Bubblewrap: Online tiling and real-time flow prediction on neural manifolds

Anne Draelos
Biostatistics & Bioinformatics
Duke University
anne.draelos@duke.edu

Pranjal Gupta
Psychology & Neuroscience
Duke University
pranjal.gupta@duke.edu

Na Young Jun
Neurobiology
Duke University
nayoung.jun@duke.edu

Chaichontat Sriworarat
Biomedical Engineering
Duke University
chaichontat.s@duke.edu

John Pearson
Biostatistics & Bioinformatics
Electrical & Computer Engineering
Neurobiology
Psychology & Neuroscience
Duke University
john.pearson@duke.edu

Abstract

While most classic studies of function in experimental neuroscience have focused on the coding properties of individual neurons, recent developments in recording technologies have resulted in an increasing emphasis on the dynamics of neural populations. This has given rise to a wide variety of models for analyzing population activity in relation to experimental variables, but direct testing of many neural population hypotheses requires intervening in the system based on current neural state, necessitating models capable of inferring neural state online. Existing approaches, primarily based on dynamical systems, require strong parametric assumptions that are easily violated in the noise-dominated regime and do not scale well to the thousands of data channels in modern experiments. To address this problem, we propose a method that combines fast, stable dimensionality reduction with a soft tiling of the resulting neural manifold, allowing dynamics to be approximated as a probability flow between tiles. This method can be fit efficiently using online expectation maximization, scales to tens of thousands of tiles, and outperforms existing methods when dynamics are noise-dominated or feature multi-modal transition probabilities. The resulting model can be trained at kiloHertz data rates, produces accurate approximations of neural dynamics within minutes, and generates predictions on submillisecond time scales. It retains predictive performance throughout many time steps into the future and is fast enough to serve as a component of closed-loop causal experiments.

1 Introduction

Systems neuroscience is in the midst of a data explosion. Advances in microscopy [1, 2] and probe technology [3, 4, 5] have made it possible to record thousands of neurons simultaneously in behaving

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

animals. At the same time, growing interest in naturalistic behaviors has increased both the volume and complexity of jointly recorded behavioral data. On the neural side, this has resulted in a host of new modeling and analysis approaches that aim to match the complexity of these data, typically using artificial neural network models as proxies for biological neural computation [6, 7, 8].

At the same time, this increase in data volume has resulted in increasing emphasis on methods for dimensionality reduction [9] and a focus on neural populations in preference to the coding properties of individual neurons [10]. However, given the complexity of neural dynamics, it remains difficult to anticipate what experimental conditions will be needed to test population hypotheses in *post hoc* analyses, complicating experimental design and reducing power. Conversely, adaptive experiments, those in which the conditions tested change in response to incoming data, have been used in neuroscience to optimize stimuli for experimental testing [11, 12, 13, 14], in closed-loop designs [15, 16, 17], and even to scale up holographic photostimulation for inferring functional connectivity in large circuits [18].

Yet, despite their promise, adaptive methods are rarely applied in practice for two reasons: First, although efficient online methods for dimensionality reduction exist [19, 20, 21, 22, 23], these methods do not typically identify *stable* dimensions to allow low-dimensional representations of data to be compared across time points. That is, when the spectral properties of the data are changing in time, methods like incremental SVD may be projecting the data into an unstable basis, rendering these projections unsuitable as inputs to further modeling. Second, while many predictive models based on the dynamical systems approach exist [6, 24, 25, 26, 27, 28, 29], including online approaches [30, 31, 16, 32], they typically assume a system with lawful dynamics perturbed by Gaussian noise. However, many neural systems of interest are noise-dominated, with multimodal transition kernels between states.

In this work, we are specifically interested in closed loop experiments in which predictions of future neural state are needed in order to time and trigger interventions like optogenetic stimulation or a change in visual stimulus. Thus, our focus is on predictive accuracy, preferably far enough into the future to compensate for feedback latencies. To address these goals, we propose an alternative to the linear systems approach that combines a fast, stable, online dimensionality reduction with a semiparametric tiling of the low-dimensional neural manifold. This tiling introduces a discretization of the neural state space that allows dynamics to be modeled as a Hidden Markov Model defined by a sparse transition graph. The entire model, which we call "Bubblewrap," can be learned online using a simple EM algorithm and handles tilings and graphs of up to thousands of nodes at kiloHertz data rates. Most importantly, this model outperforms methods based on dynamical systems in high-noise regimes when the dynamics are more diffusion-like. Training can be performed at a low, fixed latency $\approx$10ms using a GPU, while a cached copy of the model in main memory is capable of predicting upcoming states at $<$1ms latency. As a result, Bubblewrap offers a method performant and flexible enough to serve as a neural prediction engine for causal feedback experiments.

2 Stable subspaces from streaming SVD

As detailed above, one of the most pressing issues in online neural modeling is dealing with the increasingly large dimensionality of collected data — hundreds of channels per Neuropixels probe [4, 5], tens of thousands of pixels for calcium imaging. However, as theoretical work has shown [33, 34], true neural dynamics often lie on a low-dimensional manifold, so that population activity can be accurately captured by analyzing only a few variables.

Here, we combine two approaches to data reduction: In the first stage, we use sparse random projections to reduce dimensionality from an initial d dimensions (thousands) to n (a few hundred) [35, 36]. By simple scaling, for a fixed budget of N cells in our manifold tiling, we expect density (and thus predictive accuracy) to scale as $N^{\frac{1}{n}}$ in dimension n, and so we desire n to be as small as possible. However, by the Johnson-Lindenstrauss Lemma [37, 36], when reducing from d to n dimensions, the distance between vectors u_* and v_* in the reduced space is related to the distance between their original versions u and v by

$$(1-\varepsilon)\|u-v\|^2 \leq \|u_*-v_*\|^2 \leq (1+\varepsilon)\|u-v\|^2 \tag{1}$$

with probability $1-\delta$ if $n > \mathcal{O}(\log(1/\delta)/\varepsilon^2)$. Unfortunately, even for $\varepsilon \sim 0.1$ (10% relative error), the required n may be quite large, making this inappropriate for reducing to the very small numbers of effective dimensions characterizing neural datasets.

Algorithm 1 Procrustean SVD (proSVD)

1: **Given:** Initial data X_0, decay parameter $\alpha \in (0, 1]$
2: **Initialize:** QR Factorization: $X_0 = Q_0 R_0$
3:
4: **for** $t = 1 \ldots$ **do**
5: Fetch b new columns of data, X_+
6: $C \leftarrow Q_{t-1}^\top X_+, \quad X_\perp \leftarrow X_+ - Q_{t-1}C, \quad Q_\perp, R_\perp \leftarrow \texttt{QR}(X_\perp)$ ▷ Gram-Schmidt
7: $\hat{Q} \leftarrow [Q_{t-1} \quad Q_\perp], \quad \hat{R} \leftarrow \begin{bmatrix} R_{t-1} & C \\ 0 & R_\perp \end{bmatrix}$ ▷ QR of augmented data
8: $U, \Sigma, V \leftarrow \texttt{SVD}(\hat{R})$
9: $\Sigma \leftarrow \alpha\Sigma$ ▷ Discount old data
10: $M \leftarrow Q_{t-1}^\top \hat{Q} U_1 = [\mathbb{1}_{k\times k} \quad \mathbf{0}_{k\times b}]\, U_1$ ▷ U_1 contains the first k columns of U
11: $\tilde{U}, \tilde{\Sigma}, \tilde{V} \leftarrow \texttt{SVD}(M), \quad T \leftarrow \tilde{U}\tilde{V}^\top$ ▷ Orthogonal Procrustes: $\min_T \|\hat{Q}U_1T^\top - Q_{t-1}\|_F$
12: $Q_t \leftarrow \hat{Q}U_1T^\top$ ▷ Update left subspace basis
13: $Q_v, R_v \leftarrow \texttt{QR}(V), \quad R_t \leftarrow T\Sigma_1 Q_v^\top$ ▷ QR right subspace, update inner block
14: **end for**

Thus, in the second stage, we reduce from $n \sim \mathcal{O}(100)$ to $k \sim \mathcal{O}(10)$ dimensions using a streaming singular value decomposition. This method is based on the incremental block update method of [20, 22] with an important difference: While the block update method aims to return the top-k SVD at every time point, the directions of the singular vectors may be quite variable during the course of an experiment (Figure 1d–h), which implies an unstable representation of the neural manifold. However, as we show below, the top-k *subspace* spanned by these vectors stabilizes in seconds on typical neural datasets and remains so throughout the experiment. Therefore, by selecting a stable basis (instead of the singular vector basis) for the top-k subspace, we preserve the same information while ensuring a stable representation of the data for subsequent model fitting.

More specifically, let $\mathbf{x}_t \in \mathbb{R}^n$ be a vector of input data after random projections. In our streaming setup, these are processed b samples at a time, with $b = 1$ reasonable for slower methods like calcium imaging and $b = 40$ more appropriate for electrophysiological sampling rates of ∼20kHz. Then, if the data matrix X has dimension $n \times T$, adding columns over time, the incremental method of [20, 22] produces at each time step a factorization $X = QRW^\top$, where the columns of the orthogonal matrices Q and W span the left and right top-k singular subspaces, respectively. If the matrix R were diagonal, this would be equivalent to the SVD. In the incremental algorithm, R is augmented at each timestep based on new data to form $\hat{R}$, which is block diagonalized via an orthogonal matrix and truncated to the top-k subspace, allowing for an exact reduced-rank SVD (Appendix A).

However, as reviewed in [20, 22], since there are multiple choices of basis Q for for the top-k singular subspace, there are likewise multiple choices of block diagonalization for $\hat{R}$. In [20, 22], the form of this operation is chosen for computational efficiency. But an equally valid option is to select the orthogonal matrix that minimizes the change in the singular subspace basis Q from one timestep to the next:

$$\min \|Q_t - Q_{t-1}\|_F = \min_T \|\hat{Q}U_1T^\top - Q_{t-1}\|_F, \tag{2}$$

where $\hat{Q}$ is an augmented basis for the top-$(k + b)$ singular subspace, U_1 contains the first k left singular vectors of $\hat{R}$, and T is an orthogonal matrix (Appendix A). This minimization is known as the Orthogonal Procrustes problem and has a well-known solution [38]: $T = \tilde{U}\tilde{V}^\top$, where $\tilde{U}$ and $\tilde{V}$ are the left and right singular vectors, respectively, of $M \equiv Q_{t-1}^\top \hat{Q} U_1$. (See [39] for a recent application of similar ideas in brain-computer interfaces). This Procrustean SVD (proSVD) procedure is summarized in Algorithm 1. There, lines 1–8 follow [20, 22], while lines 10 and 11 perform the Orthogonal Procrustes procedure. Line 9 serves as a leak term that discounts past data as in [40].

Figures 1a-c illustrates the performance of the two-stage dimension reduction algorithm for a case of $d = 10^4$ randomly generated Gaussian data. While proSVD yields minimal distortion (due to truncation of the spectrum to $k = 6$), random projections require $k \sim \mathcal{O}(100)$ to achieve the same result (Figure 1a). By contrast, random projections are much faster (Figure 1b). Thus, we can trade off distortion against time by adjusting n, the number of intermediate dimensions. As Figure 1c shows, the optimal tradeoff occurs somewhere around $n = 200$ for this example.

