TensorSpike: Tractable Probabilistic Modeling for Neural Spike Trains with Tensor Networks

Yunyu Huang [1] **Fujia Wu** [1] **Dairui Chen** [1] **Xiaowei Gu** [2 3] **Zhe Sun** [3 4] **Chao Li** [5 3]

Abstract

Characterizing statistics in neural population activity is a central problem in systems neuroscience. We propose *TensorSpike*, a tractable probabilistic model for spike trains based on tensor networks (TNs). The model couples a tree tensor network (TTN) over neurons within each time bin with a matrix product state (MPS) over time bins. Scaled TN contractions enable stable likelihood computation, while isometric core tensors optimized on the Stiefel manifold ensure that the estimated probability distribution remains normalized throughout training. Experiments on synthetic and real neural datasets show that TensorSpike captures population-level statistics and supports exact likelihood evaluation.

1. Introduction

A central problem in systems neuroscience is to characterize statistics in the simultaneous activity of neural populations (Cunningham & Yu, 2014). Spike trains provide high-dimensional discrete records of this activity, with dependencies across neurons and time bins (Truccolo et al., 2005). A tractable probabilistic model (Choi et al., 2020) of the full joint distribution over spike trains could serve as a unified probabilistic basis for neural signal modeling, allowing exact evaluation, supporting marginal and conditional reasoning over subsets of neurons and time bins.

Existing models for neural population activity typically emphasize latent dynamics, generation, or prediction. Latent variable models (LVMs) (Yu et al., 2008; Pandarinath et al., 2018; Zoltowski et al., 2020; Valente et al., 2022; Geadah et al., 2026) describe shared population variability through low-dimensional latent states, while diffusion-based latent generative models such as LDNS (Kapoor et al., 2024) use learned latent trajectories to generate realistic spike trains. Transformer-based models (Ye & Pandarinath, 2021; Le & Shlizerman, 2022; Azabou et al., 2023; Ye et al., 2023; 2025) capture spatiotemporal dependencies for prediction and behavioral decoding. Despite their success, these methods do not primarily target tractable full joint probabilistic modeling of discrete spike trains.

To address this gap, we propose *TensorSpike*, a tractable probabilistic model for neural spike trains based on tensor networks (TNs) (Orús, 2014; Cichocki et al., 2016). Following prior tensor network generative modeling (Han et al., 2018; Miller et al., 2021), TensorSpike uses Born's rule to define a joint probability over spike trains. The underlying TN is composed of a tree tensor network (TTN) (Shi et al., 2006) over neurons within each time bin and a matrix product state (MPS) (Perez-Garcia et al., 2006) over time bins. For training, TensorSpike computes the log likelihood as a sum of conditional log likelihoods over time, using scaled TN contractions to avoid numerical instability. The core tensors are constrained to be isometric and optimized on the Stiefel manifold, so that the partition function of distribution remains equal to one throughout training.

We validate TensorSpike on a periodic synthetic dataset, an in-house resting-state rat prefrontal cortex (PFC) dataset, and the public monkey reaching dataset from the Neural Latents Benchmark (Pei et al., 2021). Our experiments test whether TensorSpike can capture statistics of spike trains in generated samples while enabling exact likelihood evaluation of recorded activity.

2. Preliminaries

Tensor networks A tensor is a multidimensional array, and its order is the number of indices, also called modes. The size of a tensor along a particular mode is called its mode dimension. Tensor contraction refers to summing over shared indices. A tensor network (TN) represents a high-order tensor as contractions of lower-order tensors, where nodes denote tensors and edges denote indices. An index summed over during contraction is called a contracted index, while an index that remains uncontracted is called

[1]Graduate School of Health Data Science, Juntendo University [2]RIKEN-CBS, Saitama, Japan [3]Faculty of Health Data Science, Juntendo University [4]Graduate School of Medicine, Juntendo University [5]RIKEN-AIP, Tokyo, Japan. Correspondence to: Chao Li <chao.li@riken.jp>.

Accepted at ICML 2026 Workshop on Connecting Low-rank Representations in AI (CoLoRAI).

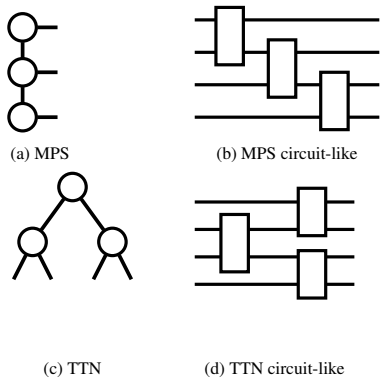


Figure 1. Tensor-network structures and their corresponding circuit-like formats. (a) Matrix product state (MPS). (b) Circuit-like format of MPS. (c) Tree tensor network (TTN). (d) Circuit-like format of TTN.

an open index. For example, matrix multiplication $C_{ij} = \sum_{k=1}^{r} A_{ik}B_{kj}$ contracts the shared index k, illustrated as i -(A)-^{k}-(B)- j . Here, k is a contracted index with mode dimension r, while i and j are open indices.

Figure 1 shows the MPS and TTN structures used in this work, together with their circuit-like formats. We use the circuit-like format as a visual representation of the same tensor contractions. For these circuit-like formats, we adopt a right-to-left contraction direction.

