NEURAL NETWORKS FOR BAYESIAN INVERSE PROBLEMS GOVERNED BY A NONLINEAR ODE*

GERMAN VILLALOBOS†, JOHANN RUDI‡, AND ANDREAS MANG†

Abstract. We investigate the use of neural networks (NNs) for the estimation of hidden model parameters and uncertainty quantification from noisy observational data for inverse parameter estimation problems. We formulate the parameter estimation as a Bayesian inverse problem. We consider a parametrized system of nonlinear ordinary differential equations (ODEs), which is the FitzHugh–Nagumo model. The considered problem exhibits significant mathematical and computational challenges for classical parameter estimation methods, including strong nonlinearities, nonconvexity, and sharp gradients. We explore how NNs overcome these challenges by approximating reconstruction maps for parameter estimation from observational data. The considered data are time series of the spiking membrane potential of a biological neuron. We infer parameters controlling the dynamics of the model, noise parameters of autocorrelated additive noise and noise modelled via stochastic differential equations, as well as the covariance matrix of the posterior distribution to expose parameter uncertainties—all with just one forward evaluation of an appropriate NN. We report results for different NN architectures and study the influence of noise on prediction accuracy. We also report timing results for training NNs on dedicated hardware. Our results demonstrate that NNs are a versatile tool to estimate parameters of the dynamical system, stochastic processes, as well as uncertainties as they propagate though the governing ODE.

Key words. Neural Networks, Deep Learning for Inverse Problems, Bayesian Inverse Problems, FitzHugh–Nagumo, Nonlinear ODEs

MSC codes. 68T07, 62F15, 34A55

1. Introduction.

We develop methodology for and test the use of neural networks (**NN**s) for the estimation of *(i)* hidden parameters of physical models, *(ii)* hidden parameters of stochastic processes, and *(iii)* model and data uncertainties from noisy observations. The present work extends our contributions described in [111] in categories *(i)* and *(ii)*; and it introduces *(iii)* (see Subsection 1.2 for specific details). We assume that the observational data can be explained through the solution of a dynamical system. The inverse problem to estimate parameters is typically formulated as a variational problem governed by the underlying system [120, 125, 83, 84, 82]. In a deterministic setting, we seek point estimates for the hidden model parameters and compute these with numerical optimization [93]. In a statistical setting, we seek samples from a posterior distribution [119, 24, 123, 70], which in practice is often a probability density function of the hidden model parameters conditioned on the data.

The considered inverse problem is governed by the FitzHugh–Nagumo model [51, 90, 28], a nonlinear system of ordinary differential equations (**ODE**s). It simulates electrical voltage spikes in biological neurons that are generated in response to current stimuli. The FitzHugh–Nagumo model is a simplification of neural dynamics with a system of two ODEs; while more complex dynamical systems utilize Hodgkin–Huxley-type models based on [66]. We consider an inverse parameter estimation problem that at first may appear computationally simple to solve, but it poses significant mathematical and computational challenges. Key *mathematical challenges* associated with the inverse parameter estimation problem are *(i)* nonlinearity of the forward model, *(ii)* a multiplicative coupling and strong nonlinear dependencies between model parameters, *(iii)* strong nonlinearities in the inverse problem, *(iv)* nonconvexity of the data mismatch term with sharp gradients [107], *(v)* and weak a priori information on adequate model parameter values [61, 104]. These challenges render the inverse problem as a global optimization problem with multiple local minima [130]. Therefore, most classical methods for the solution of this inverse problem become ineffective and motivate our work. In the present work, we explore if the use of NNs to learn the reconstruction map for hidden parameters given noisy observations allows us to reliably generate parameter predictions under the stated mathematical challenges. We illustrate the data associated with the considered parameter estimation problem in Figure 1. We show a representative optimization landscape in Figure 2 to highlight some of the

* **Funding:** This work was partly supported by the National Science Foundation (**NSF**) under the awards DMS-2012825 and DMS-2145845. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of NSF. This work was completed in part with resources provided by the Research Computing Data Core at the University of Houston. Access to the Neocortex Supercomputer at the Pittsburgh Supercomputing Center was awarded through the Neocortex Spring 2023 Call.

†Department of Mathematics, University of Houston, Houston, TX, USA (andreas@math.uh.edu).

‡Department of Mathematics, Virginia Tech, Blacksburg, VA, USA (jrudi@vt.edu).

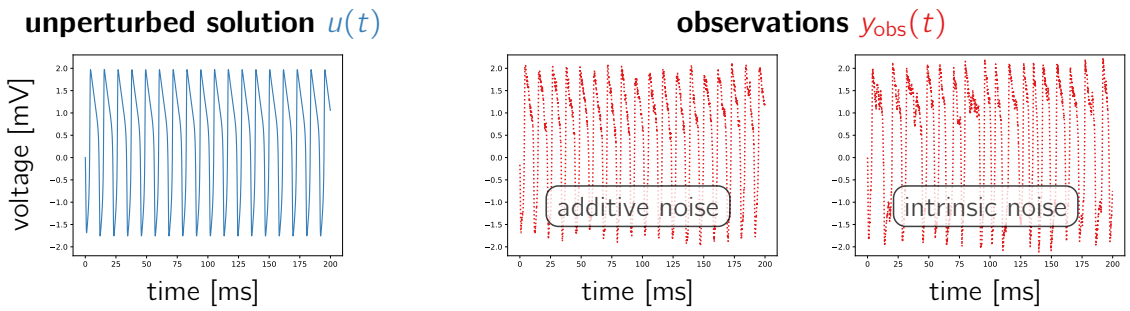

FIG. 1. *Illustration of the model dynamics and noise perturbations. We show time series of the membrane potential u for $\tau = 200\,ms$. Left: Solution of the FitzHugh–Nagumo model (membrane potential u) for representative model parameters $\boldsymbol{\theta}_{dyn} \in \mathbb{R}^3$. Right: Plots of the observational data y_{obs} for an additive and an intrinsic noise model. We limit the observational data associated with the FitzHugh–Nagumo model to the membrane potential u. Our goal is to learn a map, g_Ξ, from the observational data y_{obs} to the hidden parameters, $\boldsymbol{\theta}_{pred} = g(y_{obs})$, such that $\boldsymbol{\theta}_{pred} \approx \boldsymbol{\theta}_{true}$.*

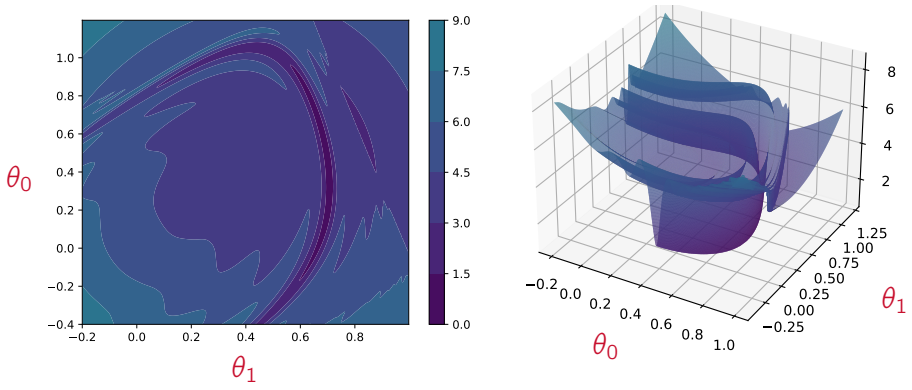

FIG. 2. *Illustration of the parameter estimation problem. We show a contour plot and the associated surface plot of the objective that enters our variational optimization problem as a function of the hidden parameters θ_0 and θ_1 of $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ that control the model output. We can observe that the optimization problem is nonconvex with a volatile change in objective values that features a narrow, flat optimality zone (dark blue region in plot on the left) with sharp gradients (surface plot on the right).*

mathematical challenges. We visualize a representative posterior distribution in Figure 3.

We note that the low-dimensional character (only three scalar parameters control the state of the dynamical system) make for an excellent testbed to explore algorithms without facing the challenge of a computationally intractable forward operator.

Estimating the parameters of the dynamical system, $\boldsymbol{\theta}_{\text{dyn}}$, is challenging on its own. It becomes even more challenging when the goal is to also infer parameters of a noise model, $\boldsymbol{\theta}_{\text{noise}}$, from noise that pollutes the observed state of the dynamical system. Estimating both simultaneously is rarely attempted but important for inverse problems on noisy measurements. In addition, we provide methodology to expose how uncertainties propagate throughout the dynamical system. These uncertainties can arise from various sources. One source of error is the discrepancy between the mathematical model subject to inference and the "true" physical system and the choice of the dynamical system class ("model form uncertainty"). These discrepancies alongside inaccuracies in specifying initial or boundary conditions of the dynamical system can introduce significant modeling bias. Measurement errors, stemming from sensor noise, finite resolution of measuring equipment, or sampling errors, are another source of error. Furthermore, numerical discretization and limited computational precision contribute to the uncertainty. There is also uncertainty in the parameters that control the dynamical system due to a lack of information or intrinsic variability. Uncertainty quantification offers a rigorous mathematical framework for exposing these errors. For effective uncertainty quantification in inverse problems governed by dynamical systems, it is essential to employ a careful, often

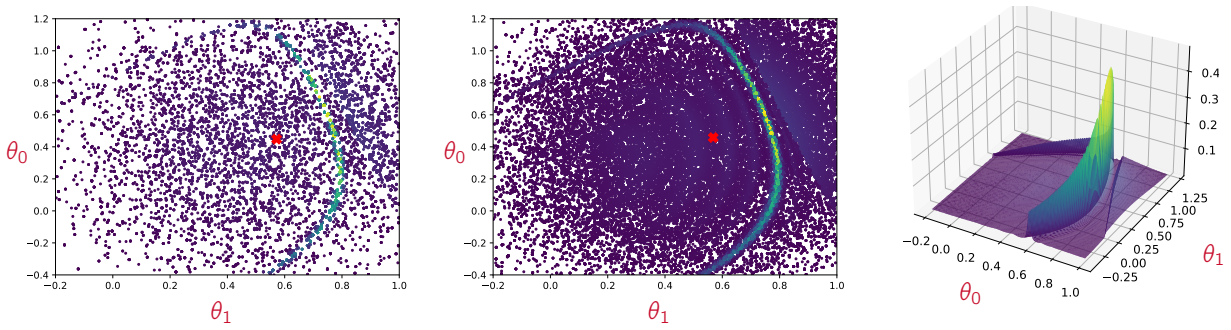

FIG. 3. *Illustration of the posterior distribution. We show samples drawn from the posterior distribution as a function of the hidden parameters θ_0 and θ_1 of $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ that control the model output. We can observe that the problem is nonconvex with a volatile change in posterior values that features a narrow optimality zone (bright region) with sharp transitions. We note that the plot in Figure 2 (the objective function) corresponds to the negative log posterior of the posterior distribution. The samples in the visualization on the left and middle are generated using a Metropolis Hastings Markov chain Monte Carlo sampling algorithm (left: 20 000 samples; middle: 100 000 samples).*

hierarchical approach to treating sources of error. Each source affects the overall uncertainty in the inferred parameters or states differently. Modern methods for inference go beyond parameter estimation and strive to quantify and manage these uncertainties, providing robust and credible inferences. Numerous approaches exist to address individual sources of error, including modeling unknowns as random variables, model averaging, and deriving a posteriori error estimates. Our work specifically aims to provide parameter estimates that control error perturbations in the measured signals, while also exposing uncertainties by estimating the covariance matrix associated with the posterior distribution.

1.1. Outline of the Method. Our work aims at designing a NN surrogate model that learns the mapping from noise observations to various quantities of interest. Let $\boldsymbol{\theta}_{\text{dyn}} \in \Theta_{\text{dyn}}$ denote the parameters of a considered dynamical system (i.e., the parameters of interest), where Θ_{dyn} is the set of feasible parameters. Let $f : \Theta_{\text{dyn}} \to \mathcal{Y}$ denote the *parameter-to-observation map*, where $\mathcal{Y}$ is the set of all realizable observations. The parameter-to-observation map is defined as the composition $f = o \circ e$. The two maps comprising f are $e : \Theta_{\text{dyn}} \to \mathcal{U}$ that maps parameters to realizable states, $\mathcal{U}$, and $o : \mathcal{U} \to \mathcal{Y}$ that restricts and/or transforms states to what can be observed. We denote $e(\boldsymbol{\theta}_{\text{dyn}})$ as the *forward map* (i.e., the solution operator of a dynamical system), $u = e(\boldsymbol{\theta}_{\text{dyn}}) \in \mathcal{U}$ as the state variable, $o(u)$ as the *observation operator*, and $y_{\text{obs}} \in \mathcal{Y}$ as observed data. We relate the parameters of the dynamical system $\boldsymbol{\theta}_{\text{dyn}}$ to the observed data $y_{\text{obs}} \in \mathcal{Y}$ by

$$y_{\text{obs}} = f(\boldsymbol{\theta}_{\text{dyn}}) + \eta = o \circ e(\boldsymbol{\theta}_{\text{dyn}}) + \eta = o(u) + \eta,$$

where η represents additive observation noise. In the *inverse parameter-estimation problem* we seek the (hidden) parameters $\boldsymbol{\theta}_{\text{dyn}}$ of the dynamical system that best explain y_{obs}, i.e., we try to find predictions $u_{\text{pred}} = e(\boldsymbol{\theta}_{\text{pred}})$ so that $f(\boldsymbol{\theta}_{\text{pred}}) \approx y_{\text{obs}}$. Informally, we can cast this problem as identifying the inverse map $g_{\text{dyn}} : \mathcal{Y} \to \Theta_{\text{dyn}}$,

$$\boldsymbol{\theta}_{\text{pred}} = g_{\text{dyn}}(y_{\text{obs}}) = f^{-1}(y_{\text{obs}}) = (o \circ e)^{-1}(y_{\text{obs}}),$$

that relates measurements y_{obs} (NN input) to parameters $\boldsymbol{\theta}_{\text{pred}}$ (NN output). We note that this mapping is not well-defined due to the ill-posedness of the inverse problem.

In the present work, we first propose and test an end-to-end machine learning framework to generate a NN approximation to g_{dyn}. That is, we learn a NN surrogate model for the *reconstruction map*. Second, we extend the NN surrogate for g_{dyn} to a reconstruction map that maps from $\mathcal{Y}$ to Θ, where Θ does not only comprise the parameters $\boldsymbol{\theta}_{\text{dyn}}$ of the dynamical system, but it also includes the parameters $\boldsymbol{\theta}_{\text{noise}}$ of different noise models and matrices $\boldsymbol{\Gamma}_{\text{post}}$. The matrices $\boldsymbol{\Gamma}_{\text{post}} \succ 0$ are positive semi-definite covariance matrices that form a Laplace approximation of the (generally non-Gaussian) posterior at the maximum a-posteriori (**MAP**) point; they are important to expose uncertainties in our Bayesian inverse problem. In summary, our target is computationally learning the NN surrogate $g_\Xi : \mathcal{Y} \to \Theta$ by optimizing the NN parameters Ξ. As

such, the dimension of the NN prediction/output space Θ significantly increases compared to the dimension of the space Θ_{dyn}.

To generate the data for training g_Ξ, we use forward simulations for samples $\boldsymbol{\theta}^i_{\text{dyn}}$, $i = 1, \dots, n_{\text{train}}$, and $\boldsymbol{\theta}^i_{\text{noise}}$ to generate y^i_{obs} and $\boldsymbol{\Gamma}^i_{\text{post}}$. Subsequently, we train a NN to learn the mapping from the NN input y_{obs} to the NN output $\boldsymbol{\theta}_{\text{pred}}$ comprised of $\boldsymbol{\theta}_{\text{dyn}}$, $\boldsymbol{\theta}_{\text{noise}}$, and $\boldsymbol{\Gamma}_{\text{post}}$.

We note that our framework does not require to draw samples from the posterior distribution to construct the training data for $\boldsymbol{\Gamma}_{\text{post}}$. We approximate $\boldsymbol{\Gamma}_{\text{post}}$ based on the inverse of the Hessian $\boldsymbol{H}$ associated with the negative log-posterior at known (true) model parameters $\boldsymbol{\theta}^i_{\text{dyn}}$. We use adjoints to approximate the Hessians [22, 103, 85, 119] resulting in a scalable method that has a fixed cost independent of the dimensions of the parameter and state spaces, Θ_{dyn} and $\mathcal{U}$, respectively.

1.2. Contributions. The considered inverse problem *poses significant computational and mathematical challenges* that render traditional methods for the solution of inverse problems inefficient. Our work extends [111] in several ways. In [111], a NN was used to infer two parameters of the dynamical system (namely, θ_0 and θ_1 in (2.1)) as well as noise parameters of an autocorrelated additive noise model. Here, we consider a second noise model based on a stochastic differential equation (**SDE**) [76, 113, 19], estimate one additional parameter of the dynamical system, and provide methodology for uncertainty quantification. In addition, we explore the performance of the proposed approach on dedicated hardware. In particular, our contributions are:

- We computationally learn an observation-to-parameter map $g_\Xi : \mathcal{Y} \to \Theta$ that is able to simultaneously estimate the parameters of the dynamical system $\boldsymbol{\theta}_{\text{dyn}} \in \mathbb{R}^3$, noise parameters $\boldsymbol{\theta}_{\text{noise}} \in \mathbb{R}^m$, $m \in \{1, 2, 3\}$, as well as the covariance matrix $\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$ of the posterior distribution from noisy observations. In [111] only two model and noise parameters were considered.
- We consider the estimation of model parameters and noise parameters for different noise models—in particular, autocorrelated additive noise and noise modelled via SDEs, and a combination thereof. We refer to the SDE noise model as "intrinsic noise." In [111] only additive noise was considered.
- We develop a computational framework for training and estimating covariance matrices of the posterior distribution of the model parameters (along with model and noise parameters) to expose uncertainties in the model and data. This framework is absent from [111].
- We consider a log-Euclidean framework to learn $\boldsymbol{\Gamma}_{\text{post}}$.
- We investigate different NN architectures for the map g_Ξ.
- We provide a detailed performance analysis of the proposed methodology. In particular, we quantify prediction accuracy as a function of varying network architecture, NN training parameters, input data, and noise perturbations.
- We report results for computational performance on dedicated hardware, in particular, the Neocortex System at the Pittsburgh Supercomputing Center, a testbed supercomputer consisting of the Cerebras CS-2 AI accelerator hardware.

1.3. Limitations. The limitations of our work include:

- The observation-to-parameter map g is not well-defined due to the ill-posedness of the inverse problem. In the absence of noise, it is reasonable to assume that the unpolluted observation lies in the range of f. However, since only polluted data is available this observation may not hold. In addition, the map f will have a non-trivial kernel; multiple instances of $\boldsymbol{\theta}_{\text{dyn}}$ will correspond to indistinguishable data y_{obs}. As such, we can interpret g_Ξ as a surrogate for a pseudo-inverse $f^\dagger$ of f.
- We consider an "end-to-end" learning approach [95, 78, 111]; we learn a direct map from observables $y_{\text{obs}} \in \mathcal{Y}$ to parameters $\boldsymbol{\theta} \in \Theta$. Therefore, our predictions (of, e.g., $\boldsymbol{\theta}_{\text{dyn}}$) are no longer informed by a likelihood [35]. However, experimental evidence (see supplementary material Subsection B.3.4) seems to suggest that the predictions are for the most part good agreement with the data, especially when an additive noise model is considered. We also do not require imposing a metric on $\mathcal{Y}$, which turns out to be a main source of concern in neural dynamics [104].
- We use an explicit time integration method for evaluating the forward and adjoint operators. Exploring more sophisticated methods (for example, adaptive time stepping; higher accuracy; etc.) remains subject to future work.
- The amount of samples for training NNs can increase significantly with the dimension of the parameter space provided sufficiently uninformative priors; and each sample incurs the cost of solving the ODE

model. If the dimension of the parameter space becomes "too large," the cost of data generation can become prohibitive.

