fixed step size Δt, where $\frac{T}{\Delta t}$ corresponds to the number of refinement iterations in standard Slot Attention (typically $T = 3$ and implicitly $\Delta t = 1$). The final slot state $S(T)$ is passed to the decoder.

Note that the system is autonomous. f_θ has no explicit dependence on t, as the scene features are precomputed and the vector field's parameters are shared across all integration steps. This autonomy is not just an implementation detail - it enables the dynamical systems analysis presented in Section 5. An autonomous ODE admits fixed points S^* satisfying $f_\theta(S^*) = 0$, and the local stability of any such point is characterized by the eigenvalues of the Jacobian $\frac{\partial f_\theta}{\partial S}\big|_{S^*}$.

3.4 Decoder

The decoder reconstructs the image from the final slot representations using spatial broadcasting [36]. Each slot s_n is independently decoded into a per-slot image and mask logit. The slot vector is broadcast onto a low-resolution spatial grid of size 8×8, augmented with soft positional embeddings, and upsampled to the target resolution through three transposed convolutional layers (stride 2, 5×5 kernels, and a ReLU nonlinearity), followed by a stride 1 convolutional layer and linear output convolution. The output for each slot is a four-channel tensor - three RGB channels $\hat{x}^n \in \mathbb{R}^{3 \times H \times W}$ and one mask logit $m^n \in \mathbb{R}^{H \times W}$.

Slot masks are obtained by applying softmax over slots at each pixel,

$$\alpha_{ij}^n = \frac{\exp(m_{ij}^n)}{\sum_{n'=1}^N \exp(m_{ij}^{n'})}, \tag{18}$$

enforcing that slots compete for pixel ownership. The final reconstruction is a mixture:

$$\hat{x} = \sum_{n=1}^{N} \alpha^n \odot \hat{x}^n. \tag{19}$$

Like the encoder, the decoder is architecturally identical to the one used in standard Slot Attention.

3.5 Training Objective

The model is trained end-to-end to minimize pixel-wise MSE between the input image and reconstruction:

$$\mathcal{L} = \frac{1}{3HW} \|x - \hat{x}\|_2^2. \tag{20}$$

No auxiliary losses, slot-level supervision, or explicit regularization toward stable dynamics are applied.

For reference, the architecture is concisely described as follows.

Algorithm 2 SlotODE

Require: Image $x \in \mathbb{R}^{3\times H\times W}$, number of slots N, integration time T, step size Δt
Ensure: Reconstruction $\hat{x}$, slot masks M

$F \leftarrow \text{Encoder}(x)$ ▷ $F \in \mathbb{R}^{M\times D_{\text{enc}}}$, $M = H \times W$
$K \leftarrow FW_K, \quad V \leftarrow FW_V$ ▷ Precomputed once, fixed during integration
$S(0) \sim \mathcal{N}(\mu, \sigma^2)$ ▷ $S(0) \in \mathbb{R}^{N\times D}$

function $f_\theta(S)$ ▷ Autonomous vector field
  $Q \leftarrow \text{LayerNorm}(S)\, W_Q$ ▷ Query projection
  $A_{ij} \leftarrow \text{Softmax}_i\big(q_i \cdot k_j \,/\, \sqrt{D}\big)$ ▷ Softmax over slots (competition)
  $\bar{A}_{ij} \leftarrow A_{ij} \,/\, \sum_{j'} A_{ij'}$ ▷ Normalize over spatial positions
  $\hat{a}_i \leftarrow \sum_j \bar{A}_{ij}\, v_j$ ▷ Weighted value aggregation
  $g_i \leftarrow \sigma\big([s_i,\, \hat{a}_i]\, W_g\big)$ ▷ Learned gate $\in (0,1)^D$
  $h_i \leftarrow \text{MLP}\big([\text{LayerNorm}(s_i),\, \hat{a}_i]\big)$ ▷ Residual MLP
  return $g \odot \hat{a} + h$ ▷ Slot velocity $\frac{dS}{dt}$
end function

$S(T) \leftarrow \text{ODESolve}\big(f_\theta,\ S(0),\ t_0{=}0,\ t_1{=}T,\ \Delta t\big)$ ▷ Euler (train) or Euler/DoPri5 (test)
$\hat{x}_i,\ m_i \leftarrow \text{Decoder}(s_i(T))$ for $i = 1, \ldots, N$ ▷ Per-slot reconstruction + mask logit
$M_i \leftarrow \text{Softmax}(m_1, \ldots, m_N)_i$ ▷ Pixel-wise mask competition
$\hat{x} \leftarrow \sum_{i=1}^{N} M_i \odot \hat{x}_i$ ▷ Composite reconstruction

4 Empirical Results

4.1 Object Decomposition

We evaluate our model on CLEVR (with masks) [47] at 64×64 resolution. All models were trained for 500K steps with batch size 64, the Adam optimizer [48] with a peak learning rate of 4×10^{-4}, a linear warmup over the first 10K steps, followed by exponential decay. All models use $N = 11$ slots, a slot dimension of $D_{\text{slot}} = 64$, and an encoder feature dimension of $D_{\text{enc}} = 64$. Evaluation uses a held-out validation split of 5K images. We report two metrics - the foreground adjusted rand index (ARI-FG) for decomposition quality and pixel-wise mean squared error (MSE) for reconstruction quality. For both metrics, SlotODE is directly compared against Slot Attention trained with an otherwise identical encoder, decoder, optimizer, and seed. The primary architectural difference is the refinement mechanism - GRU-based discrete iteration vs. ODE-based continuous integration. We sweep a number of refinement iterations $T \in \{3, 4, 5, 6\}$ for both models. For SlotODE, we also sweep Euler integration step size $\Delta t \in \{1, 0.5\}$, where $\Delta t = 1$ matches the implicit step size of Slot Attention and $\Delta t = 0.5$ corresponds to twice as many Euler steps over the same integration time. The step size axis has no analogue in the discrete baseline.

Model	Config	ARI-FG	MSE
Slot Attention	T=3	0.9840	0.000528
Slot Attention	T=4	0.9859	0.000551
Slot Attention	T=5	0.9841	0.000588
Slot Attention	T=6	0.9804	0.000712
SlotODE	T=3, Δt=1	0.9832	0.000580
SlotODE	T=4, Δt=1	0.9860	0.000575
SlotODE	T=5, Δt=1	0.9864	0.000584
SlotODE	T=6, Δt=1	0.9850	0.000613
SlotODE	T=3, Δt=0.5	0.9859	0.000558
SlotODE	T=4, Δt=0.5	0.9852	0.000662
SlotODE	T=5, Δt=0.5	0.9828	0.000755
SlotODE	T=6, Δt=0.5	0.9820	0.000703

Table 1. Object decomposition results on CLEVR across integration times and step sizes. ARI-FG measures decomposition quality (higher is better) and MSE measures reconstruction quality (lower is better). Slot Attention has 791,876 parameters and SlotODE has 783,236 parameters.

SlotODE matches Slot Attention across all shared configurations. At all T, the two models achieve near-identical ARI-FG and comparable MSE. Varying the step size from $\Delta t = 1$ to $\Delta t = 0.5$ at fixed T yields a small and consistent increase in MSE with essentially unchanged ARI-FG. These results show that the continuous-time formulation preserves object decomposition performance relative to the discrete baseline at comparable parameter counts, while exposing a new integration parameter (step size Δt).

As a sanity check against the published baseline, [18] report $98.8 \pm 0.3\%$ ARI[4] on CLEVR6 (scenes with up to 6 objects) with $T = 3$ and $N = 7$[5] slots, averaged over 5 seeds. They do not publish a quantitative ARI number for the full CLEVR dataset (CLEVR10, up to 10 objects per scene), which is the harder setting we train on with $N = 11$ slots. Our Slot Attention baseline reaches 0.9840 ARI-FG at $T = 3$, very close to the reported CLEVR6 range despite being evaluated on scenes with more objects.

Qualitative results on a held-out CLEVR scene with 7 objects are shown below. We visualize the input image, its reconstruction, and per-slot mask assignments obtained by taking the argmax over slots at each pixel. SlotODE produces clean object decompositions where each slot captures a single object and unused slots remain empty. Of the $N = 11$ slots available, Figure 1 shows exactly 7 bind to the 7 objects in the scene while Figure 2 shows the remaining 4 collapse to a shared inactive mode.

[4]Their reported ARI already excludes background pixels and is therefore directly comparable to ARI-FG as used here.

[5]Their paper uses K to denote the number of slots but we use N here for consistency with our writing.

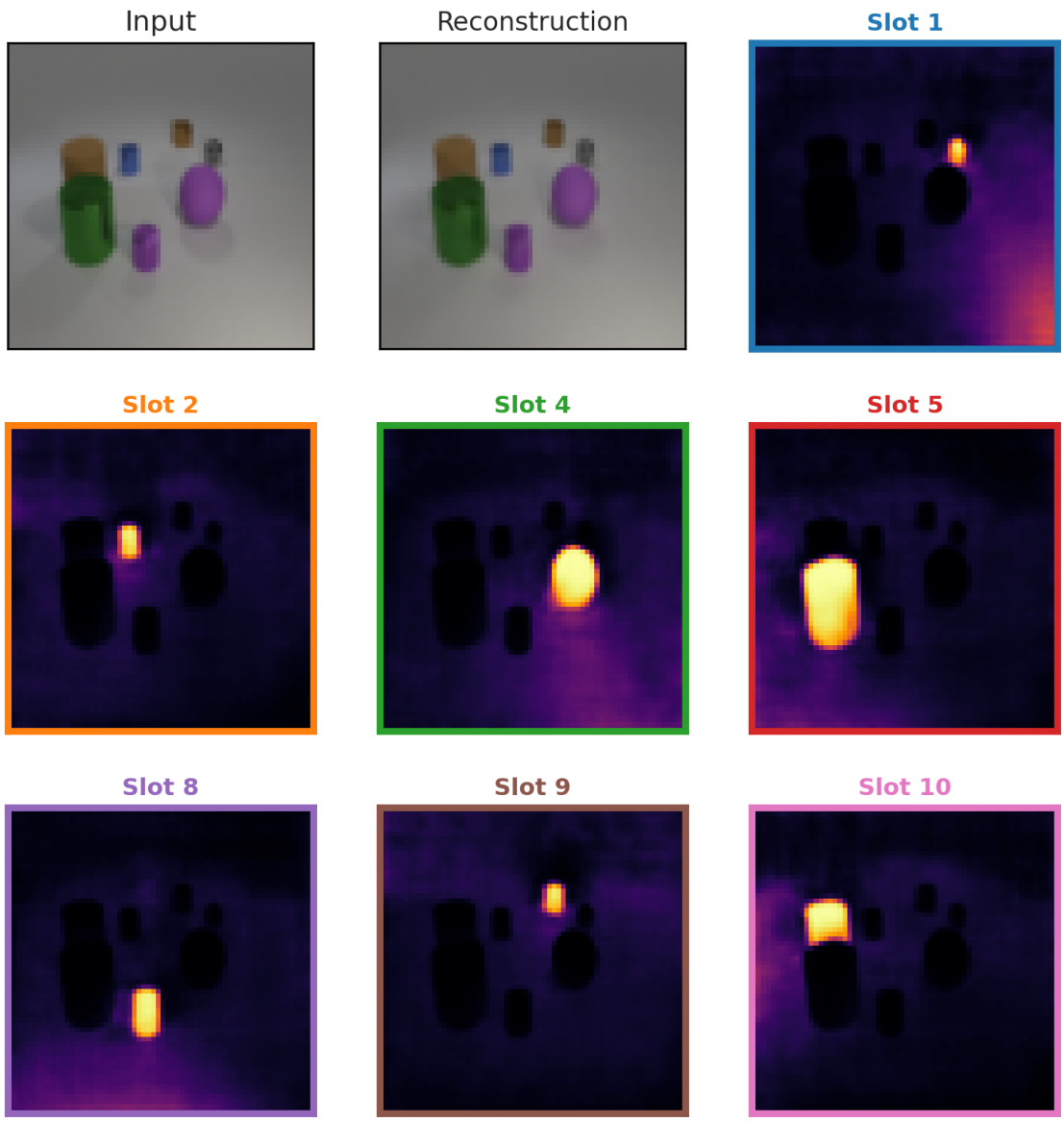

Figure 1. Qualitative object decomposition on a held-out CLEVR scene with 7 objects. We use the best SlotODE configuration based on ARI-FG ($T = 5$, $\Delta t = 1$). The top 7 slots with the highest activations are picked and visualized. Note that there is exactly one unique slot per object in the image.