Unitary core tensors Consider a complex core tensor T_{abcd}, where all indices have the same mode dimension r. Under the right-to-left contraction direction, the right indices (c, d) are grouped as the input multi-index, and the left indices (a, b) are grouped as the output multi-index. The corresponding matrix Q is defined by

$$Q_{\mu,\nu} = T_{abcd}, \qquad \mu = (c-1)r + d, \quad \nu = (a-1)r + b,$$

where $a, b, c, d \in \{1, \ldots, r\}$. The core tensor is called unitary under this orientation if $Q^\dagger Q = QQ^\dagger = I$.

Observation matrices For a binary variable $x \in \{0, 1\}$, we represent its observed value using a fixed observation matrix P_x. For an even mode dimension r, we define

$$P_0 = \begin{pmatrix} I_{r/2} & 0 \\ 0 & 0 \end{pmatrix}, \qquad P_1 = \begin{pmatrix} 0 & 0 \\ 0 & I_{r/2} \end{pmatrix}.$$

Thus, P_0 and P_1 select the two possible binary states, and they satisfy $P_0 + P_1 = I_r$.

3. Method

TensorSpike defines a normalized joint distribution over binary spike trains using a TTN over neurons within each time bin and an MPS over time bins. We describe the probability model, scaled likelihood evaluation, constrained optimization, sampling, and computational complexity.

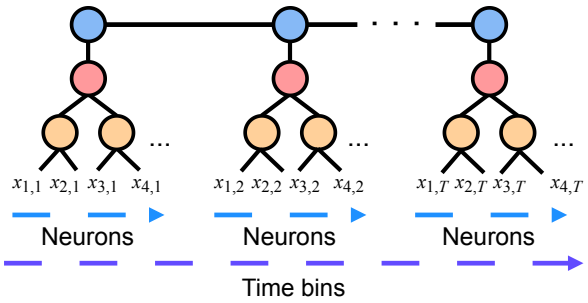


Figure 2. Tensor-network representation of f_θ.

3.1. Probabilistic modeling

TensorSpike uses Born's rule to define a probability distribution from a complex-valued function f_θ:

$$p_\theta(X) = |f_\theta(X)|^2,$$

where $X \in \{0, 1\}^{N \times T}$, N is the number of neurons, and T is the number of time bins. The TN representation of f_θ is shown in Figure 2: a TTN combines neural variables within each time bin, and an MPS connects the resulting representations across time. Using the circuit-like tensor-network representation, the squared magnitude $|f_\theta(X)|^2$ is represented directly as the probability contraction shown in Figure 3. Thus, Figure 3 visualizes the Born-rule probability $p_\theta(X)$, rather than only the amplitude $f_\theta(X)$. The learnable parameters θ are the TTN and MPS core tensors.

Each observed entry $x_{n,t}$ is inserted through the observation matrix $P_{x_{n,t}}$ defined in Section 2. In the circuit-like probability representation, these observation matrices act on the physical indices of the contraction for $|f_\theta(X)|^2$. When all binary variables are marginalized, each observation matrix becomes $P_0 + P_1 = I_r$, and the unitary core tensors defined in Section 2 make the full probability contraction collapse to one. Thus, TensorSpike defines a normalized probability distribution by construction.

3.2. Loss and scaled contraction

For a dataset $\mathcal{D} = \{X^{(m)}\}_{m=1}^{M}$, TensorSpike is trained by minimizing the negative log-likelihood (NLL).

$$\mathcal{L}(\theta) = -\frac{1}{M} \sum_{m=1}^{M} \log p_\theta\left(X^{(m)}\right).$$

For each spike train, the likelihood is evaluated by contracting the TensorSpike network described in Figure 3. Direct TN contraction can be numerically unstable for long spike trains, because intermediate tensors may become very small

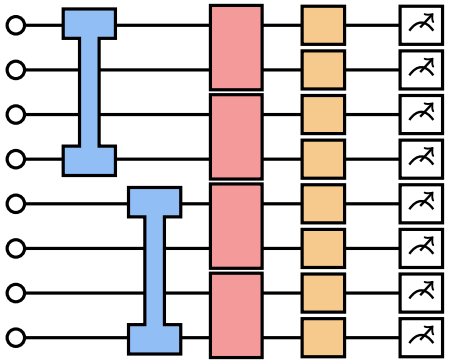


Figure 3. Circuit-like representation of the TensorSpike architecture.

or very large. TensorSpike therefore uses scaled contractions. Within each time bin, the TTN contracts the observation matrices and returns a rescaled representation M_t together with a log-scale factor. The temporal MPS then updates the contraction state using M_t. At each update, the scalar scale of the new state is separated from the normalized state and accumulated in log space. This preserves the value of the original TN contraction while avoiding numerical underflow or overflow. Details see in Appendix B.

3.3. Optimization

The learnable parameters θ are the TTN and MPS core tensors, while the observation matrices P_0 and P_1 are fixed. To preserve the normalization property established in Section 3.1, TensorSpike constrains the core tensors to be unitary under the right-to-left contraction direction defined in Section 2. Training is performed with projected or retracted gradient updates on the Stiefel manifold. After each gradient update, every core tensor is mapped back to the unitary constraint set. This ensures that the learned parameters remain within the normalized TensorSpike model class throughout training.

3.4. Sampling

Samples are generated by sequentially drawing each variable from the corresponding conditional distribution, following the standard direct sampling procedure used in tensor-network generative models (Han et al., 2018). Specifically, TensorSpike samples time bins from

$$x_t \sim p_\theta(x_t \mid x_{1:t-1}),$$

and samples neurons within each time bin one by one. To sample $x_{n,t}$, the model evaluates the two conditional values corresponding to $x_{n,t} = 0$ and $x_{n,t} = 1$. Previously sampled entries are fixed by their selected observation matrices, while unsampled entries are marginalized by replacing their input matrices with I_r.