1.4. Related Work. In [111], we estimated a subset of the parameters of the dynamical system (namely, θ_0 and θ_1 in (2.1)) and the two parameters of an autocorrelated, additive noise model. Here, we generalize our approach to include a noise model based on an SDE and we estimate the parameters of the SDE. The SDE introduces stochastic dynamics to the deterministic ODE; and inference for SDE parameters is significantly more challenging than inference for parameters of an additive noise model. Moreover, we extend our framework to enable the prediction of covariance matrices associated with the posterior distribution of the model parameters to expose model and data uncertainties. Overall, we estimate up to 12 parameters associated with our problem. While the parameters of the dynamical system can be estimated based on global (e.g., stochastic) optimization methods, the overall problem tackled in this work is more complex; it also involves estimating parameters for the noise model and entries of the covariance matrix associated with the posterior distribution of the parameters of the dynamical system conditioned on the data. The simultaneous estimation of all of those quantities of interest—ODE parameters, SDE parameters, stochastic noise parameters, and posterior covariance—has not been demonstrated to our knowledge. Modeling neural activity has a rich mathematical history [31]. So does parameter estimation for models of neural dynamics [126]. In the following, we highlight work that we view most pertinent to ours. The dynamical system that governs the parameter estimation problem is the FitzHugh–Nagumo model [51, 90]. We describe this model in greater detail in Section 2. It represents a common simplification of the Hodgkin–Huxley model to a nonlinear, coupled system of two ODEs. Traditional approaches for estimating parameters for these types of models are based on heuristics, follow trial and error strategies, and/or global optimization approaches. Examples for such approaches include simulated annealing, differential evolution, genetic algorithms, particle-swarm optimization, and brute-force grid searches [21, 20, 126, 1, 62, 45, 60]. While some of these methods are, in general, equipped with guarantees to converge to a global minimizer, their rate of convergence is typically poor. Conversely, methods that exploit local gradient information (e.g., see [44, 124, 86, 107]) suffer from strong nonlinearities and the nonconvexity of the problem.

An additional shortcoming of the deterministic methods mentioned above is that they only provide point estimates for the parameter values but no information about uncertainties or error bounds for the inferred parameters. Work on uncertainty quantification for computational models in neuroscience can be found in [122, 130]. The work in [122] uses polynomial chaos expansions instead of more traditional Markov chain Monte Carlo (**MCMC**) sampling approaches (e.g., considered in [130, 73]). Other probabilistic approaches that have been used include techniques such as state-space modeling, Kalman filters and variants thereof [67, 48, 25, 127].

Generally, the solution of inverse problems using deep NNs has recently gained attraction [2, 77, 95, 89, 54]. For a recent overview, see [12].

Our proposed approach falls into the broad class of likelihood-free inference (**LFI**) or simulation-based inference (**SBI**) [37, 79, 13] methods in the sense that we learn a surrogate model that maps from observations to hidden parameters and summary statistics without explicitly evaluating, manipulating, and/or forming a posterior or likelihood. This class of algorithms has been developed for scenarios in which the likelihood is intractable or unavailable but it is possible to generate data from the model (i.e., evaluate the forward map $e(\boldsymbol{\theta}_{\text{dyn}})$). It subsumes a large class of approaches. In our framework, we directly learn a surrogate map from observations to quantities of interest. This is different from many existing LFI/SBI algorithms, where expensive likelihood evaluations are avoided and informative simulations are selected to increase sampling efficiency [38, 116, 117]. Moreover, there are primarily three NN-based approaches to SBI: Neural Posterior Estimation (NPE) [97, 58], Neural Likelihood Estimation (NLE) [99, 55], and Neural Ratio Estimation (NRE) [63, 46]. These approaches are conceptually different from ours. We outline the differences to the potentially closest approach of the three, which is amortized NPE. In amortized NPE, a NN is trained once to learn the mapping from observations to a posterior. The NN employed by NPE typically is a normalizing flow [109]. The differences between our approach and NPE are that, first, we are free to choose any data misfit function, which takes the role of the negative-log likelihood function in the posterior. Second, we approximate posteriors with Gaussian distributions with the benefit of a reduction in computational cost, because we obtain the covariance of the Gaussian using the inverse Hessian matrix and adjoint methods; while NPE allows for non-Gaussian posteriors. Third, we train feed-forward-type NN that have a lower

computational cost in practice compared with normalizing flows. The aforementioned NN-based approaches to SBI are primarily either concerned with inference from output of a deterministic model (with some amount of added noise) or from output of a stochastic model; representative benchmark problems can be found in [79]. We go beyond this deterministic/stochastic separation, because we target the simultaneous inference of both ODE/SDE and noise parameters with just one single NN. The noise models are time-correlated and not merely scaled Gaussian white noise. The levels of noise vary across a range of parameters that go into stochastic models; therefore, our treatment of noise presents additional challenges that are not considered in the literature.

Machine learning methods for inverse problems governed by dynamical systems are discussed in, for example [106, 36, 96, 92, 81]. In the broader context of inverse problems, end-to-end approaches that directly learn the reconstruction map $g_\Xi : \mathcal{Y} \to \Theta$—as done in our work—are presented in [110, 2, 69, 131, 7, 32, 75, 112]. Instead of learning the reconstruction map, one can also learn the surrogates for the solution operators for the dynamical systems to speed up the optimization [81, 129]. Alternatively, one can use NNs to compute the gradient for updating the hidden parameters [29]. Other approaches to learn the reconstruction map use two autoencoders—one for representing the space $\mathcal{Y}$ and one for representing the space Θ. Works that consider this type of pairing of autoencoders are [35, 50, 134]. Another strategy is to learn a model representation without considering data using diffusion networks [26], generative adversarial networks [115], or variational autoencoders [56].

Classical (feed-forward) NNs do not allow to generate predictions of uncertainties; they only provide point estimates. Thus using such a NN for parameter prediction results in a point estimate, which is similar to solving an inverse problem with traditional optimization methods. In the present work, we use feed-forward NNs and consequently are only able to generate point estimates. However, we propose to learn the mapping from observational data to covariance matrices approximating the posterior distribution. As an alternative, generative NN architectures map samples (e.g., from a standard normal distribution) to samples from an unknown target distribution (e.g., a posterior). These NNs have been used to approximate the underlying likelihood density or posterior distribution. In practice, the computational cost for training the weights of generative NNs is significantly higher than for classical deep NNs. Hence, the present work proposes a computationally less demanding approach for prediction of parameter uncertainties.

The aforementioned generative NNs for approximating the underlying likelihood density or posterior distribution from samples are normalizing flow architectures [109, 97, 80, 100, 40], which can be applied to neural dynamics [57], generative adversarial networks [101, 108], and score-based diffusion NNs [14, 42]. Alternatively, one can learn the forward operator to, for instance, speed up the exploration of the posterior distribution [8].

Machine learning has not only been used to learn the solution of the inverse problem given some observational data but has also been successfully used to estimate hyperparameters that appear in the formulation of inverse problems [72, 23, 4] or learn/improve regularization operators [36, 5, 18, 33, 94, 88, 114, 77, 6]. Theoretical results associated with the use of machine learning for inference in inverse problems can be found in [105, 27]. Alternatively, machine learning has been used to accelerate optimization [30] and accelerate MCMC [52].

1.5. Outline. The paper is organized as follows. In Section 2 we introduce the methodology. The general problem formulation is discussed in Subsection 2.1. This includes the variational problem formulation in the deterministic setting (see Subsection 2.1.2) and the formulation of the inverse problem in a Bayesian setting (see Subsection 2.1.3). This allows us to derive problem specific information used in our NN framework, explore analogies between the NN approach and classical approaches, and motivate our setup of the training for the NN architectures. We present the considered NN architectures in Subsection 2.2. This includes dense NN (**DNN**) architectures (see Subsection 2.2.1) and convolutional NNs (**CNN**s; see Subsection 2.2.2). A description of the training and testing data can be found in Subsection 2.3. We present the measures to assess the performance of the proposed approach in Subsection 2.4. The results are reported in Section 3. We conclude with Section 4.

We accompany the manuscript with supplementary materials. These include the derivation of the Hessian (see Subsection B.1) and additional numerical results that are either complementary to those reported in this study, provide additional experimental evidence for some of the claims we make, or explore variations of our framework (see Subsection B.3).

2. Methods and Material. This section presents (*i*) classical variational and Bayesian problem formulations to provide a theoretical underpinning for the choices and strategies we consider in our NN approach, (*ii*) the proposed and examined NN architectures for inferring the parameters of the dynamical system, noise parameters and entries of the covariance matrix of the posterior distribution, (*iii*) the training and testing data for the NN, and (*iv*) performance measures to assess the NN predictions. We summarize the notation in Table 1.

2.1. Classical Problem Formulation. Next, we introduce the formulation of the inverse problem for estimating the parameters of the dynamical system in (2.1) from time series data y_{obs} in a variational setting [125, 120]. Subsequently, we develop the associated Bayesian formulation of the inverse problem [123, 119, 24]. We include this information for several reasons: First, we use it to illustrate mathematical challenges. Second, we use analogies to the statistical problem formulation to interpret the results obtained by the NNs. Third, we use curvature information of the variational problem formulation to generate training data for the estimation of the posterior covariances $\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$, with

$$\text{Sym}^+(n) := \left\{ \boldsymbol{M} \in \mathbb{R}^{n,n} : \boldsymbol{x}^\mathsf{T} \boldsymbol{M} \boldsymbol{x} > 0 \text{ for all } \boldsymbol{x} \in \mathbb{R}^n \setminus \{\boldsymbol{0}\} \right\}.$$

2.1.1. Forward Problem. The forward problem is given by the nonlinear system of ODEs

$$\mathrm{d}_t u - \theta_2(u - (u^3/3) + v + z) = 0 \qquad \text{in } (0, \tau], \tag{2.1a}$$

$$\mathrm{d}_t v + ((u - \theta_0 + \theta_1 v)/\theta_2) = 0 \qquad \text{in } (0, \tau], \tag{2.1b}$$

with model parameters $\boldsymbol{\theta}_{\text{dyn}} := (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$, initial conditions $u(t = 0) = u_0$, $v(t = 0) = v_0$, and state variables $u \in \mathcal{U} \subset \{w : [0, \tau] \to \mathbb{R}\}$, $v \in \mathcal{V} \subset \{w : [0, \tau] \to \mathbb{R}\}$, respectively, defined on a given time horizon $[0, \tau] \subset \mathbb{R}$, $\tau > 0$. The state variable u is related to the *membrane potential* across an axon membrane, v represents the *recovery variable* summarizing outward currents, and z is the *total membrane potential* and corresponds to a stimulus applied to the neuron. We assume that the stimulus z is a known (fixed) constant. We show representative membrane potentials u as a function of $\boldsymbol{\theta}_{\text{dyn}}$ in the supplementary materials (see Figure 10 in Subsection B.3.4).

This model is a phenomenological simplification of more complex neural models to capture qualitative dynamics. As such, the model parameters do not really have a physiological meaning. The parameters θ_0 and θ_1 dominate the dynamics. The parameter θ_0 denotes the excitability threshold; it determines the threshold above which the system enters an excited state, influencing spiking behavior. The parameter θ_1 represents the recovery scaling; it controls how the recovery variable responds to voltage changes. The parameter θ_2 is a time scale factor; it controls the time scale separation between fast and slow dynamics.

2.1.2. Variational Problem Formulation. We overview traditional, variational approaches to estimate the model parameters $\boldsymbol{\theta}_{\text{dyn}} \in \mathbb{R}^3$ that enter (2.1) given noisy observations $y_{\text{obs}} \in \mathcal{Y}$. The problem formulation is given by

$$\begin{aligned} &\underset{(u,v)\in\mathcal{U}\times\mathcal{V},\, \boldsymbol{\theta}_{\text{dyn}}\in\Theta}{\text{minimize}} && \frac{\gamma}{2}\int_0^\tau (u(t) - y_{\text{obs}}(t))^2 \,\mathrm{d}t + \frac{1}{2}\|\boldsymbol{\theta}_{\text{dyn}} - \boldsymbol{\theta}_{\text{ref}}\|_{\boldsymbol{L}}^2 \\ &\text{subject to} && (u, v) \text{ solve (2.1)}, \end{aligned} \tag{2.2}$$

The first part of the objective function in (2.2) measures the proximity between u and y_{obs}, where $\gamma > 0$ controls its contribution to the objective. We assume that y_{obs} is available at all time points $t \in [0, \tau]$. If we assume partial observations $y_{\text{obs}} \in \mathbb{R}^{n_{\text{obs}}}$, $n_{\text{obs}} \in \mathbb{N}$, a distance function that is in accordance with (2.2) can, e.g., be chosen as $(\gamma/2)\|o(u) - y_{\text{obs}}\|_2^2$, with observation operator $o : \mathcal{U} \to \mathbb{R}^{n_{\text{obs}}}$. We select y_{obs} to correspond to the membrane potential u perturbed by noise: $y_{\text{obs}}(t) = u_{\boldsymbol{\theta}}(t)$, $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\text{dyn}}, \boldsymbol{\theta}_{\text{noise}})$, for all $t \in [0, \tau]$, where $\boldsymbol{\theta}_{\text{noise}} \in \mathbb{R}^m$, $m \in \{1, 2, 3\}$, controls the amount of noise. We describe the considered noise models in Subsection 2.3.2. The second part of the objective function is a Tikhonov-type regularization model that is introduced to stabilize the solution process [47], where $\boldsymbol{L} \in \text{Sym}^+(3)$ is a regularization operator that is specified below and $\boldsymbol{\theta}_{\text{ref}} \in \mathbb{R}^3$ is a reference vector added to control the kernel of the regularization.

The reduced form of the optimization problem in (2.2) is given by

$$\underset{\boldsymbol{\theta}_{\text{dyn}}\in\Theta_{\text{ad}}}{\text{minimize}} \quad \frac{\gamma}{2}\int_0^\tau (f(\boldsymbol{\theta}_{\text{dyn}}) - y_{\text{obs}}(t))^2 \,\mathrm{d}t + \frac{1}{2}\|\boldsymbol{\theta}_{\text{dyn}} - \boldsymbol{\theta}_{\text{ref}}\|_{\boldsymbol{L}}^2. \tag{2.3}$$

TABLE 1
Common notation, symbols, and acronyms.

symbol/acronym	**meaning**
$[0,\tau] \subset \mathbb{R}$	time horizon
$\mathcal{U} \times \mathcal{V}$	state space
Θ_{dyn}	control space
$\mathcal{Y}$	observation space
Θ	quantities of interest (NN output space)
$\text{Sym}^+(n)$	space of real $n \times n$ symmetric positive definite matrices
$\text{Sym}(n)$	vector space of real $n \times n$ symmetric matrices
$u : [0,\tau] \to \mathbb{R}$	membrane potential (state variable)
$v : [0,\tau] \to \mathbb{R}$	recovery variable (state variable)
$z \in \mathbb{R}$	stimulus
$\lambda : [0,\tau] \to \mathbb{R}$	dual variable associated with state equation for u
$\nu : [0,\tau] \to \mathbb{R}$	dual variable associated with state equation for v
$\boldsymbol{\theta}_{\text{dyn}} := (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$	model parameters (dynamical system)
$\boldsymbol{\theta}_{\text{noise}} := (\delta, \sigma, \beta) \in \mathbb{R}^3$	noise parameters
$(\delta, \sigma) \in \mathbb{R}^2$	noise parameters of additive noise model
$\beta \in \mathbb{R}$	noise parameters of intrinsic noise model
$p \in \mathbb{N}$	number of unknowns; $p \in \{4, 5, 6, 10, 11, 12\}$
$f : \Theta_{\text{dyn}} \to \mathcal{Y}$	parameter-to-observation map
$e : \Theta_{\text{dyn}} \to \mathcal{U}$	forward map
$o : \mathcal{U} \to \mathcal{Y}$	observation operator
$g_{\text{dyn}} : \mathcal{Y} \to \Theta_{\text{dyn}}$	reconstruction map; "pseudo-inverse" of f
$g_\Xi : \mathcal{Y} \to \Theta$	NN surrogate parametrized by Ξ
$y_{\text{obs}} \in \mathcal{Y}$	observational data (NN input)
$\boldsymbol{y}_{\text{obs}} = (y_{\text{obs},1}, \dots, y_{\text{obs},n_t}) \in \mathbb{R}^{n_t}$	discrete observational data
$\boldsymbol{\theta}_{\text{pred}} \in \mathbb{R}^p$	NN prediction (NN output)
$\boldsymbol{\Gamma}_{\text{prior}} = \text{diag}(\boldsymbol{\sigma}_{\text{prior}}) \in \mathbb{R}^{3,3}$	prior covariance
$\bar{\boldsymbol{\theta}}_{\text{prior}} \in \mathbb{R}^3$	prior mean; $\sigma_{\text{prior}} \in \mathbb{R}^3$
$\boldsymbol{L} = \boldsymbol{\Gamma}_{\text{prior}}^{-1} \in \mathbb{R}^{3,3}$	regularization operator
$\boldsymbol{\Gamma}_{\text{noise}} = \sigma_{\text{noise}}^2 \tau^2 \boldsymbol{I} \in \mathbb{R}^{3,3}$	noise covariance; $\sigma_{\text{noise}} > 0$, $\tau > 0$, $\boldsymbol{I} = \text{diag}(1,1,1) \in \mathbb{R}^{3,3}$
$\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$	posterior covariance with entires Γ_{ij}, $i = 0, 1, 2$
$\boldsymbol{H} \in \text{Sym}^+(3)$	Hessian matrix
$n_t \in \mathbb{N}$	number of time steps (numerical time integration)
$n_{\text{layers}} \in \mathbb{N}$	number of NN layers
$n_{\text{train}} \in \mathbb{N}$	number of training data
$n_{\text{test}} \in \mathbb{N}$	number of testing data
$n_{\text{feat}} \in \mathbb{N}$	number of features
$n_f \in \mathbb{N}$	filter count (CNN)
$n_u \in \mathbb{N}$	number of units (DNN)
CDET	Coefficient of Determination; see (2.19)
(C)MSE	(Centered)Mean Squared Error; see (2.17)
(C/D)NN	(Convolutional/Dense) Neural Network
MAP	Maximum-A-Posteriori (point)
MdAPE	Median of Absolute Percentage Error; see (2.18)
MCMC	Markov Chain Monte Carlo (method)
ODE	Ordinary Differential Equation
SDE	Stochastic Differential Equation
SQB	Squared Bias (error); see (2.16)
TS	time series features (NN input)
FC	Fourier coefficient features (NN input)
TS&FC	time series and Fourier coefficient features (NN input)

In (2.3) we eliminated the constraint from the problem by replacing the model prediction with the parameter-to-observation map $f : \Theta \to \mathcal{Y}$. This formulation establish a connection to the problem formulation in a statistical setting presented next.

2.1.3. Bayesian Problem Formulation. We refer to [123, 119, 24] for an introduction to Bayesian inverse problems. With slight abuse of notation, we assume that the problem has been discretized. Consequently, the observations $y_{\text{obs}} \in \mathcal{Y}$ are given by $\boldsymbol{y}_{\text{obs}} = (y_{\text{obs},1}, \dots, y_{\text{obs},n_t}) \in \mathbb{R}^{n_t}$, where $n_t \in \mathbb{N}$ corresponds to the number of time steps used to solve the forward problem. We model all unknown variables as random variables.