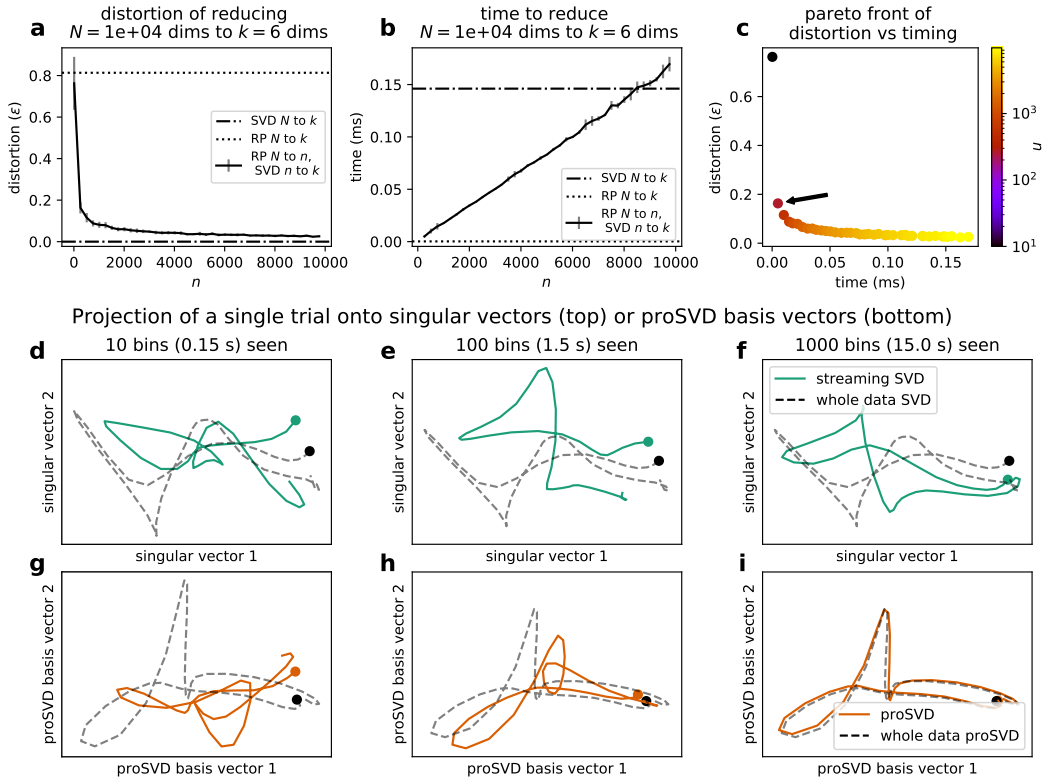

Figure 1: **Timing and stability of two-stage dimension reduction. a)** Distortion (ε) as a function of number of dimensions retained (n) for both sparse random projections and proSVD on random Gaussian data with batch size $b = 1000$. **b)** Time required for the dimensionality reduction in **(a)**, amortized for batch size. While random projections are extremely efficient, proSVD time costs grow with the number of dimensions retained. **c)** Pareto front for the time-distortion tradeoff of random projections followed by proSVD. Color indicates n, the number of dimensions retained by random projections. Black arrow indicates the particular tradeoff we chose of $n = 200$. **d–f)** Embedding of a single trial (green line) into the basis defined by streaming SVD for different amounts of data seen. Dotted line indicates the same trial embedded using SVD on the full data set. Rapid changes in estimates of singular vectors early on lead to an unstable representation. **g–i)** Same trial and conventions as **(d–f)** for the proSVD embedding. Dotted lines in the two rows are the same curve in different projections.

Figures 1d-i show results for neural data from recorded from monkey motor cortex [26] in a cued reach task. While projection of the data into the basis defined by streaming SVD remains unstable early in data collection (top), the proSVD representation is nearly equivalent to the full offline result after only a few trials (≈15s, middle). This is due to the fact that, in all data sets we examined, the top-k SVD *subspace* was identified extremely quickly; proSVD simply ensures the choice of a stable basis for that subspace.

3 Bubblewrap: a soft manifold tiling for online modeling

As reviewed above, most neural population modeling approaches are based on the dynamical systems framework, assuming a lawful equation of motion corrupted by noise. However, for animals engaged in task-free natural behavior [41, 42, 43], trajectories are likely to be sufficiently complex that simple dynamical models fail. For instance, dynamical systems models with Gaussian noise necessarily produce unimodal transition probabilities centered around the mean prediction, while neural trajectories may exhibit multimodal distributions beginning at the same system state. By contrast, we pursue an

alternative method that trades some accuracy in estimating instantaneous system state for flexibility in modeling the manifold describing neural activity.

Our approach is to produce a soft tiling of the neural manifold in the form of a Gaussian mixture model (GMM), each component of which corresponds to a single tile. We then approximate the transitions between tiles via a Hidden Markov Model (HMM), which allows us to capture multimodal probability flows. As the number of tiles increases, the model produces an increasingly finer-grained description of dynamics that assumes neither an underlying dynamical system nor a particular distribution of noise.

More specifically, let x_t be the low-dimensional system state and let $z_t \in 1 \dots N$ index the tile to which the system is assigned at time t. Then we have for the dynamics

$$p(z_t = j | z_{t-1} = i) = A_{ij} \quad p(x_t | z_t) = \mathcal{N}(\mu_{z_t}, \Sigma_{z_t}) \quad p(\mu_j, \Sigma_j) = \mathrm{NIW}(\mu_{0j}, \lambda_j, \Psi_j, \nu_j), \tag{3}$$

where we have assumed Normal-inverse-Wishart priors on the parameters of the Gaussians. Given its exponential family form and the conjugacy of the priors, online expectation maximization updates are available in closed form [44, 45, 46] for each new datum, though we opt, as in [45] for a gradient-based optimization of an estimate of the evidence lower bound

$$\begin{aligned}\mathcal{L}(A, \mu, \Sigma) = &\sum_{ij} (\hat{N}_{ij}(T) + \beta_{ij} - 1) \log A_{ij} + \sum_j (\hat{S}_{1j}(T) + \lambda_j \mu_{0j})^\top \Sigma_j^{-1} \mu_j \\ &- \frac{1}{2} \sum_j \mathrm{tr}((\Psi_j + \hat{S}_{2j}(T) + \lambda_j \mu_{0j} \mu_{0j}^\top + (\lambda_j + \hat{n}_j(T)) \mu_j \mu_j^\top) \Sigma_j^{-1}) \\ &- \frac{1}{2} \sum_j (\nu_j + \hat{n}_j(T) + d + 2) \log \det \Sigma_j\end{aligned} \tag{4}$$

with accumulating (estimated) sufficient statistics

$$\begin{aligned}\alpha_j(t) &= \sum_i \alpha_i(t-1) \Gamma_{ij}(t) & \hat{N}_{ij}(t) &= (1 - \varepsilon_t) \hat{N}_{ij}(t-1) + \alpha_i(t-1) \Gamma_{ij}(t) \\ \hat{n}_j(t) &= \sum_i \hat{N}_{ij}(t) & \hat{S}_{1j}(t) &= (1 - \varepsilon_t) \hat{S}_{1j}(t-1) + \alpha_j(t) x_t \\ & & \hat{S}_{2j}(t) &= (1 - \varepsilon_t) \hat{S}_{2j}(t-1) + \alpha_j(t) x_t x_t^\top\end{aligned} \tag{5}$$

where $\alpha_j(t) = p(z_t = j | x_{1:t})$ is the filtered posterior, $\Gamma_{ij}(t)$ is the update matrix from the forward algorithm [44], and ε_t is a forgetting term that discounts previous data. Note that even for $\varepsilon = 0$, $\mathcal{L}$ is only an estimate of the evidence lower bound because the sufficient statistics are calculated using $\alpha(t)$ and not the posterior over all observed data.

In setting Normal-Inverse-Wishart priors over the Gaussian mixture components, we take an empirical Bayes approach by setting prior means μ_{0j} to the current estimate of the data center of mass and prior covariance parameters Ψ_j to $N^{-\frac{2}{k}}$ times the current estimate of the data covariance (Appendix B). For initializing the model we use a small data buffer $M \sim \mathcal{O}(10)$. We chose effective observation numbers $(\lambda, \nu) = 10^{-3}$ and trained this model to maximize $\mathcal{L}(A, \mu, \Sigma)$ using Adam [47], enforcing parameter constraints by replacing them with unconstrained variables a_{ij} and lower triangular L_j with positive diagonal: $A_{ij} = \exp(a_{ij}) / \sum_j \exp(a_{ij})$, $\Sigma_j^{-1} = L_j L_j^\top$.

Finally, in order to prevent the model from becoming stuck in local minima and to encourage more effective tilings, we implemented two additional heuristics as part of Bubblewrap: First, whenever a new observation was highly unlikely to be in any existing mixture component ($\log p(x_t | z_t) < \theta_n$ for all z_t), we teleported a node at this data point by assigning $\alpha_J(t) = 1$ for an unused index J. For initial learning this results in a "breadcrumbing" approach where nodes are placed at the locations of each new observed datum. Second, when the number of active nodes was equal to our total node budget N, we chose to reclaim the node with the lowest value of $\hat{n}(t)$ and zeroed out its existing sufficient statistics before teleporting it to a new location. In practice, these heuristics substantially improved performance, especially early in training (Appendix D). The full algorithm is summarized in Algorithm 2.

Algorithm 2 Bubblewrap

Given: Hyperparameters $\lambda_j, \nu_j, \beta_t$, forgetting rate ε_t, teleport threshold θ, step size δ, initial data buffer M
Initialize with $\{x_1 \dots x_M\}$**:** $\mu_j \leftarrow \bar{\mu}$, $\Sigma_j \leftarrow \bar{\Sigma}$, $a_{ij} \leftarrow \frac{1}{N}$, $\alpha_j \leftarrow \lambda_j \frac{1}{N}$.

for $t = 1 \dots$ **do**
  Observe new data point x_t.
  if $\log p(x_t|z_t) < \theta \; \forall z_t$ **then** ▷ Teleport
    $\mu_J = x_t, \alpha_J(t) = 1$ for $J = \arg\min_j \hat{n}_j(t)$
  end if
  Calculate $\Gamma_{ij}(t)$ via forward filtering [44].
  Update sufficient statistics via (5). ▷ E step
  $\bar{\mu} \leftarrow \frac{\sum_j \hat{S}_{1j}}{\sum_j \hat{n}_j}$, $\overline{\Sigma} \leftarrow \frac{\sum_j \hat{S}_{2j}}{\sum_j \hat{n}_j} - \bar{\mu}\bar{\mu}^T$ ▷ Global mean and covariance update
  $\epsilon_j \sim \mathcal{N}(0, \eta^2)$, $\mu_{0j} \leftarrow a\mu_{0j} + (1-a)\bar{\mu} + \epsilon_j$, $\Psi_j \leftarrow \frac{\overline{\Sigma}}{N^{\frac{2}{k}}}$ ▷ Update priors (Appendix B)
  Perform gradient-based update of $\mathcal{L}(A, \mu, \Sigma)$ (4) ▷ M step
end for

4 Experiments

We demonstrated the performance of Bubblewrap on both simulated non-linear dynamical systems and experimental neural data. We compared these results to two existing online learning models for neural data, both of which are based on dynamical systems [30, 32]. To simulate low-dimensional systems, we generated noisy trajectories from a two-dimensional Van der Pol oscillator and a three-dimensional Lorenz attractor. For experimental data, we used four publicly available datasets from a range of applications: 1) trial-based spiking data recorded from primary motor cortex in monkeys performing a reach task [48, 49] preprocessed by performing online jPCA [49]; 2) continuous video data and 3) trial-based wide-field calcium imaging from a rodent decision-making task [50, 51]; 4) high-throughput Neuropixels data [52, 53].

For each data set, we gave each model the same data as reduced by random projections and proSVD. For comparisons across models, we quantified overall model performance by taking the mean log predictive probability over the last half of each data set (Table 1). For Bubblewrap, prediction T steps into the future gives

$$\begin{aligned}\log p(x_{t+T}|x_{1:t}) &= \log \sum_{i,j} p(x_{t+T}|z_{t+T}=j)p(z_{t+T}=j|z_t=i)p(z_t=i|x_{1:t}) \\ &= \log \sum_{i,j} \mathcal{N}(x_{t+1}; \mu_j, \Sigma_j)(A^T)_{ij}\alpha_i(t),\end{aligned} \tag{6}$$

where A^T is the T-th power of the transition matrix. Conveniently, these forward predictions can be efficiently computed due to the closed form (6), while similar predictions in comparison models [30, 32] must be approximated by sampling (Appendix C). In addition, for Bubblewrap, which is focused on coarser transitions between tiles, we also report the entropy of predicted transitions:

$$H(t,T) = -\sum_j p(z_{t+T}=j|x_{1:t}) \log p(z_{t+T}=j|x_{1:t}) = -\sum_{ij} (A^T)_{ij}\alpha_i(t) \log \sum_k (A^T)_{kj}\alpha_k(t). \tag{7}$$

Additional detailed experimental results and benchmarking of our GPU implementation in JAX [54] are in Appendix D. We compared performance of our algorithm against both [30] (using our own implementation in JAX) and Variational Joint Filtering [32] (using the authors' implementation). Our implementation of Bubblewrap, as well as code to reproduce our experiments, is open-source and available online at `http://github.com/pearsonlab/Bubblewrap`.