3.5. Computational complexity

Let N be the number of neurons, T the number of time bins, and r the mode dimension. Computing the log-likelihood of one spike train requires $O(TNr^3)$ operations in the current contraction scheme. Sequential sampling is more expensive because conditional scores are evaluated for individual binary variables. In the current implementation, sampling one spike train requires $O(TN^2r^3)$ operations.

The memory cost is dominated by the TTN and MPS core tensors and the forward contraction state. For four-index core tensors with mode dimension r, each core tensor has $O(r^4)$ entries. With shared TTN and MPS core tensors, parameter storage is therefore $O(Nr^4 + r^4)$.

4. Experiments

We evaluate TensorSpike on synthetic and real neural population datasets. The experiments test whether the model captures population-level spike statistics, single-neuron temporal statistics, and explicit likelihoods of binary spike trains.

4.1. Experimental setup

Datasets. We use three datasets. The first is a synthetic periodic dataset with known temporal structure. It contains 200 samples, each with 100 time bins, generated from four neurons with phase-shifted periodic firing patterns. The second is an in-house rat prefrontal cortex (PFC) dataset recorded during resting-state electrophysiology. It contains 200 samples from 112 recorded units, with each sample represented as a binary sequence of 100 time bins using a bin size of 5 ms. The third is the MC Maze dataset from the Neural Latents Benchmark (NLB) (Pei et al., 2021). We use 2008 trials with 182 neurons, where each trial is represented as a binary sequence of 140 time bins using a bin size of 5 ms. All datasets are converted into binary spike-event sequences.

Evaluation metrics. We compare real and generated samples using four statistics: population spike count, pairwise neuronal correlation, mean inter-spike interval (ISI), and standard deviation of ISI. Population spike count evaluates global activity levels, pairwise correlation evaluates dependencies among neurons, and ISI statistics evaluate single-neuron temporal firing patterns. For quantitative comparison, we report a histogram discrepancy for population spike counts and RMSE for pairwise correlation, mean ISI, and standard deviation of ISI. Lower values indicate better agreement with real data. Metric definitions are provided in Appendix A.

Baselines. We compare TensorSpike with representative models from the three families discussed in the introduc-

tion. LFADS (Pandarinath et al., 2018) is a recurrent latent-variable model that infers low-dimensional neural dynamics; we use the AutoLFADS training pipeline (Keshtkaran et al., 2022). LDNS (Kapoor et al., 2024) is a diffusion-based latent generative model for neural spike trains. STNDT (Le & Shlizerman, 2022) is a Transformer-based neural sequence model that uses attention to capture spatiotemporal dependencies in population activity. These baselines cover latent-variable, diffusion-based, and attention-based approaches. In contrast, TensorSpike directly represents a joint distribution over binary spike trains through tensor network contractions.

Training details. TensorSpike uses a TTN over neurons within each time bin and an MPS across time bins. It is trained by minimizing the negative log-likelihood using the scaled contraction procedure in Section 3.2. After training, samples are generated sequentially as described in Section 3.4. For all methods, generated sample statistics are computed using the same evaluation scripts. The main TensorSpike hyperparameters are summarized in Table 1; additional computing details are provided in Appendix C.

Table 1. Training hyperparameters.

Dataset	Learning rate	Mode dim. r	Batch size	Epochs
Synthetic	1×10^{-4}	12	200	200
Rat PFC	5×10^{-4}	26	20	1000
MC Maze	5×10^{-4}	24	20	300

4.2. Controlled validation on synthetic periodic activity

The synthetic dataset provides a controlled setting where the temporal structure is known. A successful model should recover both the marginal firing patterns of individual neurons and the phase-shifted coordination across neurons.

Figure 4 shows that TensorSpike reproduces the periodic spiking structure. The generated raster preserves the phase-shifted firing patterns, and the population spike count histogram closely matches the training distribution. TensorSpike also preserves pairwise correlations and ISI statistics, indicating that it captures both cross-neuron coordination and single-neuron temporal regularity.

4.3. Real neural population modeling

We next evaluate TensorSpike on two real neural population datasets. These experiments test whether the explicit probability model can generate realistic binary spike trains beyond controlled synthetic structure.

4.3.1. Rat PFC dataset

The rat PFC dataset contains sparse and irregular resting-state activity. Figure 5 shows that TensorSpike generates samples with activity patterns similar to the real recordings. The population spike count distribution closely matches the empirical distribution, and pairwise correlations are largely preserved. The mean and standard deviation of ISI are also well aligned with the real data, suggesting that TensorSpike captures both population-level activity and single-unit temporal firing statistics in this dataset.

4.3.2. MC Maze dataset

The MC Maze dataset is higher dimensional and contains task-dependent motor cortex activity. Figure 6 shows that TensorSpike remains stable on this dataset and generates sparse population activity. The population spike count distribution closely follows the empirical distribution, and pairwise correlations are partially preserved.

The ISI statistics show larger deviations than in the rat PFC dataset. One likely reason is that MC Maze contains non-stationary activity across different phases of the reaching task. The current TensorSpike architecture uses a shared temporal MPS core, which limits its ability to model time-varying neuron-specific temporal statistics. This motivates future extensions with time-dependent temporal cores or richer temporal parameterizations.