For the likelihood model we assume that observational uncertainty (i.e., uncertainty in the data related to measurement errors) and the uncertainty in the model (i.e., the uncertainty in $\boldsymbol{f}(\boldsymbol{\theta}_{\text{dyn}})$ due to modeling errors) are each centered, additive, and Gaussian. We integrate both errors into a single noise model. More precisely, we assume $\boldsymbol{y}_{\text{obs}} \sim \mathcal{N}(\boldsymbol{f}(\boldsymbol{\theta}_{\text{dyn}}), \boldsymbol{\Gamma}_{\text{noise}})$, with parameter-to-observation map $\boldsymbol{f} : \mathbb{R}^3 \to \mathbb{R}^{n_t}$ and covariance $\boldsymbol{\Gamma}_{\text{noise}} \in \text{Sym}^+(n_t)$, $\boldsymbol{\Gamma}_{\text{noise}} = \sigma_{\text{noise}}^2 \tau^2 \boldsymbol{I}$, $\boldsymbol{I} = \text{diag}(1, \dots, 1) \in \mathbb{R}^{n_t \times n_t}$, $\sigma_{\text{noise}} > 0$. We obtain

$$\boldsymbol{y}_{\text{obs}} = \boldsymbol{f}(\boldsymbol{\theta}_{\text{dyn}}) + \boldsymbol{\eta} \tag{2.4}$$

with $\boldsymbol{\eta} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Gamma}_{\text{noise}})$. Under the above assumptions, the likelihood probability density function is given by

$$\pi_{\text{like}}(\boldsymbol{y}_{\text{obs}} \,|\, \boldsymbol{\theta}_{\text{dyn}}) \propto \exp\left(-\frac{1}{2}\left\langle \boldsymbol{y}_{\text{obs}} - \boldsymbol{f}(\boldsymbol{\theta}_{\text{dyn}}), \boldsymbol{\Gamma}_{\text{noise}}^{-1}(\boldsymbol{y}_{\text{obs}} - \boldsymbol{f}(\boldsymbol{\theta}_{\text{dyn}}))\right\rangle\right). \tag{2.5}$$

We also have to decide on a prior distribution for $\boldsymbol{\theta}_{\text{dyn}}$. We model each component of $\boldsymbol{\theta}_{\text{dyn}}$ as normal and independent identically distributed random variable; $\boldsymbol{\theta}_{\text{dyn}} \sim \mathcal{N}(\bar{\boldsymbol{\theta}}_{\text{prior}}, \boldsymbol{\Gamma}_{\text{prior}})$, $\bar{\boldsymbol{\theta}}_{\text{prior}} \in \mathbb{R}^3$, $\boldsymbol{\Gamma}_{\text{prior}} = \text{diag}(\boldsymbol{\sigma}_{\text{prior}}) \in \text{Sym}^+(3)$, with standard deviation $\boldsymbol{\sigma}_{\text{prior}} \in \mathbb{R}^3$. The associated probability density function is given by

$$\pi_{\text{prior}}(\boldsymbol{\theta}_{\text{dyn}}) \propto \exp\left(-\frac{1}{2}\left\langle \boldsymbol{\theta}_{\text{dyn}} - \bar{\boldsymbol{\theta}}_{\text{prior}}, \boldsymbol{\Gamma}_{\text{prior}}^{-1}(\boldsymbol{\theta}_{\text{dyn}} - \bar{\boldsymbol{\theta}}_{\text{prior}})\right\rangle\right). \tag{2.6}$$

Remark 2.1. We use the prior distribution to draw samples for the training of the NNs. The selection of the hyper-parameters $\boldsymbol{\Gamma}_{\text{prior}}^{-1}$ and $\bar{\boldsymbol{\theta}}_{\text{prior}}$ of π_{prior} in (2.6) is a challenge in itself. In a Bayesian setting, one can treat these hyperparameters as unknown random variables and estimate them along with the parameters $\boldsymbol{\theta}_{\text{dyn}}$. This leads to so-called *hierarchical methods* (or *hyperprior models*) [70, 123]. Since we know little about the range of the parameters $\boldsymbol{\theta}_{\text{dyn}}$, we use a rather uninformative prior distribution with loose bounds on $\boldsymbol{\sigma}_{\text{prior}}$. More details regarding the parameter selection are provided in Subsection 2.3.3.

We use a Gaussian prior distribution to draw samples for the parameters $\boldsymbol{\theta}_{\text{dyn}}$. Existing work that considers a Gaussian prior for the parameters for the FitzHugh–Nagumo model includes [68]. The work in [113] uses uniform distributions. Since we use the samples drawn from the prior distribution solely to generate NN inputs (i.e., evaluate the forward operator e), our framework is, in general, amendable to any type of prior distribution.

We assume Gaussian distributions when approximating the posterior at each (known) "true solution" $\boldsymbol{\theta}_{\text{dyn}}^i$ (a sample chosen from the prior). The mean of the Gaussian is $\boldsymbol{\theta}_{\text{dyn}}$ and the covariance is computed from the Hessian of the negative log-posterior. This quadratic approximation (or linearization) is adequate, since we evaluate the model during data generation at the "true solution."

According to Bayes theorem, the posterior distribution π_{post} of $\boldsymbol{\theta}_{\text{dyn}}$ conditioned on the data $\boldsymbol{y}_{\text{obs}}$ is given by

$$\pi_{\text{post}}(\boldsymbol{\theta}_{\text{dyn}} \,|\, \boldsymbol{y}_{\text{obs}}) \propto \pi_{\text{like}}(\boldsymbol{y}_{\text{obs}} \,|\, \boldsymbol{\theta}_{\text{dyn}})\ \pi_{\text{prior}}(\boldsymbol{\theta}_{\text{dyn}}). \tag{2.7}$$

The MAP point $\boldsymbol{\theta}_{\text{map}} \in \mathbb{R}^3$ can be found by minimizing the negative log posterior

$$-\log \pi_{\text{post}}(\boldsymbol{\theta}_{\text{dyn}} \,|\, \boldsymbol{y}_{\text{obs}}) = \phi(\boldsymbol{\theta}_{\text{dyn}}) + \text{const},$$

with $\phi : \mathbb{R}^3 \to \mathbb{R}$,

$$\phi(\boldsymbol{\theta}_{\text{dyn}}) = \frac{1}{2}\|\boldsymbol{y}_{\text{obs}} - \boldsymbol{f}(\boldsymbol{\theta}_{\text{dyn}})\|^2_{\boldsymbol{\Gamma}_{\text{noise}}^{-1}} + \frac{1}{2}\|\boldsymbol{\theta}_{\text{dyn}} - \bar{\boldsymbol{\theta}}_{\text{prior}}\|^2_{\boldsymbol{\Gamma}_{\text{prior}}^{-1}}. \tag{2.8}$$

The function ϕ is a discrete version of the objective function in (2.3) with $\gamma = 1/(\sigma^2_{\text{noise}}\tau^2)$, precision matrix $\boldsymbol{L} = \boldsymbol{\Gamma}^{-1}_{\text{prior}}$ and $\boldsymbol{\theta}_{\text{ref}} = \bar{\boldsymbol{\theta}}_{\text{prior}}$. This establishes a direct connection to the deterministic optimization problem in (2.3). We use this connection in Subsection 2.3 to derive an approximation of the covariance matrix $\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$ of the posterior distribution in (2.7).

Remark 2.2. The proposed *NN-based approach* discussed next is not guaranteed to estimate the MAP point $\boldsymbol{\theta}_{\text{map}}$ or any other estimates such as the maximum likelihood or conditional mean.

We also note that we cannot *a priori* guarantee that our NN predictor is unbiased. According to classical ML theory, there exists a bias-variance tradeoff that is controlled by the complexity (or capacity) of the ML model [53]. If the ML model has a strong bias it is not sufficiently complex; we expect to observe poor performance during the training, validation and testing phase (*underfitting*). Conversely, increasing model complexity will increase the variance of the predictor, leading to an improved performance during training but a decrease in performance on the validation and testing data (*overfitting*). Overall, large bias or variance of the ML predictor will lead to a large generalization error. Other sources of generalization error include insufficient or biased training data, a poor validation strategy, or instabilities in the optimization strategy (e.g., poor convergence, bad local minima). Consequently, in the absence of any theoretical guarantees, we have to perform an empirical study to assess the generalization error on validation and testing data; we do so in the experimental part of this manuscript.

We emphasize that there exists recent evidence that modern learning architectures can escape the classical bias-variance trade-off; bias and variance can decrease with increasing network complexity [15, 91, 132, 133].

2.2. Neural Network Architectures. Next, we describe the NN architectures. As we have mentioned in Subsection 1.1, we want to learn the *reconstruction map* $g_\Xi : \mathcal{Y} \to \Theta$. Our approach is supervised. The architectures are in line with our past work [111]. Consequently, we keep their discussion brief.

The inputs (features) $\boldsymbol{y}_{\text{obs}}$ to the NNs include, e.g., time series that correspond to the solution of the ODE in (2.1) perturbed by noise. We specify the training and testing data as well as the noise models considered in this work in Subsection 2.3. The output of the NNs are predictions $\boldsymbol{\theta}_{\text{pred}} \in \mathbb{R}^p$ of hidden parameters for a given noisy observation $\boldsymbol{y}_{\text{obs}} \in \mathbb{R}^{n_t}$. We define the NN-based reconstruction map by

$$\boldsymbol{\theta}_{\text{pred}} = \boldsymbol{y}_n, \quad \boldsymbol{y}_l = \boldsymbol{g}_l(\boldsymbol{y}_{l-1}) \quad \text{for } 1 \le l \le n_{\text{layers}}, \quad \boldsymbol{y}_0 = \boldsymbol{y}_{\text{obs}}, \tag{2.9}$$

where $\boldsymbol{g}_l$ denotes the lth layer of the NN. In this sense, we treat the reconstruction map as a sequence of recursive functions, where each function $\boldsymbol{g}_l$ corresponds to the output of the lth network layer.

We describe the different architectures considered for the layers $\boldsymbol{g}_l$ in (2.9) in Subsection 2.2.1 and Subsection 2.2.2, respectively. We summarize the NN parameters in Table 2. Some of the settings that are shared across architectures are the loss function (mean squared error; **MSE**) and the ADAM optimization algorithm [71]. We consider a learning rate of 0.002 (we explored other choices without noticeable benefit) and a SWISH activation function defined by

$$\boldsymbol{h}_l(\boldsymbol{x}) = \boldsymbol{x} \oslash (\boldsymbol{e}_{n_l} + \mathbf{exp}(-\boldsymbol{x})), \quad \boldsymbol{e}_{n_l} = (1, \dots, 1) \in \mathbb{R}^{n_l}, \tag{2.10}$$

for any $\boldsymbol{x} \in \mathbb{R}^{n_l}$. Here, $\oslash$ denotes the Hadamard division and $\mathbf{exp} : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ is applied element-wise.

2.2.1. Dense Neural Networks. This architecture consists of a sequence of fully connected layers. Each layer l consists of $n_u := n_u(l) \in \mathbb{N}$ units or nodes. Each $\boldsymbol{g}_l$ is modelled as the composition of an affine mapping with a nonlinear activation function: $\boldsymbol{g}_l(\boldsymbol{y}_{l-1}) = \boldsymbol{h}_l(\boldsymbol{W}_l\boldsymbol{y}_{l-1} + \boldsymbol{b}_l)$, where $\boldsymbol{W}_l$ denotes an $n_u(l) \times n_u(l-1)$ matrix of weights, and $\boldsymbol{b}_l$ is the $n_u(l) \times 1$ bias vector for the lth layer. The activation function $\boldsymbol{h}_l : \mathbb{R}^{n_u(l)} \to \mathbb{R}^{n_u(l)}$ is defined in (2.10). We consider 4–24 layers and 4–256 units (see Table 2) to cover a somewhat wide range of network sizes. This leads to a total of 8,079 up to 2,026,243 trainable NN parameters. For experiments in Subsection 3.5, we utilize a DNN composed of 12 layers and 128 units per hidden layer, consisting of a total 695,179 trainable NN parameters.

2.2.2. Convolutional Neural Networks. The second architecture are CNNs. This architecture allows us to exploit temporal locality of the considered time series based on convolutional layers. Each filter is applied to a subinterval of the time series. The filters of one layer are connected to all neighboring layers. This is similar to the units of the DNNs described in Subsection 2.2.1. The weights of these connections

TABLE 2

DNN and CNN configurations. Entries that are sets denote different choices that we use in the exploration of NN architectures.

architecture	option	value(s)
DNN & CNN	loss function	MSE
	optimizer	ADAM [71]
	learning rate	0.002
	batch size	32
	epochs	64
	activation function	Swish/SiLU
DNN	layers	{2,4,8,12,16,20}
	nodes/units n_u	{4,8,16,32,64,128,256}
CNN	layers	{2,3,4,5,6}
	filters setting n_f	{2,4,8,16,32,64,128}
	kernel size	3
	kernel stride	2
	pooling type	average
	pooling size	2
	pooling stride	2
	post-flattening layers	2 dense layers with $n_u = 32$

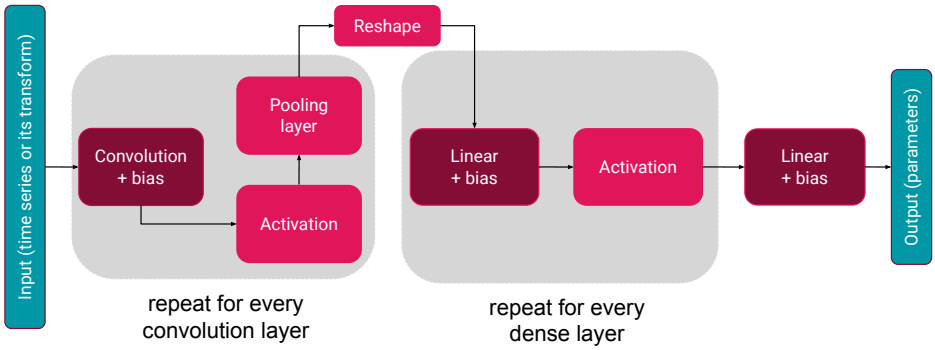

FIG. 4. *Illustration of the CNN architecture. The input is passed into a sequence of convolutional block (each consisting of a convolution, activation, and pooling layers); the output of the last convolutional block is passed to a sequence of dense blocks (each consisting of a dense and activation layers). The last dense layer, without activation, predicts parameters.*

represent the NN parameters to be learned during training. Since the weights are shared across time, significantly less weights have to be estimated. Pooling layers, which are nontrainable, are often used in CNNs for their reduction in spatial dimensionality.

The considered architecture is based on [74]. We interleave convolutional layers with pooling layers to aggregate a small block from a convolutional step into a single value. We determine the number of filters for the lth layer using a multiplier based on the depth (i.e., the layer number). That is, we select $n_f \times (1, 2, 4, 8, \ldots)$ filters. For example, for a CNN with $n_{\text{layers}} = 3$ convolutional layers, we have $n_f \times 1$ filters in the first layer, $n_f \times 2$ filters in the second layer, $n_f \times 4$ filters in the third layer, and so on. Each convolutional layer reduces the size of the output compared with the input by a factor of 1/2 (via a stride equals to two). The reduction by 1/2 also holds for the pooling layers and is complemented by an increase in the filter count n_f by a factor of 2. The output of these layers is passed to a sequence of two dense layers with $n_u = 32$ units. The diagram in Figure 4 illustrates our CNN architecture. Across our experiments, the number of trainable parameters for this setup ranges between 17,095 and 33,662,371. For our main experiments, we utilize a CNN composed of 5 layers with $n_f = 8$, reaching 38,516 trainable NN parameters.

2.3. Training and Testing Data. Next, we describe how to generate the training and testing data. In analogy to the Bayesian problem formulation in Subsection 2.1.3, we generate the NN-outputs (labels) for the training and testing data

$$\{\Theta_{\text{train}}, \Theta_{\text{test}}\} = \left\{\{\boldsymbol{\theta}^i_{\text{train}}\}_{i=1}^{n_{\text{train}}}, \{\boldsymbol{\theta}^i_{\text{test}}\}_{i=1}^{n_{\text{test}}}\right\} \subset \mathbb{R}^p \tag{2.11}$$

by drawing samples of the model parameters $\boldsymbol{\theta}_{\text{dyn}} \in \mathbb{R}^3$ from the prior distribution (2.6). The specific choices for $\boldsymbol{\sigma}_{\text{prior}}$ and $\bar{\boldsymbol{\theta}}_{\text{prior}}$ can be found in Subsection 2.3.3. We discard any samples for $\boldsymbol{\theta}_{\text{dyn}}$ that are outside of

predefined parameter bounds identified in (2.15). In addition, we also include noise parameters $\boldsymbol{\theta}_{\text{noise}}$ and approximations to the posterior covariances $\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$, $\boldsymbol{\Gamma}_{\text{post}} = (\Gamma_{ij})_{i,j=0,1,2}$, associated with $\boldsymbol{\theta}_{\text{dyn}}$ into the training and testing datasets, respectively. Since $\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$, we only consider the six upper triangular entries of $\boldsymbol{\Gamma}_{\text{post}}$ (see Subsection 2.3.1 for more details). Overall, p in (2.11) ranges from 5 to 12, depending on the experiment. To generate the observations associated with the dataset $\{\Theta_{\text{train}}, \Theta_{\text{test}}\}$, we solve the forward problem in (2.1) and perturb the state variables according to the noise models specified in Subsection 2.3.2. We obtain the NN-inputs (features)

$$\{Y_{\text{train}}, Y_{\text{test}}\} = \left\{ \{\boldsymbol{y}^i_{\text{train}}\}_{i=1}^{n_{\text{train}}}, \{\boldsymbol{y}^i_{\text{test}}\}_{i=1}^{n_{\text{test}}} \right\} \subset \mathbb{R}^{n_{\text{feat}}},$$

corresponding to the training in testing dataset, respectively. We generate a total of 15,000 samples. We use fixed subsets (with randomly chosen entries) of these training and testing samples. As we increase the size of training data, we keep the samples we included in the smaller datasets.

If we only consider time series data as features, we have $n_{\text{feat}} = n_t$, where n_t corresponds to the number of time steps used to integrate (2.1) in time; we use an explicit, second-order accurate Runge–Kutta method. The time series are limited to $\tau = 200\,\text{ms}$. The number of time steps is set to $n_t = 2{,}000$, which yields a time step size of $\tau/n_t = 0.1\,\text{ms}$. The identifier for these features is **TS** (for "time series"). Consequently, the examples $\boldsymbol{y}^i_{\text{train}}$ or $\boldsymbol{y}^i_{\text{test}}$ are the time series data associated with the ith sample of the model parameters $\boldsymbol{\theta}_{\text{dyn}}$ in Θ_{train} or Θ_{test}, respectively. We also consider the associated $n_{\text{feat}} = n_t$ Fourier coefficients as features. We denote this feature setting by **FC** (for "Fourier coefficients"). Lastly, we consider a combination of these features, resulting in $n_{\text{feat}} = 2n_t$ features for each training or testing sample, respectively. The identifier for this setting is **TS&FC**.

Likewise to traditional methods for parameter estimation, we expect the performance of the proposed approach to deteriorate as a function of increasing noise. Using NNs for parameter estimation, a question that arises is how the presence of noise in the training data affects performance. To explore this, we consider the following settings: *(i)* training and testing data is noise-free (identifier: $(0 \mid 0)$), *(ii)* training data is noise-free, testing data is noisy (identifier: $(0 \mid 1)$), and *(iii)* training data and testing data are noisy (identifier: $(1 \mid 1)$).

Remark 2.3. This question was already explored in [111]. Like in [111], we observed that including noise in the training data yields better predictions for noisy observations. Although our setup is different from [111], we decided to omit these experiments in the main manuscript and fix the setting to $(1 \mid 1)$. We report results in the supplementary materials that support this decision (see Subsection B.3.2).

Next, we detail how we generate training data for estimating the covariance matrix $\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$ of π_{post} in (2.7). This enables us to expose uncertainties as they propagate through our framework. One approach to construct the dataset $\{\{\boldsymbol{\Gamma}^i_{\text{train}}\}_{i=1}^{n_{\text{train}}}, \{\boldsymbol{\Gamma}^i_{\text{test}}\}_{i=1}^{n_{\text{train}}}\}$ for $\boldsymbol{\Gamma}_{\text{post}}$ is to draw samples from π_{post} in the vicinity of a trial $\boldsymbol{\theta}_{\text{dyn}}$. We tested several MCMC sampling strategies (see [128] for details). In the present case, sampling from π_{prior} is *not* intractable by virtue of the simple forward model (2.1). However, to obtain an accurate estimate for $\boldsymbol{\Gamma}_{\text{post}}$ we still require a large number of samples, and consequently a large number of evaluations of $\boldsymbol{f}$. To avoid this, we adopt ideas from [119, 22, 85] and approximate $\boldsymbol{\Gamma}_{\text{post}}$ based on the inverse of the Hessian $\boldsymbol{H}$ associated with (2.8). We compared this strategy to the results obtained via MCMC sampling. The estimates were in good agreement. Thus, we only report results for the data generated via $\boldsymbol{H}^{-1} \approx \boldsymbol{\Gamma}_{\text{post}}$.