When tested on low-dimensional dynamical systems, Bubblewrap successfully learned tilings of both neural manifolds, outperforming VJF [32] on both datasets (Figure 2a,b) while it was comparable to the algorithm of [30] on one of the 2D (but neither of the 3D) cases (Figure 2). This is surprising, since both comparison methods assume an underlying dynamical system and attempt to predict

Table 1: Model comparison results as mean $\pm$ standard deviation of the log predictive probability over the last half of the dataset. Asterisks (*) indicate models that degenerated to a random walk.

	Log predictive probability		
Dataset	Bubblewrap	VJF [32]	ZP (2016) [30]
2D Van der Pol, 0.05	0.965 ± 1.123	-0.338 ± 0.427	0.121 ± 0.857
2D Van der Pol, 0.20	-1.088 ± 1.184	-1.140 ± 0.879	-0.506 ± 0.964
3D Lorenz, 0.05	-7.338 ± 1.289	-16.98 ± 1.923	$-12.39 \pm 1.723*$
3D Lorenz, 0.20	-7.474 ± 1.279	-17.30 ± 2.112	$-12.42 \pm 1.708*$
Monkey reach	3.046 ± 4.959	-5.159 ± 0.987	3.818 ± 9.118
Wide-field calcium	5.974 ± 2.979	3.768 ± 6.204	1.613 ± 4.083
Mouse video	-10.93 ± 2.386	-15.86 ± 1.084	$-10.65 \pm 4.145*$
Neuropixels	-12.84 ± 6.017	-12.06 ± 5.244	-12.28 ± 4.567

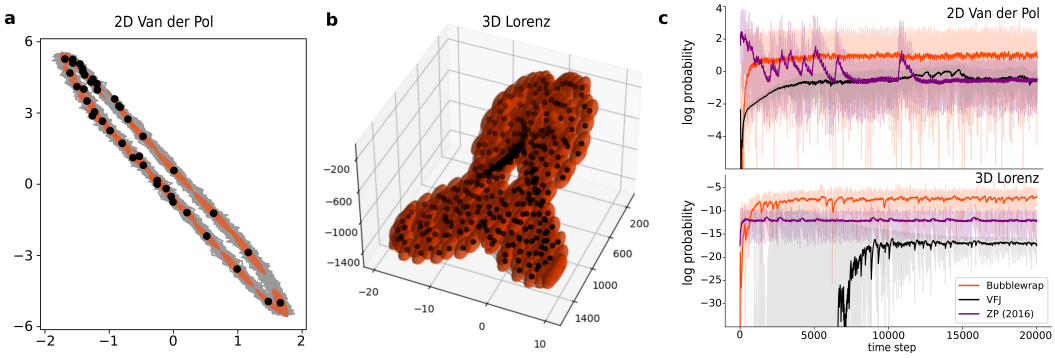

Figure 2: **Modeling of low-dimensional dynamical systems.** **a)** Bubblewrap end tiling of a 2D Van der Pol oscillator (data in gray; 5% noise case corresponding to line 1 of Table 1). Tile center locations are in black with covariance 'bubbles' for 3 sigma in orange. **b)** Bubblewrap end tiling of a 3D Lorenz attractor (5% noise), where tiles are plotted similarly to (a). **c)** Log predictive probability across all timepoints for each comparative model for the 2D Van der Pol, 0.05 case (top) and for the 3D Lorenz, 0.05 case (bottom).

differences between individual data points, while Bubblewrap only attempts to localize data to within a coarse area of the manifold.

We next tested each algorithm on more complex data collected from neuroscience experiments. These data exhibited a variety of structure, from organized rotations (Figure 3a) to rapid transitions between noise clusters (Figure 3b) to slow dynamics (Figure 3c). In each case, Bubblewrap learned a tiling of the data that allowed it to equal or outperform state predictions from the comparison algorithms (Figure 3d–f, blue). In some cases, as with the mouse dataset, the algorithm of [30] produced predictions for x_t by degenerating to a random walk model (Table 1 marked with *; Appendix D). Regardless, Bubblewrap's tiling generated transition predictions with entropies far below those of a random walk (Figure 3d–f, green), indicating it successfully identified coarse structure, even in challenging datasets. Thus, even though these data are noise-dominated and lack much of the typical structure identified by neural population models, coarse-graining identifies some reliable patterns.

We additionally considered the capability of our algorithm to scale to high-dimensional or high-sampling rate data. As a case study, we considered real-time processing (including random projections, proSVD, and Bubblewrap learning) of Neuropixels data comprising 2688 units with 74,330 timepoints from 30 ms bins. As Figure 4 shows, Bubblewrap once again learns a tiling of the data manifold (a), capturing structure in the probability flow within the space (b) with predictive performance comparable to finer-grained methods (Table 1). More importantly, all these steps can be performed well within the 30ms per sample time of the data (c). In fact, when testing on representative examples of $d = 10^4$ dimensions, 1 kHz sampling rates, or $N = 20,000$ tiles, our algorithm was able to maintain amortized per-sample processing times below those of data acquisition. In practice, we

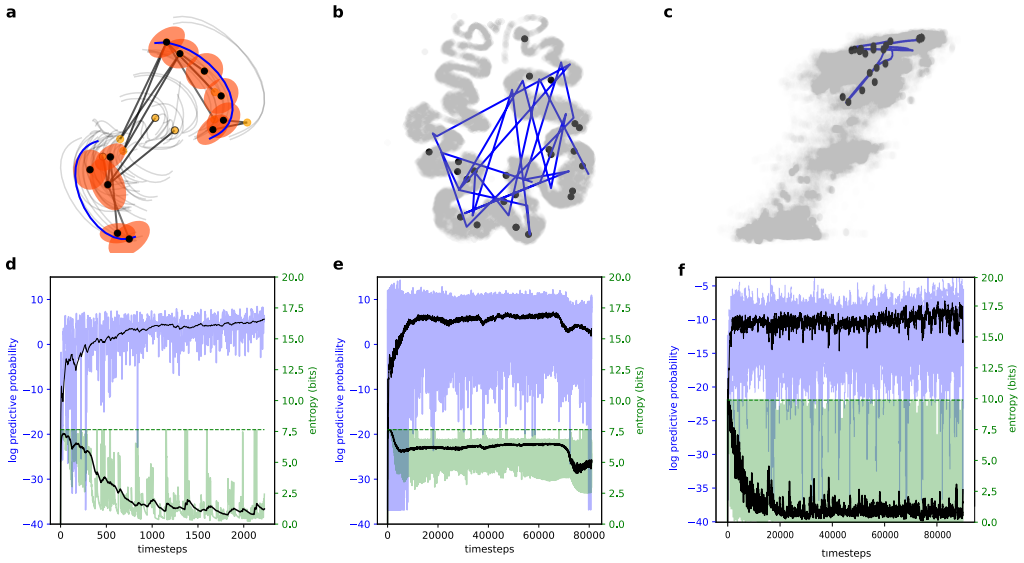

Figure 3: **Bubblewrap results on experimental datasets. a)** Bubblewrap results for example trials (blue) from the monkey reach dataset [48, 49], projected onto the first jPCA plane. All trials are shown in gray. The tile center locations which were closest to the trajectories are plotted along with their covariance "bubbles." Additionally, large transition probabilities from each tile center are plotted as black lines connecting the nodes. Bubblewrap learns both within-trial and across-trial transitions, as shown by the probability weights. **b)** Bubblewrap results on widefield calcium imaging from [50, 51], visualized with UMAP. A single trajectory comprising ≈1.5s of data is shown in blue. Covariance "bubbles" and transition probabilities omitted for clarity. **c)** Bubblewrap results when applied to videos of mouse behavior [50, 51], visualized by projection onto the first SVD plane. Blue line: 3.3s of data. **d, e, f)** Log predictive probability (blue) and entropy (green) over time for the respective datasets in (a,b,c). Black lines are exponential weighted moving averages of the data. Dashed green line indicates maximum entropy ($\log_2(N)$).

found that even in higher-dimensional datasets (as in the Neuropixels case), only 1-2 thousand tiles were used by the model, making it easy to run at kHz data rates. What's more, while learning involved round trip GPU latencies to perform gradient updates, online predictions using slightly stale estimates of Bubblewrap parameters could be performed far faster, in tens of microseconds.

Just as importantly, when used for closed loop experiments, algorithms must be able to produce predictions far enough into the future for interventions to be feasible. Thus we examined the performance of our algorithm and comparison models for predicting T steps ahead into the future. Bubblewrap allows us to efficiently calculate predictions even many time steps into the future using (6), whereas the comparison models require much costlier sampling approaches. Figure 5 shows the mean log predictive probabilities for all models many steps into the future for each experimental dataset (top row), and the entropy of the predicted transitions using Bubblewrap (bottow row). Our algorithm consistently maintains performance even when predicting 10 steps ahead, providing crucial lead time to enable interventions at specific points along the learned trajectory. In comparison, predictive performance of [30], which initially matches or exceeds Bubblewrap for two datasets, rapidly declines, while Variational Joint Filtering [32], with lower log likelihood, also exhibits a slow decay in accuracy.

5 Discussion

While increasing attention has been paid in neuroscience to population hypotheses of neural function [10], and while many methods for modeling these data offline exist, surprisingly few methods function online, though presumably online methods will be needed to test some population dynamics hypotheses [17]. While the neural engineering literature has long used online methods based on

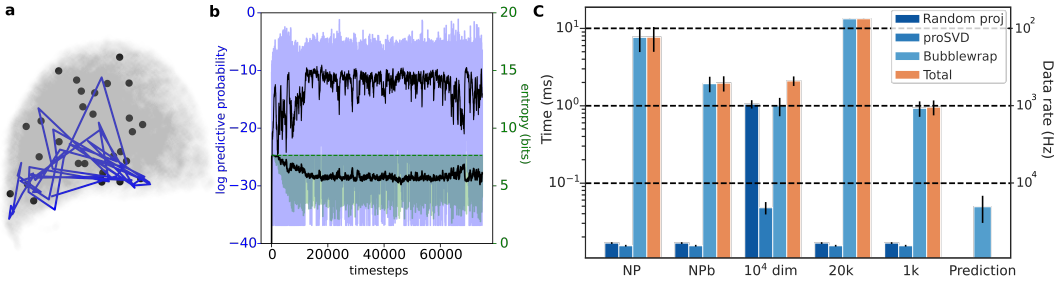

Figure 4: **High-throughput data & benchmarking. a)** Bubblewrap results for example trajectories (blue) in the Neuropixels dataset [52, 53] (data in gray) visualized with UMAP. **b)** Log predictive probability (blue) and entropy (green) over time. Black lines are exponential weighted moving averages of the data. Dashed green line indicates maximum entropy. **c)** Average cycle time (log scale) during learning or prediction (last bar) for each timepoint. Neuropixels (NP) is run as in (a,b) with no optimization and all heuristics, and Bubblewrap is easily able to learn at rates much faster than acquisition (30 ms). By turning off the global mean and covariance and priors updates and only taking a gradient step for $\mathcal{L}$ every 30 timepoints, we are able to run at close to 1 kHz (NPb). All other bars show example timings from Van der Pol synthetic datasets optimized for speed: 10^4 dim, where we randomly project down to 200 dimensions and used proSVD to project to 10 dimensions for subsequent Bubblewrap modeling learning; N = 20k, 10k, and 1k nodes, showing how our algorithm scales with the number of tiles; and Prediction, showing the time cost to predict one step ahead for the N = 1k case.

Kalman filtering, (e.g., [16]), and these methods are known to work well in many practical cases, they also imply strong assumptions about the evolution of activity within these systems. Thus, many studies that employ less constrained behavior or study neural activity with less robust dynamics may benefit from more flexible models that can be trained while the experiment is running.