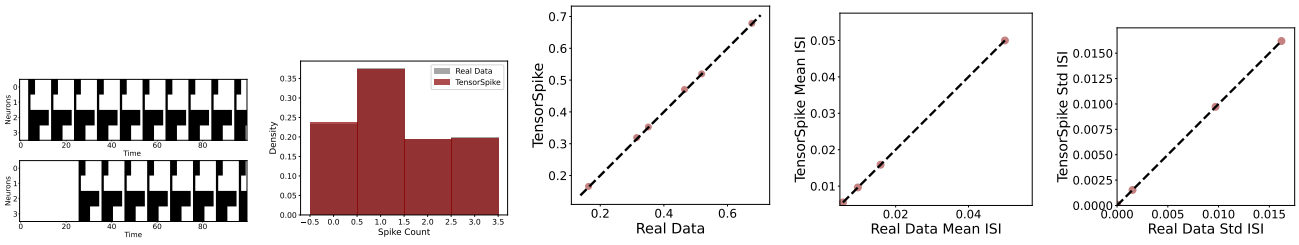
4.4. Sample-based baseline comparison

Table 2 compares TensorSpike with LFADS, LDNS, and STNDT. On the rat PFC dataset, TensorSpike achieves the lowest population spike count discrepancy and the lowest errors for both ISI metrics, while LFADS gives the best pairwise correlation error. This suggests that TensorSpike captures the overall sparsity and single-neuron temporal statistics of the resting-state recordings well.

On the MC Maze dataset, TensorSpike again achieves the lowest population spike count discrepancy, indicating that it captures the global activity level of the motor-cortex population. LDNS performs best on pairwise correlation and ISI metrics, suggesting stronger modeling of fine-grained temporal structure in this task-dependent dataset. Overall, TensorSpike is strongest on population spike count agreement and remains competitive on several temporal statistics, while more expressive temporal components may be needed for non-stationary reaching activity.

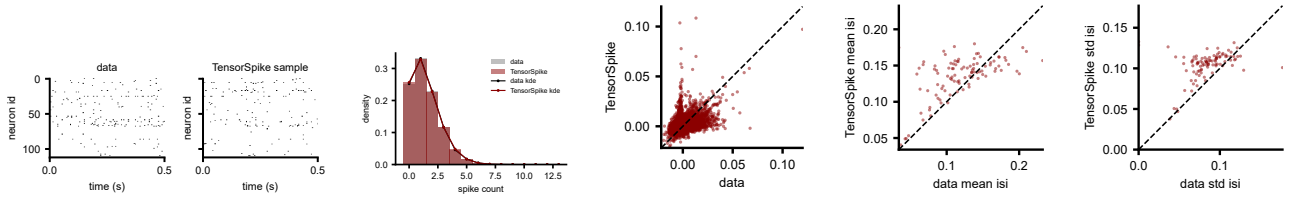
4.5. Explicit likelihood results

Beyond sample-based statistics, TensorSpike directly assigns probabilities to binary spike trains. This enables likelihood evaluation through tensor network contractions, which



(a) *(b)* *(c)* *(d)* *(e)*

Figure 4. **TensorSpike recovers structured periodic spiking activity.** (a) Spike raster comparison. (b) Population spike count histogram. (c) Pairwise neuronal correlations. (d,e) Mean and standard deviation of ISI.



(a) *(b)* *(c)* *(d)* *(e)*

Figure 5. **TensorSpike generates realistic resting-state population activity in the rat PFC dataset.** (a) Spike raster comparison. (b) Population spike count distribution. (c) Pairwise neuronal correlations. (d,e) Mean and standard deviation of ISI.

Table 2. Quantitative comparison between real and generated spike train statistics across 5 fold sampled with replacement. Count denotes population spike count histogram discrepancy. Corr., Mean ISI, and Std ISI denote RMSE. Lower is better.

Dataset	Method	Count	Corr.	Mean ISI	Std ISI
Rat PFC	LFADS	$0.0034 \pm 3.0\text{e}{-4}$	$\mathbf{0.0088 \pm 1.7\text{e}{-4}}$	0.054 ± 0.004	0.036 ± 0.001
	LDNS	$0.0166 \pm 3.3\text{e}{-3}$	$0.0093 \pm 2.7\text{e}{-4}$	0.051 ± 0.007	0.037 ± 0.002
	STNDT	$0.0912 \pm 1.9\text{e}{-3}$	$0.0091 \pm 1.5\text{e}{-4}$	0.079 ± 0.004	0.047 ± 0.004
	TensorSpike	$\mathbf{0.0008 \pm 1.6\text{e}{-4}}$	$0.0097 \pm 1.3\text{e}{-4}$	$\mathbf{0.049 \pm 0.003}$	$\mathbf{0.035 \pm 0.002}$
MC Maze	LFADS	$0.0040 \pm 2.2\text{e}{-4}$	$0.0026 \pm 1.25\text{e}{-5}$	0.039 ± 0.003	0.029 ± 0.001
	LDNS	$0.0039 \pm 3.9\text{e}{-4}$	$\mathbf{0.0025 \pm 1.1\text{e}{-5}}$	$\mathbf{0.037 \pm 0.001}$	$\mathbf{0.023 \pm 0.001}$
	STNDT	$0.5210 \pm 2.0\text{e}{-3}$	$0.0031 \pm 1.0\text{e}{-5}$	0.097 ± 0.003	0.051 ± 0.0005
	TensorSpike	$\mathbf{0.0020 \pm 6.0\text{e}{-5}}$	$0.0029 \pm 1.1\text{e}{-5}$	0.062 ± 0.001	0.032 ± 0.0005

Table 3. Negative log-likelihood of TensorSpike. Values are normalized by the number of neuron-time bins. Lower is better.