Under the assumption that the parameter-to-observation map $\boldsymbol{f}$ is differentiable, we can linearize the right hand side of (2.4) around a given sample $\boldsymbol{\theta}^i_{\text{dyn}}$ to obtain

$$\boldsymbol{y}_{\text{obs}} \approx \boldsymbol{f}(\boldsymbol{\theta}^i_{\text{dyn}}) + \boldsymbol{F}^i(\boldsymbol{\theta}_{\text{dyn}} - \boldsymbol{\theta}^i_{\text{dyn}}) + \boldsymbol{\eta},$$

where $\boldsymbol{F}^i$ is the Jacobian of $\boldsymbol{f}$ evaluated at $\boldsymbol{\theta}^i_{\text{dyn}} \in \{\Theta_{\text{train}}, \Theta_{\text{test}}\}$. It follows that we can approximate the posterior covariance matrix for the i-th sample of our training dataset as

$$\boldsymbol{\Gamma}^i_{\text{post}} \approx \boldsymbol{\Gamma}^i_{\text{train}} = ((\boldsymbol{F}^i)^{\mathsf{H}} \boldsymbol{\Gamma}^{-1}_{\text{noise}} \boldsymbol{F}^i + \boldsymbol{\Gamma}^{-1}_{\text{prior}})^{-1}.$$

Comparing this linearization to a quadratic approximation of the objective function in (2.8) reveals that we can approximate $\boldsymbol{\Gamma}^i_{\text{post}}$ associated with a sample $\boldsymbol{\theta}^i_{\text{dyn}}$ based on the inverse of the Hessian matrix

$\boldsymbol{H}^i \in \mathbb{R}^{3,3}$,

$$\boldsymbol{\Gamma}^i_{\text{post}} \approx (\boldsymbol{H}^i)^{-1} = (\boldsymbol{H}^i_{\text{dat}} + \boldsymbol{H}^i_{\text{reg}})^{-1}.$$

If the parameter-to-observation map $\boldsymbol{f}$ were linear this approximation would be exact. We present the derivation as well as steps necessary to compute the Hessian matrix $\boldsymbol{H}$ for (2.2) in the supplementary materials in Subsection B.1. The matrices $\boldsymbol{H}_{\text{dat}}$ and $\boldsymbol{H}_{\text{reg}}$ are discretized versions of the continuous operators $\mathcal{H}_{\text{reg}}$ and $\mathcal{H}_{\text{dat}}$ of the reduced space Hessian operator $\mathcal{H}$ associated with (2.2). Since $\boldsymbol{H}$ is a 3×3 matrix, we can simply invert it using direct methods.

2.3.1. Mapping $\text{Sym}^+(3)$ into $\mathbb{R}^6$. The proposed approach for learning the posterior covariance matrices builds upon ideas described in [9, 10, 11, 102]. Since we use MSE as a loss function, we assume that the parameters $\boldsymbol{\theta}$ live in an Euclidean space. This does not hold for the posterior covariance $\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$. The space $\text{Sym}^+(3)$ when endowed with a Riemannian metric forms a Riemannian manifold that is locally similar to an Euclidean space. Computing distances between two elements of $\text{Sym}^+(3)$ does, in general, require the computation of a geodesic. While the geodesics can, in general, be evaluated in closed form for $\text{Sym}^+(3)$, this strategy would introduce additional complications to our NN framework. To simplify the associated computations, we consider a log-Euclidean metric on $\text{Sym}^+(3)$ [9, 10, 11]. Using this approach allows us to use classical Euclidean computations in the domain of matrix logarithms [11]. In particular, we are mapping elements of $\text{Sym}^+(3)$ to the space of symmetric matrices $\text{Sym}(3)$. These matrices can uniquely be represented as a vector in $\mathbb{R}^6$. This is the data we use for training our NN. There are several alternative strategies; this approach is particularly simple and does not require us to change the loss function for the NN or the NN architecture.

In particular, we use this log-Euclidean framework to map elements in $\text{Sym}^+(3)$ to their tangent space using the Riemannian logarithmic map $\log_{\boldsymbol{\Sigma}} : \text{Sym}^+(3) \to T_{\boldsymbol{\Sigma}} \text{Sym}^+(3)$,

$$\log_{\boldsymbol{\Sigma}}(\boldsymbol{W}) = \boldsymbol{\Sigma}^{1/2} \log(\boldsymbol{\Sigma}^{-1/2} \boldsymbol{W} \boldsymbol{\Sigma}^{-1/2}) \boldsymbol{\Sigma}^{1/2}$$

for $\boldsymbol{\Sigma} \in \text{Sym}^+(3)$ [102]; here, log denotes the usual matrix logarithm. This allows us to perform classical Euclidean computations in the domain of matrix logarithms. The inverse of the logarithmic map $\log_{\boldsymbol{\Sigma}}$is given by the Riemannian exponential map $\exp_{\boldsymbol{\Sigma}} : T_{\boldsymbol{\Sigma}} \text{Sym}^+(3) \to \text{Sym}^+(3)$,

$$\exp_{\boldsymbol{\Sigma}}(\boldsymbol{V}) = \boldsymbol{\Sigma}^{1/2} \exp(\boldsymbol{\Sigma}^{-1/2} \boldsymbol{V} \boldsymbol{\Sigma}^{-1/2}) \boldsymbol{\Sigma}^{1/2}$$

for $\boldsymbol{\Sigma} \in \text{Sym}^+(3)$ that associates to each $\boldsymbol{V} \in T_\Sigma \text{Sym}^+(3)$ a point of the manifold $\text{Sym}^+(3)$ [102]. Here, $\exp(\cdot)$ denotes the usual matrix exponential. Notice that the exponential map and its inverse, the logarithm, are both smooth and invertible maps, i.e., diffeomorphisms [11].

We evaluate $\log_{\boldsymbol{\Sigma}}$ and $\exp_{\boldsymbol{\Sigma}}$ at $\boldsymbol{\Sigma} = \mathbf{Id} = \text{diag}(1, 1, 1) \in \mathbb{R}^{3,3}$. After applying $\log_{\mathbf{Id}}$, we are working with symmetric matrices with $(3(3 + 1)/2) = 6$ independent coefficients (e.g., the upper triangular part). As such, we can map these matrices from $\text{Sym}(3)$ to $\mathbb{R}^6$. Let $\boldsymbol{W} = \log_{\mathbf{Id}}(\boldsymbol{\Gamma}_{\text{post}})$. The associated assignment $\text{vec}_{\mathbf{Id}} : \text{Sym}(3) \to \mathbb{R}^6$ is given by

$$\text{vec}_{\mathbf{Id}}(\boldsymbol{W}) = (w_{00}, \sqrt{2}w_{01}, \sqrt{2}w_{02}, w_{11}, \sqrt{2}w_{12}, w_{22}) \in \mathbb{R}^6.$$

We apply this mapping to the entire training and testing dataset. Consequently, the entries predicted by the NN architecture are all vectors in $\mathbb{R}^6$, and by that elements of $\text{Sym}(3) = T_{\mathbf{Id}} \text{Sym}^+(3)$ after appropriate rescaling and re-arrangement. To obtain the prediction of $\boldsymbol{\Gamma}_{\text{post}}$ we apply the inverse of the projection $\text{vec}_{\mathbf{Id}}$ and subsequently the Riemannian exponential $\exp_{\mathbf{Id}}$ to map from $T_{\mathbf{Id}} \text{Sym}^+(3)$ to $\text{Sym}^+(3)$.

2.3.2. Noise Models. Our first noise model is first-order autoregressive in time, following [111]. These types of models are widely used for representing time series. The autoregressive process is parametrized by a correlation parameter ρ, which determines the dependence of the process on its previous value [87]. The jth entry of $\boldsymbol{y}_{\text{obs}} \in \mathbb{R}^{n_t}$ for the additive noise model is given by

$$y_{\text{obs},j} = u(t^j) + \eta(t^j), \quad \eta(t^j) = \rho\eta(t^{j-1}) + \varepsilon(t^j), \tag{2.12}$$

at time point $t^j = jh_t \in [0, \tau]$, $j = 1, \ldots, n_t$, with $h_t = \tau/n_t$, $\eta(t^1) \sim \mathcal{N}(0, \sigma^2/h_t^2)$ and $|\rho| < 1$. Notice that the variance of the noise model depends on the time step size $h_t = \tau/n_t$. Moreover, the variance of η is constant

TABLE 3
Parameters considered for generating the training and testing data.

variable	meaning	value/range	equation
τ	final time	200 ms	see (2.1)
n_t	number of time steps	2,000	—
z	membrane potential	-0.4	see (2.1)
θ_0	model parameter (excitability threshold)	$[-0.2, 1.0]$	see (2.1)
θ_1	model parameter (recovery scaling)	$[-0.4, 1.2]$	see (2.1)
θ_2	model parameter (time scale separation)	$[2.0, 5.0]$	see (2.1)
$\boldsymbol{\sigma}_{\text{prior}}$	prior standard deviation	$(0.3, 0.4, 0.4)$	see (2.6)
$\bar{\boldsymbol{\theta}}_{\text{prior}}$	prior mean	$(0.3, 0.4, 3.4)$	see (2.6)
ρ	noise correlation	$\sim \mathcal{N}(0.8, 0.05^2)$	see (2.12)
σ^2	noise variance	$\sim \mathcal{N}(0.07, 0.01^2)$	see (2.12)
β^2	noise variance	$\sim \mathcal{N}(0.15, 0.05^2)$	see (2.13)
n_{train}	number of training samples	1,000, 2,000, 4,000, 8,000	—
n_{test}	number of testing samples	4,000	—
n_{val}	number of validation samples	2,000	—

across time since the process is stationary due to $|\rho| < 1$ (i.e., $\eta(t^j) \sim \mathcal{N}(0, \sigma^2/h_t^2)$ for all $j = 1, \dots, n_t$). Moreover, we select $\varepsilon(t^j) \sim \mathcal{N}(0, (1-\rho^2)\sigma^2/h_t^2)$. The noise parameters are $\boldsymbol{\theta}_{\text{noise}} = (\rho, \sigma) \in \mathbb{R}^2$.

In our second noise model, which we call intrinsic noise, we replace the equation for u in (2.1) by the SDE

$$\mathrm{d}Y_t - \mu(Y_t, t)\mathrm{d}t - \beta \mathrm{d}W_t = 0, \tag{2.13}$$

with initial condition $Y_0 = u_0$. The term $\mu(Y_t, t)$ can be interpreted as the expectation and is given by $\mu(Y_t, t) = \theta_2(Y_t - (Y_t^3/3) + v + z)$. The parameter $\beta > 0$ can be interpreted as the variance of the model, where $\{W_t\}_{t \geq 0}$ is a Wiener process (standard Brownian motion). The jth entry of $\boldsymbol{y}_{\text{obs}} \in \mathbb{R}^{n_t}$ for the intrinsic noise model is given by $y_{\text{obs},j} = Y_{t^j}$ for all $t^j = jh_t$, $j = 1, \dots, n_t$. We use a Euler–Maruyama scheme (e.g., see [64]) for the integration of the SDE in (2.13). We have $\boldsymbol{\theta}_{\text{noise}} = \beta \in \mathbb{R}$.

2.3.3. Parameter Selection. We discuss the selection of the ODE model parameters $\boldsymbol{\theta}_{\text{dyn}}$ first. We use

$$\boldsymbol{\sigma}_{\text{prior}} = (0.3, 0.4, 0.4) \quad \text{and} \quad \bar{\boldsymbol{\theta}}_{\text{prior}} = (0.4, 0.4, 3.4), \tag{2.14}$$

respectively, such that $[\bar{\theta}_{\text{prior},i} - 2\sigma_{\text{prior},i}, \bar{\theta}_{\text{prior},i} + 2\sigma_{\text{prior},i}]$ for $i = 0, 1, 2$. We additionally limit the samples from the prior distribution for each component of $\boldsymbol{\theta}_{\text{dyn}}$ to construct $\{\Theta_{\text{train}}, \Theta_{\text{test}}\}$ to the ranges

$$\theta_0 \in [-0.2, 1.0], \quad \theta_1 \in [-0.4, 1.2], \quad \theta_2 \in [2.0, 5.0]. \tag{2.15}$$

We discard samples for $\boldsymbol{\theta}_{\text{dyn}}$ that do not fall into these ranges.

The values of the parameter ρ and σ controlling the additive noise perturbation in (2.12) are varied randomly across different samples $\boldsymbol{y}_{\text{obs}}^i$. We generate 100 independent samples, with $\rho \sim \mathcal{N}(0.8, 0.05^2)$ and $\sigma \sim \mathcal{N}(0.07, 0.01^2)$. Selecting ρ this way allows us to achieve a correlation of 0.65 to 0.95. Likewise, selecting σ this way yields a noise level of 4% to 10%. For each pair (ρ^j, σ^j), independent replicates of the process η are generated and added to training and/or testing data. The parameter β that controls the intrinsic noise in (2.13) is chosen according to $\beta \sim \mathcal{N}(0.15, 0.05^2)$. In addition, we restrict the samples for β to the range $[0.01, 0.27]$. This range has been determined empirically. We summarize these (and other) parameters in Table 3.

2.4. Performance Measures. We assess the performance of the proposed approach qualitatively and quantitatively. We note that, since we use simulations to generate our data we have access to the true parameters and hence can use this data to evaluate the performance of the proposed methodology. We split the generated data into training and testing data to provide an unbiased estimate of the model's predictive capabilities. We report results in Section 3. Here, we summarize quantitative performance measures. We consider various measures of model performance to get a better understanding of the sources of error in our predictive model. We explain the benefits and differences for each measure below. We decompose the MSE into its squared bias (**SQB**) and centered MSE (**CMSE**) components to assess the contributions of the mean

and of the variability of the prediction mismatch [121]. In particular, a large SQB (high bias) indicates that the model is too simple (i.e., is indicative of underfitting) whereas a large CMSE (high variance) suggests that the model is too sensitive to the training data (i.e., is indicative of overfitting). These measures are given by

$$\varepsilon_{\text{SQB}}(\{\boldsymbol{\theta}^i_{\text{pred}}\}_{i=1}^{n_{\text{test}}}) = \|\bar{\boldsymbol{\theta}}_{\text{test}} - \bar{\boldsymbol{\theta}}_{\text{pred}}\|_2^2 \tag{2.16}$$

with means $\bar{\boldsymbol{\theta}}_{\text{test}} = \sum_{i=1}^{n_{\text{test}}} \boldsymbol{\theta}^i_{\text{test}}/n_{\text{test}}$ and $\bar{\boldsymbol{\theta}}_{\text{pred}} = \sum_{i=1}^{n_{\text{test}}} \boldsymbol{\theta}^i_{\text{pred}}/n_{\text{test}}$, and

$$\varepsilon_{\text{CMSE}}(\{\boldsymbol{\theta}^i_{\text{pred}}\}_{i=1}^{n_{\text{test}}}) = \frac{1}{n_s}\sum_{i=1}^{n_{\text{test}}} \left\|(\boldsymbol{\theta}^i_{\text{test}} - \bar{\boldsymbol{\theta}}_{\text{test}}) - (\boldsymbol{\theta}^i_{\text{pred}} - \bar{\boldsymbol{\theta}}_{\text{pred}})\right\|_2^2. \tag{2.17}$$

We also report values for the median of absolute percentage error (**MdAPE**) given by

$$\varepsilon_{\text{MdAPE}}(\{\boldsymbol{\theta}^i_{\text{pred}}\}_{i=1}^{n_{\text{test}}}) = \text{median}_{i=1,\dots,n_{\text{test}}} \left\{\|\boldsymbol{\theta}^i_{\text{test}} - \boldsymbol{\theta}^i_{\text{pred}}\|_2/\|\boldsymbol{\theta}^i_{\text{test}}\|_2\right\} \tag{2.18}$$

This yields a scale invariant performance measure that is less sensitive to outliers (we use the median) compared to the mean absolute percentage error (**MAPE**) and MSE, i.e., it is more robust. Since MdAPE is a relative error it is interpretable. We also report values for the coefficient of determination (**CDET**), also called R^2, given by

$$\varepsilon_{\text{CDET}}(\{\boldsymbol{\theta}^i_{\text{pred}}\}_{i=1}^{n_{\text{test}}}) = 1 - \sum_{i=1}^{n_{\text{test}}} \|\boldsymbol{\theta}^i_{\text{test}} - \boldsymbol{\theta}^i_{\text{pred}}\|_2^2 / \sum_{i=1}^{n_{\text{test}}} \|\boldsymbol{\theta}^i_{\text{test}} - \bar{\boldsymbol{\theta}}_{\text{test}}\|_2^2. \tag{2.19}$$

This measure is used to assess the percentage of variability explained by the prediction. It takes values in $(-\infty, 1]$ with 1 being optimal. The closer $\varepsilon_{\text{CDET}}$ is to a value of 1, the more variability is explained by the prediction. This measure is more sensitive to outliers than MdAPE. However, compared to the other measures it accounts for class imbalance. Throughout our experiments, the number of training samples n_{train} is varied ranging from 1,000 to 8,000. We summarize these choices in Table 3.

3. Numerical Experiments. We report results under various conditions. If not otherwise stated, the parameters for the NN architectures are as in Table 2; the parameters for data generation are as in Table 3. We consider a $(1 \mid 1)$ noise configuration for all experiments (training and testing samples contain noise). Due to page limitations we have moved additional experiments with varying configurations to the supplementary materials (see Subsection B.3). The software environment is described in Section A.

3.1. ODE Model and Noise Parameter Estimation. We report results for point estimates for the ODE parameters $\boldsymbol{\theta}_{\text{dyn}} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ along with the parameters $\boldsymbol{\theta}_{\text{noise}}$ controlling the noise. We consider additive and/or intrinsic noise. The experiment in Subsection 3.1.1 explores how different NN architectures affect the estimation of $\boldsymbol{\theta}_{\text{dyn}}$. Subsequently, we study the performance when additionally estimating additive noise parameters (see Subsection 3.1.2) and a combination of additive and intrinsic noise parameters (see Subsection 3.1.3). The results for solely estimating intrinsic noise parameters (i.e., without additive noise) can be found in the supplementary materials (see Subsection B.3.3).

3.1.1. Exploration of NN Architectures. This experiment assesses the predictive capabilities of a variety of NN architectures when we restrict ourselves to the estimation of model parameters $\boldsymbol{\theta}_{\text{dyn}}$; hence, each NN's output dimension is $p = 3$. We intend to empirically identify the architecture with optimal performance to be used for the remaining numerical study. We refer to [128] for a more detailed analysis of the NN architecture and hyperparameter turning.

Setup: As reported in [111], a learning rate of 0.002, a batch size of 32, and 64 epochs (i.e., 8,000 optimization steps) are adequate for training. We train with 4,000 samples, which gives representative results for selecting NN architectures, and validate with 2,000 samples. The input data (features) are of type TS. For DNN architectures we vary *(i)* the number of layers from 4 to 20 and *(ii)* the number of units per layer n_u from 4 to 256. For CNN architectures we vary *(i)* the number of convolutional layers from 2 to 6 and *(ii)* the filter setting n_f from 2 to 128. Recall that our CNN architecture has two additional dense layers after the convolutions (see details in Subsection 2.2.2).

TABLE 4

Exploration of DNN architectures. Each cell shows the prediction accuracy on validation data for estimating $\boldsymbol{\theta}_{dyn}$ using the MdAPE and CDET (in brackets) measures. We vary the number of nodes/units n_u per layer (rows) and the number of layers (columns).

n_u	**4 layers**	**8 layers**	**12 layers**	**16 layers**	**20 layers**
4	0.130 (0.652)	0.128 (0.635)	0.165 (0.525)	0.180 (0.450)	0.189 (0.412)
8	0.116 (0.727)	0.103 (0.768)	0.102 (0.774)	0.135 (0.660)	0.184 (0.454)
16	0.082 (0.851)	0.080 (0.859)	0.076 (0.845)	0.078 (0.858)	0.111 (0.711)
32	0.070 (0.869)	0.067 (0.878)	0.075 (0.867)	0.084 (0.849)	0.109 (0.818)
64	0.064 (0.894)	0.072 (0.873)	0.067 (0.887)	0.067 (0.893)	0.080 (0.849)
128	0.073 (0.878)	0.069 (0.888)	0.065 (0.892)	0.068 (0.885)	0.208 (0.432)
256	0.066 (0.892)	0.062 (0.892)	0.109 (0.714)	0.106 (0.717)	0.173 (0.511)

TABLE 5

Exploration of CNN architectures. Each cell shows the prediction accuracy on validation data for estimating $\boldsymbol{\theta}_{dyn}$ using the MdAPE and CDET (in brackets) measures. We vary the filter setting n_f per layer (rows) and the number of convolutional layers (columns). Recall that a full CNN has an additional two dense layers after the convolutions.

n_f	**2 layers**	**3 layers**	**4 layers**	**5 layers**	**6 layers**
2	0.059 (0.900)	0.053 (0.924)	0.053 (0.931)	0.053 (0.926)	0.052 (0.928)
4	0.061 (0.895)	0.054 (0.923)	0.050 (0.934)	0.051 (0.936)	0.052 (0.927)
8	0.063 (0.894)	0.055 (0.913)	0.051 (0.930)	0.052 (0.935)	0.053 (0.931)
16	0.070 (0.885)	0.061 (0.904)	0.059 (0.915)	0.055 (0.923)	0.057 (0.919)
32	0.068 (0.885)	0.064 (0.898)	0.058 (0.918)	0.055 (0.916)	0.058 (0.918)
64	0.074 (0.881)	0.064 (0.895)	0.068 (0.903)	0.061 (0.918)	0.061 (0.906)
128	0.071 (0.873)	0.068 (0.890)	0.060 (0.908)	0.063 (0.912)	0.954 (−7.268)

Results: We report the MdAPE and CDET evaluation measures for the DNN architectures in Table 4 and for the CNN in Table 5.