Here, to address this need, we have introduced both a new dimension reduction method that rapidly produces stable estimates of features and a method for rapidly mapping and charting transitions on neural manifolds. Rather than focus on moment-by-moment prediction, we focus on estimating a coarse tiling and probability flow among these tiles. Thus, Bubblewrap may be less accurate than methods based on dynamical systems when state trajectories are accurately described by smooth vector fields with Gaussian noise. Conversely, when noise dominates, is multimodal, or only large-scale probability flow is discernible over longer timescales, Bubblewrap is better poised to capture these features. We saw this in our experiments, where the model of [30] exhibited better overall performance in the mouse video dataset (Figure 3e) when it did not learn to predict and degenerated to a random walk. Indeed, the most relevant comparison to the two approaches is the duality between stochastic differential equations and Fokker-Planck equations, where ours is a (softly) discretized analog of the latter. Nonetheless, in many of the cases we consider, Bubblewrap produces superior results even for state prediction. Nonetheless, like many similar models, ours includes multiple hyperparameters that require setting. While we did not experience catastrophic failure or sensitive dependence on parameters in our testing, and while our methods adapt to the scale and drift of the data, some tuning was required in practice.

As detailed above, while many methods target population dynamics, and a few target closed-loop settings [31, 16, 55], very few models are capable of being trained online. Thus, the most closely related approaches are those in [30, 32], to which we provide extensive comparisons. However, these comparisons are somewhat strained by the fact that we provided all models with the same proSVD-reduced low-dimensional data, while [32] is capable of modeling high-dimensional data in its own right and [30] was targeted at inferring neural computations from dynamical systems. We thus view this work as complementary to the dynamical systems approach, one that may be preferred when small distinctions among population dynamics are less important than characterizing highly noisy, non-repeating neural behavior.

Finally, we showed that online training of Bubblewrap can be performed fast enough for even kiloHertz data acquisition rates if small latencies are tolerable and gradient steps can be performed for small numbers of samples at a time. Yet, for real-time applications, it is not training time but the

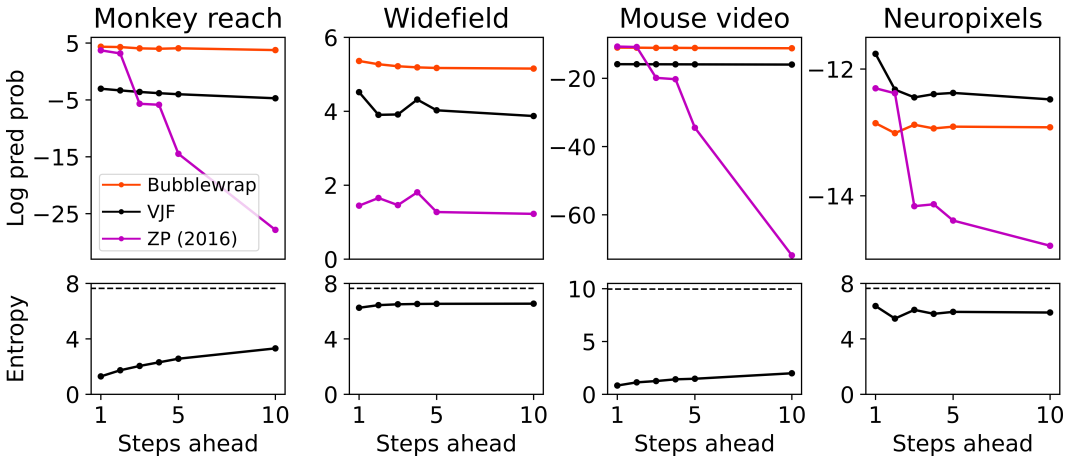

Figure 5: **Multi-step ahead predictive performance. (top)** Mean log predictive probability as a function of the number of steps ahead used for prediction for each of the four experimental datasets studied. Colors indicate model. **(bottom)** Bubblewrap entropy as a function of the number of steps ahead used for prediction. Higher entropy indicates more uncertainty about future states. Dashed lines denote maximum entropy for each dataset (log of the number of tiles).

time required to make predictions that is relevant, and we demonstrate prediction times of tens of microseconds. Moreover, Bubblewrap is capable of producing effective predictions multiple time steps into the future, providing ample lead time for closed-loop interventions. Thus, coarse-graining methods like ours open the door to online manipulation and steering of neural systems.

Acknowledgments and Disclosure of Funding

Research reported in this publication was supported by a NIH BRAIN Initiative Planning Grant (R34NS116738; JP), and a Swartz Foundation Postdoctoral Fellowship for Theory in Neuroscience (AD). AD also holds a Career Award at the Scientific Interface from the Burroughs Wellcome Fund.

References

[1] Misha B Ahrens, Michael B Orger, Drew N Robson, Jennifer M Li, and Philipp J Keller. Whole-brain functional imaging at cellular resolution using light-sheet microscopy. *Nature methods*, 10(5):413–420, 2013.

[2] Valentina Emiliani, Adam E Cohen, Karl Deisseroth, and Michael Häusser. All-optical interrogation of neural circuits. *Journal of Neuroscience*, 35(41):13917–13926, 2015.

[3] Ian H Stevenson and Konrad P Kording. How advances in neural recording affect data analysis. *Nature neuroscience*, 14(2):139–142, 2011.

[4] Nicholas A Steinmetz, Christof Koch, Kenneth D Harris, and Matteo Carandini. Challenges and opportunities for large-scale electrophysiology with neuropixels probes. *Current opinion in neurobiology*, 50:92–100, 2018.

[5] Nicholas A Steinmetz, Cagatay Aydin, Anna Lebedeva, Michael Okun, Marius Pachitariu, Marius Bauza, Maxime Beau, Jai Bhagat, Claudia Böhm, Martijn Broux, et al. Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *bioRxiv*, 2020.

[6] Valerio Mante, David Sussillo, Krishna V Shenoy, and William T Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *nature*, 503(7474):78–84, 2013.

[7] Kanaka Rajan, Christopher D Harvey, and David W Tank. Recurrent network models of sequence generation and memory. *Neuron*, 90(1):128–142, 2016.

[8] H Francis Song, Guangyu R Yang, and Xiao-Jing Wang. Training excitatory-inhibitory recurrent neural networks for cognitive tasks: a simple and flexible framework. *PLoS computational biology*, 12(2):e1004792, 2016.

[9] John P Cunningham and M Yu Byron. Dimensionality reduction for large-scale neural recordings. *Nature neuroscience*, 17(11):1500–1509, 2014.

[10] R Becket Ebitz and Benjamin Y Hayden. The population doctrine revolution in cognitive neurophysiology. *arXiv preprint arXiv:2104.00145*, 2021.

[11] Eric T Carlson, Russell J Rasquinha, Kechen Zhang, and Charles E Connor. A sparse object coding scheme in area v4. *Current Biology*, 21(4):288–293, 2011.

[12] Christopher DiMattina and Kechen Zhang. Adaptive stimulus optimization for sensory systems neuroscience. *Frontiers in neural circuits*, 7:101, 2013.

[13] Benjamin Cowley, Ryan Williamson, Katerina Clemens, Matthew Smith, and Byron M Yu. Adaptive stimulus selection for optimizing neural population responses. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[14] Reza Abbasi-Asl, Yuansi Chen, Adam Bloniarz, Michael Oliver, Ben DB Willmore, Jack L Gallant, and Bin Yu. The deeptune framework for modeling and characterizing neurons in visual cortex area v4. *bioRxiv*, page 465534, 2018.

[15] Zihui Zhang, Lloyd E Russell, Adam M Packer, Oliver M Gauld, and Michael Häusser. Closed-loop all-optical interrogation of neural circuits in vivo. *Nature methods*, 15(12):1037–1040, 2018.

[16] Michael F Bolus, Adam A Willats, Christopher J Rozell, and Garrett B Stanley. State-space optimal feedback control of optogenetically driven neural activity. *Journal of Neural Engineering*, 2020.

[17] Diogo Peixoto, Jessica R Verhein, Roozbeh Kiani, Jonathan C Kao, Paul Nuyujukian, Chandramouli Chandrasekaran, Julian Brown, Sania Fong, Stephen I Ryu, Krishna V Shenoy, et al. Decoding and perturbing decision states in real time. *Nature*, pages 1–6, 2021.

[18] Anne Draelos and John Pearson. Online neural connectivity estimation with noisy group testing. *Advances in Neural Information Processing Systems*, 33, 2020.

[19] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. *Computer Vision?ECCV 2002*, pages 707–720, 2002.

[20] Christopher G Baker. A block incremental algorithm for computing dominant singular subspaces. Master's thesis, Florida State University, 2004.

[21] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006.

[22] Christopher G Baker, Kyle A Gallivan, and Paul Van Dooren. Low-rank incremental methods for computing dominant singular subspaces. *Linear Algebra and its Applications*, 436(8):2866–2888, 2012.

[23] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(1), 2010.

[24] Evan Archer, Il Memming Park, Lars Buesing, John Cunningham, and Liam Paninski. Black box variational inference for state space models. *arXiv preprint arXiv:1511.07367*, 2015.

[25] Yuanjun Gao, Evan Archer, Liam Paninski, and John P Cunningham. Linear dynamical neural population models through nonlinear embeddings. *arXiv preprint arXiv:1605.08454*, 2016.

[26] Chethan Pandarinath, Daniel J O'Shea, Jasmine Collins, Rafal Jozefowicz, Sergey D Stavisky, Jonathan C Kao, Eric M Trautmann, Matthew T Kaufman, Stephen I Ryu, Leigh R Hochberg, et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, 15(10):805–815, 2018.

[27] Scott Linderman, Matthew Johnson, Andrew Miller, Ryan Adams, David Blei, and Liam Paninski. Bayesian learning and inference in recurrent switching linear dynamical systems. In *Artificial Intelligence and Statistics*, pages 914–922, 2017.

[28] Scott W. Linderman*, Matthew J. Johnson*, Andrew C. Miller, Ryan P. Adams, David M. Blei, and Liam Paninski. Bayesian learning and inference in recurrent switching linear dynamical systems. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[29] Josue Nassar, Scott W Linderman, Monica Bugallo, and Il Memming Park. Tree-structured recurrent switching linear dynamical systems for multi-scale modeling. *arXiv preprint arXiv:1811.12386*, 2018.

[30] Yuan Zhao and Il Memming Park. Interpretable nonlinear dynamic modeling of neural trajectories. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[31] Yuxiao Yang, Allison T Connolly, and Maryam M Shanechi. A control-theoretic system identification framework and a real-time closed-loop clinical simulation testbed for electrical brain stimulation. *Journal of neural engineering*, 15(6):066007, 2018.

[32] Yuan Zhao and Il Memming Park. Variational online learning of neural dynamics. *Frontiers in computational neuroscience*, 14, 2020.

[33] Peiran Gao, Eric Trautmann, Byron Yu, Gopal Santhanam, Stephen Ryu, Krishna Shenoy, and Surya Ganguli. A theory of multineuronal dimensionality, dynamics and measurement. *BioRxiv*, page 214262, 2017.

[34] Eric M Trautmann, Sergey D Stavisky, Subhaneil Lahiri, Katherine C Ames, Matthew T Kaufman, Daniel J O'Shea, Saurabh Vyas, Xulu Sun, Stephen I Ryu, Surya Ganguli, et al. Accurate estimation of neural population dynamics without spike sorting. *Neuron*, 103(2):292–308, 2019.

[35] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[36] Ping Li, Trevor J Hastie, and Kenneth W Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296, 2006.

[37] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[38] Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[39] Alan D Degenhart, William E Bishop, Emily R Oby, Elizabeth C Tyler-Kabara, Steven M Chase, Aaron P Batista, and M Yu Byron. Stabilization of a brain–computer interface via the alignment of low-dimensional spaces of neural activity. *Nature biomedical engineering*, 4(7):672–685, 2020.

[40] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International journal of computer vision*, 77(1):125–141, 2008.

[41] Gordon J Berman, Daniel M Choi, William Bialek, and Joshua W Shaevitz. Mapping the stereotyped behaviour of freely moving fruit flies. *Journal of The Royal Society Interface*, 11(99):20140672, 2014.

[42] Gordon J Berman, William Bialek, and Joshua W Shaevitz. Predictability and hierarchy in drosophila behavior. *Proceedings of the National Academy of Sciences*, 113(42):11943–11948, 2016.