Dataset	N	T	NLL / bin
Synthetic	4	100	0.009
Rat PFC	112	100	0.065
MC Maze	182	140	0.107

is not directly available for implicit generators or latent generative baselines in the same binary joint-distribution form.

Table 3 reports the negative log-likelihood of TensorSpike, normalized by the number of neuron-time bins. Lower values indicate higher likelihood assigned to the evaluated spike trains.

5. Conclusion

6. Conclusion

We introduced TensorSpike, a probabilistic model for binary neural spike trains using TNs. TensorSpike represents the probability distribution with a TTN over neurons within each time bin and an MPS over time bins, enabling normalized probability evaluation and stable likelihood computation through constrained and scaled TN contractions.

Experiments on synthetic periodic activity, rat PFC recordings, and the MC Maze benchmark show that TensorSpike captures important statistics of neural spike trains while supporting explicit likelihood evaluation. On the synthetic dataset, TensorSpike recovers the known phase-shifted temporal structure. On the rat PFC dataset, it matches population spike count distributions and single-neuron ISI statistics well. On the MC Maze dataset, TensorSpike captures global

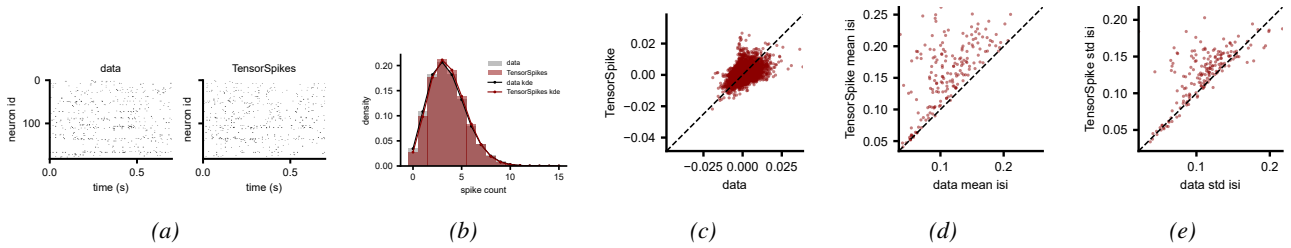


Figure 6. **TensorSpike generates sparse motor-cortex population activity in the MC Maze reaching task.** (a) Spike raster comparison. (b) Population spike count distribution. (c) Pairwise neuronal correlations. (d,e) Mean and standard deviation of ISI.

spike count statistics but shows larger deviations in fine-grained ISI statistics, suggesting that more expressive temporal parameterizations may be needed for non-stationary task-related activity.

Overall, TensorSpike provides a tractable approach to full joint probabilistic modeling of neural spike trains.

Limitations. Although TensorSpike avoids the exponential cost of enumerating all spike-train patterns, sequential sampling remains more expensive than likelihood evaluation because conditional scores are computed autoregressively. The current shared temporal MPS core also limits the modeling of time-varying neural dynamics. Future work will explore more expressive temporal cores and more efficient sampling procedures.

Broader impact. TensorSpike may support neuroscience research by enabling simulation and probabilistic analysis of neural population activity. However, generated neural activity and likelihood-based scores should not be used as substitutes for real recordings or for clinical and behavioral decisions without careful validation.

Acknowledgements

This work was supported by Japan Agency for Medical Research and Development Brain Minds 2.0 Project under Grant Number 24wm0625411h0001, 25wm0625411h0002, and 26wm0625411h0003.

References

Azabou, M., Arora, V., Ganesh, V., Mao, X., Nachimuthu, S., Mendelson, M., Richards, B., Perich, M., Lajoie, G., and Dyer, E. A unified, scalable framework for neural population decoding. *Advances in Neural Information Processing Systems*, 36:44937–44956, 2023.

Choi, Y., Vergari, A., and Van den Broeck, G. Probabilistic circuits: A unifying framework for tractable probabilistic models. *UCLA. URL: http://starai. cs. ucla. edu/papers/ProbCirc20. pdf*, 6, 2020.

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D. P. Tensor networks for dimensionality reduction and large-scale optimization part 1 low-rank tensor decompositions. *Foundations and Trends in Machine Learning*, 9(4-5):249–429, 2016.

Cunningham, J. P. and Yu, B. M. Dimensionality reduction for large-scale neural recordings. *Nature neuroscience*, 17(11):1500–1509, 2014.

Geadah, V., Nejatbakhsh, A., Lipshutz, D., Pillow, J., and Williams, A. Modeling neural activity with conditionally linear dynamical systems. *Advances in Neural Information Processing Systems*, 38:87635–87665, 2026.

Han, Z.-Y., Wang, J., Fan, H., Wang, L., and Zhang, P. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3):031012, 2018.

Kapoor, J., Schulz, A., Vetter, J., Pei, F., Gao, R., and Macke, J. H. Latent diffusion for neural spiking data. *Advances in Neural Information Processing Systems*, 37: 118119–118154, 2024.

Keshtkaran, M. R., Sedler, A. R., Chowdhury, R. H., Tandon, R., Basrai, D., Nguyen, S. L., Sohn, H., Jazayeri, M., Miller, L. E., and Pandarinath, C. A large-scale neural network training framework for generalized estimation of single-trial population dynamics. *Nature methods*, 19 (12):1572–1577, 2022.