Observations: DNN architectures yield reasonable prediction accuracy with CDET > 0.80 for a wide range of settings, namely 4 to 16 layers and for n_u between 16 and 128. The peak accuracy over all DNNs is $\sim$0.89. However, across CNN architectures prediction accuracy is significantly better with CDET > 0.90 for a wide range of networks: 3 to 6 convolutional layers and for n_f between 2 and 128. The peak accuracy over all CNNs reaches $\sim$0.93, which is 0.04 points better than for the peak for DNNs; since CDET is close to its maximum value of one, an improvement of 0.04 is significant. Complementary results in the supplementary materials show that the CMSE metric behaves analogously to CDET and reveal the same trends for DNN and CNN architectures as we observe here; hence, both measures are adequate to assess prediction accuracy. The comparison between DNN and CNN architectures suggests that the use of convolution operations is advantageous for the considered time series data. We attribute this to the fact that convolutions take into account the locality of neighboring time steps in the data, which is not true for dense layers. The CNNs have a lower number of trainable weights, because convolution kernels are 3×3 matrices. For instance, the CNN with 5 layers and $n_f = 8$ has 38,195 trainable parameters, compared with the (slightly worse performing) DNN with 12 layers and $n_u = 128$, which amounts to 695,179 trainable parameters. On the other hand, DNNs have an advantage in terms of computational efficiency since their arithmetic operations can utilize hardware accelerated kernels for matrix-matrix multiplications. Based on these observations we restrict the remaining experiments to a CNN architecture with 5 layers and $n_f = 8$.

3.1.2. Estimating ODE and Noise Parameters – Additive Noise. We assess the performance of NN predictions for the simultaneous estimation of ODE model parameters $\boldsymbol{\theta}_{\text{dyn}} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ and noise parameters $\boldsymbol{\theta}_{\text{noise}} = (\rho, \sigma) \in \mathbb{R}^2$ of the additive noise model (see Subsection 2.3.2); hence, each NN will have output dimension $p = 5$. This setup allows to quantify properties of the noise of a data example at nearly the same computational cost as estimating only $\boldsymbol{\theta}_{\text{dyn}} \in \mathbb{R}^3$. As will be demonstrated, the NNs are able to estimate parameters of noise—at effectively no additional cost—along with parameters of the ODE, which is a unique capability compared with other methods for parameter estimation or inference.

Setup: We consider the three different types of NN inputs: TS, FC, and TS&FC, where the latter is obtained through stacking of TS and FC. We will see that choice of features has an impact on the accuracy of estimating $\boldsymbol{\theta}_{\text{noise}}$. In addition, we vary the number of training samples between 500 and 8,000 to study the effects on estimating $\boldsymbol{\theta}_{\text{dyn}}$ and $\boldsymbol{\theta}_{\text{noise}}$ as the amount of training data increases. The trained CNNs are assessed with 4,000 testing samples that are distinct from the validation data set used in Subsection 3.1.1.

TABLE 6

Estimation accuracy of CNN on testing data when inferring $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ and $\boldsymbol{\theta}_{noise} = (\rho, \sigma) \in \mathbb{R}^2$. Each cell shows MdAPE and CDET (in parenthesis). Blocks of three rows have a varying number of training samples n_{train}, while within each block the input data to the NN is different (TS: time series data; FC: Fourier coefficients; TS&FC: combination of TS and FC).

n_{train}	**input**	θ_0	θ_1	θ_2	ρ	σ
500	TS	0.101 (0.934)	0.266 (0.807)	0.030 (0.730)	0.058 (−0.287)	0.090 (−0.194)
	FC	0.263 (0.416)	0.429 (0.360)	0.047 (0.548)	0.027 (0.669)	0.060 (0.554)
	TS&FC	0.114 (0.928)	0.351 (0.697)	0.035 (0.647)	0.028 (0.622)	0.061 (0.532)
1,000	TS	0.101 (0.939)	0.257 (0.816)	0.029 (0.766)	0.054 (−0.147)	0.086 (−0.126)
	FC	0.262 (0.505)	0.396 (0.553)	0.039 (0.663)	0.024 (0.703)	0.054 (0.659)
	TS&FC	0.093 (0.950)	0.242 (0.826)	0.026 (0.777)	0.022 (0.759)	0.048 (0.718)
2,000	TS	0.099 (0.937)	0.206 (0.872)	0.028 (0.794)	0.050 (−0.146)	0.089 (0.007)
	FC	0.217 (0.581)	0.358 (0.628)	0.034 (0.730)	0.022 (0.759)	0.048 (0.702)
	TS&FC	0.094 (0.954)	0.193 (0.887)	0.023 (0.824)	0.021 (0.788)	0.044 (0.754)
4,000	TS	0.094 (0.945)	0.186 (0.890)	0.025 (0.826)	0.047 (−0.005)	0.065 (0.480)
	FC	0.200 (0.533)	0.320 (0.665)	0.033 (0.721)	0.020 (0.790)	0.045 (0.745)
	TS&FC	0.071 (0.966)	0.166 (0.901)	0.021 (0.859)	0.018 (0.834)	0.040 (0.804)
8,000	TS	0.070 (0.966)	0.160 (0.916)	0.022 (0.870)	0.026 (0.685)	0.045 (0.762)
	FC	0.187 (0.572)	0.291 (0.700)	0.030 (0.770)	0.019 (0.815)	0.041 (0.788)
	TS&FC	0.070 (0.970)	0.158 (0.919)	0.020 (0.876)	0.016 (0.862)	0.035 (0.835)

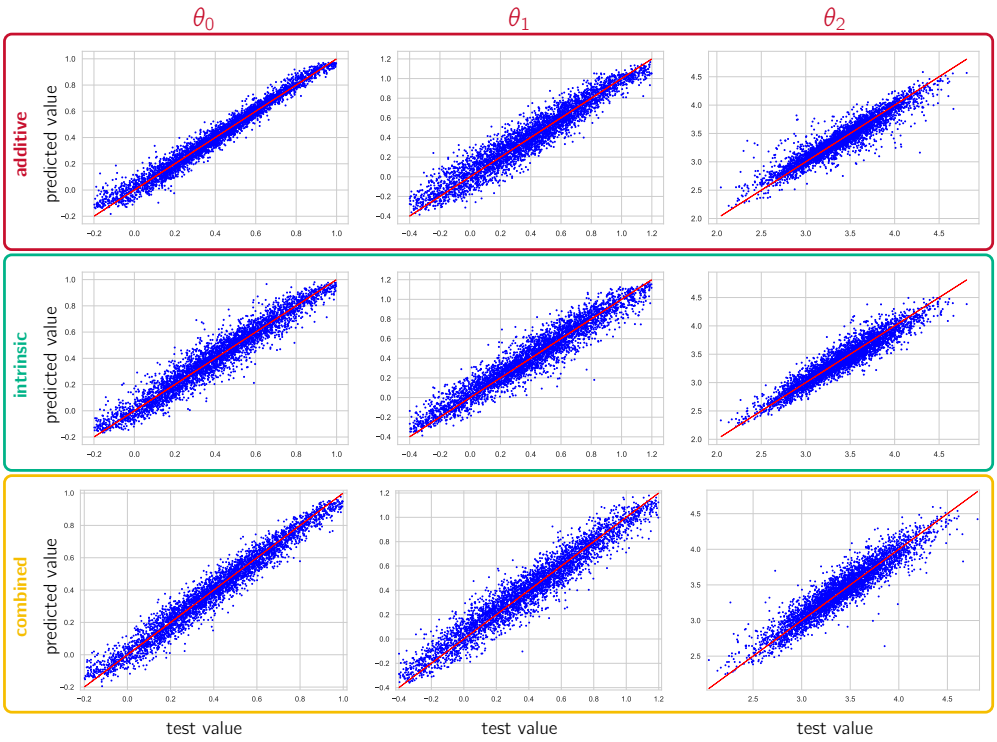

FIG. 5. *Scatter plots for CNN predictions using testing data when estimating $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ for the data type TS&FC and a training size of $n_{train} = 8{,}000$. The CNN also infers $\boldsymbol{\theta}_{noise}$, but this is not shown. The three vertical blocks correspond to additive noise (top block), intrinsic noise (middle block), and combined noise (bottom block). We plot the value of the test samples (x-axis) versus the predicted values (y-axis) for θ_0 (left), θ_1 (center), and θ_2 (right). The diagonal red line indicates optimal predictions. Overall, the CNNs deliver similar predictions across different noise configurations.*

The selection of the NN architecture is based on the experiments reported in Subsection 3.1.1; we choose a CNN with 5 layers and $n_f = 8$.

Results: We report the MdAPE and CDET evaluation measures in Table 6. We show scatter plots for the prediction of the model parameters $\boldsymbol{\theta}_{\text{dyn}}$ in Figure 5 for different noise models, data type TS&FC, and a training size of $n_{\text{train}} = 8{,}000$; top block "additive." The horizontal and vertical axes correspond to the true and predicted values, respectively. Optimal predictions are indicated by a diagonal red line.

TABLE 7

Estimation accuracy of CNN on testing data when inferring $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2)$ and $\boldsymbol{\theta}_{noise} = (\rho, \sigma, \beta)$. Each cell shows MdAPE (and CDET in parenthesis). Blocks of three rows have varying number of training samples n_{train}, while within each block the input data to NN is different (TS: time series data; FC: Fourier coefficients; TS&FC: combination of TS and FC).

n_{train}	**input**	θ_0	θ_1	θ_2	β	ρ	σ
500	TS	0.088 (0.931)	0.152 (0.843)	0.037 (0.690)	0.263 (−0.103)	0.048 (−0.112)	0.111 (−0.696)
	FC	0.227 (0.392)	0.290 (0.440)	0.053 (0.470)	0.150 (0.646)	0.027 (0.654)	0.057 (0.617)
	TS&FC	0.101 (0.943)	0.197 (0.883)	0.033 (0.774)	0.132 (0.697)	0.024 (0.705)	0.050 (0.665)
1,000	TS	0.093 (0.928)	0.140 (0.881)	0.034 (0.753)	0.213 (0.236)	0.039 (0.236)	0.114 (−0.656)
	FC	0.214 (0.349)	0.201 (0.605)	0.041 (0.669)	0.132 (0.697)	0.024 (0.702)	0.052 (0.638)
	TS&FC	0.098 (0.944)	0.195 (0.869)	0.029 (0.834)	0.123 (0.752)	0.022 (0.754)	0.046 (0.707)
2,000	TS	0.075 (0.946)	0.107 (0.913)	0.028 (0.813)	0.182 (0.434)	0.034 (0.434)	0.114 (−0.671)
	FC	0.203 (0.363)	0.200 (0.635)	0.036 (0.741)	0.127 (0.723)	0.023 (0.727)	0.050 (0.668)
	TS&FC	0.098 (0.942)	0.177 (0.907)	0.027 (0.855)	0.118 (0.767)	0.021 (0.768)	0.045 (0.712)
4,000	TS	0.075 (0.952)	0.099 (0.928)	0.026 (0.859)	0.164 (0.527)	0.029 (0.528)	0.111 (−0.624)
	FC	0.179 (0.413)	0.184 (0.690)	0.034 (0.778)	0.123 (0.753)	0.022 (0.754)	0.050 (0.679)
	TS&FC	0.088 (0.959)	0.164 (0.921)	0.025 (0.868)	0.116 (0.778)	0.020 (0.777)	0.045 (0.717)
8,000	TS	0.073 (0.959)	0.088 (0.943)	0.024 (0.871)	0.149 (0.651)	0.027 (0.651)	0.105 (−0.379)
	FC	0.170 (0.422)	0.164 (0.723)	0.032 (0.808)	0.123 (0.754)	0.021 (0.754)	0.047 (0.696)
	TS&FC	0.088 (0.958)	0.148 (0.929)	0.025 (0.877)	0.109 (0.794)	0.019 (0.795)	0.043 (0.729)

Observations: The Fourier coefficients play a critical role in the recovery of the noise parameters $\boldsymbol{\theta}_{\text{noise}}$. For instance, in Table 6 with $n_{\text{train}} = 2{,}000$, we obtain a CDET for ρ and σ with TS inputs that is at or below zero—a very poor accuracy—but FC inputs yield CDET values > 0.70. However, the Fourier coefficients by themselves (FC rows in the table) do not yield as good of an accuracy for $\boldsymbol{\theta}_{\text{dyn}}$ as with the TS data. The best results for both $\boldsymbol{\theta}_{\text{dyn}}$ and $\boldsymbol{\theta}_{\text{noise}}$ are consequently obtained with TS&FC, a concatenation of the TS and FC features.

3.1.3. Estimating ODE and Noise Parameters – Combined Noise. We extend the previous experiments in Subsection 3.1.2 to include intrinsic noise in addition to additive noise. Therefore, we jointly estimate ODE model parameters $\boldsymbol{\theta}_{\text{dyn}} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ and noise parameters $\boldsymbol{\theta}_{\text{noise}} = (\rho, \sigma, \beta) \in \mathbb{R}^3$. Consequently, each NN will have output dimension $p = 6$.

Setup: This experiment uses the same configuration for data and CNN as the experiment in Subsection 3.1.2. Since we combine two noise models, the intensity in each noise model is reduced by 50% to obtain a similar signal-to-noise ratio as in the former experiment.

Results: We report the MdAPE and CDET evaluation measures in Table 7. We complement the table with scatter plots for the prediction of the model parameters $\boldsymbol{\theta}_{\text{dyn}}$ in Figure 5, bottom block "combined."

Observations: The overall behavior is in line with previous experiments: *(i)* relatively low CDET for the recovery of noise parameters with TS data; *(ii)* FC data significantly improves the accuracy for the noise parameters; *(iii)* TS&FC data maintains the same accuracy for ODE model parameters as for pure TS data, and achieves similar accuracy for noise parameters as FC data. Comparing the accuracy obtained here with the experiment in Subsection 3.1.2, we observe slightly worse values for MdAPE (and CDET) for the noise parameters. For example, the best accuracy in Subsection 3.1.2 reached CDET values $(0.862, 0.835)$ for (ρ, σ) versus $(0.795, 0.729)$ now. This indicates that the presence of both autocorrelated and intrinsic noise makes the parameter estimation more challenging. However, the accuracy for the ODE model parameters $\boldsymbol{\theta}_{\text{dyn}}$ remains mostly similar to the experiment in Subsection 3.1.2, which is also confirmed by the scatter plots in the top and bottom rows of Figure 5.

3.2. Uncertainty Quantification – Posterior Covariance Estimation. We jointly estimate ODE parameters $\boldsymbol{\theta}_{\text{dyn}} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$, noise parameters $\boldsymbol{\theta}_{\text{noise}} = (\beta, \rho, \sigma) \in \mathbb{R}^3$ of different noise conditions (additive noise, intrinsic noise, and a combination thereof), and entries of the covariance matrix $\boldsymbol{\Gamma}_{\text{post}} \in \text{Sym}^+(3)$. For the latter we use a Laplace approximation of the posterior distribution of the model parameters $\boldsymbol{\theta}_{\text{dyn}}$ conditioned on $\boldsymbol{y}_{\text{obs}}$. Estimating $\boldsymbol{\Gamma}_{\text{post}}$ allows us to assess uncertainties locally around $\boldsymbol{\theta}_{\text{dyn}}$. By combining the three output vectors/matrices, the NN will have a total output dimension $p = 12$, where the covariance matrix $\boldsymbol{\Gamma}_{\text{post}}$ contributes six values $(\Gamma_{00}, \Gamma_{01}, \Gamma_{02}, \Gamma_{11}, \Gamma_{21}, \Gamma_{22})$; we note that we use a log-Euclidean framework to project the data from the Riemannian manifold $\text{Sym}^+(3)$ to $\mathbb{R}^6$. This allows us to perform classical Euclidean computations in the domain of matrix logarithms (see Subsection 2.3.1). With the task to predict $\boldsymbol{\Gamma}_{\text{post}}$, we are asking the question if the CNN is capable to determine a property that is not directly determined from its input $\boldsymbol{y}_{\text{obs}}$ but predict additional information that we impose, namely

TABLE 8

Estimation accuracy of CNNs on testing data when inferring $\boldsymbol{\theta}_{dyn}$, $\boldsymbol{\theta}_{noise}$, and $\boldsymbol{\Gamma}_{post}$. Each cell shows MdAPE (and CDET in parenthesis). We report values for the model and noise parameters (A: additive noise; I: intrinsic noise; C: combined noise) in the top table and for the covariance entries in the bottom table. The training sizes differ from previous experiments; we refer to the text for more details.

n_{train}	**noise**	θ_0	θ_1	θ_2	β	ρ	σ
432	A	0.120 (0.919)	0.280 (0.777)	0.029 (0.689)	—	0.032 (0.504)	0.071 (0.336)
	I	0.152 (0.877)	0.232 (0.836)	0.043 (0.657)	0.089 (0.871)	—	—
	C	0.108 (0.931)	0.210 (0.874)	0.037 (0.680)	0.144 (0.654)	0.026 (0.672)	0.056 (0.614)
872	A	0.094 (0.948)	0.229 (0.847)	0.026 (0.795)	—	0.026 (0.669)	0.055 (0.628)
	I	0.138 (0.886)	0.201 (0.861)	0.036 (0.747)	0.082 (0.889)	—	—
	C	0.097 (0.944)	0.185 (0.893)	0.031 (0.795)	0.127 (0.740)	0.023 (0.738)	0.049 (0.660)
1,742	A	0.092 (0.954)	0.194 (0.879)	0.024 (0.830)	—	0.026 (0.714)	0.055 (0.625)
	I	0.126 (0.906)	0.191 (0.875)	0.032 (0.810)	0.075 (0.904)	—	—
	C	0.090 (0.954)	0.163 (0.915)	0.028 (0.828)	0.117 (0.787)	0.021 (0.785)	0.050 (0.673)
3,480	A	0.093 (0.956)	0.166 (0.913)	0.022 (0.864)	—	0.020 (0.813)	0.042 (0.784)
	I	0.149 (0.887)	0.214 (0.875)	0.029 (0.836)	0.073 (0.919)	—	—
	C	0.090 (0.956)	0.159 (0.924)	0.024 (0.878)	0.107 (0.806)	0.019 (0.805)	0.043 (0.748)
6,928	A	0.062 (0.973)	0.136 (0.939)	0.017 (0.890)	—	0.016 (0.863)	0.040 (0.813)
	I	0.126 (0.907)	0.165 (0.906)	0.027 (0.862)	0.065 (0.936)	—	—
	C	0.089 (0.952)	0.163 (0.903)	0.025 (0.880)	0.102 (0.824)	0.018 (0.824)	0.044 (0.745)

n_{train}	**noise**	Γ_{00}	Γ_{01}	Γ_{02}	Γ_{11}	Γ_{12}	Γ_{22}
432	A	0.057 (0.924)	0.450 (0.784)	0.126 (0.930)	0.132 (0.780)	0.349 (0.793)	0.058 (0.875)
	I	0.087 (0.738)	0.416 (0.740)	0.149 (0.837)	0.131 (0.749)	0.346 (0.796)	0.083 (0.799)
	C	0.065 (0.864)	0.341 (0.832)	0.128 (0.918)	0.132 (0.798)	0.294 (0.847)	0.067 (0.872)
872	A	0.049 (0.946)	0.338 (0.864)	0.106 (0.957)	0.111 (0.845)	0.276 (0.861)	0.052 (0.910)
	I	0.088 (0.749)	0.382 (0.766)	0.148 (0.851)	0.113 (0.787)	0.304 (0.847)	0.074 (0.817)
	C	0.061 (0.895)	0.338 (0.856)	0.116 (0.934)	0.097 (0.852)	0.266 (0.881)	0.059 (0.899)
1,742	A	0.040 (0.955)	0.303 (0.892)	0.094 (0.963)	0.097 (0.885)	0.231 (0.889)	0.049 (0.926)
	I	0.079 (0.782)	0.387 (0.793)	0.130 (0.868)	0.108 (0.815)	0.270 (0.865)	0.072 (0.831)
	C	0.053 (0.911)	0.300 (0.880)	0.103 (0.943)	0.089 (0.880)	0.235 (0.896)	0.055 (0.912)
3,480	A	0.045 (0.952)	0.285 (0.909)	0.089 (0.964)	0.086 (0.907)	0.196 (0.917)	0.043 (0.940)
	I	0.089 (0.753)	0.442 (0.761)	0.137 (0.855)	0.112 (0.820)	0.304 (0.854)	0.075 (0.831)
	C	0.052 (0.903)	0.286 (0.897)	0.097 (0.943)	0.086 (0.897)	0.227 (0.914)	0.054 (0.916)
6,928	A	0.033 (0.970)	0.223 (0.935)	0.078 (0.977)	0.077 (0.926)	0.161 (0.937)	0.039 (0.959)
	I	0.079 (0.778)	0.352 (0.819)	0.123 (0.865)	0.092 (0.843)	0.246 (0.885)	0.076 (0.827)
	C	0.051 (0.901)	0.317 (0.876)	0.103 (0.945)	0.095 (0.878)	0.252 (0.889)	0.058 (0.912)

$\boldsymbol{\Gamma}_{\text{post}}$.