[43] Talmo D Pereira, Joshua W Shaevitz, and Mala Murthy. Quantifying behavior to understand the brain. *Nature Neuroscience*, pages 1–13, 2020.

[44] Gianluigi Mongillo and Sophie Deneve. Online learning with hidden markov models. *Neural computation*, 20(7):1706–1716, 2008.

[45] Olivier Cappé and Eric Moulines. On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613, 2009.

[46] Sylvain Le Corff, Gersende Fort, et al. Online expectation maximization based algorithms for inference in hidden markov models. *Electronic Journal of Statistics*, 7:763–792, 2013.

[47] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] Mark Churchland. Churchland lab code. `"https://churchland.zuckermaninstitute.columbia.edu/content/code"`.

[49] Mark M Churchland, John P Cunningham, Matthew T Kaufman, Justin D Foster, Paul Nuyujukian, Stephen I Ryu, and Krishna V Shenoy. Neural population dynamics during reaching. *Nature*, 487(7405):51–56, 2012.

[50] Simon Musall, Matthew T Kaufman, Ashley L Juavinett, Steven Gluf, and Anne K Churchland. Single-trial neural dynamics are dominated by richly varied movements. *Nature neuroscience*, 22(10):1677–1686, 2019.

[51] Simon Musall, Matthew T. Kaufman, Ashley L. Juavinett, Steven Gluf, and Anne K. Churchland. Single-trial neural dynamics are dominated by richly varied movements: dataset. Technical report, October 2019.

[52] Nick Steinmetz, Marius Pachitariu, Carsen Stringer, Matteo Carandini, and Kenneth Harris. Eight-probe neuropixels recordings during spontaneous behaviors, Mar 2019.

[53] Carsen Stringer, Marius Pachitariu, Nicholas Steinmetz, Charu Bai Reddy, Matteo Carandini, and Kenneth D Harris. Spontaneous behaviors drive multidimensional, brainwide activity. *Science*, 364(6437), 2019.

[54] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[55] Omid G Sani, Hamidreza Abbaspourazad, Yan T Wong, Bijan Pesaran, and Maryam M Shanechi. Modeling behaviorally relevant neural dynamics enabled by preferential subspace identification. Technical report, Nature Publishing Group, 2020.

A Details of proSVD algorithm

We follow the notation of [20, 22] with the exception that we use X for the data matrix rather than A, with new data X_+ of dimension $n \times b$ rather than $m \times l$. Here, we focus on the updates of the left singular subspace spanned. Details for the right singular subspace are similar and covered in the original works. Matrix sizes are listed for convenience in Table 2.

Table 2: Matrix dimensions for incremental SVD.

matrix	rows	columns
X	n	T
X_0	n	k
X_+	n	b
C	k	b
Q_t	n	k
R_t	k	k
$\hat{Q}$	n	$k+b$
$\hat{R}$	$k+b$	$k+b$
U	$k+b$	$k+b$
U_1	$k+b$	k
U_2	$k+b$	b
T_u	k	k
S_u	b	b
G_u	$k+b$	$k+b$
G_{u_1}	$k+b$	k

The goal of the algorithm is to maintain an approximation of the data up to the present moment as

$$X \approx QRW^\top \tag{8}$$

with Q and W orthogonal but R not necessarily diagonal. The requirement that (8) be equivalent to the top-k SVD requires that the top-k SVD of X can easily be computed via the SVD of the small matrix $R = U\Sigma V^\top$.

The algorithm begins with an initial data matrix X_0, which is factorized via the QR decomposition

$$X_0 = [Q_0 \quad Q_\perp] \begin{bmatrix} R_0 \\ 0 \end{bmatrix} \mathbb{1}, \tag{9}$$

which has the form (8) if we identify $W_0 = \mathbb{1}$. Thus Q_0 forms the initial candidate for a basis for the top-k subspace. On subsequent iterations, the procedure is as follows:

1. Observe a new $n \times b$ data matrix X_+.
2. Perform a Gram-Schmidt Orthogonalization of this new data, obtaining C, a projection into the previous basis Q_{t-1} and a remainder, $X_\perp$.
3. Perform a QR decomposition $X_\perp = Q_\perp R_\perp$. This gives rise to a new factorization

$$X_t = [X_{t-1} \quad X_+] = \hat{Q}\hat{R}\hat{W}^T \tag{10}$$

with

$$\hat{Q} \equiv [Q_{t-1} \quad Q_\perp] \tag{11}$$

$$\hat{R} \equiv \begin{bmatrix} R_{t-1} & C \\ 0 & R_\perp \end{bmatrix}. \tag{12}$$

4. From this new factorization, the goal is to block diagonalize $\hat{R}$ and truncate to the upper left block, which is the new top-k singular subspace. This is done by first performing an SVD, $\hat{R} = U\Sigma V^\top$.
5. To allow for old and stale data to decay in influence over time and to prevent unbounded accumulation of variance in Σ, we follow [40] in multiplying Σ by a discount parameter α at each step.

6. As in [20, 22], the goal is to find a matrix G_u (and a counterpart on the right, G_v) satisfying

$$G_u^\top U = \begin{bmatrix} T_u & 0 \\ 0 & S_u \end{bmatrix} \tag{13}$$

or the equivalent condition

$$G_u^\top U_1 = \begin{bmatrix} T_u \\ 0 \end{bmatrix} \tag{14}$$

with U_1 the first k columns of U, which would yield

$$G_u^\top \hat{R} G_v = \begin{bmatrix} T_u \Sigma_1 T_v^\top & 0 \\ 0 & S_u \Sigma_2 S_v^\top \end{bmatrix}. \tag{15}$$

7. This done, the matrices can once again be truncated back to their top-k versions:

$$R_t = T_u \Sigma_1 T_v^\top \tag{16}$$

$$Q_t = \hat{Q} G_{u_1} = \hat{Q} U_1 T_u^\top. \tag{17}$$

What is most important to note in this is that the solution to (14) is not unique. Many choices of G_u (equivalently T_u) are possible. In [20, 22], a particular solution to this equation is chosen for computational efficiency. By contrast, proSVD seeks to solve

$$\min_{T_u} \|Q_t - Q_{t-1}\|_F = \min_{T_u} \|\hat{Q} U_1 T_u^\top - Q_{t-1}\|_F, \tag{18}$$

with $\|\cdot\|_F$ denoting the Frobenius norm. As previously stated, this is an Orthogonal Procrustes problem with solution $T = \tilde{U}\tilde{V}^\top$, where $\tilde{U}$ and $\tilde{V}$ are defined by the SVD [38]:

$$\tilde{U}\tilde{\Sigma}\tilde{V}^\top = Q_{t-1}^\top \hat{Q} U_1 = [\mathbb{1}_{k\times k} \quad \mathbf{0}_{k\times b}]\, U_1. \tag{19}$$

That is, T can be found by performing the SVD of the upper left $k \times k$ block of U, and the only additional cost of our formulation relative to [20, 22] is the $\mathcal{O}(k^3)$ cost of this SVD, which is negligible in practice for k small (Figure 1b).

B Details of Bubblewrap initialization and heuristics

Our implementation of Bubblewrap neither normalizes nor assumes a scale for incoming data. Thus, in order to choose priors that adjust to the scale of the data, we adopt an empirical Bayes approach. That is, we adjust some parameters of the Normal-Inverse-Wishart priors for each bubble, μ_{0j} and Ψ_j, based on data. More specifically, we first calculate empirical estimates $\bar{\mu}$ and $\overline{\Sigma}$ of the data mean and covariance online, as in line 11 of Algorithm 2. We then update the priors as follows:

Random walk dynamics on μ_{0j}: If μ_{0j} is fixed and identical across all bubbles, then points passing through this concentration of Gaussians will be assigned equally to all, which creates problems for tiling. Here, the idea is to allow the μ_{0j} to randomly walk, thereby breaking degeneracy among nodes with no data points. However, in order to bound the random walk, we include a decay back toward the current center of mass. More specifically, each of the μ_{0j} is governed by a biased random walk:

$$\mu_{0j}(t) = (1-\lambda)\mu_{0j}(t-1) + \lambda\bar{\mu}(t-1) + \epsilon_{jt} \tag{20}$$

with $\lambda \in [0,1]$ and $\epsilon_{jt} \sim \mathcal{N}(0, \eta^2)$, $\mathbb{E}[\epsilon_{jt}\epsilon_{j't'}] = \eta^2 \delta_{jj'}\delta_{tt'}$. Here, η is a step size for the random walk that may differ by dimension. Note that, when $\bar{\mu}$ is fixed, the distribution of μ_{0j} converges to $\mathcal{N}(\bar{\mu}, \mathrm{diag}(\eta^2)/\lambda)$.

In practice, we chose $\lambda = 0.02$ and $\eta = \sqrt{\lambda\ \mathrm{diag}(\overline{\Sigma})}$, so that the diffusion range of bubbles with no data is set by the scale of the data distribution.

Bubble packing for Ψ_j: Here, the basic idea is to choose Ψ_j, which controls the covariance around which the prior concentrates, so that the total volume of bubbles scales as the volume of the data. Specifically, let $\Psi_j = \sigma^2 \mathbb{1}$, so that the covariance prior is spherical with radius σ. We also assume

that, in k dimensions, data take up $\mathcal{O}(L)$ space along each dimension, so that $\mathrm{vol}(data) \sim L^k$. Then the volume of each bubble is $\sim \sigma^k$ and the total volume of all bubbles is

$$N\sigma^k \sim L^k \quad \Rightarrow \quad \sigma \sim \frac{L}{N^{\frac{1}{k}}} \tag{21}$$

which yields

$$\Psi_j = \frac{L^2}{N^{\frac{2}{k}}}\mathbb{1}. \tag{22}$$

More generally, we want Ψ_j to reflect the estimated data covariance $\overline{\Sigma}$, so we set

$$\Psi_j = \frac{1}{N^{\frac{2}{k}}}\overline{\Sigma}. \tag{23}$$

C Predictive distribution calculations

Both [30] and [32] use a nonlinear dynamical system model parameterized as

$$\mathbf{x}_{t+1} = \mathbf{x}_t + f(\mathbf{x}_t) + \mathbf{B}_t(\mathbf{x}_t)\mathbf{u}_t + \epsilon_{t+1} \tag{24}$$

$$\mathbf{y}_t \sim P(g(\mathbf{C}\mathbf{x}_t + \mathbf{b})) \tag{25}$$

$$f(\mathbf{x}) = \mathbf{W}\boldsymbol{\phi}(\mathbf{x}) - e^{-\tau^2}\mathbf{x}_t \tag{26}$$

$$\mathbf{x}_0, \epsilon_t \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbb{1}), \tag{27}$$

with $\boldsymbol{\phi}$ a set of basis functions and g a link function. In [32], $\mathbf{B}$ is assumed constant and $\tau \to \infty$. More importantly, the filtered probability $p(\mathbf{x}_t|\mathbf{y}_{\leq t})$ is approximated by a posterior $q(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \mathrm{diag}(\mathbf{s}_t))$ with $\boldsymbol{\mu}_t$ and $\mathbf{s}_t$ defined by the output of a neural network trained to optimize a variational lower bound on the log evidence.

In [30], the loss function is mean squared error in the prediction (24), which is equivalent to the likelihood model

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_t + f(\mathbf{x}_t) + \mathbf{B}_t(\mathbf{x}_t)\mathbf{u}_t, \sigma^2\mathbb{1}), \tag{28}$$

with $\sigma^2 = \mathrm{var}[\epsilon_t]$ estimated from the data. In [30], there is no separate observation model (25), so $\mathbf{y} = \mathbf{x}$, and we use (28) (with σ^2 estimated from the residuals of (24) via an exponential smooth) to calculate predictive likelihood for our comparisons.