Le, T. and Shlizerman, E. Stndt: Modeling neural population activity with spatiotemporal transformers. *Advances in Neural Information Processing Systems*, 35:17926–17939, 2022.

Miller, J., Rabusseau, G., and Terilla, J. Tensor networks for probabilistic sequence modeling. In *International Conference on Artificial Intelligence and Statistics*, pp. 3079–3087. PMLR, 2021.

Orús, R. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of physics*, 349:117–158, 2014.

Pandarinath, C., O'Shea, D. J., Collins, J., Jozefowicz, R., Stavisky, S. D., Kao, J. C., Trautmann, E. M., Kaufman, M. T., Ryu, S. I., Hochberg, L. R., et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, 15(10):805–815, 2018.

Pei, F., Ye, J., Zoltowski, D., Wu, A., Chowdhury, R. H., Sohn, H., O'Doherty, J. E., Shenoy, K. V., Kaufman, M. T., Churchland, M., et al. Neural latents benchmark'21: evaluating latent variable models of neural population activity. *arXiv preprint arXiv:2109.04463*, 2021.

Perez-Garcia, D., Verstraete, F., Wolf, M. M., and Cirac, J. I. Matrix product state representations. *arXiv preprint quant-ph/0608197*, 2006.

Shi, Y.-Y., Duan, L.-M., and Vidal, G. Classical simulation of quantum many-body systems with a tree tensor network. *Physical Review A—Atomic, Molecular, and Optical Physics*, 74(2):022320, 2006.

Truccolo, W., Eden, U. T., Fellows, M. R., Donoghue, J. P., and Brown, E. N. A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects. *Journal of neurophysiology*, 93(2):1074–1089, 2005.

Valente, A., Pillow, J. W., and Ostojic, S. Extracting computational mechanisms from neural data using low-rank rnns. *Advances in Neural Information Processing Systems*, 35:24072–24086, 2022.

Ye, J. and Pandarinath, C. Representation learning for neural population activity with neural data transformers. *arXiv preprint arXiv:2108.01210*, 2021.

Ye, J., Collinger, J., Wehbe, L., and Gaunt, R. Neural data transformer 2: multi-context pretraining for neural spiking activity. *Advances in Neural Information Processing Systems*, 36:80352–80374, 2023.

Ye, J., Rizzoglio, F., Smoulder, A., Mao, H., Ma, X., Marino, P., Chowdhury, R., Moore, D., Blumenthal, G., Hockeimer, W., et al. A generalist intracortical motor decoder. *bioRxiv*, 2025.

Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S., Shenoy, K. V., and Sahani, M. Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. *Advances in neural information processing systems*, 21, 2008.

Zoltowski, D., Pillow, J., and Linderman, S. A general recurrent state space framework for modeling neural dynamics during decision-making. In *International Conference on Machine Learning*, pp. 11680–11691. PMLR, 2020.

A. Evaluation Metrics

We evaluate generated spike trains using population-level and single-neuron statistics. These metrics are computed for both real data and generated samples, and the discrepancies between them are used for quantitative comparison in Table 2.

A.1. Population spike count Distribution

For a spike train $X \in \{0,1\}^{N\times T}$, the population spike count at time bin t is defined as

$$C_t = \sum_{n=1}^{N} x_{n,t},$$

where N is the number of neurons. The set of counts $\{C_t\}_{t=1}^{T}$ summarizes the distribution of global population activity across time. For multiple trials or samples, we aggregate the spike counts across all trials and time bins.

We compare the empirical population spike count distribution of real data and generated samples using histogram discrepancy. In our quantitative tables, this entry is denoted as Count. When using KL divergence, the discrepancy is computed as

$$D_{\mathrm{KL}}(P_{\mathrm{real}} \,\|\, P_{\mathrm{gen}}) = \sum_{c} P_{\mathrm{real}}(c) \log \frac{P_{\mathrm{real}}(c)}{P_{\mathrm{gen}}(c)+\epsilon},$$

where P_{real} and P_{gen} are the normalized spike count histograms of real and generated data, and ϵ is a small constant for numerical stability.

A.2. Pairwise Neuronal Correlation

To evaluate dependencies among neurons, we compute the Pearson correlation coefficient between spike trains of neuron i and neuron j:

$$\rho_{i,j} = \frac{\mathbb{E}_t\left[(x_{i,t}-\mu_i)(x_{j,t}-\mu_j)\right]}{\sigma_i \sigma_j},$$

where μ_i and σ_i are the mean and standard deviation of neuron i's activity across time bins. For trial-based datasets, correlations are computed after concatenating spike trains across trials.

Each point in the pairwise correlation plots corresponds to one neuron pair (i,j), comparing $\rho_{i,j}^{\mathrm{real}}$ and $\rho_{i,j}^{\mathrm{gen}}$. The quantitative discrepancy is reported as RMSE:

$$\mathrm{RMSE}_{\mathrm{corr}} = \sqrt{\frac{2}{N(N-1)} \sum_{1\le i<j\le N} \left(\rho_{i,j}^{\mathrm{real}} - \rho_{i,j}^{\mathrm{gen}}\right)^2}.$$

A.3. Inter-Spike Interval Statistics

For each neuron, the inter-spike interval (ISI) is defined as the time difference between consecutive spike events. Let

$$\mathcal{T}_n = \{t : x_{n,t} = 1\}$$

denote the set of spike times for neuron n. If $\mathcal{T}_n = \{t_1^{(n)}, t_2^{(n)}, \ldots, t_{K_n}^{(n)}\}$ with $t_1^{(n)} < t_2^{(n)} < \cdots < t_{K_n}^{(n)}$, then the ISIs are

$$\mathrm{ISI}_k^{(n)} = t_{k+1}^{(n)} - t_k^{(n)}, \qquad k = 1, \ldots, K_n - 1.$$