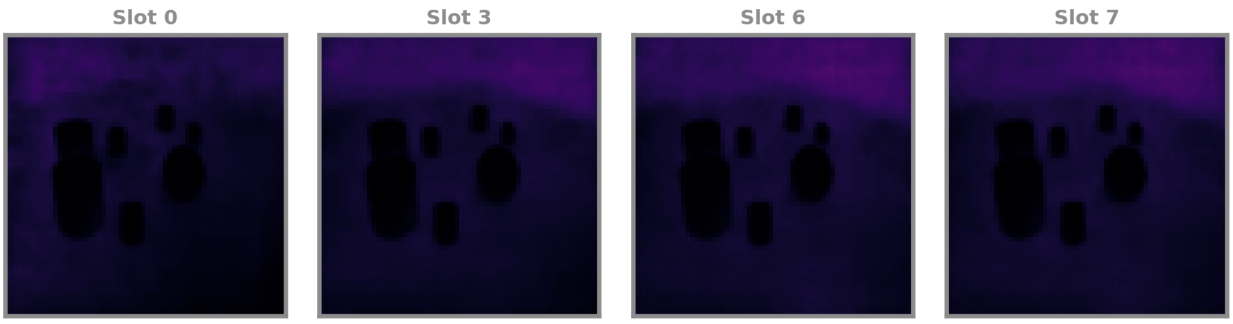

Figure 2. The 4 unused slots omitted from Figure 1.

Because SlotODE refines slots by integrating a continuous vector field rather than applying a fixed number of discrete updates, we can also view the decomposition *forming* over integration time. Figure 3 shows the decoder mask of each used slot from the same scene, decoded at seven approximately evenly-spaced points $t \in \{0.0, 0.8, 1.6, 2.5, 3.3, 4.1, 5.0\}$ along the trajectory with $\Delta t = 0.1$. At $t = 0$, all slots produce diffuse, image-wide masks - the slot states are random samples from the learned prior and have not yet bound to any region. By $t \approx 1$ each used slot has localized to a coarse spatial region, and by $t = 3$ the masks largely correspond to individual objects. Further integration progressively sharpens boundaries with the masks at $t = 5$ matching those in Figure 1. The bottom row shows one of the four inactive slots which collapses to a uniform low-mass mask. The colored borders match Figure 1.

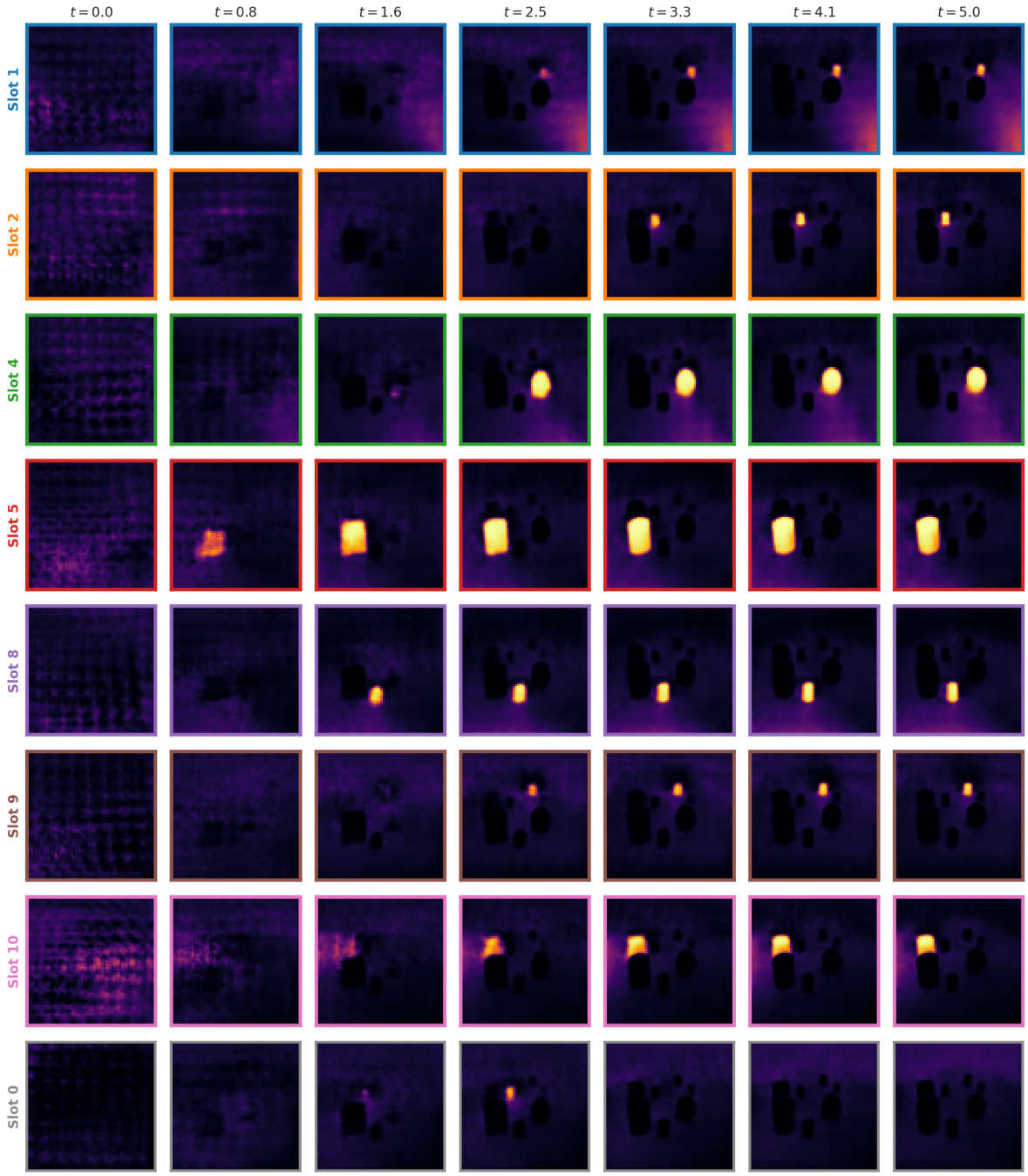

Figure 3. Decoder mask evolution along the SlotODE trajectory (evenly-spaced integration times from $t = 0$ to $t = 5$ with $\Delta t = 0.1$) for the 7-object scene from Figure 1. Masks evolve from diffuse, image-wide attention at $t = 0$ to per-object localized masks by $t \approx 3$, then continue to sharpen until $t = 5$.

Together, these results establish that SlotODE reproduces Slot Attention's object decomposition at matched parameter counts across the swept integration times, and that the continuous-time reformulation preserves performance. Notably, SlotODE achieves this with a simpler, memoryless sigmoid gate in place of the GRU update (Section 3.3). This suggests that the hidden gating dynamics of the GRU are not load-bearing for object decomposition once refinement is cast as integration of a learned vector field, though the gate and continuous-time reformulation are swapped jointly here and a controlled isolation of the two is left to future work. Training convergence under an otherwise identical optimization setup is indistinguishable between the two models (see Appendix B, Figure 11).

4.2 Object Segmentation

The slot masks $A \in \mathbb{R}^{N \times P}$ produced by Slot Attention and SlotODE assign each pixel a probability distribution over slots, and taking the argmax across them yields a hard segmentation. This naive segmentation is poor as pixels where no slot is clearly responsible - backgrounds, object boundaries, or empty regions - are still assigned to whichever slot has the marginally highest probability, producing fragmented and noisy masks even when the underlying object decomposition is correct.

A simple thresholding procedure recovers clean segmentations. Each pixel p is assigned to its argmax slot only if the maximum slot probability exceeds a confidence threshold $\tau \in [0, 1]$:

$$\hat{s}(p) = \begin{cases} \arg\max_n A_{np} & \text{if } \max_n A_{np} \geq \tau, \\ \text{unassigned} & \text{otherwise.} \end{cases} \tag{21}$$

The threshold introduces a "none of the above" class that absorbs background and low-confidence pixels, leaving only confidently-explained object regions in the segmentation.

The same procedure applies to any model that produces per-pixel slot probabilities, so we can directly compare Slot Attention and SlotODE under matched conditions. For each model we sweep $\tau \in [0, 0.95]$ at increments of 0.025 on 5,000 held-out CLEVR validation images, computing mean intersection-over-union (mIoU) against ground-truth masks - selecting the $\tau^\star$ that maximizes mIoU. We report results for Slot Attention ($T = 5$) and the two SlotODE variants at the same T ($\Delta t = 1$ and $\Delta t = 0.5$).

Table 2 reports mIoU at $\tau = 0$ (argmax) and per-model $\tau^\star$. Argmax assignment is poor for all three models, with mIoU between 0.16 and 0.20 - confirming that slot competition without thresholding does not produce a clean segmentation on its own. After thresholding, all three models cross 0.80 mIoU, but SlotODE outperforms the discrete baseline by a clear margin at 0.861 ($\Delta t = 1$) versus 0.801 at the same T, an improvement of +6.0 mIoU. SlotODE also reaches its optimum at a lower threshold ($\tau^\star = 0.550$ vs. 0.625), indicating that a smaller fraction of the image needs to be excluded to recover a clean segmentation.

Model	Config	mIoU ($\tau = 0$)	mIoU (best)	$\tau^\star$
Slot Attention	$T = 5$	0.198	0.801	0.625
SlotODE	$T = 5,\ \Delta t = 1$	0.187	0.861	0.550
SlotODE	$T = 5,\ \Delta t = 0.5$	0.164	0.855	0.575

Table 2. Unsupervised segmentation on CLEVR via thresholded slot masks. Each model is swept independently over τ. We report mIoU at $\tau = 0$ (argmax) and per-model optimum $\tau^\star$. SlotODE outperforms Slot Attention at matched $T = 5$ by +6.0 mIoU and reaches its optimum at a lower threshold.