For [32], following the procedure outlined there, we can compute the log predictive probability via sampling

$$\mathbf{x}_t^s \sim q(\mathbf{x}_t) \tag{29}$$

$$\mathbf{x}_{t+1}^s \sim \mathcal{N}(\mathbf{x}_t^s + f(\mathbf{x}_t^s) + \mathbf{B}_t\mathbf{u}_t, \sigma^2) \tag{30}$$

$$\log p(\mathbf{y}_{t+1}|\mathbf{y}_{\leq t}) \approx \log\left[\frac{1}{S}\sum_{s=1}^{S} P(\mathbf{y}_{t+1}; g(\mathbf{C}\mathbf{x}_{t+1}^s + \mathbf{b}))\right] \tag{31}$$

with $S = 100$ samples. However, for direct comparison with our method, which operates in a reduced-dimensional space, we compared the predictive probability by feeding all methods the same data as reduced by proSVD, so that $\dim(\mathbf{y}) = \dim(\mathbf{x}) = \dim(\mathrm{data})$.

When predicting more than one time step ahead, we use sequential sampling for both models. For the model of [30], we iterate (24) to produce S trajectories $T-1$ time steps into the future (ending at x_{t+T-1}^s and calculate

$$p(\mathbf{x}_{t+T}|\mathbf{x}_{\leq t}) = \frac{1}{S}\sum_{s=1}^{S} p(\mathbf{x}_{t+T}|\mathbf{x}_{t+T-1}^s) = \frac{1}{S}\sum_{s=1}^{S}\mathcal{N}(\mathbf{x}_{t+T-1}^s + f(\mathbf{x}_{t+T-1}^s), \sigma^2\mathbb{1}) \tag{32}$$

using $S = 100$ trajectories. For the model of [32], we have

$$
\begin{aligned}
p(\mathbf{y}_{t+T}|\mathbf{y}_{\leq t}) &= \int \prod_{t'=t+1}^{T} d\mathbf{x}_{t'}\; p(\mathbf{y}_{t+T}|\mathbf{x}_{t+1:T}) p(\mathbf{x}_{t+1:T}|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{y}_{\leq t}) \\
&\approx \int \prod_{t'=t+1}^{T} d\mathbf{x}_{t'}\; p(\mathbf{y}_{t+T}|\mathbf{x}_{t+1:T}) p(\mathbf{x}_{t+1:T}|\mathbf{x}_t) q(\mathbf{x}_t) \\
&\approx \frac{1}{S} \sum_{s=1:S} p(\mathbf{y}_{t+T}|\mathbf{x}^s_{t+T}) \\
&= \frac{1}{S} \sum_{s=1}^{S} P(\mathbf{y}_{t+T}; g(\mathbf{C}\mathbf{x}^s_{t+T} + \mathbf{b})),
\end{aligned} \tag{33}
$$

where again we have used sampling to marginalize over the intervening latent states.

D Additional Experiments

All experimental simulations were run on custom-built desktop machines running either Ubuntu 18.04.4 LT or Ubuntu 20.04.2. The computers ran with either 64GB or 128 GB of system memory, and various CPUs including a 4 or 8 core 4.0 GHz Intel i7-6700K processor and 14 core 3.1 GHz Intel i9-7940X processors. GPUs used included an NVIDIA GeForce GTX 1080 Ti (11 GB), 2x NVIDIA RTX 2080 Ti (11 GB), an NVIDIA Titan Xp (12 GB), and an NVIDIA RTX 3090 (24 GB).

D.1 Datasets

The monkey reach dataset can be found at `https://churchland.zuckermaninstitute.columbia.edu/content/code` [48], and was originally presented in [49]. The widefield calcium imaging and mouse video datasets can both be found at `https://dx.doi.org/10.14224/1.38599` [51], and were originally presented in [50]. The neuropixels dataset can be found at `https://doi.org/10.25378/janelia.7739750.v4` [52], and has a (CC BY-NC 4.0) license. It was originally presented in [53].

To motivate the number of dimensions of the low-dimensional subspace onto which we project the data, we show the cumulative variance explained by each eigenvector of the real datasets in Figure S1. These spectra show that keeping on the order of ~ 10 components retains 40% of variance of the monkey reach and neuropixels datasets, and retains $> 99\%$ variance of the widefield and mouse video datasets.

D.2 proSVD stability

Here we apply proSVD and the incremental block update method of [20, 22] (referred to as streaming SVD) to the experimental datasets used in the main text. The first column of Figure S2 shows that the proSVD basis vectors stabilize after a few tens of seconds for these datasets, with only small gradients thereafter. The second column shows that proSVD basis vectors stabilize to their final (offline) positions faster than streaming SVD vectors.

D.3 Log predictive probability & entropy

Here we include results for log predictive probability and entropy over time for the datasets not included in the main text. Figure S3 shows the log predictive probability across all time points for all datasets except the 2D Van der Pol and 3D Lorenz (5 % noise) datasets (cp. Figure 2). Figure S4 plots the entropy over time for the 2D Van der Pol (0.05, 0.20) and the 3D Lorenz (0.05, 0.20) datasets ; all other entropy results were shown in the main text (Figure 3d-f, Figure 4b).

D.4 Final transition matrices

Figure S5 shows the eigenspectra of the final learned transition matrices A for each of the four experimental datasets studied in the main text. The monkey reach and mouse video results yielded

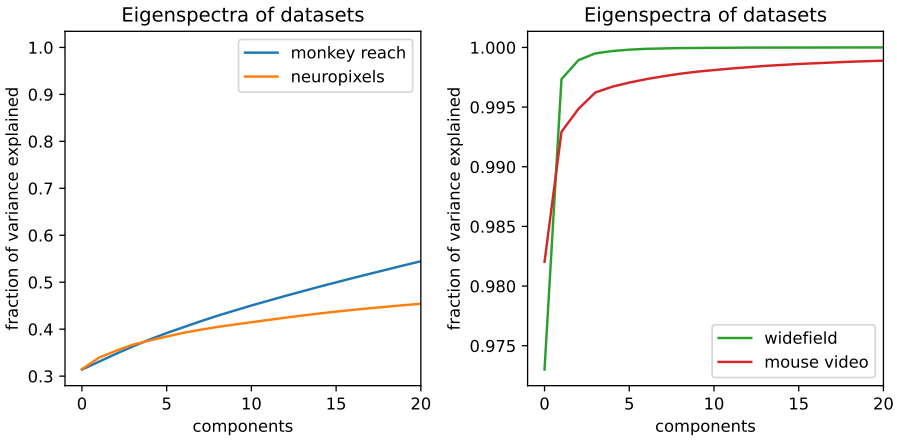

Figure S1: **Eigenspectra of datasets.** Cumulative variance explained by retaining differing numbers of components in linear dimensionality reduction. Labels reference datasets as in the main text.

slowly-decaying eigenspectra, suggesting long-range predictive power, whereas the widefield and neuropixels eigenspectra decayed much faster (and thus yielded less long-range predictive power over time). Figures 3 and 4 in the main text also show this distinction: the monkey reach and mouse video entropy results reached much lower values than the other two datasets, corresponding to lower-uncertainty predictions.

Additionally, we visualize the final node locations and transition entries in A for each experimental dataset in Figure S6. Note that while all timepoints of the data are plotted (grey), only the end result of Bubblewrap is shown (black dots, blue lines).

D.5 Degeneration of dynamical systems to random walks

For the Lorenz and Mouse video datasets, the model of [30] outperformed both Bubblewrap and VJF in log predictive probability. However, as shown in Figure S7, this is because the predicted step size ($f(x)$ in (24)) drops to near 0. That is, the model degenerates to a random walk. Datasets where this occurs are marked with a * in Table 1.

D.6 Model benchmarking

Here we present average runtimes for the comparison models on selected datasets, similar to Figure 4c in the main text. Note that we gave all models the already dimension-reduced data. Figure S8 shows only the time needed to update the model for each new datapoint of the 2D Van der Pol (VdP) oscillator dataset for two different numbers of basis functions (labeled '500' and '50'), which are roughly equivalent to the notion of nodes in Bubblewrap. Next, we include the time to generate predictions via sampling, increasing the total computation time, for the Neuropixels dataset ('NP') and the Van der Pol oscillator (0.05 noise), again with 500 or 50 basis functions ('VdP 500' and 'VdP 50'). Finally, we plot the average time to fit Bubblewrap (including prediction time) for the base VdP ('VdP') case using 1k nodes. All comparison model cases, using our implementations, showed that models ran at rates slower than 1 kiloHerz.

D.7 Bubblewrap model evolution

As shown in the above log predictive probability plots, Bubblewrap takes only a small amount of data to successfully fit the underlying neural manifold. Figure S9 shows our model fit as a function of the number of timepoints seen for the 2D Van der Pol and 3D Lorenz simulated datasets (0.05), with tiling locations settling in after observing a few hundred data points and uncertainty being refined throughout the course of the dataset.

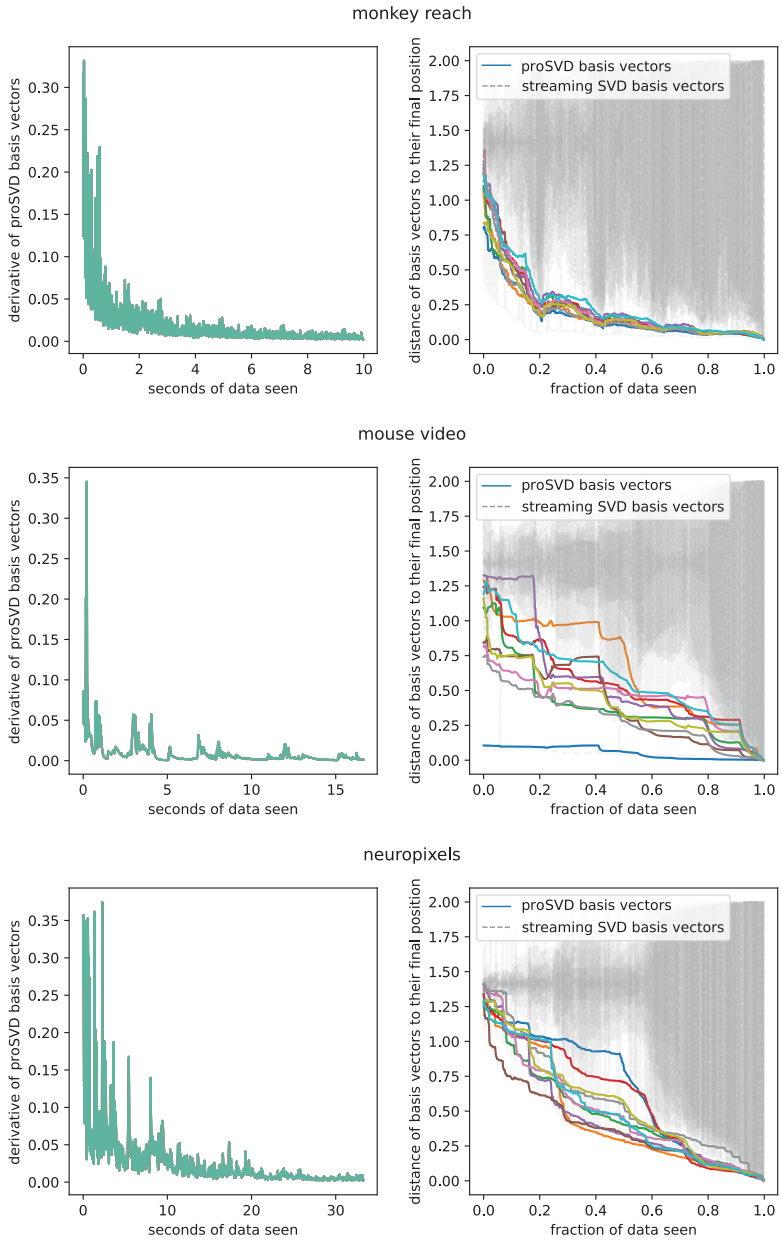

Figure S2: **Rapid stability of proSVD.** Titles reference data as in the main text. First column shows the derivatives of the proSVD basis vectors over time, i.e., the norms of the first difference of the basis vectors learned over time. Derivatives of streaming SVD vectors not shown for clarity. Second column shows how close the streaming vectors are to their final positions, measured by the Euclidean distance from the whole data basis vectors to the most recently learned basis vectors. $k = 10$ vectors shown.

D.8 Different seeds

To examine the consistency in the performance of Bubblewrap and the two comparison models, we generated a set of 100 trajectories per dataset using different random seeds to set initial conditions. Figure S10 shows the mean log predictive probability for each of these trajectories for the 2D Van der Pol and 3D Lorenz (0.05) datasets.