We compute two single-neuron summary statistics:

$$\mathrm{MeanISI}_n = \frac{1}{K_n - 1} \sum_{k=1}^{K_n-1} \mathrm{ISI}_k^{(n)},$$

and

$$\mathrm{StdISI}_n = \sqrt{\frac{1}{K_n - 1} \sum_{k=1}^{K_n-1} \left(\mathrm{ISI}_k^{(n)} - \mathrm{MeanISI}_n\right)^2}.$$

The mean ISI measures the typical firing interval of a neuron, while the standard deviation of ISI measures temporal variability. In the quantitative comparison, we compute RMSE between real and generated per-neuron statistics:

$$\mathrm{RMSE}_{\mathrm{MeanISI}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\mathrm{MeanISI}_n^{\mathrm{real}} - \mathrm{MeanISI}_n^{\mathrm{gen}}\right)^2},$$

and

$$\mathrm{RMSE}_{\mathrm{StdISI}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\mathrm{StdISI}_n^{\mathrm{real}} - \mathrm{StdISI}_n^{\mathrm{gen}}\right)^2}.$$

Neurons with fewer than two spikes do not have a well-defined ISI. In practice, we exclude such neurons from ISI-based RMSE computation for both real and generated data.

B. Equivalence of Scaled Contraction

This appendix explains why the scaled likelihood computation in Section 3 is equivalent to the original tensor-network probability.

The main idea is simple: tensor contractions are multilinear. Therefore, if an intermediate tensor is multiplied or divided by a scalar, the final contraction is multiplied or divided by the same scalar. We can therefore remove scalar factors during contraction for numerical stability, as long as we add them back in the log-likelihood.

TTN scaling. Within each time bin t, the TTN contracts the observation matrices $\{P_{x_{n,t}}\}_{n=1}^{N}$ from bottom to top. During this contraction, some intermediate tensors may become very small. We rescale these intermediate tensors and record the removed scale in log form. Let α_t denote the total log-scale removed during the TTN contraction at time bin t.

After rescaling, the TTN outputs a normalized intermediate tensor M_t. The removed scale is not discarded; it is added to the final log-likelihood as α_t.

Temporal MPS scaling. After the TTN contraction, the temporal MPS processes the sequence of tensors $M_1, \ldots, M_T$. At time bin t, the MPS update produces an intermediate contraction $\widetilde{H}_t$. Its trace gives the scale contributed by the current time bin:

$$c_t = \mathrm{Tr}(\widetilde{H}_t).$$

Equivalently, this is the tensor-network version of computing conditional probabilities one step at a time: the scale c_t corresponds to $p_\theta(x_t \mid x_{1:t-1})$. We then normalize the state by

$$H_t = \frac{\widetilde{H}_t}{c_t}.$$

Because the following MPS contractions are multilinear in H_t, dividing $\widetilde{H}_t$ by c_t only removes a scalar factor from the forward contraction. This scalar factor is stored separately as $\log c_t$. Thus, the normalization prevents numerical underflow without changing the probability assigned by the original tensor network.

Combining the TTN scale and the temporal MPS scale, the log-likelihood becomes

$$\log p_\theta(X) = \sum_{t=1}^{T}\left(\alpha_t + \log c_t\right),$$

which is the same value as the unscaled full tensor-network contraction, but computed in a numerically stable way.

C. Computing Environment

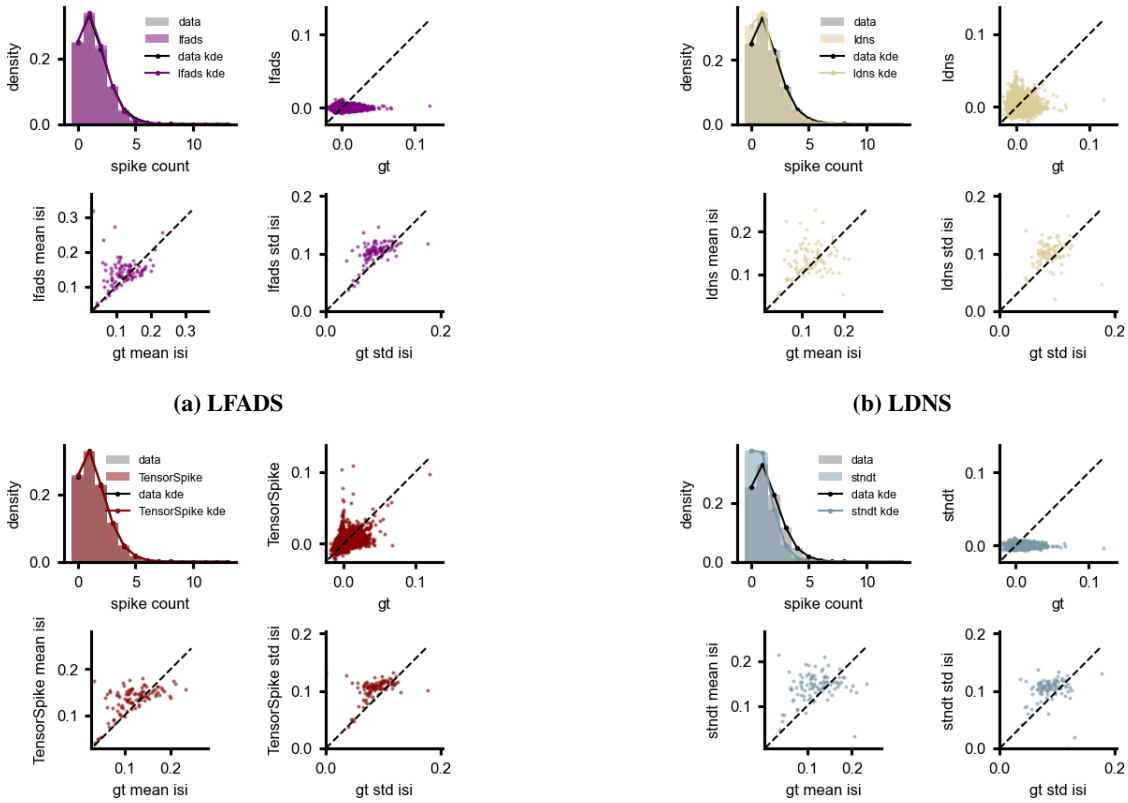
Table 4 summarizes the hardware used for training and sampling. TB and STNDT are trained and sampled on a single NVIDIA H100 GPU. LFADS and LDNS are trained and sampled on a single NVIDIA RTX 4090 GPU.