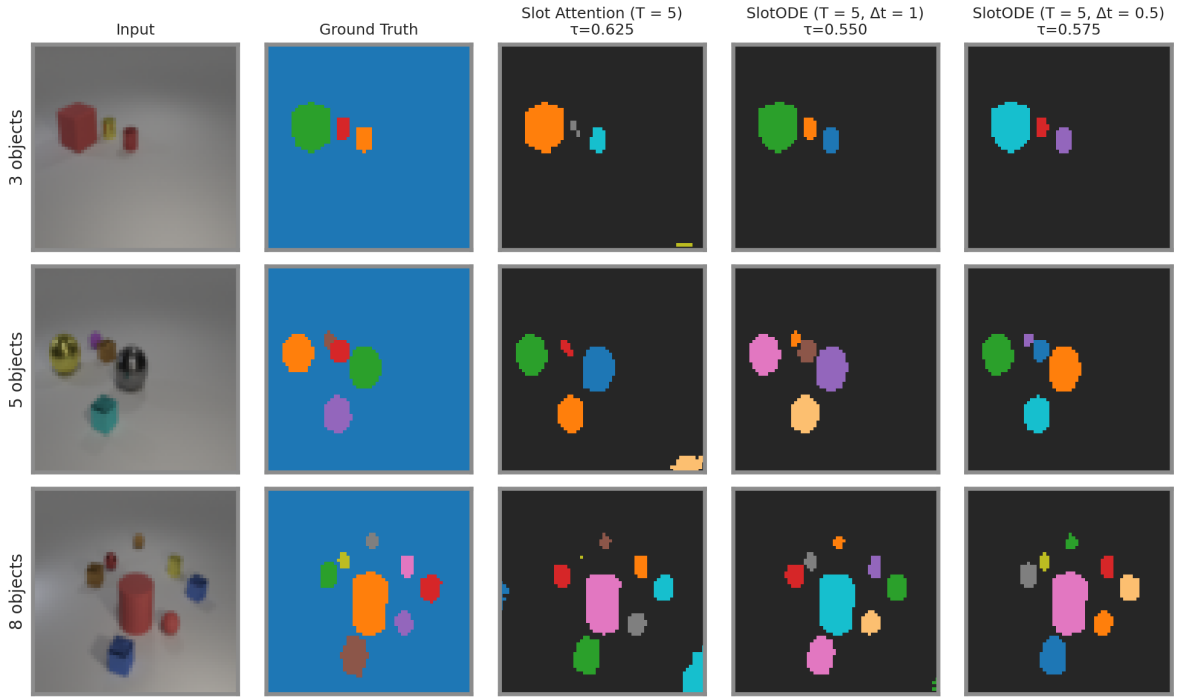

Figure 4. Unsupervised segmentation on CLEVR scenes with 3, 5, and 8 objects. Columns (from left to right) are the input image, ground-truth masks, and thresholded segmentations at each model's $\tau^\star$ (Slot Attention $T = 5$, SlotODE $T = 5$, $\Delta t = 1$, and SlotODE $T = 5$, $\Delta t = 0.5$).

The thresholding procedure uses a single scalar hyperparameter with no supervision or retraining. We believe continuous-time refinement produces sharper attention masks where each pixel is more confidently bound to a single slot leading to the performance gap between Slot Attention and SlotODE.

4.3 Adaptive Computation

In Slot Attention, the number of refinement steps T is a fixed architectural choice baked into training. Each scene receives the same compute regardless of how many objects it contains or how difficult the decomposition is - a five and ten-object scene both pay T GRU updates. The continuous-time reformulation removes this constraint. At inference, we no longer have to use the same fixed-step Euler solver used for training. Adaptive solvers that vary their step size based on local error estimates are now an option.

A precondition for adaptive integration is that the vector field must carry useful information along the entire trajectory, not just at t values seen during training. We probe this directly - starting from the trained SlotODE ($T = 5$, $\Delta t = 1$) and integrating the learned vector field with a finer step size $\Delta t_f = 0.1$ - evaluating the decomposition quality (ARI-FG) at intermediate times. Figure 5 shows the result on 200 held-out CLEVR scenes. ARI-FG rises monotonically from $t = 0$ to $t = 5$, with no collapse or plateaus in between. Each probed sub-step of the learned vector field continues to improve decomposition, even though training only ever saw points at $\frac{T}{\Delta t}$ intervals. The dynamics are smooth enough to allow intermediate, partially-resolved decompositions - which is exactly what an adaptive solver needs.

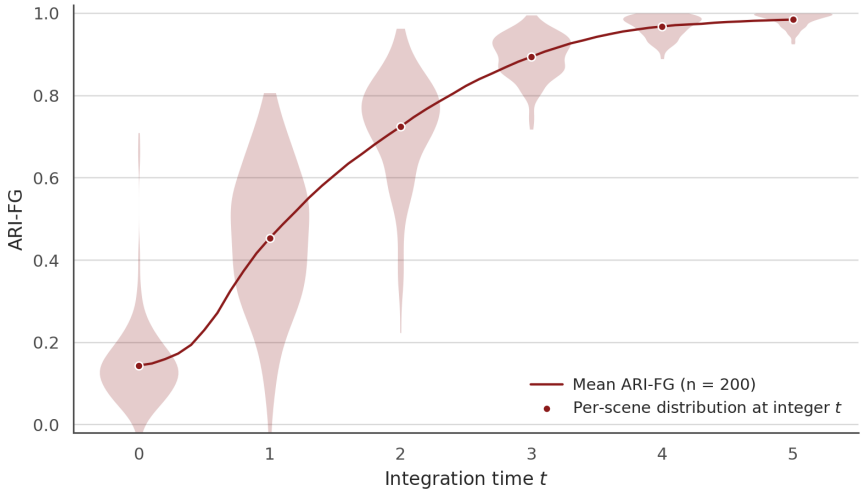

Figure 5. Decomposition quality along the SlotODE trajectory. We integrate the trained $T = 5$, $\Delta t = 1$ model with a finer step size $\Delta t_f = 0.1$ and decode the slot state at every sub-step, evaluating ARI-FG against ground-truth masks. The red line is the mean across 200 held-out CLEVR scenes at all 51 fine timepoints and the shaded violins show per-scene distribution at each integer training step $t \in \{0, 1, \ldots, 5\}$. The mean rises monotonically from $t = 0$ to $t = 5$, the per-scene distributions shift upward and tighten, and there is smooth interpolation between integer t values where training was not explicit.

We swap the fixed-step Euler solver used during training with the Dormand-Prince fifth-order method (Dopri5) at inference. We use the PID step size controller (see `Diffrax` [49] documentation) at relative tolerance 10^{-3} and absolute tolerance 10^{-4}. The solver runs from $t = 0$ to $t = T_{\text{train}}$ with no fine-tuning or other

changes. We record the number of integration steps per scene on 500 held-out CLEVR validation images grouped by object count.

Figure 6 shows the result for the four $\Delta t = 1$ models. In every configuration, the mean number of solver steps increases monotonically with scene object count - from approximately 8 steps on three-object scenes ($T = 3$) up to around 21 steps on ten-object scenes ($T = 6$). The same trend holds across all four T values. No part of the model was trained to produce this behavior. The total number of function evaluations is six times the step count, since Dopri5 has six effective stages.

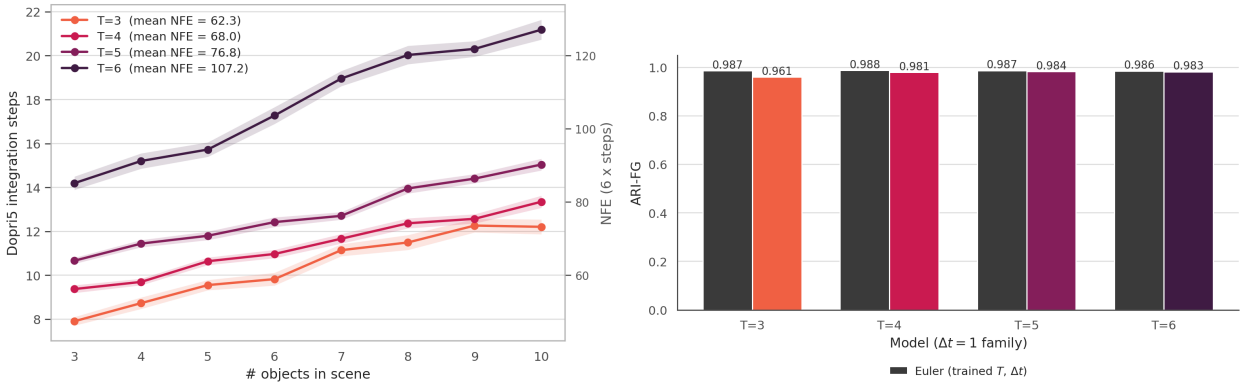

Figure 6. Adaptive compute and decomposition quality across SlotODE models ($\Delta t = 1$). Mean Dopri5 integration steps vs. # objects per scene on left. Compute cost grows with scene complexity for every T. Per-model ARI-FG from fixed-step Euler training vs. Dopri5 adaptive integration at inference on right. Decomposition quality is mostly preserved.

Table 3 displays the full sweep, including $\Delta t = 0.5$ models. The same trend across object counts appears in every row. Note that longer training horizons T generally cost more steps at inference, which is expected for a longer integration interval over the same vector field. Models trained at $\Delta t = 0.5$ require substantially more adaptive steps than their $\Delta t = 1$ counterparts at matched T, with ratios ranging from roughly 1.7x to 3.3x. Euler at step Δt is stable only when the Jacobian eigenvalues of the vector field satisfy $|\lambda| < \frac{2}{\Delta t}$, so training at a smaller Δt permits the model to fit a stiffer field. Adaptive solvers at fixed tolerance pay for that stiffness in additional steps.

	# objects in scene							
Config	3	4	5	6	7	8	9	10
$T = 3,\ \Delta t = 1$	7.90	8.80	9.55	9.85	11.14	11.48	12.31	12.35
$T = 3,\ \Delta t = 0.5$	13.61	14.75	15.51	16.71	18.20	19.76	20.00	20.73
$T = 4,\ \Delta t = 1$	9.37	9.69	10.64	10.97	11.66	12.37	12.57	13.35
$T = 4,\ \Delta t = 0.5$	18.95	20.66	22.13	21.94	23.06	23.62	23.48	24.80
$T = 5,\ \Delta t = 1$	10.66	11.44	11.80	12.42	12.70	13.95	14.40	15.04
$T = 5,\ \Delta t = 0.5$	36.71	39.92	41.42	42.39	43.27	44.86	45.43	44.73
$T = 6,\ \Delta t = 1$	14.19	15.20	15.74	17.27	18.99	20.02	20.31	21.16
$T = 6,\ \Delta t = 0.5$	35.26	36.97	39.42	37.97	40.38	40.43	40.08	39.00

Table 3. Mean Dopri5 integration steps per scene, grouped by object count, across all eight SlotODE configurations. Tolerances rtol = 10^{-3} and atol = 10^{-4}, integrated from $t = 0$ to $t = T_{\text{train}}$. The number of function evaluations (NFEs) per scene equals six times the step count. Across every row, mean steps grow with object count.

The discrete Slot Attention baseline cannot produce any analogue of Table 3. Its computation is a fixed sequence of T GRU updates with no notion of step size and no mechanism for allocating compute based on input. Adaptive computation is a property SlotODE provides directly that emerges without any changes to the training objective. The model learns a vector field that happens to be smoother on simple scenes and stiffer on complex ones, and an adaptive solver picks up the difference at inference. Do note that the number of objects in a scene is not the only measure of scene complexity. Accounting for scene properties like object sizes and occlusion is left to future work.

Lastly, note that we do not extensively sweep tolerances to minimize NFEs or develop a method to do so. Our goal is to show that SlotODE learns a vector field that encodes a representation expressive enough to automatically account for scene complexity. Further, Dopri5 is just one of many adaptive solvers. Optimizing for *efficient* adaptive computation is a separate engineering problem on its own.