Setup: We utilize TS&FC as NN inputs. For each sample of $\boldsymbol{\theta}_{\text{dyn}}$ we compute $\boldsymbol{\Gamma}_{\text{post}}$ by inverting the Hessian matrix $\boldsymbol{H}$ of the optimization problem Equation (2.3) at the (known) solution $\boldsymbol{\theta}_{\text{dyn}}$ (see Subsection 2.3 for more details). While the matrix $\boldsymbol{H}$ associated with each sample is expected to be positive definite, $\boldsymbol{H} \in \text{Sym}^+(3)$, we observe that singular matrices can occur in practice. This is likely caused by the weak prior term (see Subsection 2.3.3) combined with a highly varying data misfit term (see Figure 2) and numerical inaccuracies. We train only on the subset of samples satisfying $\boldsymbol{H} \in \text{Sym}^+(3)$; we discard samples that violate that condition, because $\boldsymbol{H}$ does not satisfy conditions to represent an inverse covariance matrix. We decided to not regenerate discarded samples, because our goal was to remain consistent with previous experiments. As a consequence, the number of the training and testing samples varies slightly compared with the previous experiments: Overall, our dataset consists of 15,000 samples. 110 samples yielded negative definite Hessians. An additional 1,893 Hessians were ill-conditioned. This leaves a total of 12,997 samples for which we construct approximations for the associated covariances $\boldsymbol{\Gamma}_{\text{post}}$. We provide additional details in the supplementary material accompanying this manuscript.

Results: We report MdAPE and CDET evaluation measures in Table 8. Each block of three rows corresponds to different training sizes n_{train}. Each row in each block corresponds to a different noise setting (additive, intrinsic, and combined).

Observations: The most important observation is that the NN-based approach for inference is able to scale to larger output dimensions without increasing training size or NN architecture.

The accuracy of predicting model parameters (see columns $(\theta_0, \theta_1, \theta_2)$) remains nearly identical to the experiment in Subsection 3.1.3, where only $\boldsymbol{\theta}_{\text{dyn}}$ and $\boldsymbol{\theta}_{\text{noise}}$ were NN outputs (i.e., without $\boldsymbol{\Gamma}_{\text{post}}$; see Table 7). The accuracy of predicting noise parameters is generally consistent with the previous experiment. We observe in many cases a slightly lower accuracy (see columns (β, ρ, σ) and rows of combined noise type) compared with Table 7. This reduction in accuracy only shows up in CDET values at lower training sizes $n_{\text{train}} < 2{,}000$.

TABLE 9

Sensitivity to initialization. We report the mean and standard deviation for the NN-predictions of the model parameters $\boldsymbol{\theta}_{dyn}$ and covariance matrix entries across different seed initializations. The top block corresponds to the DNN architecture. The bottom block corresponds to the CNN architecture.

architecture	metric	parameter recovery		covariance recovery	
		mean	std dev	mean	std dev
CNN	CDET	0.856	6.3e−3	0.791	9.7e−3
	MdAPE	0.072	2.0e−3	0.174	3.1e−3
	CMSE	0.030	1.8e−3	1.3e−3	6.5e−5
	SQB	4.2e−4	3.4e−4	7.0e−6	3.0e−6
DNN	CDET	0.782	6.2e−3	0.646	4.2e−3
	MdAPE	0.104	1.6e−3	0.227	2.6e−3
	CMSE	0.069	1.1e−3	2.1e−3	3.6e−5
	SQB	9.2e−4	1.4e−3	2.3e−5	9.0e−6

For larger training sizes ($n_{\text{train}} \in \{3{,}480, 6{,}928\}$) recovery results across all parameters ($\boldsymbol{\theta}_{\text{dyn}}$ and $\boldsymbol{\theta}_{\text{noise}}$) are comparable to those without trying to predict covariances. This observation holds for all noise conditions. Overall, nearly the same performance for different NN outputs—when compared with the experiment in Subsection 3.1.3—shows a remarkable property of NN to scale to larger data dimensions without sacrificing accuracy.

When estimating the covariance entries, we observe that NN predictions achieve an accuracy that is comparable to the one for the ODE and/or noise parameters. An increase in training size also increases accuracy, which is a trend seen in previous experiments. At the largest training size, $n_{\text{train}} = 6{,}928$, all but one of the covariance entry reach CDET values above 0.8.

3.3. Sensitivity to Initialization. In this experiment, we assess the sensitivity of the NN architectures to the initialization of the network.

Setup: We report results for different random seeds (i.e., weight initializations), focusing on recovery of model parameters, noise parameters, and covariance entries under our combined noise model. We consider 10 different random initialization. We use a DNN with $n_u = 128$ nodes per layer and 12 hidden layers, and a CNN with 5 hidden layers and $n_f = 8$. For these experiments, training was conducted for 100 epochs using our largest training set size ($n_{\text{train}} = 6928$ samples) and tested on $n_{\text{test}} = 3456$ samples.

Results: Table 9 summarize the recovery performance of the CNN and DNN architectures across seeds to evaluate the effects of random weight initialization. We report detailed results in the supplementary material in Subsection B.3.6 and Subsection B.3.7, respectively. We also include results for a 6-fold cross validation in the supplementary material.

Observations: Across all seeds tested, model performance was consistent, with no significant impact from random weight initialization. Both CNN and DNN architectures exhibited stable metrics (e.g., CDET and MdAPE) with low variability. No anomalies or outliers were observed. The CNN generally achieved slightly higher CDET values and lower errors. Additionally, training times were also consistent across seeds, with the CNN averaging 73.43 seconds total, and the DNN averaging 48.44 seconds total.

3.4. Sensitivity of NN Predictions to Noise. We explore the performance of the proposed framework as a function of increasing noise perturbations. We expect the prediction accuracy to deteriorate as the noise level increases. We note that these results are baked into those reported in prior experiments. Here, we attempt to disentangle the errors and report prediction performance as a function of increasing noise perturbations. Note that we draw 100 noise parameter samples; and each noise parameter is used multiple times to generate multiple different realizations of noise. Consequently, we report statistics averaged across different predictions for each sample of noise parameter.

Setup: These results correspond to experiments in Subsection 3.2, in particular, those reported in Table 8, where inference is performed for model parameter, noise parameters, and covariance entries.

Results: Since the intrinsic noise is controlled by a single parameter β, reporting the error as a function of increasing β is straightforward. We provide box-whisker plots for the error in the ODE parameters and the covariance matrix entries in Figure 6. We report the relative ℓ^2-error, which corresponds to MdAPE in (2.18). For the additive noise model the noise is controlled by two parameters $(\rho, \sigma) \in \mathbb{R}^2$. We provide box-whisker plots for the estimation errors as each of these parameters increases in Figure 7. We provide 2D plots in the supplementary material (see Subsection B.3.8).

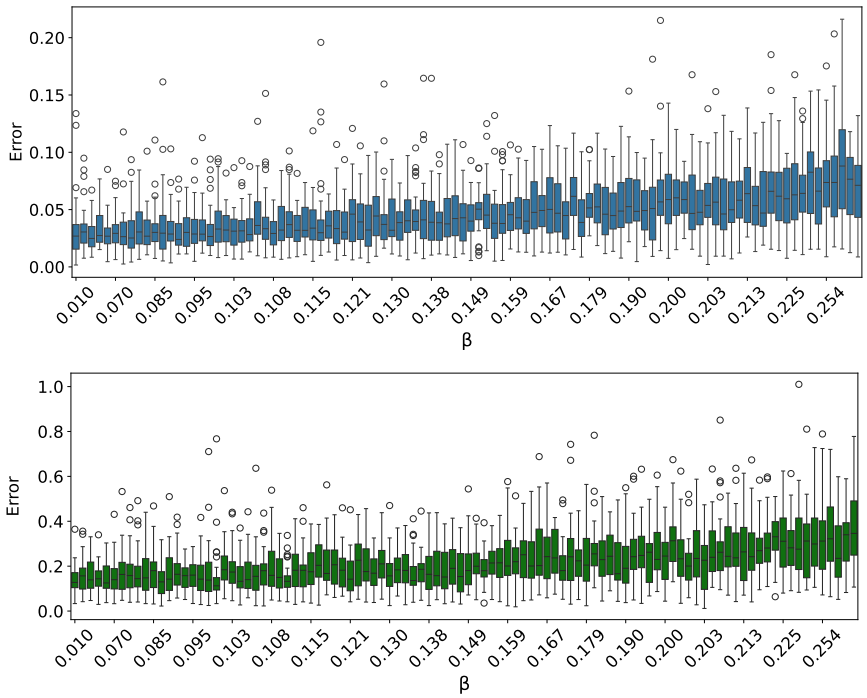

FIG. 6. *Statistics of the relative ℓ^2-error for the intrinsic noise model as a function of the noise parameter β. We report the ℓ^2-norm for the predicted entries of the model parameter vector $\boldsymbol{\theta}_{dyn}$ (top; blue) and the entries of the covariance matrix (bottom; green). The boundaries of each of these boxes correspond to the 25th and 75th percentile (lower and upper quartile), respectively. The line inside the box represents the 50th percentile. The whiskers correspond to the data points that are within 1.5 times the interquartile range from the lower and upper quartiles. The circles above or below the whiskers denote points outside this range, i.e. outliers.*

Observations: For the results for the intrinsic noise model, we can observe that increasing β results in a slight increase (on average) in the prediction error (see Figure 6). We note that the errors overall remain relative stable for the wide range of values for β considered in this study.

For the results reported for the additive noise model in Figure 7 we cannot observe a clear trend as each individual noise parameter increases. This can be attribute to the fact that these plots mix results for two parameters. The 2D plots included in the supplementary material seem to indicate that as both parameters increase, the errors seem to increase. But even for these plots the trend is not striking.

Lastly, these plots reaffirm that our framework struggles more when it comes to predicting the entries for the covariance matrices.

3.5. Performance on High-Performance Computing Hardware. The goal of these experiments is to scale up the sizes the NN architectures in terms of their trainable parameters and measure improvements in accuracy of NN predictions. Furthermore, we document the performance of NN training on high-performance computing (**HPC**) hardware. The main computational cost for NN-based parameter inference, after generating a data set for training, is the optimization of the NN parameters during the training phase. Here, we investigate this cost on two different types of HPC hardware. We compare the contemporary graphics processing unit (GPU), NVIDIA A100, with the dedicated NN hardware, Cerebras CS-2. The Cerebras CS-2 is a system that is optimized for the workloads of training deep neural networks. More information about the two platforms can be found in Section A. The TensorFlow code that we run is identical on both platforms.

Setup: The inputs and outputs of the NN are the same as in the previous experiment; see Subsection 3.2 with additive noise. Namely, we utilize TS&FC as NN inputs and estimate ODE model parameters $\boldsymbol{\theta}_{\text{dyn}} = (\theta_0, \theta_1, \theta_2)$, noise parameters $\boldsymbol{\theta}_{\text{noise}} = (\rho, \sigma)$ of additive noise noise, and a covariance matrix $\boldsymbol{\Gamma}_{\text{post}} \in \mathbb{R}^{3\times 3}$. Hence, the NN will have the total output dimension $p = 11$. The training size is $n_{\text{train}} = 6{,}928$, which is the largest quantity in the previous experiment of Subsection 3.2.

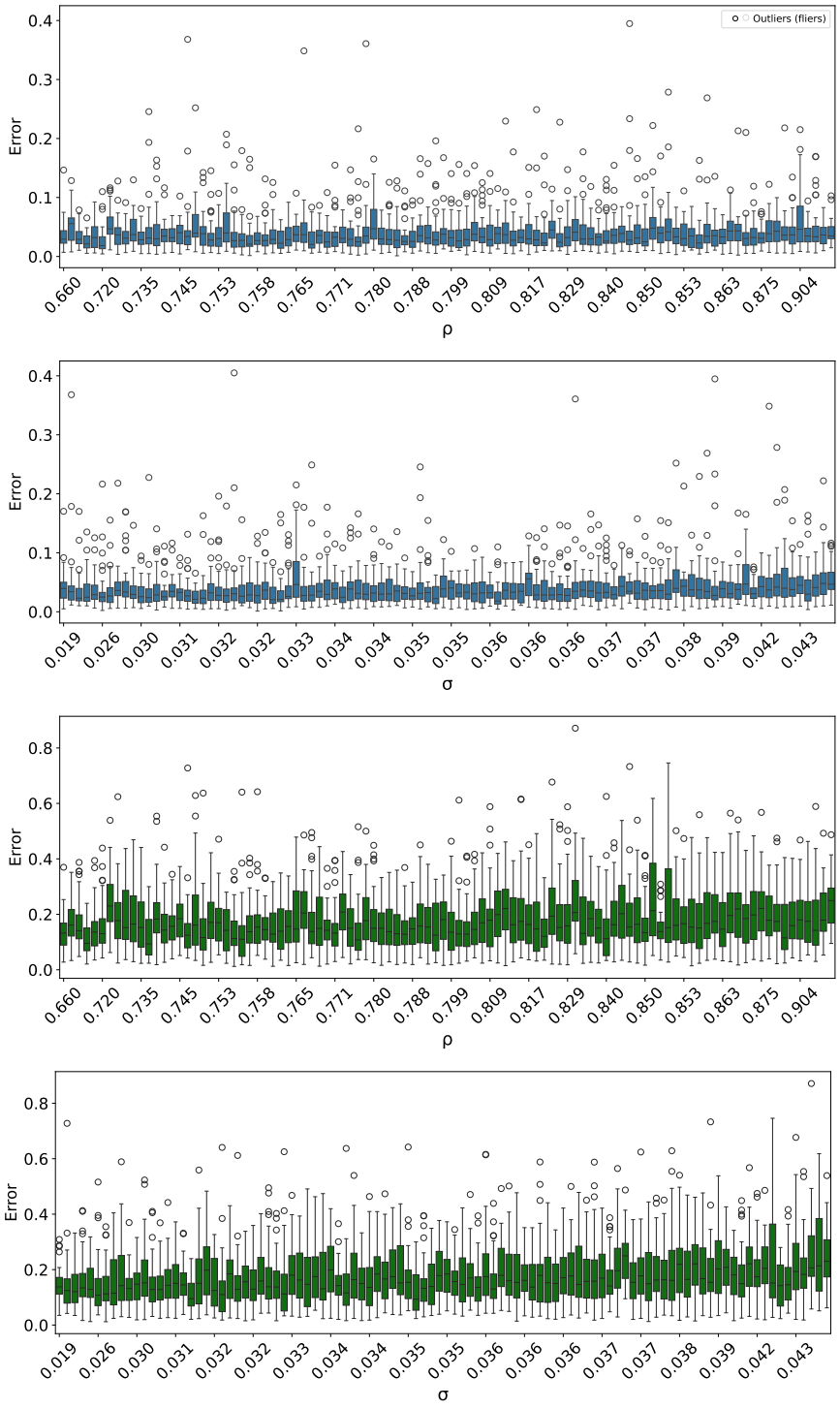

FIG. 7. *Statistics of the relative ℓ^2-error for the additive noise model as a function of the noise parameter ρ and σ. We report the ℓ^2-norm for the predicted entries of the model parameter vector $\boldsymbol{\theta}_{dyn}$ (top two blocks in blue) and the entries of the covariance matrix (bottom two blocks in green). The boundaries of each box correspond to the 25th and 75th percentile (lower and upper quartile), respectively. The line inside the box represents the 50th percentile. The whiskers correspond to the data points that are within 1.5 times the interquartile range from the lower and upper quartiles. The circles above or below the whiskers denote points outside this range, i.e. outliers.*

TABLE 10

DNN and CNN configurations for scaling and performance. For brevity, this table lists only the differences relative to the baseline settings in Table 10.

architecture	option	value(s)
DNN & CNN	n_{train}	6,928
	batch size	250
	optimization steps	12800 ($\approx$ 462 epochs)
	learning rate	10^{-4}
DNN	layers	12
	nodes/units n_u	{128,256,512,1024,2048,4096}
	dropout	0.2
CNN	convolution layers	5
	filters setting n_f	{2,4,8,16,32,64,128}
	kernel size	3
	kernel stride	2
	pooling type	*none*
	post-flattening layers	8 dense layers with $n_u = 1024$
	dropout	0

The goal of this section is to scale NN sizes to measure the effects on accuracy and the runtime performance on HPC systems. Therefore, we employ CNNs and DNNs architectures that are slightly modified compared with the ones previously used in Subsection 3.1.1. The new configurations of NNs and the settings for training are summarized in Table 10, which lists only the differences to the baseline configurations shown in Table 2 from Subsection 2.2. Notably, the dense units and the filter settings increase to up to $n_u = 4096$ and $n_f = 128$, respectively, amouting to up to $n_f \times 2^5 = 2048$ filters in the CNN. The increased complexity of the DNNs requires additional considerations for regularization and stability of the optimization. To this end, we employ dropout layers [118] with probability 0.2 between hidden layers of DNNs. Note that the dropout probability is zero for CNNs. Furthermore, we decrease the learning rate along with an increase in optimization steps and batch size (see Table 10).

We would ideally employ CNN architectures on the Cerebras CS-2, but the CS-2 software stack of the particular system that we have access to does not support the 1-dimensional convolutions required by our models.[1] Therefore, we carry out performance experiments on the CS-2 with only DNN architectures, while performance with CNN and DNN architectures are shown for the GPU.

Results: The results are presented in three tables. Table 11 for CNNs and Table 12 for DNNs show the number of trainable parameters and the associated accuracy on the testing data using the metrics MSE (i.e., loss function) and CDET. The best results are in bold and they are reached for $n_f = 64$ for CNNs and $n_u = 2048$ for DNNs. In addition, those two tables show the runtime for training on the NVIDIA A100 GPU. We list the total training time for all optimizations steps (lower is better) and the average number of samples processed per second (higher is better). The latter is a relative metric that is independent of the number of optimizaiton steps.

Table 13 shows the performance on the Cerebras CS-2 hardware for the same DNNs as in Table 12 making the performance on the GPU comparable to the CS-2. The results for CS-2 encompass additional quantities. The utilization of the CS-2 chip is the percentage of programmable fabric and compute cores that are utilized to execute a program. Furthermore once before training, the CS-2 requires compilation of TensorFlow code followed by a setup, when the CS-2 fabric is configured to execute the program. Note that comilation and setup phases do not exist when running on the GPU.

Observations: We first note the effect of scaling NN sizes on the accuracy of predictions. We grow the number of trainable parameters exponentially and are able to reach a pivotal point in prediction accuracy for both CNNs and DNNs. The accuracy is better with CNNs by around 5% compared with DNNs, which supports our observations in Subsection 3.1.1. On the flipside, is is significantly faster to train DNNs: for the best models of each class of architecture, the DNN ($n_u = 2048$ in Table 12) takes around 1/3 of the training time as the CNN ($n_f = 64$ in Table 11).

The training time on the GPU can depend on the NN size. For the "smaller" NN sizes with $< 10^5$ parameters, the time is nearly constant. It only increases for sufficiently large networks (e.g., CNNs with $n_f \geq 8$ and DNNs with $n_u \geq 2048$). The effect of steady training time despite larger NNs is even more

[1] Our additional efforts to find workarounds by using 2-dimensional convolutions, while not being ideal for the type of our data, have not been successful.