Table 4. Computing environment.

Model	Hardware
TB	Single NVIDIA H100 GPU
STNDT	Single NVIDIA H100 GPU
LFADS	Single NVIDIA RTX 4090 GPU
LDNS	Single NVIDIA RTX 4090 GPU

D. Baseline Comparison Results

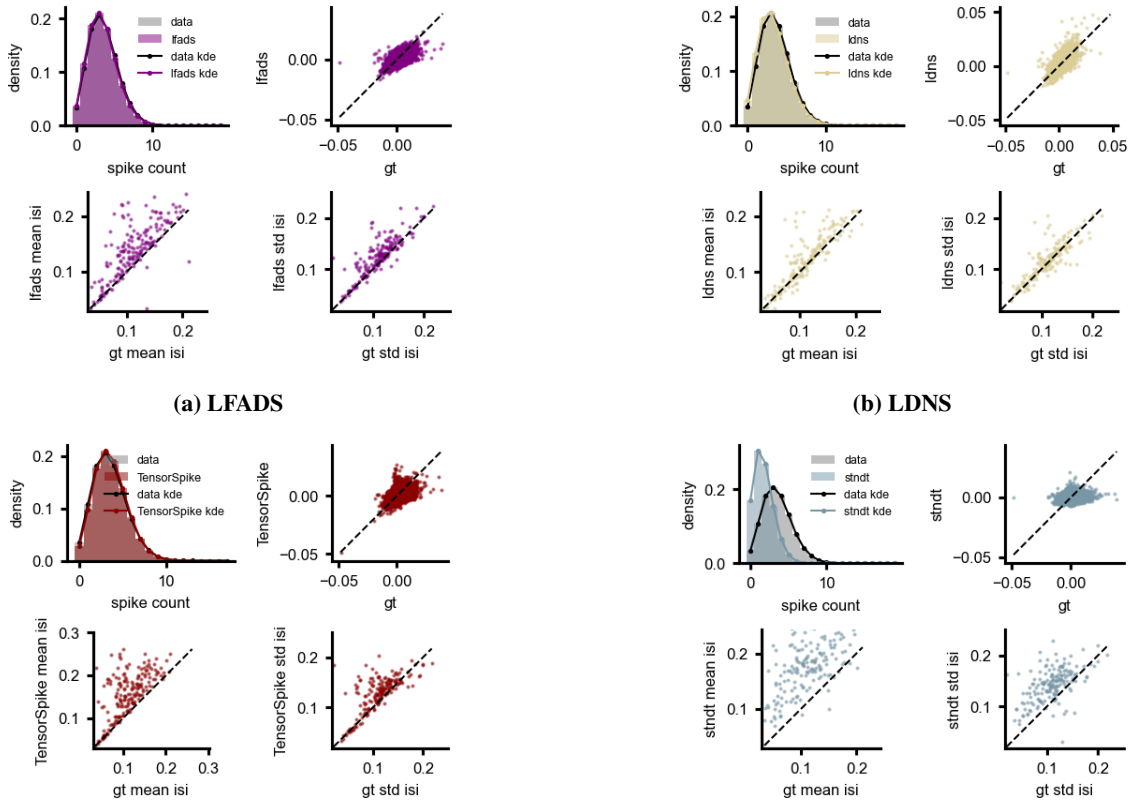
We provide additional metric plots comparing TensorSpike with LDNS, LFADS, and STNDT on the rat PFC dataset and the MC Maze dataset. For each method, we use the same evaluation metrics as in the main text: population spike count histograms, pairwise correlation comparisons, mean ISI, and standard deviation of ISI. LFADS is trained using the AutoLFADS pipeline.



(a) LFADS **(b) LDNS**

(c)TensorSpike **(d) STNDT**

Figure 7. Baseline comparison on the rat PFC dataset. Each method is evaluated using four statistics: population spike count histogram, pairwise correlation, mean ISI, and standard deviation of ISI.



(a) LFADS **(b) LDNS**

(c) TensorSpike **(d) STNDT**

Figure 8. Baseline comparison on the MC Maze dataset. Each method is evaluated using four statistics: population spike count histogram, pairwise correlation, mean ISI, and standard deviation of ISI.