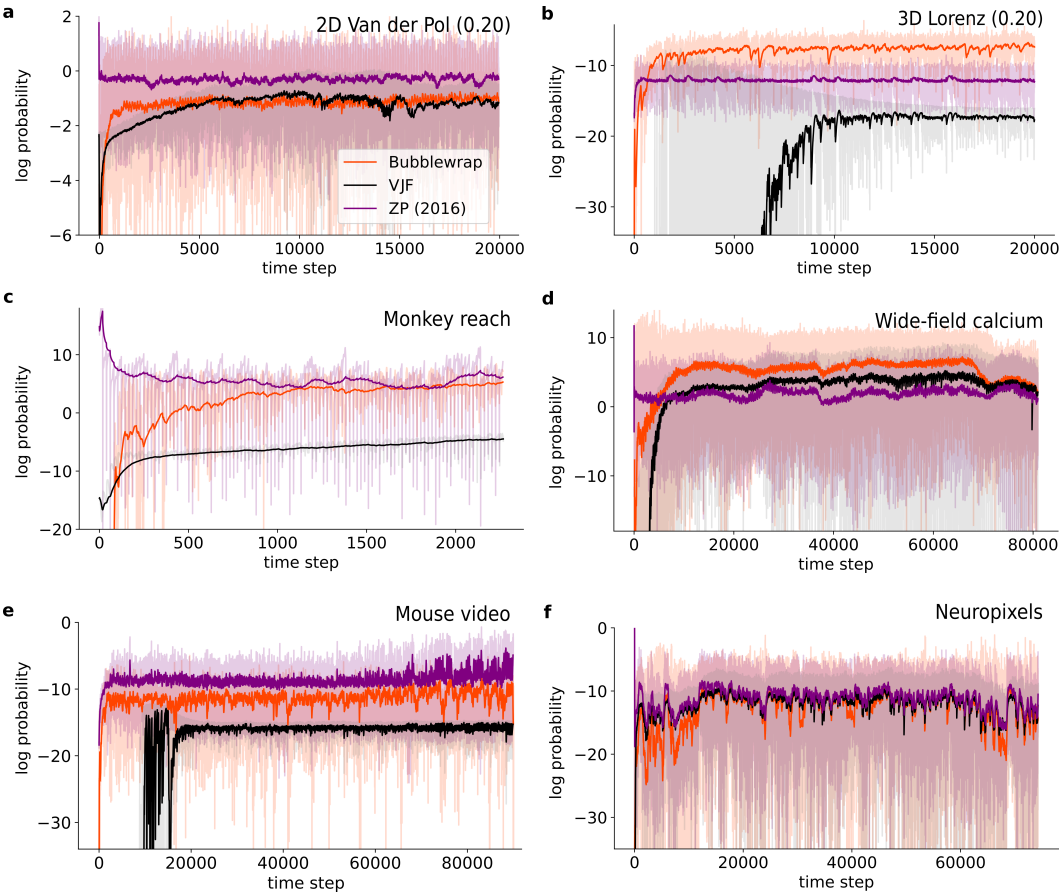

Figure S3: **Log predictive probability results** for all systems not shown in the main text. Log predictive probability across all timepoints for **a)** 2D Van der Pol (0.20 noise), **b)** 3D Lorenz (0.20 noise), **c)** Monkey reach, **d)** Wide-field calcium, **e)** Mouse video, and **f)** Neuropixels datasets. Conventions are as in Figure 2.

D.9 Benchmarking

Figure S11 displays a typical breakdown of timing for three separate steps within the Bubblewrap algorithm for single step (a) and batch mode (b). For single step updates, the algorithm observes a new data point, updates the data mean and covariance estimates, and uses these to update the node priors ('Update priors'). The algorithm then performs forward filtering and computing sufficient statistics ('E step'). Finally, the M step involves computing and applying gradients of $\mathcal{L}$. ('$\mathcal{L}$ gradient'). In batch mode, prior updates and gradient calculations are performed only once per batch, amortizing the cost of these updates. Finally, we display the time for calculating the model's predicted probability distribution over the tile index at the next time step $p(z_{t+1}|x_{1:t})$ ('Prediction'). This cost is on the order of microseconds.

D.10 Bubblewrap without heuristics

As noted in the main text, we employed heuristics in addition to the online EM updates to improve Bubblewrap's performance. Figure S12 shows results on the 2D Van der Pol (0.05) dataset with some features removed; 'Bubblewrap' in (a) is plotted for comparison with all features enabled.

When tiles are initialized, their priors place them at the center of mass of the earliest few data points. With each new data point, we re-estimate the data mean and covariance and use these to update the priors for μ_j and Σ_j. This includes random walk dynamics on μ_{0j} (cf. (20)) to break degeneracy among tiles that have seen the same or no data. If we remove this feature, we are still able to

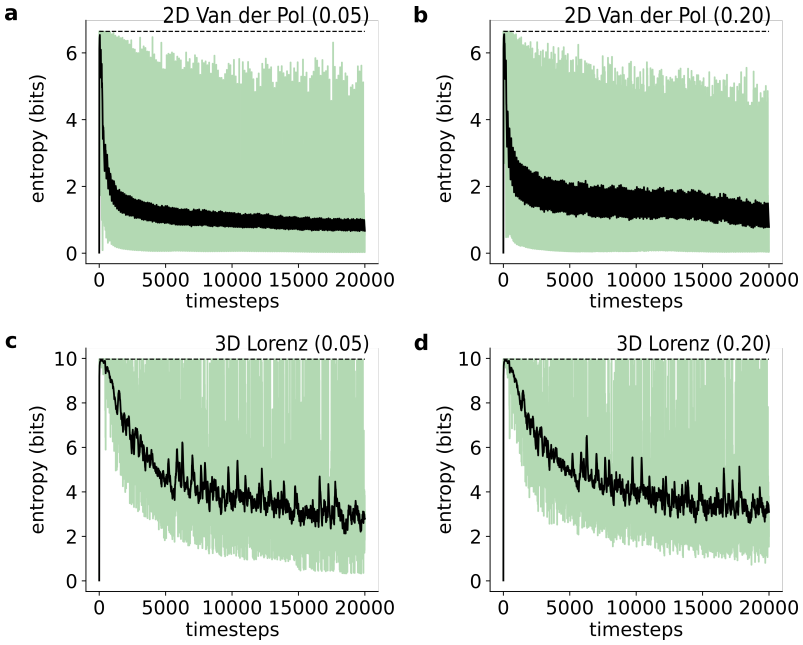

Figure S4: **Entropy** across all timepoints for each comparative model for all systems not shown in main text. Black lines are exponential weighted moving averages of the data. **a)** 2D Van der Pol (0.05), **b)** 2D Van der Pol (0.20), **c)** 3D Lorenz (0.05), **d)** and 3D Lorenz (0.20) datasets.

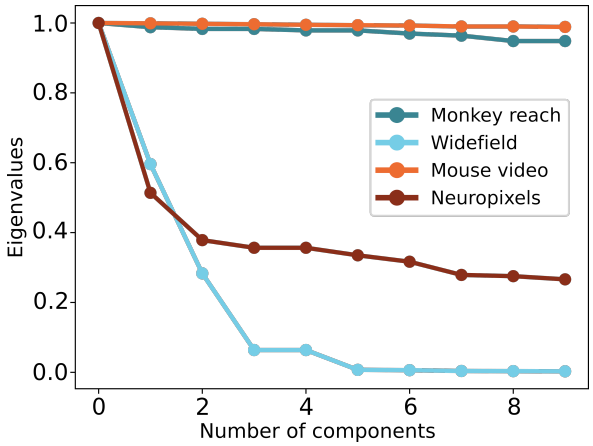

Figure S5: **Eigenspectra of final learned transition matrices.** Labels reference datasets as in the main text. Both the mouse video and monkey reach datasets have many eigenvalues close to 1, implying slow mixing of the corresponding Markov chain and long-range predictive power. By contrast, the widefield calcium and neuropixels datasets, with faster-decaying eigenspectra, quickly lose predictive information over time.

effectively tile the space ('No update priors', Fig. S12b) but a number of redundant, overlapping nodes are left in their initial state (black arrow). For timing considerations, we note that turning off these prior updates provides a speed benefit without drastically diminishing performance in some cases (Fig. S12e).

A second important heuristic we employed was to begin by marking all nodes as available for teleportation and thereby 'breadcrumb' the initial set of incoming datapoints. With this teleport feature turned off ('No teleporting', Fig. S12c), again Bubblewrap is able to effectively learn the correct tiling, though at a slower rate during initial learning (Fig. S12e). Additionally, because

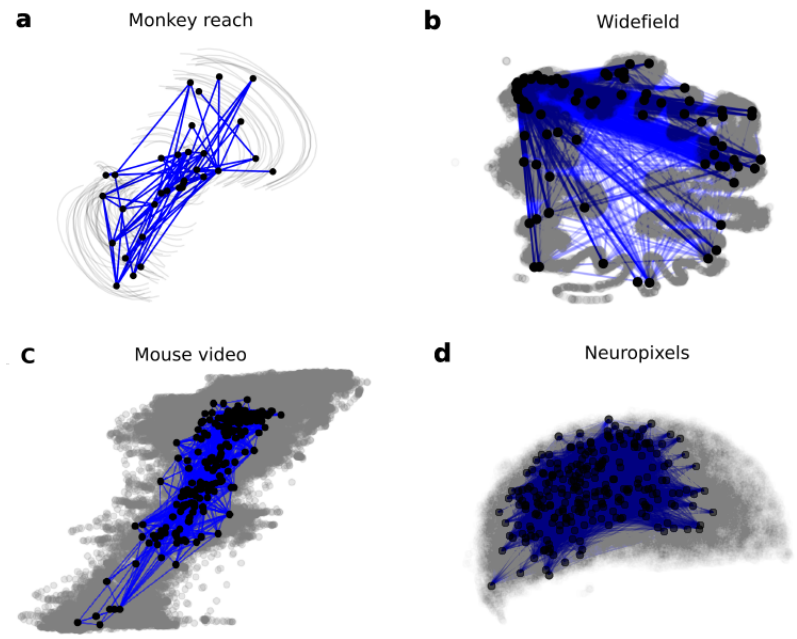

Figure S6: **Final transition matrices for each experimental dataset.** Labels reference datasets as in the main text. Data are plotted in grey as trajectories **(a)** or individual points **(b-d)**. Black dots are the final node locations and blue lines are entries in the final transition matrix A greater than the initialization value $\frac{1}{N}$.

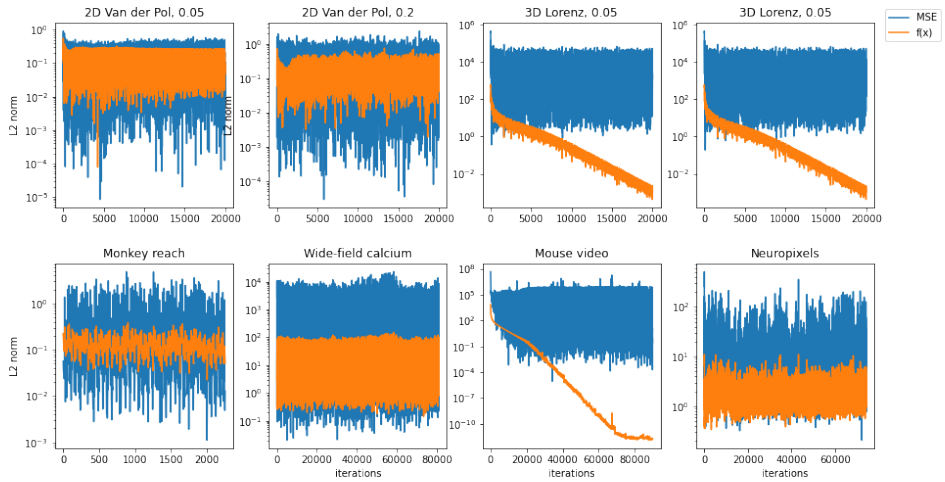

Figure S7: **ZP2016 degenerates to a random walk for some data sets.** Plots of ZP2016 prediction error (MSE, blue curves) and predicted step size ($f(x)$, orange curves, cf. (24)) for **a)** 2D Van der Pol (0.05), **b)** 2D Van der Pol (0.2), **c)** 3D Lorenz (0.05), **d)** 3D Lorenz (0.2), **e)** Monkey reach, **f)** Wide-field calcium, **g)** Mouse video, and **h)** Neuropixels data sets.