5 Theoretical Analysis

The continuous-time formulation introduced in Section 3.3 affords a small but useful set of tools borrowed from the analysis of nonlinear dynamical systems. We treat these tools as descriptive vocabulary for an empirical analysis rather than as theorems with exact proofs. Before applying them to trained SlotODE models, we briefly setup relevant ideas.

5.1 Framework

Throughout this section we treat the learned slot dynamics

$$\frac{dS}{dt} = f_\theta(S), \qquad S \in \mathbb{R}^{N \times D_{\text{slot}}}, \tag{22}$$

as an *autonomous* ordinary differential equation. The vector field f_θ depends only on the slot state S as the scene features (K, V) are precomputed from the input image and held constant throughout integration. Unlike a non-autonomous system, an autonomous ODE admits a notion of *fixed points* - configurations of the state where dynamics stop moving. Formally, $S^\star \in \mathbb{R}^{N \times D_{\text{slot}}}$ is a fixed point if

$$f_\theta(S^\star) = 0. \tag{23}$$

At a fixed point the slot trajectory $S(t)$ is stationary - if the system arrives at $S^\star$ it stays there. The empirical question in the rest of this section is whether the integrator terminates in a region whose linearization is locally attractor-like, and what the geometry of that region looks like.

State-space Jacobian. Local behavior near a fixed point is characterized by the linearization of f_θ at $S^\star$. The state-space Jacobian is an $(N \times D_{\text{slot}}) \times (N \times D_{\text{slot}})$ matrix

$$J(S^\star) = \left.\frac{\partial f_\theta}{\partial S}\right|_{S=S^\star}. \tag{24}$$

Asymptotic stability. The eigenvalue spectrum of $J(S^\star)$ determines whether nearby trajectories are attracted to or repelled from $S^\star$. By the Hartman-Grobman theorem and Lyapunov's indirect method [50], if every eigenvalue λ_i of $J(S^\star)$ satisfies $\text{Re}(\lambda_i) < 0$, then $S^\star$ is locally asymptotically stable - there exists a neighborhood of $S^\star$ in which trajectories converge to $S^\star$ exponentially in t. If any eigenvalue has a positive real part, $S^\star$ is unstable. Eigenvalues on the imaginary axis correspond to neutral or marginal directions and require higher-order analysis. We do not prove stability. We measure $J(s_t)$ along sampled trajectories and report the eigenvalue distribution. Under the standard linearization implication, $\text{Re}(\lambda) < 0$ at the terminal state is consistent with local contraction toward the reached state. Whether this holds globally, or uniformly across the data distribution, is an empirical observation, not a theorem.

Log-norm and contraction. Asymptotic stability is a long-time statement - trajectories eventually converge, but they may transiently expand before contracting. A stronger, instantaneous notion is *contraction*. The *logarithmic norm* of J is

$$\mu(J) = \lambda_{\max}\left(\tfrac{1}{2}\left(J + J^\top\right)\right), \tag{25}$$

the largest eigenvalue of the symmetric part of J. If $\mu(J(S)) \leq -\alpha < 0$ in a region, then any two trajectories in that region satisfy $\|S_1(t) - S_2(t)\| \leq e^{-\alpha t}\,\|S_1(0) - S_2(0)\|$ - distances between trajectories shrink monotonically [51]. Note that $\mathrm{Re}(\lambda_i) < 0$ for all i does not imply $\mu(J) < 0$. The gap between the two reflects how *non-normal* J is, i.e., how far J is from commuting with $J^\top$. A non-normal Jacobian permits transient growth even when the eigenvalue spectrum is fully stable, and we see this distinction in the trained model.

Why does this matter for SlotODE? Each scene defines its own autonomous dynamics through its precomputed K and V. Fixed points, Jacobians, and log-norms are therefore properties of the model *as evaluated on a particular input*. The discrete Slot Attention update rule does not allow any of these analyses in their continuous-time form. It has no explicit vector field and Jacobian, no log-norm in the contraction sense, and no notion of basin of attraction under integration. The reframing of refinement as integration of a learned vector field is what makes the rest of this section possible.

Studying the state of a trained network through fixed points and their Jacobians has a long history in computational neuroscience and recurrent network analysis [52], with recent extensions to deep equilibrium models [21], object-centric iterative refinement [46], and very recently, general autoencoders [53]. SlotODE is closest in spirit to [46], but defines refinement as the continuous integration of an autonomous learned vector field rather than a discrete iteration.

5.2 Trajectory geometry

The framework of Section 5.1 treats slot refinement as integration of an autonomous vector field. Before looking at the linearization $J(s_t)$, we visualize the trajectory $S(t)$ itself in slot space. Each scene defines its own trajectory in $\mathbb{R}^{N\times D_{\text{slot}}}$ with $N = 11$ slots whose individual states $s_n(t) \in \mathbb{R}^{64}$ evolve from a shared sampled initial distribution to a scene-specific final configuration. To project this 704-dimensional trajectory into a comprehensible space, we run principal component analysis on the stacked slot states $\{s_n(t)\}_{n,t}$ for a single representative scene and plot the per-slot trajectories using the top-3 principal components. This is a per-scene, per-trajectory PCA - the axes have no meaning across scenes.

Figure 7 shows the result. We separate slots by usage at $t = T$ where a slot is *used* on this scene if its decoded mask attains a peak above the per-image mean and *unused* otherwise. All N slots begin in a tight cluster - the i.i.d. initialization places them near a single point in the PCA frame. The K used slots (here $K = 7$, matching the scene's object count) diverge along distinct trajectories to spatially separated terminal points where each used slot is bound to exactly one object and arrives at an object-specific location in slot space. Lastly, the $N - K = 4$ unused slots diverge to inactive endpoints.

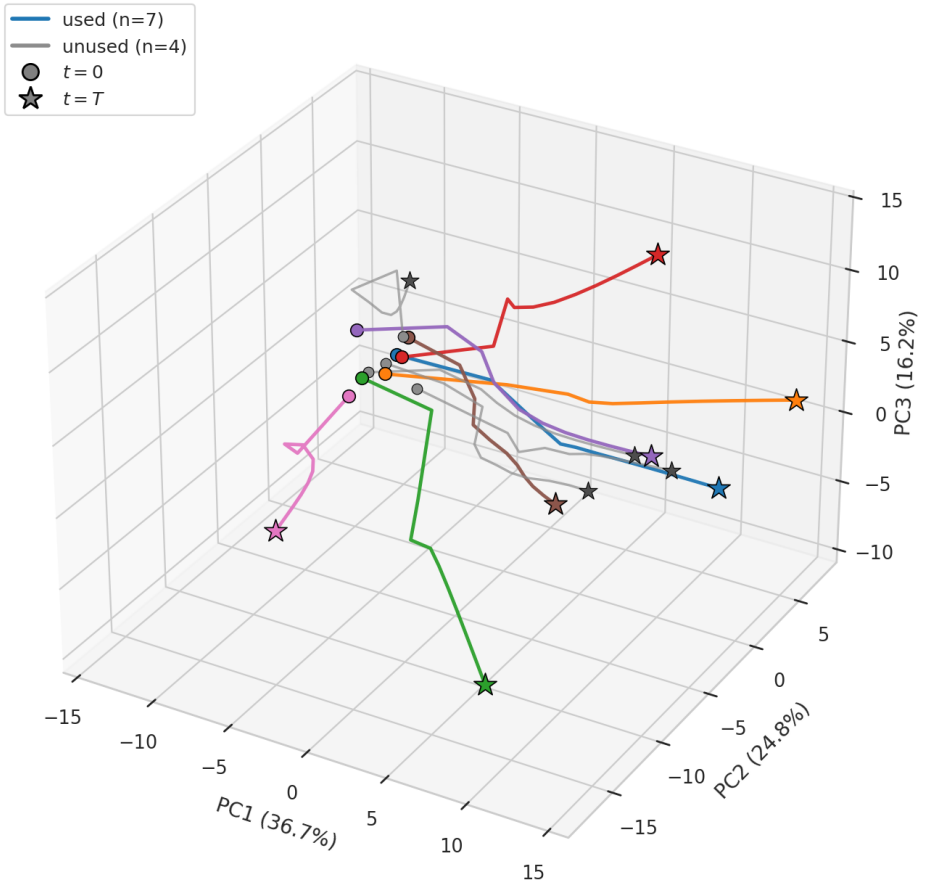

Figure 7. Per-slot trajectories of a representative validation scene ($K = 7$ objects) projected into the top-3 PCA components of per-scene slot states. Colored curves are the K used slots, gray curves are the $N - K$ unused slots. Circles mark $t = 0$ and stars mark $t = T$. All slots begin in a shared cluster. Used slots fan out to object-specific endpoints while unused slots diverge to inactive endpoints.

5.3 Spectrum and contraction

We now move from looking at the trajectory itself to its local linearization. At each saved state s_t along the integrated trajectory, we compute the state-space Jacobian $J(s_t) = \frac{\partial f_\theta}{\partial S} \in \mathbb{R}^{(N \times D_{\text{slot}}) \times (N \times D_{\text{slot}})}$ holding the precomputed K and V fixed. With $N = 11$ and $D_{\text{slot}} = 64$, $J(s_t)$ is a real 704×704 matrix and admits up to 704 complex eigenvalues. We integrate at a finer step size $\Delta t = 0.5$ on the SlotODE checkpoint trained at $T = 5$, $\Delta t = 1$, and save every step. We summarize $J(s_t)$ along the trajectory through three views - the spectrum on a representative scene, the trace $\text{tr}\,(J) = \sum_i \text{Re}(\lambda_i)$, and the logarithmic norm $\mu(J)$.

Figure 8 shows the distribution of $\mathrm{Re}(\lambda)$ across the saved time steps for the representative scene as a stack of kernel densities, $t = 0$ at the top and $t = T$ at the bottom. The spectrum both *narrows* and *shifts*. At $t = 0$, the real-part distribution is broad and by $t = T$ it tightens into a sharper peak just left of zero, with 98.3% of eigenvalues in the open left half-plane and the largest real part reduced to ≈ 0.45. The trajectory terminates in a region whose linearization is consistent with local contraction toward the reached state. We cannot say for sure that s_T is a fixed point of f_θ since we have not solved $f_\theta(S^\star) = 0$. But - we can say that wherever the integrator stopped - the local geometry is overwhelmingly stable in the eigenvalue sense.