TABLE 11

Performance and scaling of CNNs on NVIDIA A100 GPU. Rows correspond to increasing NN sizes via n_f, while keeping fixed: 5 convolutional layers, 8 dense layers, $n_u = 1024$ units per dense layer. The performance is expressed in total training time (seconds) and in a nondimensional speed: samples/second. The accuracy on the testing data (MSE, CDET) is shown when inferring $(\boldsymbol{\theta}_{dyn}, \boldsymbol{\theta}_{noise}, \boldsymbol{\Gamma}_{post})$. (For comparison, the NN used in Subsection 3.2 achieves CDET = 0.838).

CNN size		**Testing accuracy**		**NVIDIA A100**	
n_f	parameters	MSE	CDET	training [sec]	samples/sec
2	11,455,499	0.007579	0.798	168.9	18 942.6
4	15,551,499	0.006571	0.825	165.3	19 365.2
8	23,743,499	0.006688	0.822	230.7	13 872.8
16	40,127,499	0.006050	0.838	267.1	11 980.6
32	72,895,499	0.004958	0.868	274.6	11 654.7
64	138,431,499	**0.004578**	**0.878**	418.5	7646.6
128	269,503,499	0.004968	0.868	772.2	4144.3

TABLE 12

Performance and scaling of DNNs on NVIDIA A100 GPU. Rows correspond to increasing NN sizes via n_u, while fixing 12 dense layers. The performance is expressed in total training time (seconds) and in a nondimensional speed: samples/second. The accuracy on the testing data (MSE, CDET) is shown when inferring $(\boldsymbol{\theta}_{dyn}, \boldsymbol{\theta}_{noise}, \boldsymbol{\Gamma}_{post})$.

DNN size		**Testing accuracy**		**NVIDIA A100**	
n_u	parameters	MSE	CDET	training [sec]	samples/sec
128	695,179	0.009745	0.740	128.7	24 906.5
256	1,750,795	0.006169	0.836	130.7	24 532.8
512	4,943,371	0.005819	0.845	130.6	24 550.7
1,024	15,653,899	0.005687	0.848	132.5	24 178.3
2,048	54,376,459	**0.005456**	**0.855**	151.1	21 205.2
4,096	201,027,595	0.005631	0.850	284.9	11 238.8

TABLE 13

Performance and scaling of DNNs on Cerebras CS-2. We increase utilization of the CS-2 chip from 1% to 94% while maintaining a high throughput that only weakly depends on the NN size (see samples/sec), because it decreases by only ∼25%. Rows correspond to increasing NN sizes via n_u, while fixing 12 dense layers. Explanations for utilization and compile, setup, and training times are given in the text. The CS-2 processes 8–14.5 times more samples per second compared with the A100 GPU (see Table 12).

DNN size		**Cerebras CS-2**				
n_u	parameters	utilization [%]	compile [sec]	setup [sec]	training [sec]	samples/sec
128	695,179	1.1	204.7	53.7	31.1	215 375.0
256	1,750,795	2.1	234.8	55.1	31.3	214 106.3
512	4,943,371	4.9	228.2	59.6	32.2	215 527.1
1,024	15,653,899	12.7	279.1	73.2	30.5	201 337.7
2,048	54,376,459	39.6	1132.2	125.9	34.0	197 587.8
4,096	201,027,595	94.1	1445.7	250.8	51.8	162 383.4

impressive on the CS-2, because the training time decreases only slightly as the DNN size grows exponentially to more than 50 million parameters. It is only near the limit of CS-2 utilization, when the training slows by around 40% ($n_u = 2048$ vs. $n_u = 4096$ in Table 13). The training performance when considering samples per second on Cerebras CS-2 is better than NVIDIA A100 by a factor of ∼8 when DNNs have $n_u = 128, \ldots, 512$ units. As the DNN sizes increase further, the gap between the two systems widens to a factor of ∼14.5 at $n_u = 4{,}096$. Currently, the speed of computations of the CS-2 comes with an overhead cost for program compilation and setup of the programmable hardware. Additional limitations are caused by the less mature software stack compared with established GPU hardware. If these limitations can be lifted by the vendor in the future, we see greater potential for using such specialized hardware for scientific applications.

4. Conclusions. We have presented a study of the performance of NNs for inference of model parameters, noise parameters, and approximate parameter uncertainties in inverse parameter estimation problems governed by nonlinear ODEs. We chose this problem as a testing bed since it poses significant challenges for traditional variational approaches or sampling methods for Bayesian inference; the associated optimization landscape features sharp gradients, a narrow, flat optimality zone, and is strongly nonconvex. In addition, this nonlinear dynamical system is simple enough to gain greater insight into the performance of our ap-

proach than more complex systems would allow. Moreover, the forward and adjoint operators are relatively cheap to evaluate and manipulate. This provides an excellent benchmark for the proposed NN framework for statistical inference.

We reported results for a joint estimation of model parameters and noise parameters. We considered different noise models; an autocorrelated additive noise model, an intrinsic noise model (based on an SDE), as well as a combination thereof. In addition, we developed a framework that allows us to estimate the covariance of the posterior distribution of the model parameters conditioned on the data. To provide training data for the NN, we have *(i)* used the prior of the parameters and *(ii)* exploited a Laplace approximation that allows us to efficiently generate approximations of the covariance of the posterior distribution. We have not only considered different noise settings but also explored the performance of the considered NNs for different features (time series, spectral coefficients, and a combination thereof). We considered a log-Euclidean framework to project the covariance data from a Riemannian manifold into an Euclidean space in an attempt to simplify our computations and guarantee that the estimated covariances are symmetric positive definite matrices.

The *most important observations* of our experiments are as follows:

- We obtain an excellent agreement (i.e., relative errors are at single-digit percentages) between NN predictions and the true, underlying model parameters, noise parameters, and covariance entries for an extremely challenging problem.
- The performance of our methodology is robust with respect to different noise settings in observational data of the inverse problem.
- To reliably estimate model and noise parameters we require a combination of spectral coefficients and time series data.
- A minimum training size of 2,000 training samples is required to achieve what we consider acceptable results.
- The NN framework scales with increasing output dimension, where accuracy remains high without significant increase in training data.
- We could reduce the training time by a factor of 8–14 when utilizing dedicated hardware platforms for training NNs.

In this work, we have limited the our explorations to DNN and CNN architectures. These architectures performed well for our purposes. In future work, we intend to push the proposed methodology to large-scale inverse problems, for which the forward and adjoint operators are more challenging to evaluate. In this context, we plan to explore modern ML frameworks that have gained popularity in the inverse problems community. Potential approaches include generative adversarial networks [3, 101, 108], normalizing flows [109, 98, 41], or diffusion models [43, 39, 34].

Appendix A. Software and Hardware Environment.

The numerical experiment in Subsection 3.1 and Subsection 3.2 are implemented in `Python 3.7.10`. We utilize the following software environment: *(i)* `Numpy 1.18.5`, *(ii)* `Scipy 1.4.1`, *(iii)* `Sklearn 0.24.2`, *(iv)* `Tensorflow 2.3.0`. We execute these runs on UH's Sabine system. Each node is equipped with a 28 core Intel Xeon E5-2680v4 CPU with 128 or 256 GB memory.

The GPU system used in Subsection 3.5 is hosted by the Department of Mathematics at Virginia Tech. The GPU is the NVIDIA A100 connected via PCIe; it contains 108 Multiprocessors and 64 CUDA Cores per Multiprocessor (6912 CUDA Cores in total), 40GB of device memory and a memory bandwidth of 1.56TB/s. The processor has 54.2 billion transistors over a silicon area of 826mm^2. The software environment is *(i)* `CUDA 11`, *(ii)* `cuDNN 8.9.6.50`, *(iii)* `cuBLAS 11.11.3.6`, *(iv)* `TensorFlow-GPU 2.9.3`.

The Cerebras CS-2 system used in Subsection 3.5 is hosted at the Pittsburgh Supercomputing Center within their Neocortex system, which houses two Cerebras CS-2 systems. The CS-2 main component is the processor: Cerebras Wafer Scale Engine 2 (WSE2). The WSE2 consists of 850,000 compute cores in total that are interconnected via a so-called programmable fabric, where each core contains a programmable router to communicate with its four neighboring cores. The WSE2 has 40GB of on-chip memory with a memory bandwidth of 20PB/s. The processor has 2.6 trillion transistors over a silicon area of 46,225mm^2. The software environment is *(i)* `Cerebras Model Zoo 1.6.0`, *(ii)* `TensorFlow 2.2.0`.

Appendix B. Supplementary Material. The supplementary materials provide additional background information on the computation of the reduced-space Hessian and complementary results to those

reported in the main manuscript. In Subsection B.1, we discuss the reduced-space Hessian. First, we outline the steps required to evaluate the Hessian matvec in Subsection B.1.1. We formally derive the reduced-space Hessian in Subsection B.1.2. We address issues with ill-conditioning, negative definite, and indefinite Hessians in the training data in Subsection B.2. We report additional numerical results in Subsection B.3. In particular, we provide additional results for exploring the NN architectures in Subsection B.3.1 We study the effects of including or omitting noise in the training data in Subsection B.3.2. We report results for estimating noise parameters for the intrinsic noise in Subsection B.3.3. In Subsection B.3.4 we explore if the end-to-end predictions of the parameters of the dynamical system yield state predictions that are in on good agreement with the data and the true state of the system. We discuss issues with an off-the-shelf prediction of covariance matrices in Subsection B.3.5. We provide additional results for the evaluation of the sensitivity of the NN predictions with respect to the initialization of the NN in Subsection B.3.6. We include k-fold cross-validation results in Subsection B.3.7 to shed additional light on how our framework generalizes to unseen data. We provide additional results for the sensitivity of our NN framework with respect to the amount of noise perturbations in Subsection B.3.8.

B.1. Curvature Information. As outlined in Subsection 2.3, we use the Hessian $\boldsymbol{H} \in \mathrm{Sym}^+(3)$ associated with the variational problem formulation in (2.2) to approximate the covariance matrix $\boldsymbol{\Gamma}_{\text{post}} \in \mathrm{Sym}^+(3)$ of the posterior distribution. In the following, we discuss the computations for the construction of the Hessian.

We consider an *optimize-then-discretize* approach to derive the second-order derivatives [59]. That is, we compute first- and second-order variations in the continuum and subsequently discretize the resulting mathematical expressions. To handle the ODE constraints in (2.2), we introduce the Lagrange multipliers $\lambda : [0,\tau] \to \mathbb{R}$ for (2.1a) and $\nu : [0,\tau] \to \mathbb{R}$ for (2.1b). Let $\boldsymbol{\Xi}$ denote the collection of the parameter vector, and the primal and dual variables, i.e., $\boldsymbol{\Xi} := (\boldsymbol{\theta}_{\text{dyn}}, u, v, \lambda, \nu)$. The Lagrangian functional ℓ is given by

$$
\begin{aligned}
\ell(\boldsymbol{\Xi}) = \frac{\gamma}{2}\int_0^\tau (u(t) - y_{\text{obs}}(t))^2 \,\mathrm{d}t + \frac{1}{2}\|\boldsymbol{\theta}_{\text{dyn}} - \boldsymbol{\theta}_{\text{ref}}\|^2_{\boldsymbol{L}} \\
+ \int_0^1 \lambda(\mathrm{d}_t u - \theta_2(u - (u^3/3) + v + z))\,\mathrm{d}t + \lambda(t=0)(u(t=0) - u_0) \\
+ \int_0^1 (\mathrm{d}_t v + (u - \theta_0 - \theta_1 v)/\theta_2)\nu \,\mathrm{d}t + \nu(t=0)(v(t=0) - v_0),
\end{aligned}
\tag{B.1}
$$

where $\gamma > 0$, $\boldsymbol{L} \in \mathrm{Sym}^+(3)$, $\boldsymbol{\theta}_{\text{ref}} \in \mathbb{R}^3$, $z > 0$, $u_0 \in \mathbb{R}$ and $v_0 \in \mathbb{R}$ are fixed variables defined in Section 2.

Below, we denote the perturbations of the parameter vector, and the primal and dual variables associated with the first variations by $\hat{\boldsymbol{\Xi}} := (\hat{\boldsymbol{\theta}}, \hat{u}, \hat{v}, \hat{\lambda}, \hat{\nu})$. For the second variations, we use $\tilde{\boldsymbol{\Xi}} := (\tilde{\boldsymbol{\theta}}, \tilde{u}, \tilde{v}, \tilde{\lambda}, \tilde{\nu})$. We refer to $\tilde{\boldsymbol{\theta}}$ as the incremental parameter vector and to $\tilde{u}$, $\tilde{v}$, $\tilde{\lambda}$, and $\tilde{\nu}$, as the incremental state and adjoint variables. In the context of optimization, these variables correspond to the search directions associated with the Newton step.

In Subsection B.1.1 we overview the steps associated with constructing the Hessian matrix $\boldsymbol{H}$. In Subsection B.1.2 we formally derive the first- and second-order variations.

B.1.1. Computation of Hessian. We consider a reduced space method to compute the Hessian [16, 17]. The Hessian matvec, i.e., the action of the Hessian on a vector $\tilde{\boldsymbol{\theta}}$, is given by

$$
\mathcal{H}\tilde{\boldsymbol{\theta}} = (\mathcal{H}_{\text{reg}} + \mathcal{H}_{\text{dat}})\tilde{\boldsymbol{\theta}} := \boldsymbol{L}\tilde{\boldsymbol{\theta}} - \int_0^\tau \boldsymbol{Q}\begin{pmatrix}\tilde{\lambda}\\ \tilde{\nu}\end{pmatrix} + \tilde{\boldsymbol{Q}}\begin{pmatrix}\lambda\\ \nu\end{pmatrix}\mathrm{d}t,
\tag{B.2}
$$

where $\mathcal{H}_{\text{reg}} = \boldsymbol{L} \in \mathrm{Sym}^+(3)$ is the contribution of the regularization model and

$$
\mathcal{H}_{\text{dat}} = -\int_0^\tau \boldsymbol{Q}\begin{pmatrix}\tilde{\lambda}\\ \tilde{\nu}\end{pmatrix} + \tilde{\boldsymbol{Q}}\begin{pmatrix}\lambda\\ \nu\end{pmatrix}\mathrm{d}t
$$

represents the data misfit term. The expressions for $\boldsymbol{Q}$ and $\tilde{\boldsymbol{Q}}$ are given by

$$
\boldsymbol{Q} = \begin{pmatrix} 0 & 1/\theta_2 \\ 0 & -v/\theta_2 \\ u - (u^3/3) + v + z & (u - \theta_0 + \theta_1 v)/\theta_2^2 \end{pmatrix}
$$

and

$$\tilde{\boldsymbol{Q}} = \begin{pmatrix} 0 & -\tilde{\theta}_2/\theta_2^2 \\ 0 & -(\tilde{v}/\theta_2) + (v\tilde{\theta}_2/\theta_2^2) \\ (1-u^2)\tilde{u} + \tilde{v} & ((\tilde{u} + \theta_1\tilde{v} + v\tilde{\theta}_1 - \tilde{\theta}_0)/\theta_2^2) - (2(u - \theta_0 + \theta_1 v)\tilde{\theta}_2/\theta_2^3) \end{pmatrix},$$

respectively. Since the discrete representation $\boldsymbol{H}$ of $\mathcal{H}$ is a 3×3 matrix, we can explicitly form and store $\boldsymbol{H}$. To do so, we sample the columns of $\boldsymbol{H}$ by applying the discretized version of $\mathcal{H}$ in (B.2) to the standard basis vectors $\boldsymbol{e}_k \in \mathbb{R}^3$, $k = 1, 2, 3$. That is, we invoke (B.2) by replacing $\tilde{\boldsymbol{\theta}}$ with $\boldsymbol{e}_k$. To evaluate (B.2) we require all variables of $\boldsymbol{\Xi}$ and $\tilde{\boldsymbol{\Xi}}$, respectively. The candidate vector $\boldsymbol{\theta}_{\text{dyn}} \in \mathbb{R}^3$ is given. We find the state variables $u : [0, \tau] \to \mathbb{R}$ and $v : [0, \tau] \to \mathbb{R}$ for the candidate $\boldsymbol{\theta}_{\text{dyn}}$ by solving (2.1) forward in time. Subsequently, we compute $\lambda : [0, \tau] \to \mathbb{R}$ and $\nu : [0, \tau] \to \mathbb{R}$ by solving the adjoint equations

$$-\mathrm{d}_t\lambda - \theta_2(1-u^2)\lambda + (\nu/\theta_2) + u - y_{\text{obs}} = 0 \qquad \text{in } [0, \tau), \tag{B.3a}$$

$$-\mathrm{d}_t\nu + (\theta_1\nu/\theta_2) - \theta_2\lambda = 0 \qquad \text{in } [0, \tau), \tag{B.3b}$$

with final conditions $\lambda(t = \tau) = 0$ and $\nu(t = \tau) = 0$, respectively, backward in time. The incremental state variables $\tilde{u} : [0, \tau] \to \mathbb{R}$ and $\tilde{v} : [0, \tau] \to \mathbb{R}$ are found by solving

$$\begin{aligned} \mathrm{d}_t\tilde{u} - \theta_2((1-u^2)\tilde{u} + \tilde{v}) - \tilde{\theta}_2(u - (u^3/3) + v + z) = 0 \quad &\text{in } (0, \tau], \\ \mathrm{d}_t\tilde{v} + [\theta_1\tilde{v} + (\tilde{u} - \tilde{\theta}_0 + \tilde{\theta}_1 v) - (\tilde{\theta}_2(u - \theta_0 + \theta_1 v)/\theta_2)]/\theta_2 = 0 \quad &\text{in } (0, \tau], \end{aligned}$$

with initial conditions $\tilde{u}(t = 0) = 0$ and $\tilde{v}(t = 0) = 0$, respectively, forward in time. The incremental adjoint variables $\tilde{\lambda} : [0, \tau] \to \mathbb{R}$ and $\tilde{\nu} : [0, \tau] \to \mathbb{R}$ are found by solving

$$\begin{aligned} -\mathrm{d}_t\tilde{\lambda} - \theta_2(1-u^2)\tilde{\lambda} - \tilde{\theta}_2(1-u^2)\lambda + (\tilde{\nu}/\theta_2) - (\nu\tilde{\theta}_2/\theta_2^2) + (1 + 2\theta_2 u\lambda)\tilde{u} = 0 \quad &\text{in } [0, \tau), \\ -\mathrm{d}_t\tilde{\nu} + ([\theta_1\tilde{\nu} + \tilde{\theta}_1\nu - (\theta_1\tilde{\theta}_2\nu/\theta_2)]/\theta_2) - \theta_2\tilde{\lambda} - \tilde{\theta}_2\lambda = 0 \quad &\text{in } [0, \tau), \end{aligned}$$

with final conditions $\tilde{\lambda}(t = \tau) = 0$ and $\tilde{\nu}(t = \tau) = 0$, respectively, backward in time.

B.1.2. Variations. Below, we formally derive the first- and second-order variations. We assume that all variables that appear in the derivations below satisfy suitable regularity requirements to carry out the necessary computations.