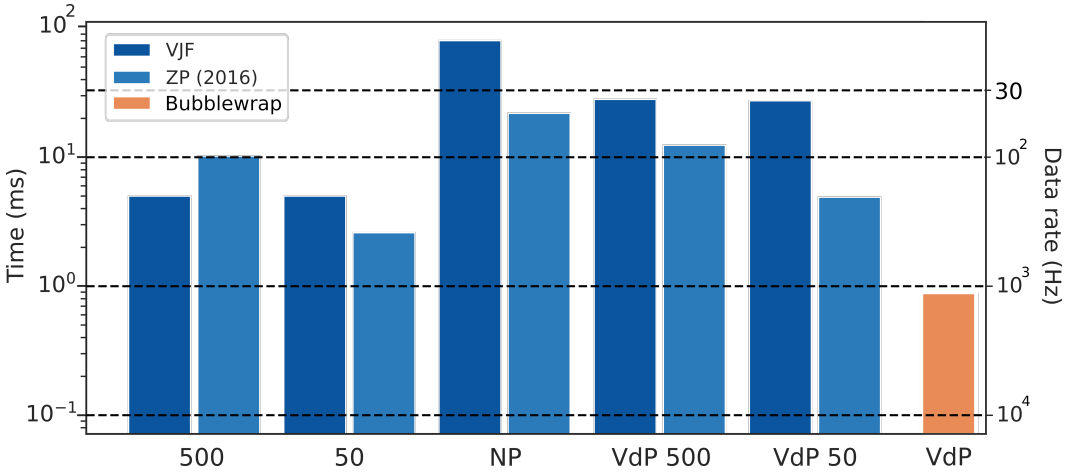

Figure S8: **Benchmarking of comparison models.** Average runtimes for the time to update the model for 500 ('500') or 50 ('50') basis functions on the 2D Van der Pol (0.05 noise) dataset; for the total time including prediction on the Neuropixels dataset ('NP'), the 2D Van der Pol dataset with 500 ('VdP 500') or 50 ('Vdp 50') basis functions; and for Bubblewrap on the base case of the 2D Van der Pol dataset using 1k nodes ('VdP').

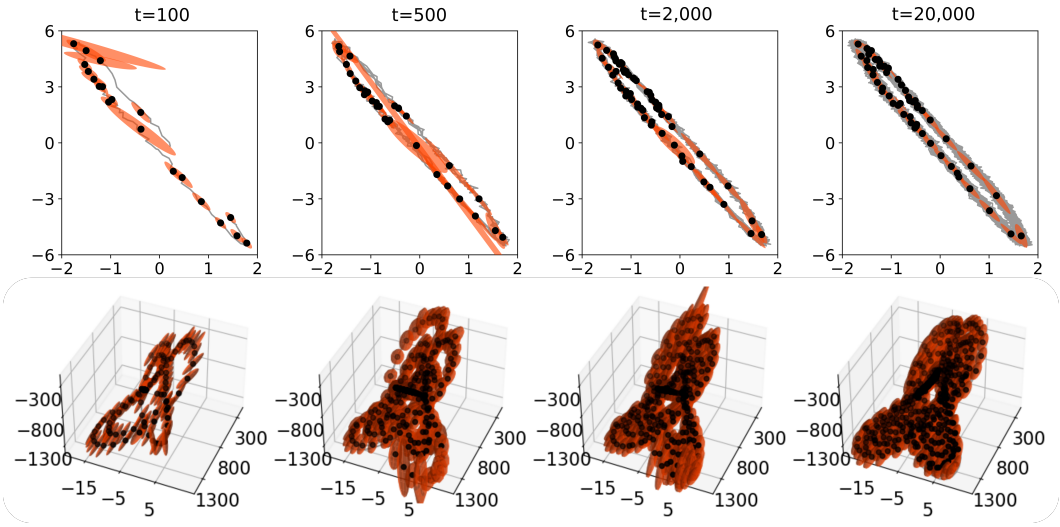

Figure S9: **Bubblewrap over time for simulated datasets.** Model results after t=100, 500, 2000, and 20000 (entire dataset) data points seen. Top row: 2D Van der Pol; bottom row: 3D Lorenz; both 0.05 noise cases.

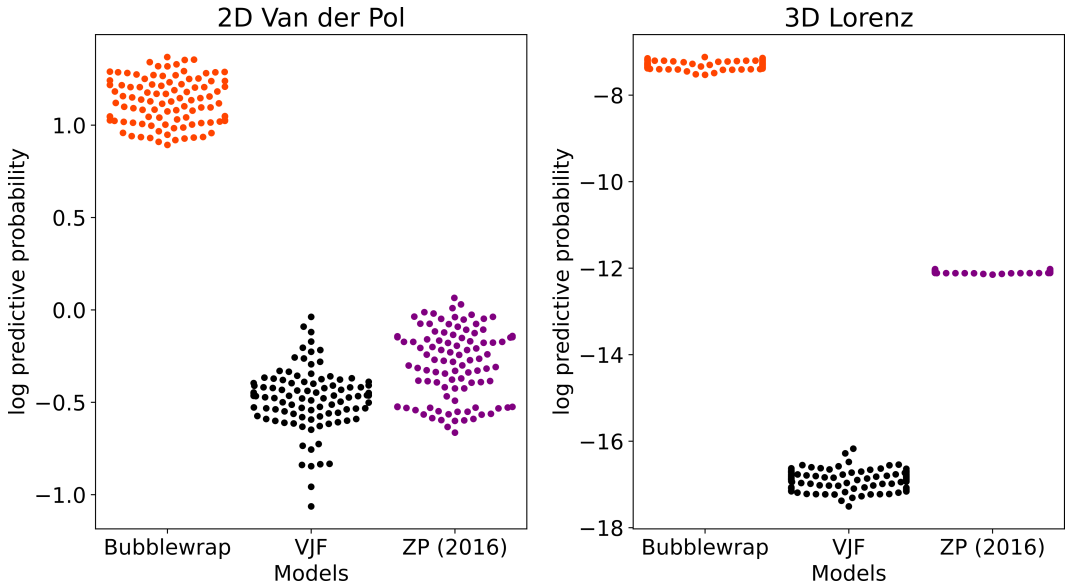

Figure S10: **Mean log predictive probability across seeds.** Model results for 100 different simulated trajectories of the 2D Van der Pol (0.05 noise) and 3D Lorenz (0.05) datasets.

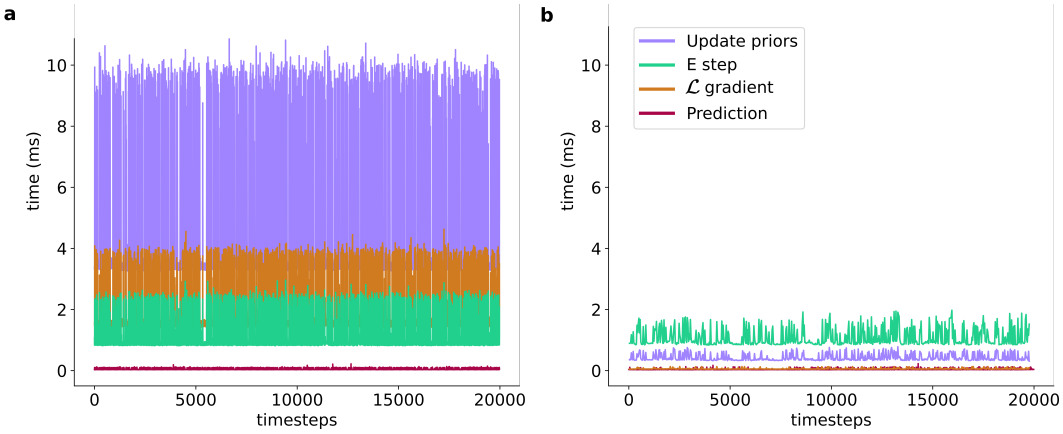

Figure S11: **Timing for different steps inside Bubblewrap. a)** Raw step times for updating priors, E step, gradient update of $\mathcal{L}$, and prediction for the 2D Van der Pol datatset. **b)** Same as in (a), but in batch mode, where prior updates and the gradient step are performed every 30 points.

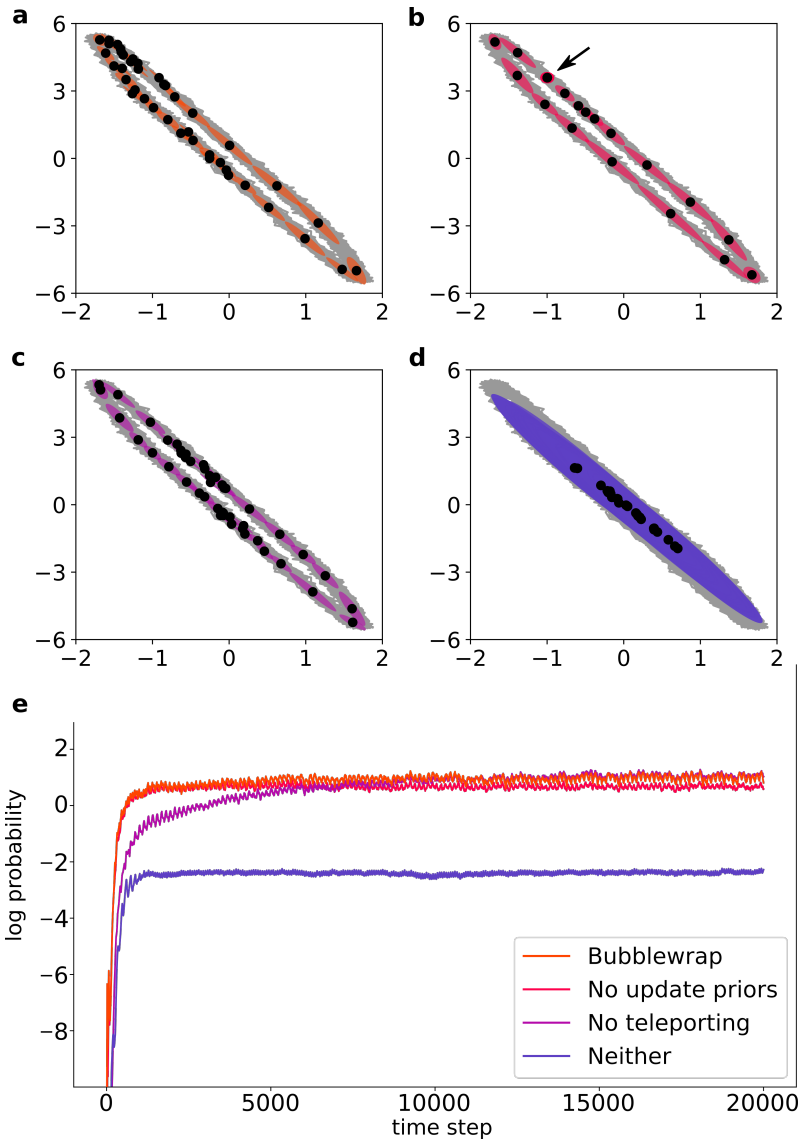

Figure S12: **Bubblewrap feature comparisons.** Model results for the 2D Van der Pol (0.05 noise) data set shown for different combinations of Bubblewrap heuristics. **a)** Reproduction of Figure 2a) with all features enabled. **b)** Results if priors are not updated with data estimates. **c)** Results if teleporting is not employed. **d)** Results if neither prior updates nor teleporting features are enabled. **e)** Smoothed log predictive probability across all time points for the 4 cases shown above (raw data omitted for clarity).

tile locations must be gradually updated through gradient updates on $\mathcal{L}$ rather than instantaneously updated via teleporting, the entire computation time per new data point is roughly 3 times slower.

Finally, if we employ neither of the above features, we can fall into less-optimal tilings with lower log predictive probability results ('Neither', Fig. S12d,e). Here many nodes have all tried to encompass all of the data simulataneously, with heavily overlapping covariance bubbles.