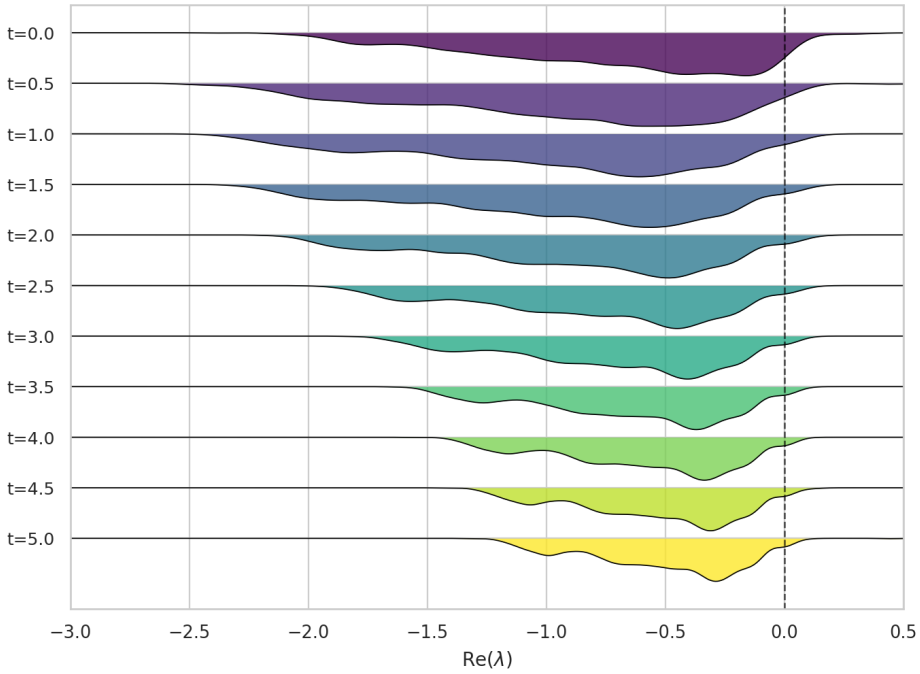

Figure 8. Distribution of $\mathrm{Re}(\lambda)$ for the eigenvalues of $J(s_t)$ along the integration trajectory on the representative validation scene. Each ridge is a kernel density over the 704 eigenvalues at one saved time t, $t = 0$ at the top, and $t = T$ at the bottom. The spectrum both narrows and shifts into the left half-plane as t increases.

Figure 9 aggregates the spectral picture into two scalar functions of $J(s_t)$ - plotted as median and interquartile range over 50 randomly drawn validation scenes. The trace $\mathrm{tr}\,(J(s_t)) = \sum_i \mathrm{Re}(\lambda_i)$ is the rate at which an infinitesimal volume around s_t shrinks (a negative trace indicates a dissipative flow). The log-norm $\mu(J(s_t)) = \lambda_{\max}\left(\frac{1}{2}(J + J^\top)\right)$ is the worst-case instantaneous expansion rate in ℓ_2. By construction $\mu(J) \geq \max_i \mathrm{Re}(\lambda_i)$ for every J and equality holds when J is normal (the gap measures non-normality).

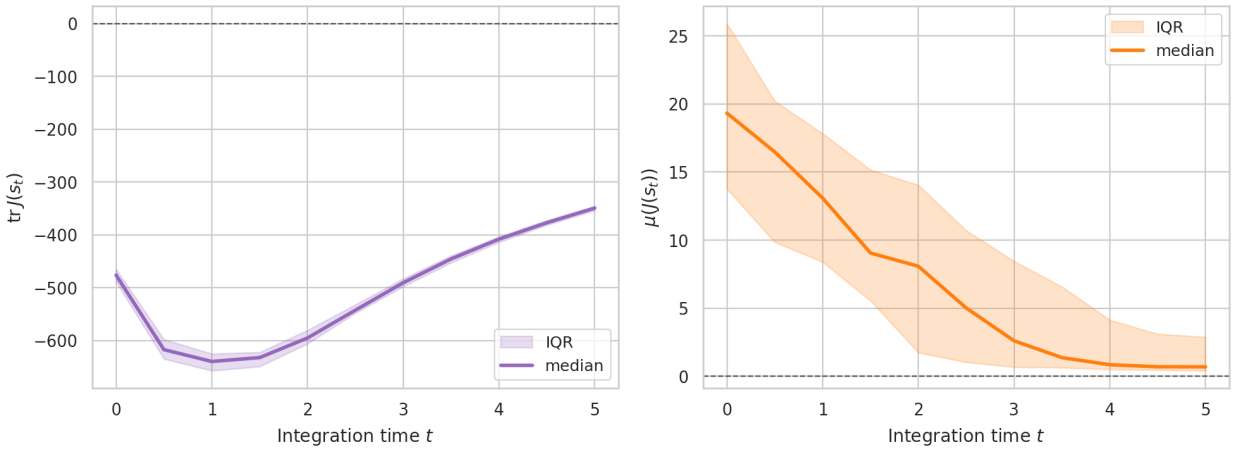

Figure 9. Scalar Jacobian summaries along the trajectory (median and interquartile range) over 50 validation scenes. On the left is trace $\mathrm{tr}\,(J(s_t))$, persistently negative and minimized in the early phase. On the right is log-norm $\mu(J(s_t))$, decaying from ≈ 19 at $t = 0$ to ≈ 0.7 at $t = T$.

The curves reveal a two-phase trajectory:

- *Early phase ($t \lesssim 1$).* The trace deepens from $\mathrm{tr}\,(J) \approx -476$ at $t = 0$ to a minimum of ≈ -639 near $t = 1$ indicating most slot-space modes contract aggressively. At the same time - log-norm starts large ($\mu(J) \approx 19$) which means the symmetric part of J has a strongly expansive direction. The combination of heavy contraction with a few sharply expansive non-normal directions is exactly the geometry needed to break i.i.d. slot initialization. Small differences between slot states are amplified along unstable directions.

- *Late phase ($t \gtrsim 3$).* Both summaries relax toward zero. The trace lifts back to ≈ -349 at $t = T$ which is still negative, but no longer minimized. The log-norm $\mu(J)$ collapses to ≈ 0.7. The non-normal expansion that drove early symmetry breaking dissipates and the linearization at the reached state is mainly mild contraction.

The terminal state spectrum is tightly distributed across scenes. Over 50 randomly drawn validation scenes the fraction of eigenvalues with $\mathrm{Re}(\lambda) < 0$ at $t = T$ has mean 0.9903 and lies in $[0.983, 0.996]$ on every scene. Therefore, the picture of a near-fully-stable terminal spectrum is not a property of any one scene, but of a trained SlotODE model. The log-norm tells a complementary story - $\mu(J(s_T)) > 0$ on every one of the 50 scenes - not a single scene reaches strict ℓ_2 contraction $\mu(J) < 0$. The terminal-state geometry across a trained SlotODE model is consistently *eigenvalue-stable* but *not* log-norm-contractive. Essentially, a trained SlotODE model is eventually stable but not instantaneously stable. Every scene converges in the long run, but the path there leaves room for small perturbations to transiently grow. Again, the model was not trained with any explicit regularization for stability or contraction - the two-phase trajectory and near-uniformly stable terminal spectrum emerge from the reconstruction objective alone.

6 Future Work

The reformulation of slot refinement as an autonomous neural ODE opens several natural directions. We discuss five here, ordered roughly by how directly they extend our current work.

Stochastic slot dynamics for real-world scenes. A core limitation of the current work is that SlotODE, like Slot Attention, has been evaluated exclusively on synthetic scenes where objects are well-separated, textures are uniform, and the encoder features cleanly reflect object boundaries. Real-world scenes violate all three assumptions. A natural first step toward real-world scenes is to swap the CNN encoder for a pretrained vision foundation model such as DINO, following [38], who demonstrated that strong encoder features alone - without any changes to slot dynamics - are sufficient to close much of the gap between synthetic and real-world object decomposition. If DINO features suffice, the dynamics are not the bottleneck. The more interesting question is whether SlotODE's continuous-time refinement provides additional benefit on top of strong encoder features, or whether the performance gap observed on CLEVR-derived benchmarks collapses once the encoder is no longer the limiting factor.

We think the answer depends on scene complexity. On moderately complex real scenes, a strong encoder may dominate and the dynamics matter little. On scenes with heavy occlusion, ambiguous boundaries, or many objects, we expect the deterministic vector field f_θ to develop spurious stable attractors - fixed points S^* satisfying $f_\theta(S^*) = 0$ whose basins of attraction correspond to degenerate decompositions (slot collapse, background dominance, or ambiguous boundary assignment) rather than valid object segmentations. The dynamical systems tools developed earlier can provide a direct path to characterizing this failure mode where Jacobian spectra near spurious attractors should differ from spectra near valid decompositions, and a perturbation analysis (a natural extension of the tools developed here, left to future work) would reveal high sensitivity to initialization in scenes where decomposition fails. Establishing a mechanistic picture where failure on hard scenes is a dynamical property of the learned vector field and not just a representational limitation of the encoder would motivate principled fixes. We propose extending SlotODE to use a stochastic differential equation by adding a state-dependent diffusion term to slot dynamics:

$$dS = f_\theta(S)\,dt + \sigma(S,t)\,dW_t \tag{26}$$

where W_t is a standard Wiener process and $\sigma(S,t) \in \mathbb{R}^{N\times D_{\text{slot}}}$ is a learned or annealed noise schedule. This is a natural extension of the autonomous ODE formulation where the drift term is unchanged, and the existing vector field training carries over. The diffusion term plays a specific role as high noise early in integration encourages exploration of the slot assignment landscape and prevents premature convergence to spurious attractors, while noise annealed toward zero as $t \to T$ allows the system to settle into a sharp, stable decomposition. The proposed direction is to first establish whether continuous-time dynamics provide measurable benefit over Slot Attention when both are paired with DINO features - then, in scenes where deterministic SlotODE still fails, characterize the failure dynamically using the analytical tools developed here - and finally, introduce stochastic dynamics as a principled fix.

Energy-constrained dynamics. A natural architectural followup is to parameterize the vector field as a gradient flow $f_\theta = -\nabla E_\theta$ on a learned scalar energy $E_\theta : \mathbb{R}^{N \times D_{\text{slot}}} \to \mathbb{R}$. This imposes contraction *by construction*, gives a Lyapunov function for free, and recovers the modern Hopfield analysis of softmax attention as a special case [54]. The obvious empirical question would ask if the additional structural constraint costs decomposition quality on standard benchmarks. More interestingly, we could explore whether basins of E_θ correspond to valid scene decompositions where relabeling slots gives the same decomposition at the same energy. Any other minima would be failure modes worth studying.

Video and temporal scenes. Slot Attention has a natural video extension in SAVi [25], which carries slots forward across frames by re-running the discrete refinement at each step. SlotODE replaces this re-refinement with continuous-time integration of a single shared vector field, so a video extension would integrate f_θ across frames rather than restart at each frame. This makes precise the question of *what slots do between frames*, which has no analogue in the discrete formulation. Adaptive solvers should also pay off more here as per-frame integration cost can scale with how much the scene is changing.

A scene-conditional vector field via hypernetworks. In our model, the projections W_K, W_Q, and W_V are global learned matrices. This means, different scenes share the same query, key, and value transformations - only differing through encoded features K and V. A natural extension is to generate these projections per-scene via a hypernetwork [55] conditioned on a scene-level summary of the encoded features, $\{W_K, W_Q, W_V\} = h_\phi(F)$. This makes the vector field f_θ genuinely scene-specific in its functional form rather than only in its inputs - the geometry of the competition itself can adapt to scene complexity. The cost is additional parameters and a generalization risk if h_ϕ overfits per-scene. The open question is whether scene-conditional projections produce qualitatively different basin structure or are just a more expressive reparameterization of the same dynamics.

Adjoint-method training. Our training pipeline backpropagates through unrolled Euler steps, which costs $O(\frac{T}{\Delta t})$ memory in slot state and activations. The adjoint method from [33] computes the same gradients by integrating a separate ODE backwards in time that is $O(1)$ in memory at the cost of one additional forward pass. The savings are modest at the $T = 5$, $\Delta t = 1$ scale studied here but become significant for the video extension above (where integration time is multiplied by the number of frames) and for higher-resolution scenes (where slot state and decoder activations dominate memory). Implementing adjoint training cleanly with the existing pipeline is an interesting engineering exercise rather than a research question, but still an important prerequisite for scaling the continuous-time formulation to cases where the unrolled computation graph does not fit in memory.

A Dataset

All experiments use a 64×64 variant of the CLEVR-with-masks dataset from Google DeepMind's *Multi-Object Datasets* release of CLEVR [56]. The original dataset provides 240×320 rendered scenes containing 3 to 10 objects of varying shape (cube, sphere, and cylinder), color (8 different ones), and materials (metal and rubber), along with per-object segmentation masks and visibility flags.

Preprocessing. We download the raw TFRecords and resize to 64×64. Images are bilinearly downsampled, masks are downsampled by nearest-neighbor interpolation to preserve binary boundaries. We retain all 11 mask channels per scene (one per object slot in the original rendering padded with empty masks for scenes that have fewer objects), along with the corresponding visibility vector. Images (PNGs) are normalized to $[-1, 1]$ at load time via $x \leftarrow \frac{x}{127.5-1}$. Masks and visibility vectors are stored as NumPy arrays.

Train and validation splits. The downsampled dataset contains 95,000 training and 5,000 validation images, matching the original CLEVR-with-masks split. We do not use a held-out test set as the original release does not provide one. We train using the training split and report all numbers on the validation split.

Scene and object feature distributions. Each scene contains between 3 and 10 visible objects, with a roughly uniform distribution over object counts. The distribution of object color and shape is also roughly uniform across all scenes. The visibility vector is a length-11 binary indicator that marks which of the 11 mask channels actually contains an object - the first slot is always background and the remaining ten are 1 for slots holding a rendered object and 0 for unused padding slots.

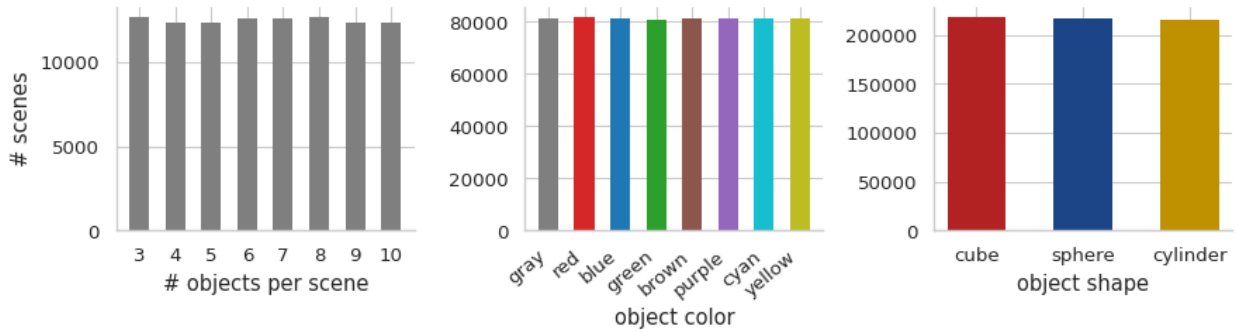

Figure 10. Distributions of scene and object features over the entire CLEVR-with-masks dataset. The number of visible objects per scene (left), per-object color with bars colored to match (center), and per-object shape (right). All three are approximately uniform across their respective categories - so, per-object count, per-color, and per-shape metrics are not biased by class imbalance.

B Training Setup

Architecture. All SlotODE runs use $N = 11$ slots (matching the maximum object count plus one background slot), slot dimension $D_{\text{slot}} = 64$, and encoder hidden dimension $D_{\text{enc}} = 64$. Encoder, decoder, and slot initialization parameters follow the standard Slot Attention configuration of [18], with the GRU-based slot update replaced by the autonomous neural ODE described in Section 3.3. The discrete Slot Attention baseline uses $T = 3$ refinement iterations, matching [18]. SlotODE variants are trained at $T \in \{3, 4, 5, 6\}$ with Euler step sizes $\Delta t \in \{0.5, 1\}$, giving an effective number of training-time refinement steps $\frac{T}{\Delta t}$ ranging from 3 to 12 across configurations.

Loss. The training objective is pixel-wise MSE between the input image and per-slot mixture reconstruction. No auxiliary losses, slot-level supervision, or stability regularization are applied. The objective is identical across SlotODE variants and the Slot Attention baseline.

Optimization. We train with Adam [48] at $\beta_1 = 0.9$, $\beta_2 = 0.95$, and global-norm gradient clipping at 1.0. The lower β_2 (vs. the typical 0.999) follows [18] and shortens the second-moment averaging window to track the rapidly-shifting gradient scale produced by competing slots early in training. Clipping is applied to raw gradients before the Adam update. The learning rate follows a linear warmup from 0 to a peak of 4×10^{-4} over 10,000 steps, then exponential decay with rate 0.5 over 100,000-step intervals. We use a batch size of 64 and train for 500,000 total steps, checkpointing every 10,000 and validating every 5,000. The random seed is fixed at 42 across all reported runs.

Convergence. Figure 11 reports validation ARI-FG and reconstruction MSE over training for both SlotODE variants ($T = 3$, $\Delta t = 1$ and $\Delta t = 0.5$) along with the discrete Slot Attention baseline ($T = 3$). All three models converge to near-identical final performance. The convergence trajectories are closely matched. The baseline reaches an ARI-FG of 0.80 slightly earlier (10K vs. 20K steps for both SlotODE variants), while SlotODE at $\Delta t = 1$ crosses 0.97 before the baseline (100K vs. 130K steps). Coarser discretization $\Delta t = 1$ converges marginally faster than $\Delta t = 0.5$ in step count. Training and validation curves remain tightly coupled throughout, with no evidence of overfitting. Reparameterizing the discrete update as a neural ODE does not degrade convergence.

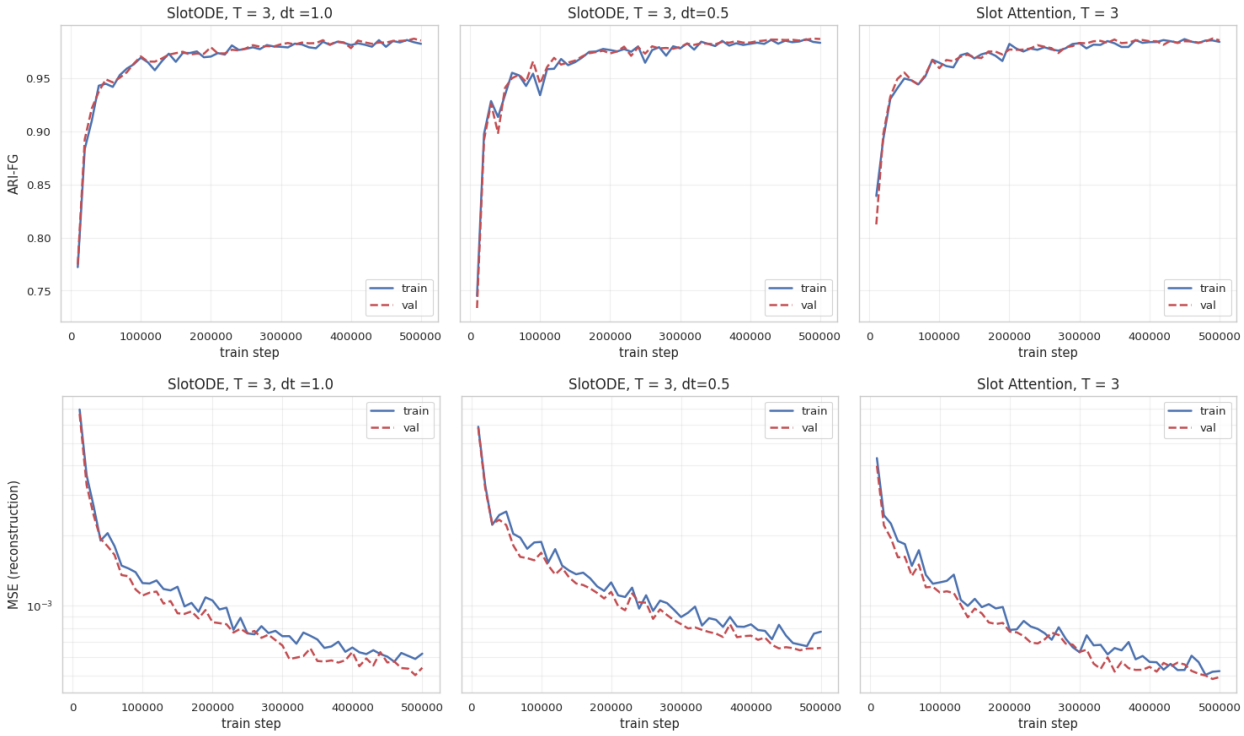

Figure 11. Validation ARI-FG over training steps for SlotODE ($\Delta t = 1$, $\Delta t = 0.5$) and the Slot Attention baseline, all trained at $T = 3$. The solid blue line tracks training and the red dashed line tracks validation. All three models converge to indistinguishable final quality (ARI-FG $\approx 0.98+$).

Hardware and runtime. All SlotODE training runs were performed on a TPU v4 pod with default `bfloat16` mixed precision. The discrete Slot Attention baseline was trained on a single NVIDIA A100 GPU on Modal. End-to-end wall-clock training time for a 500,000-step run is approximately 4 to 7 hours on a TPU v4 pod and approximately 24 to 36 hours on a single NVIDIA A100 GPU.

C Implementation

The SlotODE codebase is written in JAX. We use Equinox [57] to define neural network modules as PyTrees, Optax [58] for gradient-based optimization, and Diffrax [49] for ODE integration. Parameterizing slot dynamics as a vector field needs a differential equation library that composes cleanly with autodiff and `jit` with Diffrax being a natural fit on top of JAX. The same framework also supports adaptive solvers and stochastic differential equations with minimal code changes.

Equinox represents every module as a frozen dataclass whose fields are JAX arrays (parameters) or static metadata. This makes the entire model a single PyTree - gradients, optimizer state, and serialized checkpoints all share the model's tree structure. Filtered transformations (`eqx.filter_jit`, `eqx.filter_grad`, and `eqx.filter_vmap`) automatically partition arrays from non-array fields, so the same module can be passed unchanged through `jit`, `grad`, and `vmap`.

The slot dynamics live in a single `eqx.Module` whose `__call__` matches the signature Diffrax expects for an `ODETerm`, which is $f(t, y, \text{args})$. Time is unused (the system is autonomous) and the precomputed keys and values are passed through `args`, so they remain fixed across integration.

```python
class SlotODEFunc(eqx.Module):
    W_q: jax.Array
    W_gate: jax.Array
    W_ff0: jax.Array
    W_ff1: jax.Array
    norm_attn: eqx.nn.LayerNorm
    norm_ff: eqx.nn.LayerNorm
    scale: float
    slot_dim: int = eqx.field(static=True)
    mlp_hidden: int = eqx.field(static=True)

    def __init__(self, slot_dim: int, mlp_hidden: int = 128, *, key: jax.Array):
        k1, k2, k3, k4 = jax.random.split(key, 4)

        self.scale = slot_dim ** -0.5
        self.slot_dim = slot_dim
        self.mlp_hidden = mlp_hidden

        self.W_q = jax.random.normal(k1, (slot_dim, slot_dim)) * (slot_dim ** -0.5)
        self.W_gate = jax.random.normal(k2, (slot_dim, 2 * slot_dim)) * ((2 * slot_dim) ** -0.5)
        self.W_ff0 = jax.random.normal(k3, (mlp_hidden, 2 * slot_dim)) * ((2 * slot_dim) ** -0.5)
        self.W_ff1 = jax.random.normal(k4, (slot_dim, mlp_hidden)) * (mlp_hidden ** -0.5)

        self.norm_attn = eqx.nn.LayerNorm(slot_dim)
        self.norm_ff = eqx.nn.LayerNorm(slot_dim)

```

```
    def __call__(self, t, slots, args):
        # slots: [B, N, D],  args = (k, v) with k, v: [B, M, D]
        k, v = args

        slots_norm = jax.vmap(jax.vmap(self.norm_attn))(slots)
        q = jnp.einsum('bnd,od->bno', slots_norm, self.W_q)

        att_logits = jnp.einsum('bnd,bmd->bnm', q, k) * self.scale
        att = jax.nn.softmax(att_logits, axis=1) # softmax over slots
        att = att / (att.sum(axis=-1, keepdims=True) + 1e-8) # normalize over features
        f_attn = jnp.einsum('bnm,bmd->bnd', att, v)

        gate_in = jnp.concatenate([slots_norm, f_attn], axis=-1)
        gate = jax.nn.sigmoid(jnp.einsum('bnd,od->bno', gate_in, self.W_gate))

        slots_ff = jax.vmap(jax.vmap(self.norm_ff))(slots)
        h = jnp.einsum('bnd,od->bno', jnp.concatenate([slots_ff, f_attn], axis=-1), self.W_ff0)
        h = jax.nn.relu(h)
        h = jnp.einsum('bnd,od->bno', h, self.W_ff1)

        return gate * f_attn + h
```

The softmax over slots (line 20, `axis=1`) is the source of slot competition for input features, and is the only place where slots interact with each other inside one evaluation of f_θ. Both LayerNorms are applied per-slot via a double `vmap` (the inner one over slots and outer one over batch).

The integration is set up once per forward pass - term, solver, step-size controller, and save points are constructed locally so they can be configured per-call (e.g. `return_traj` switches between saving only the terminal state and saving at every Euler step). We can explicitly request `SaveAt(ts=...)` aligned with integration steps to avoid any interpolation of intermediate states.

```
def __call__(self, enc_feat, key, return_traj=False):
    feat = jax.vmap(jax.vmap(self.norm_input))(enc_feat)
    feat = jax.vmap(jax.vmap(self.fc_input))(feat)
    k = jax.vmap(jax.vmap(self.to_k))(feat)
    v = jax.vmap(jax.vmap(self.to_v))(feat)

    slots_0 = self.initialize_slots(enc_feat.shape[0], key)

    term = diffrax.ODETerm(self.slot_ode_func)
    solver = diffrax.Euler()
    stepctrl = diffrax.ConstantStepSize()

    if return_traj:
```

```
        ts = jnp.clip(jnp.arange(0.0, self.T + self.dt0, self.dt0), 0.0, self.T)
        saveat = diffrax.SaveAt(ts=ts)
    else:
        saveat = diffrax.SaveAt(t1=True)

    n_steps = int(self.T / self.dt0)
    sol = diffrax.diffeqsolve(
        term, solver,
        t0=0.0, t1=self.T, dt0=self.dt0,
        y0=slots_0, args=(k, v),
        saveat=saveat, stepsize_controller=stepctrl, max_steps=n_steps + 16,
    )
    return sol.ys[-1] if return_traj else sol.ys[0]
```

Because Diffrax unrolls the solver under `jit`, the entire forward pass - encoder, integration, decoder - is a single fused computation graph. The backward pass backpropagates through the unrolled Euler steps, which is what we use throughout this work. Switching to the adjoint method requires replacing the default `adjoint` argument with `diffrax.BacksolveAdjoint()`, with no changes to the model itself.

Training is one filtered `jit` that wraps a value-and-grad pass and an Optax update. `eqx.filter_value_and_grad` traces the loss with respect to the array leaves of the model and treats everything else as static. `eqx.filter` extracts the array partition needed by the optimizer.

```
@eqx.filter_jit
def train_step(model, opt_state, optimizer, images, key):
    def loss_fn(model):
        recon, masks, slots = model(images, key=key)
        return jnp.mean((recon - images) ** 2)

    loss, grads = eqx.filter_value_and_grad(loss_fn)(model)

    updates, opt_state = optimizer.update(
        grads, opt_state, eqx.filter(model, eqx.is_array)
    )

    model = eqx.apply_updates(model, updates)

    return model, opt_state, loss
```

The optimizer is built with Optax. Optax expresses optimizers as compositions of stateless gradient transformations, each mapping (gradients, state, and params) to (updates and new state). `optax.chain` threads these transformations in sequence. Here, we apply global-norm gradient clipping before the Adam update so clipping operates on raw gradients rather than on Adam's normalized step. See Appendix B for specifics on the learning rate schedule and gradient clipping.

```
schedule = optax.join_schedules(
    [
        optax.linear_schedule(0.0, args.lr, args.warmup_steps),
        optax.exponential_decay(init_value=args.lr,
                                transition_steps=args.decay_steps,
                                decay_rate=args.decay_rate)
    ],
    boundaries=[args.warmup_steps],
)

optimizer = optax.chain(
    optax.clip_by_global_norm(args.grad_clip),
    optax.adam(schedule, b2=0.95),
)

opt_state = optimizer.init(eqx.filter(model, eqx.is_array))
```

TPU runs use the JAX default of `bfloat16` for matrix multiplication and `float32` for accumulation (no manual casting is required). On GPU, the same defaults use `float32` throughout. The GPU runs match TPU runs within numerical noise on the held-out validation set, where final ARI-FG can drift by roughly 10^{-3} between the two precisions - small but non-zero. However, this does not meaningfully affect the results and drawn conclusions for any experiments.

JAX's PRNG is functional - every stochastic operation requires an explicit key that is split rather than reseeded. Dataset shuffling, slot initialization, and validation sampling each use their own subkey derived from a fixed seed (`42` for all reported runs). This makes training runs a deterministic function of seed, configuration, and JAX version - the only sources of untracked randomness that remain are the order of nonassociative reductions in CUDA/XLA and hardware-level `bfloat16` rounding behavior on TPUs. None of these have a meaningful affect on any result at the precision we report.

References

[1] Anne Treisman. "The Binding Problem". In: *Current Opinion in Neurobiology* 6.2 (1996), pp. 171–178. DOI: 10.1016/S0959-4388(96)80070-5.

[2] Christoph von der Malsburg. "The What and Why of Binding: The Modeler's Perspective". In: *Neuron* 24 (Sept. 1999), pp. 95–104.

[3] Robert Desimone and John Duncan. "Neural Mechanisms of Selective Visual Attention". In: *Annual Review of Neuroscience* 18 (1995), pp. 193–222. DOI: 10.1146/annurev.ne.18.030195.001205.

[4] Hong Zhou, Howard S. Friedman, and Rüdiger Von Der Heydt. "Coding of Border Ownership in Monkey Visual Cortex". In: *The Journal of Neuroscience* 20.17 (Sept. 1, 2000), pp. 6594–6611. ISSN: 0270-6474, 1529-2401. DOI: 10.1523/JNEUROSCI.20-17-06594.2000. URL: https://www.jneurosci.org/lookup/doi/10.1523/JNEUROSCI.20-17-06594.2000.

[5] Tai Sing Lee and David Mumford. "Hierarchical Bayesian inference in the visual cortex". In: *Journal of the Optical Society of America A* 20.7 (July 1, 2003), p. 1434. ISSN: 1084-7529, 1520-8532. DOI: 10.1364/JOSAA.20.001434. URL: https://opg.optica.org/abstract.cfm?URI=josaa-20-7-1434.

[6] Hugh R. Wilson and Jack D. Cowan. "Excitatory and Inhibitory Interactions in Localized Populations of Model Neurons". In: *Biophysical Journal* 12.1 (Jan. 1972), pp. 1–24. ISSN: 00063495. DOI: 10.1016/S0006-3495(72)86068-5. URL: https://linkinghub.elsevier.com/retrieve/pii/S0006349572860685.

[7] Peter W. Battaglia et al. *Relational inductive biases, deep learning, and graph networks*. Oct. 17, 2018. DOI: 10.48550/arXiv.1806.01261. arXiv: 1806.01261[cs]. URL: http://arxiv.org/abs/1806.01261.

[8] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. *On the Binding Problem in Artificial Neural Networks*. Dec. 9, 2020. DOI: 10.48550/arXiv.2012.05208. arXiv: 2012.05208[cs]. URL: http://arxiv.org/abs/2012.05208.

[9] Fucai Ke et al. *Explain Before You Answer: A Survey on Compositional Visual Reasoning*. Aug. 27, 2025. DOI: 10.48550/arXiv.2508.17298. arXiv: 2508.17298[cs]. URL: http://arxiv.org/abs/2508.17298.

[10] Kexin Yi et al. "CLEVRER: CoLlision Events for Video REpresentation and Reasoning". In: *International Conference on Learning Representations (ICLR)*. arXiv:1910.01442. arXiv, Mar. 8, 2020. DOI: 10.48550/arXiv.1910.01442. arXiv: 1910.01442[cs]. URL: http://arxiv.org/abs/1910.01442.

[11] Tejas Kulkarni et al. "Unsupervised Learning of Object Keypoints for Perception and Control". In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1906.11883. arXiv, Nov. 20, 2019. DOI: 10.48550/arXiv.1906.11883. arXiv: 1906.11883[cs]. URL: http://arxiv.org/abs/1906.11883.

[12] Chen Sun et al. *Stochastic Prediction of Multi-Agent Interactions from Partial Observations*. Feb. 25, 2019. DOI: 10.48550/arXiv.1902.09641. arXiv: 1902.09641[cs]. URL: http://arxiv.org/abs/1902.09641.

[13] OpenAI et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. Dec. 13, 2019. DOI: 10.48550/arXiv.1912.06680. arXiv: 1912.06680[cs]. URL: http://arxiv.org/abs/1912.06680.

[14] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (Nov. 14, 2019), pp. 350–354. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-019-1724-z. URL: https://www.nature.com/articles/s41586-019-1724-z.

[15] Peter W. Battaglia et al. “Interaction Networks for Learning about Objects, Relations and Physics”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1612.00222. arXiv, Dec. 1, 2016. DOI: 10.48550/arXiv.1612.00222. arXiv: 1612.00222[cs]. URL: http://arxiv.org/abs/1612.00222.

[16] Damian Mrowca et al. “Flexible Neural Representation for Physics Prediction”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1806.08047. arXiv, Oct. 27, 2018. DOI: 10.48550/arXiv.1806.08047. arXiv: 1806.08047[cs]. URL: http://arxiv.org/abs/1806.08047.

[17] Alvaro Sanchez-Gonzalez et al. “Learning to Simulate Complex Physics with Graph Networks”. In: *International Conference on Machine Learning (ICML)*. arXiv:2002.09405. arXiv, Sept. 14, 2020. DOI: 10.48550/arXiv.2002.09405. arXiv: 2002.09405[cs]. URL: http://arxiv.org/abs/2002.09405.

[18] Francesco Locatello et al. “Object-Centric Learning with Slot Attention”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:2006.15055. arXiv, Oct. 14, 2020. DOI: 10.48550/arXiv.2006.15055. arXiv: 2006.15055[cs]. URL: http://arxiv.org/abs/2006.15055.

[19] Yilun Du and Igor Mordatch. “Implicit Generation and Generalization in Energy-Based Models”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1903.08689. arXiv, June 30, 2020. DOI: 10.48550/arXiv.1903.08689. arXiv: 1903.08689[cs]. URL: http://arxiv.org/abs/1903.08689.

[20] Yann LeCun et al. “A Tutorial on Energy-Based Learning”. In: (2006).

[21] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. “Deep Equilibrium Models”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1909.01377. arXiv, Oct. 28, 2019. DOI: 10.48550/arXiv.1909.01377. arXiv: 1909.01377[cs]. URL: http://arxiv.org/abs/1909.01377.

[22] Marcin Andrychowicz et al. “Learning to learn by gradient descent by gradient descent”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1606.04474. arXiv, Nov. 30, 2016. DOI: 10.48550/arXiv.1606.04474. arXiv: 1606.04474[cs]. URL: http://arxiv.org/abs/1606.04474.

[23] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *International Conference on Machine Learning (ICML)*. arXiv:1703.03400. arXiv, July 18, 2017. DOI: 10.48550/arXiv.1703.03400. arXiv: 1703.03400[cs]. URL: http://arxiv.org/abs/1703.03400.

[24] Erin Grant et al. “Recasting Gradient-Based Meta-Learning as Hierarchical Bayes”. In: *International Conference on Learning Representations (ICLR)*. arXiv:1801.08930. arXiv, Jan. 26, 2018. DOI: 10.48550/arXiv.1801.08930. arXiv: 1801.08930[cs]. URL: http://arxiv.org/abs/1801.08930.

[25] Gamaleldin F. Elsayed et al. “SAVi++: Towards End-to-End Object-Centric Learning from Real-World Videos”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:2206.07764. arXiv, Dec. 23, 2022. DOI: 10.48550/arXiv.2206.07764. arXiv: 2206.07764[cs]. URL: http://arxiv.org/abs/2206.07764.

[26] Sjoerd van Steenkiste et al. “Relational Neural Expectation Maximization: Unsupervised Discovery of Objects and their Interactions”. In: *International Conference on Learning Representations (ICLR)*. arXiv:1802.10353. arXiv, Feb. 28, 2018. DOI: 10.48550/arXiv.1802.10353. arXiv: 1802.10353[cs]. URL: http://arxiv.org/abs/1802.10353.

[27] Rishi Veerapaneni et al. “Entity Abstraction in Visual Model-Based Reinforcement Learning”. In: *Conference on Robot Learning (CoRL)*. arXiv:1910.12827. arXiv, May 6, 2020. DOI: 10.48550/arXiv.1910.12827. arXiv: 1910.12827[cs]. URL: http://arxiv.org/abs/1910.12827.

[28] Thomas Kipf et al. “Conditional Object-Centric Learning from Video”. In: *International Conference on Learning Representations (ICLR)*. arXiv:2111.12594. arXiv, Mar. 15, 2022. DOI: 10.48550/arXiv.2111.12594. arXiv: 2111.12594[cs]. URL: http://arxiv.org/abs/2111.12594.

[29] Jascha Sohl-Dickstein et al. “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. In: *International Conference on Machine Learning (ICML)*. arXiv:1503.03585. arXiv, Nov. 18, 2015. DOI: 10.48550/arXiv.1503.03585. arXiv: 1503.03585[cs]. URL: http://arxiv.org/abs/1503.03585.

[30] Yang Song and Stefano Ermon. “Generative Modeling by Estimating Gradients of the Data Distribution”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1907.05600. arXiv, Oct. 10, 2020. DOI: 10.48550/arXiv.1907.05600. arXiv: 1907.05600[cs]. URL: http://arxiv.org/abs/1907.05600.

[31] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising Diffusion Probabilistic Models”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:2006.11239. arXiv, Dec. 16, 2020. DOI: 10.48550/arXiv.2006.11239. arXiv: 2006.11239[cs]. URL: http://arxiv.org/abs/2006.11239.

[32] Nikolay Savinov et al. “Step-unrolled Denoising Autoencoders for Text Generation”. In: *International Conference on Learning Representations (ICLR)*. arXiv:2112.06749. arXiv, Apr. 19, 2022. DOI: 10.48550/arXiv.2112.06749. arXiv: 2112.06749[cs]. URL: http://arxiv.org/abs/2112.06749.

[33] Ricky T. Q. Chen et al. “Neural Ordinary Differential Equations”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1806.07366. arXiv, Dec. 14, 2019. DOI: 10.48550/arXiv.1806.07366. arXiv: 1806.07366[cs]. URL: http://arxiv.org/abs/1806.07366.

[34] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:1706.03762. arXiv, Aug. 2, 2023. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762[cs]. URL: http://arxiv.org/abs/1706.03762.

[35] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. arXiv:1406.1078. arXiv, Sept. 3, 2014. DOI: 10.48550/arXiv.1406.1078. arXiv: 1406.1078[cs]. URL: http://arxiv.org/abs/1406.1078.

[36] Nicholas Watters et al. *Spatial Broadcast Decoder: A Simple Architecture for Learning Disentangled Representations in VAEs*. Aug. 14, 2019. DOI: 10.48550/arXiv.1901.07017. arXiv: 1901.07017[cs]. URL: http://arxiv.org/abs/1901.07017.

[37] Gautam Singh, Fei Deng, and Sungjin Ahn. "Illiterate DALL-E Learns to Compose". In: *International Conference on Learning Representations (ICLR)*. arXiv:2110.11405. arXiv, Mar. 14, 2022. DOI: 10.48550/arXiv.2110.11405. arXiv: 2110.11405[cs]. URL: http://arxiv.org/abs/2110.11405.

[38] Maximilian Seitzer et al. "Bridging the Gap to Real-World Object-Centric Learning". In: *International Conference on Learning Representations (ICLR)*. arXiv:2209.14860. arXiv, Mar. 6, 2023. DOI: 10.48550/arXiv.2209.14860. arXiv: 2209.14860[cs]. URL: http://arxiv.org/abs/2209.14860.

[39] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. arXiv:1512.03385. arXiv, Dec. 10, 2015. DOI: 10.48550/arXiv.1512.03385. arXiv: 1512.03385[cs]. URL: http://arxiv.org/abs/1512.03385.

[40] Eldad Haber and Lars Ruthotto. "Stable Architectures for Deep Neural Networks". In: *Inverse Problems* 34.1 (Jan. 1, 2018), p. 014004. ISSN: 0266-5611, 1361-6420. DOI: 10.1088/1361-6420/aa9a90. arXiv: 1705.03341[cs]. URL: http://arxiv.org/abs/1705.03341.

[41] Yiping Lu et al. "Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations". In: *International Conference on Machine Learning (ICML)*. arXiv:1710.10121. arXiv, Mar. 23, 2020. DOI: 10.48550/arXiv.1710.10121. arXiv: 1710.10121[cs]. URL: http://arxiv.org/abs/1710.10121.

[42] Yiping Lu et al. *Understanding and Improving Transformer From a Multi-Particle Dynamic System Point of View*. June 6, 2019. DOI: 10.48550/arXiv.1906.02762. arXiv: 1906.02762[cs]. URL: http://arxiv.org/abs/1906.02762.

[43] Borjan Geshkovski et al. *A mathematical perspective on Transformers*. Aug. 21, 2025. DOI: 10.48550/arXiv.2312.10794. arXiv: 2312.10794[cs]. URL: http://arxiv.org/abs/2312.10794.

[44] Borjan Geshkovski et al. *The emergence of clusters in self-attention dynamics*. Feb. 12, 2024. DOI: 10.48550/arXiv.2305.05465. arXiv: 2305.05465[cs]. URL: http://arxiv.org/abs/2305.05465.

[45] Anh Tong et al. *Neural ODE Transformers: Analyzing Internal Dynamics and Adaptive Fine-tuning*. Apr. 16, 2025. DOI: 10.48550/arXiv.2503.01329. arXiv: 2503.01329[cs]. URL: http://arxiv.org/abs/2503.01329.

[46] Michael Chang, Thomas L. Griffiths, and Sergey Levine. "Object Representations as Fixed Points: Training Iterative Refinement Algorithms with Implicit Differentiation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:2207.00787. arXiv, Jan. 1, 2023. DOI: 10.48550/arXiv.2207.00787. arXiv: 2207.00787[cs]. URL: http://arxiv.org/abs/2207.00787.

[47] Rishabh Kabra et al. *Multi-Object Datasets*. https://github.com/deepmind/multi-object-datasets/. 2019.

[48] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. arXiv:1412.6980. arXiv, Jan. 30, 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980[cs]. URL: http://arxiv.org/abs/1412.6980.

[49] Patrick Kidger. "On Neural Differential Equations". PhD thesis. University of Oxford, 2021.

[50] Hassan K. Khalil. *Nonlinear Systems*. 3rd. Prentice Hall, 2002.

[51] Winfried Lohmiller and Jean-Jacques E. Slotine. “On Contraction Analysis for Non-Linear Systems”. In: *Automatica* 34.6 (1998), pp. 683–696.

[52] David Sussillo and Omri Barak. “Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks”. In: *Neural Computation* 25.3 (2013), pp. 626–649.

[53] Marco Fumero et al. “Navigating the Latent Space Dynamics of Neural Models”. In: *International Conference on Learning Representations (ICLR)*. 2026.

[54] Hubert Ramsauer et al. “Hopfield Networks is All You Need”. In: *International Conference on Learning Representations (ICLR)*. 2021. URL: https://arxiv.org/abs/2008.02217.

[55] David Ha, Andrew M. Dai, and Quoc V. Le. “HyperNetworks”. In: *International Conference on Learning Representations (ICLR)*. 2017. URL: https://arxiv.org/abs/1609.09106.

[56] Justin Johnson et al. “CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. arXiv:1612.06890. arXiv, Dec. 20, 2016. DOI: 10.48550/arXiv.1612.06890. arXiv: 1612.06890[cs]. URL: http://arxiv.org/abs/1612.06890.

[57] Patrick Kidger and Cristian Garcia. “Equinox: neural networks in JAX via callable PyTrees and filtered transformations”. In: *Differentiable Programming workshop at Neural Information Processing Systems 2021* (2021).

[58] DeepMind et al. *The DeepMind JAX Ecosystem*. 2020. URL: http://github.com/google-deepmind.

[59] Christopher P. Burgess et al. *MONet: Unsupervised Scene Decomposition and Representation*. Jan. 22, 2019. DOI: 10.48550/arXiv.1901.11390. arXiv: 1901.11390[cs]. URL: http://arxiv.org/abs/1901.11390.

[60] Jiarui Xu et al. “GroupViT: Semantic Segmentation Emerges from Text Supervision”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. arXiv:2202.11094. arXiv, July 18, 2022. DOI: 10.48550/arXiv.2202.11094. arXiv: 2202.11094[cs]. URL: http://arxiv.org/abs/2202.11094.

[61] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[62] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[63] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[64] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[65] Jason Ansel et al. “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation”. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS ’24)*. ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. URL: https://docs.pytorch.org/assets/pytorch2-2.pdf.

[66] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/jax-ml/jax.

[67] Matei A. Zaharia et al. "Accelerating the Machine Learning Lifecycle with MLflow". In: *IEEE Data Eng. Bull.* 41 (2018), pp. 39–45. URL: https://api.semanticscholar.org/CorpusID:83459546.

[68] Klaus Greff et al. "Kubric: A Scalable Dataset Generator". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022. URL: https://arxiv.org/abs/2203.03570.