The first variations with respect to the perturbations $\hat{\lambda}$, $\hat{\nu}$, of the Lagrange multipliers λ, ν, are given by

$$\begin{aligned} \ell_\lambda[\boldsymbol{\Xi}](\hat{\lambda}) &= \int_0^\tau (\mathrm{d}_t u - \theta_2(u - (u^3/3) + v + z))\hat{\lambda}\,\mathrm{d}t + (u(t = 0) - u_0)\hat{\lambda}(t = 0), \\ \ell_\nu[\boldsymbol{\Xi}](\hat{\nu}) &= \int_0^\tau (\mathrm{d}_t v + (u - \theta_0 - \theta_1 v)/\theta_2)\hat{\nu}\,\mathrm{d}t + (v(t = 0) - v_0)\hat{\nu}(t = 0). \end{aligned}$$

The variations with respect to the perturbations $\hat{u}$, $\hat{v}$, of the state variables u, v, are given by

$$\begin{aligned} \ell_u[\boldsymbol{\Xi}](\hat{u}) &= \int_0^\tau (-\mathrm{d}_t\lambda - \theta_2(1-u^2)\lambda + (\nu/\theta_2) + u - y_{\text{obs}})\hat{u}\,\mathrm{d}t + \lambda(t = \tau)\hat{u}(t = \tau), \\ \ell_v[\boldsymbol{\Xi}](\hat{v}) &= \int_0^\tau (-\mathrm{d}_t\nu + (\theta_1\nu/\theta_2) - \theta_2\lambda)\hat{v}\,\mathrm{d}t + \nu(\tau = 1)\hat{v}(\tau = 1). \end{aligned}$$

The variations with respect to the perturbations $\hat{\boldsymbol{\theta}} = (\hat{\theta}_0, \hat{\theta}_1, \hat{\theta}_2) \in \mathbb{R}^3$ of the parameter vector $\boldsymbol{\theta}_{\text{dyn}} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ are given by

$$\ell_{\boldsymbol{\theta}}[\boldsymbol{\Xi}](\hat{\boldsymbol{\theta}}) = \langle \boldsymbol{L}\boldsymbol{\theta}_{\text{dyn}}, \hat{\boldsymbol{\theta}} \rangle - \int_0^\tau \left\langle \begin{pmatrix} 0 & 1/\theta_2 \\ 0 & -v/\theta_2 \\ u - (u^3/3) + v + z & (u - \theta_0 + \theta_1 v)/\theta_2^2 \end{pmatrix} \begin{pmatrix} \lambda \\ \nu \end{pmatrix}, \hat{\boldsymbol{\theta}} \right\rangle \mathrm{d}t.$$

At optimality, i.e., for any primal-dual optimal $\boldsymbol{\Xi}^\star = (u^\star, v^\star, \nu^\star, \lambda^\star, \boldsymbol{\theta}_{\text{dyn}}^\star)$, these variations vanish. Setting $\ell_\lambda[\boldsymbol{\Xi}](\hat{\lambda})$, $\ell_\nu[\boldsymbol{\Xi}](\hat{\nu})$, $\ell_u[\boldsymbol{\Xi}](\hat{u})$ and $\ell_v[\boldsymbol{\Xi}](\hat{v})$ to zero yields the strong form of the state and adjoint equations

for the variational problem (2.2) (see (2.1) and (B.3)). Their numerical time integration allows us to compute the state variables u and v as well as the adjoint variables λ and ν for a candidate $\boldsymbol{\theta}_{\text{dyn}}$.

Since we are interested in deriving curvature information for training the NNs for the prediction of the posterior covariance matrices we have to compute second variations. These follow from the first variations in a straight-forward way.

The second variations of the Lagrangian in with respect to perturbations $\tilde{\lambda}$ in λ are given by

$$\begin{aligned}
\ell_{\lambda\lambda}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\lambda}) &= 0,\\
\ell_{\nu\lambda}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\nu}) &= 0,\\
\ell_{u\lambda}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{u}) &= \int_0^\tau (-\mathrm{d}_t\tilde{\lambda} - \theta_2(1-u^2)\tilde{\lambda})\hat{u}\,\mathrm{d}t + \tilde{\lambda}(t=\tau)\hat{u}(t=\tau),\\
\ell_{v\lambda}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{v}) &= \int_0^\tau (-\theta_2\tilde{\lambda})\hat{v}\,\mathrm{d}t,\\
\ell_{\boldsymbol{\theta}\lambda}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\boldsymbol{\theta}}) &= \int_0^\tau -(u-(u^3/3)+v+z)\tilde{\lambda}\hat{\theta}_2\,\mathrm{d}t.
\end{aligned}$$

The second variations of the Lagrangian in with respect to perturbations $\tilde{\nu}$ in ν are given by

$$\begin{aligned}
\ell_{\lambda\nu}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\lambda}) &= 0,\\
\ell_{\nu\nu}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\nu}) &= 0,\\
\ell_{u\nu}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{u}) &= \int_0^\tau (\tilde{\nu}/\theta_2)\hat{u}\,\mathrm{d}t,\\
\ell_{v\nu}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{v}) &= \int_0^\tau (-\mathrm{d}_t\tilde{\nu} + (\theta_1\tilde{\nu}/\theta_2))\hat{v}\,\mathrm{d}t + \tilde{\nu}(t=\tau)\hat{v}(t=\tau),\\
\ell_{\boldsymbol{\theta}\nu}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\boldsymbol{\theta}}) &= \int_0^\tau (-\tilde{\nu}/\theta_2)\hat{\theta}_0 + (v\tilde{\nu}/\theta_2)\hat{\theta}_1 - ((u-\theta_0+\theta_1 v)\tilde{\nu}/\theta_2^2)\hat{\theta}_2\,\mathrm{d}t.
\end{aligned}$$

The second variations of the Lagrangian in with respect to perturbations $\tilde{u}$ in u are given by

$$\begin{aligned}
\ell_{\lambda u}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\lambda}) &= \int_0^\tau (\mathrm{d}_t\tilde{u} - \theta_2(1-u^2)\tilde{u})\hat{\lambda}\,\mathrm{d}t + \tilde{u}(t=0)\hat{\lambda}(t=0),\\
\ell_{\nu u}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\nu}) &= \int_0^\tau (\tilde{u}/\theta_2)\hat{\nu}\,\mathrm{d}t,\\
\ell_{uu}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{u}) &= \int_0^\tau (1+2\theta_2 u\lambda)\tilde{u}\hat{u}\,\mathrm{d}t,\\
\ell_{vu}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{v}) &= 0,\\
\ell_{\boldsymbol{\theta}u}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\boldsymbol{\theta}}) &= \int_0^\tau -((1-u^2)\lambda\tilde{u} + (\nu\tilde{u}/\theta_2^2))\hat{\theta}_2\,\mathrm{d}t.
\end{aligned}$$

The second variations of the Lagrangian in with respect to perturbations $\tilde{v}$ in v are given by

$$\begin{aligned}
\ell_{\lambda v}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\lambda}) &= \int_0^\tau -\theta_2\tilde{v}\hat{\lambda}\,\mathrm{d}t,\\
\ell_{\nu v}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\nu}) &= \int_0^\tau (\mathrm{d}_t\tilde{v} + (\theta_1\tilde{v}/\theta_2))\hat{\nu}\,\mathrm{d}t + \tilde{v}(t=0)\hat{\nu}(t=0),\\
\ell_{uv}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{u}) &= 0,\\
\ell_{vv}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{v}) &= 0,\\
\ell_{\boldsymbol{\theta}v}[\boldsymbol{\Xi},\tilde{\boldsymbol{\Xi}}](\hat{\boldsymbol{\theta}}) &= \int_0^\tau (\nu\tilde{v}/\theta_2)\hat{\theta}_1 - (\lambda\tilde{v} + (\theta_1\nu\tilde{v}/\theta_2^2))\hat{\theta}_2\,\mathrm{d}t.
\end{aligned}$$

The second variations of the Lagrangian in with respect to perturbations $\tilde{\boldsymbol{\theta}} = (\tilde{\theta}_0,\tilde{\theta}_1,\tilde{\theta}_2) \in \mathbb{R}^3$ in

TABLE 14

Training samples after removal of Hessian matrices that were negative definite (110 samples) and severely ill-conditioned (1,893 samples) from our original dataset of 15,000 samples. We kept the same training sets that we considered in other experiments. Consequently, the size of each training dataset was reduced.

n_{train} (original)	n_{train} (after removal)	samples dropped
500	432 (80%)	68
1,000	872 (87%)	128
2,000	1,742 (87%)	258
4,000	3,480 (87%)	520
8,000	6,928 (86%)	1,072

$\boldsymbol{\theta}_{\text{dyn}} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ are given by

$$\ell_{\lambda\boldsymbol{\theta}}[\boldsymbol{\Xi}, \tilde{\boldsymbol{\Xi}}](\hat{\lambda}) = \int_0^\tau -(u - (u^3/3) + v + z)\tilde{\theta}_2\hat{\lambda}\,\mathrm{d}t,$$

$$\ell_{\nu\boldsymbol{\theta}}[\boldsymbol{\Xi}, \tilde{\boldsymbol{\Xi}}](\hat{\nu}) = \int_0^\tau [(-(u - \theta_0 + \theta_1 v)\tilde{\theta}_2/\theta_2^2) - (\tilde{\theta}_0/\theta_2) + (v\tilde{\theta}_1/\theta_2)]\hat{\nu}\,\mathrm{d}t,$$

$$\ell_{u\boldsymbol{\theta}}[\boldsymbol{\Xi}, \tilde{\boldsymbol{\Xi}}](\hat{u}) = \int_0^\tau [-(1 - u^2)\lambda\tilde{\theta}_2 - (\nu\tilde{\theta}_2/\theta_2^2)]\hat{u}\,\mathrm{d}t,$$

$$\ell_{v\boldsymbol{\theta}}[\boldsymbol{\Xi}, \tilde{\boldsymbol{\Xi}}](\hat{v}) = \int_0^\tau [-\tilde{\theta}_2\lambda - (\theta_1\tilde{\theta}_2\nu/\theta_2^2) + (\tilde{\theta}_1\nu/\theta_2)]\hat{v}\,\mathrm{d}t,$$

$$\ell_{\boldsymbol{\theta}\boldsymbol{\theta}}[\boldsymbol{\Xi}, \tilde{\boldsymbol{\Xi}}](\hat{\boldsymbol{\theta}}) = \langle \boldsymbol{L}\tilde{\boldsymbol{\theta}}, \hat{\boldsymbol{\theta}}\rangle + \int_0^\tau \left\langle \frac{\nu}{\theta_2^2} \begin{pmatrix} \tilde{\theta}_2 \\ -v\tilde{\theta}_2 \\ (2(u - \theta_0 + \theta_1 v)\tilde{\theta}_2/\theta_2) + \tilde{\theta}_0 - v\tilde{\theta}_1 \end{pmatrix}, \hat{\boldsymbol{\theta}} \right\rangle \mathrm{d}t.$$

B.2. Instabilities in Covariance Dataset. We generate 15,000 samples for the model parameter $\boldsymbol{\theta}_{\text{dyn}} \in \mathbb{R}^3$. Since we evaluate the Hessian $\boldsymbol{H}$ at the solution of our problem (i.e., for a given $\boldsymbol{\theta}_{\text{dyn}}$ that was used to generate the observation $\boldsymbol{y}_{\text{obs}}$), we expect that $\boldsymbol{H} \in \text{Sym}^+(3)$. However, we do not necessarily observe this in computation. Due to numerical errors, the Hessians $\boldsymbol{H}$ we compute is only symmetric up to a certain level. That is, $\|\boldsymbol{H} - \boldsymbol{H}^\mathsf{T}\|$ is not exactly zero. As a remedy, we set $\boldsymbol{H}$ to $(\boldsymbol{H} + \boldsymbol{H}^\mathsf{T})/2$. In addition, we observed that for certain parameters $\boldsymbol{\theta}_{\text{dyn}}$ the associated Hessian is no longer positive definite. Because the number of negative definite matrices is relatively small (110 samples), we decided to exclude these examples for the experiments reported in Subsection 3.2 of the main manuscript.

We also observed that some of the Hessian matrices are severely ill-conditioned. This is a significant challenge, since we use the inverse of the Hessian to approximate the covariance $\boldsymbol{\Gamma}_{\text{post}}$ of the negative log posterior. Let $\boldsymbol{H} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\mathsf{T}$ with $\boldsymbol{S} = \text{diag}(s_1, s_2, s_3)$, $s_1 \geq s_2 \geq s_3 > 0$. We observed that the first singular values s_1 associated with a subset of the training samples for $\boldsymbol{\theta}_{\text{dyn}}$ were extremely large. We show box-whisker plots for the singular values s_i, $i = 1, 2, 3$, in Figure 8, with the green horizontal lines further highlighting the whiskers of the plots. The upper whiskers are computed by adding 1.5 times the interquartile range (IQR) to the 75th percentile, while the lower whiskers are determined by subtracting 1.5 times the IQR from the 25th percentile. Notably, significant variations are observed in the ranges among singular values. A high ratio between the largest and smallest singular values can lead to numerical instabilities, indicating an ill-conditioned matrix. To mitigate potential issues, we exclude Hessian matrices with a first singular value s_1 exceeding the upper whisker, as well as those with a third singular value s_3 below the lower whisker. Overall, these thresholds result in an exclusion of a total of 1,893 samples. This leaves a total of 12,997 sample pairs $\Theta_{\text{train}} = \{\boldsymbol{\theta}^i_{\text{dyn}}, \boldsymbol{\theta}^i_{\text{noise}}, \boldsymbol{\Gamma}^i\}$, with $\boldsymbol{\Gamma}^i_{\text{post}} \approx \boldsymbol{\Gamma}^i = (\boldsymbol{H}^i)^{-1}$. To remain consistent with other experiments reported in the main manuscript (where we estimate noise and model parameters), we kept the same samples, resulting in different training sizes. For example, we keep 3,456 of the matrices of the dataset that compromises 4,000 samples. We summarize the remaining training data sizes in Table 14.

The entries of the resulting matrices $\boldsymbol{\Gamma} \in \text{Sym}^+(3)$ approximating the covariance matrices $\boldsymbol{\Gamma}_{\text{post}}$ are shown in Figure 9.

To determine if a matrix is symmetric positive definite, we first assess symmetry by comparing the matrix to its transpose. In particular, we quantify the relative error calculated as the norm of the difference divided by the norm of the matrix itself. We use a tolerance of 1.0e−3. To assess if a matrix is positive

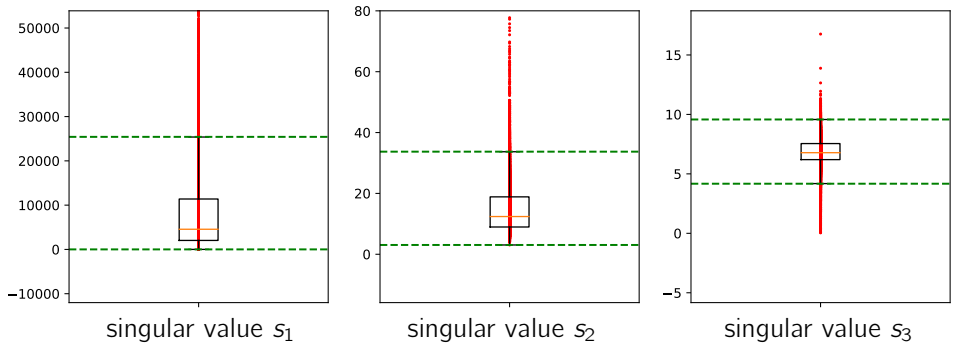

FIG. 8. *Singular values s_i, $i = 1, 2, 3$, for the positive-definite Hessian matrices $\boldsymbol{H}$. From left to right we show the box-whisker plots for the singular values s_1, s_2, and s_3, respectively. The original dataset consists of* 15,000 *samples. 110 of these matrices were negative definite in computation. Here, we show the singular values for the remaining* 14,890 *matrices. Dashed horizontal lines correspond to the upper and lower whiskers of the plots.*

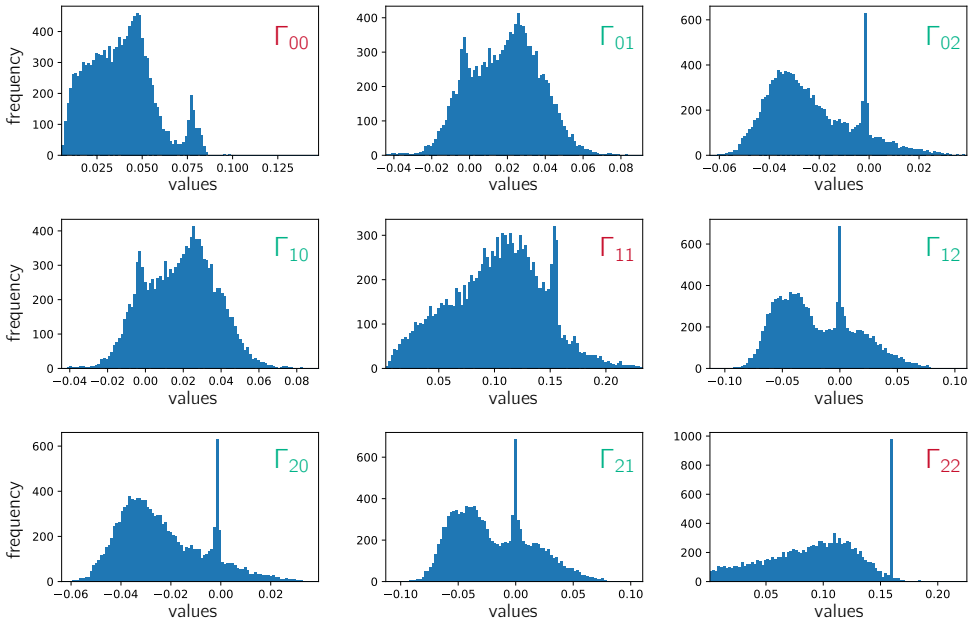

FIG. 9. *Histograms for the approximations $\boldsymbol{\Gamma} = (\boldsymbol{H})^{-1}$ of the posterior covariance matrices $\boldsymbol{\Gamma}_{post}$ based on the Hessian matrices $\boldsymbol{H}$ included in our dataset. We show the frequency for each individual entry Γ_{ij}, $i, j = 0, 1, 2$.*

definite we compute an eigenvalue decomposition.

B.3. Additional Numerical Results. Below, we provide more insight into the performance of our method.

B.3.1. Exploration of NN Architectures. Here, we report additional results for the exploration of the DNN and CNN architectures. In the main manuscript we have reported values for the MdAPE and CDET score. Here, we report values for the SQB and CMSE. The values for the DNN are reported in Table 15. The values for the CNN are reported in Table 16. In the main manuscript we selected a CNN architecture with 5 layers and $n_f = 8$ based on these experiments.

B.3.2. Effects of Noise in Training Data. In the main manuscript, we consistently included noise in both training and testing data. Here, we provide some additional experiments to identify if this was indeed necessary. While the setup is slightly different (we invert for three and not two model parameters),

TABLE 15

Exploration of DNN architectures. We report values for the SQB and the CMSE (in brackets). We consider time series data as features (TS setting). We report results for a varying number of nodes/units n_u per layer, where n_u ranges from 4 to 256. We vary the number of layers from 2 to 20.

n_u	**4 layers**	**8 layers**	**12 layers**	**16 layers**	**20 layers**
4	1.7e−4 (0.140)	1.1e−3 (0.149)	6.1e−4 (0.183)	1.3e−3 (0.213)	6.9e−5 (0.226)
8	1.1e−3 (0.108)	2.1e−3 (0.089)	5.8e−4 (0.089)	2.1e−3 (0.135)	3.2e−4 (0.207)
16	1.0e−4 (0.060)	5.2e−5 (0.057)	1.2e−4 (0.063)	5.5e−4 (0.056)	2.9e−5 (0.116)
32	3.0e−4 (0.052)	3.1e−4 (0.049)	2.4e−3 (0.051)	3.2e−3 (0.057)	3.4e−3 (0.068)
64	4.1e−5 (0.043)	1.1e−3 (0.050)	2.7e−4 (0.046)	5.4e−5 (0.043)	1.2e−3 (0.060)
128	3.9e−4 (0.049)	4.5e−3 (0.041)	3.5e−4 (0.043)	2.4e−3 (0.044)	2.3e−3 (0.212)
256	5.2e−4 (0.043)	1.6e−4 (0.044)	2.3e−3 (0.114)	5.1e−3 (0.108)	9.6e−4 (0.185)

TABLE 16

Exploration of CNN architectures. We report values for the SQB and the CMSE (in brackets). We consider time series data as features (TS setting). We report results for a varying number of filters n_f per layer, where n_f ranges from 2 to 128. We vary the number of layers from 2 to 6.

n_f	**2 layers**	**3 layers**	**4 layers**	**5 layers**	**6 layers**
2	7.3e−5 (0.040)	2.6e−4 (0.031)	3.4e−4 (0.028)	7.0e−5 (0.030)	3.1e−4 (0.029)
4	7.0e−5 (0.043)	3.5e−4 (0.031)	2.1e−4 (0.026)	6.1e−4 (0.026)	3.8e−4 (0.029)
8	6.0e−4 (0.042)	4.0e−4 (0.035)	1.2e−4 (0.028)	4.5e−4 (0.026)	5.6e−4 (0.027)
16	3.1e−4 (0.046)	3.1e−4 (0.038)	1.0e−4 (0.034)	2.1e−4 (0.031)	2.2e−4 (0.032)
32	2.2e−4 (0.046)	1.7e−4 (0.041)	1.2e−4 (0.033)	6.6e−4 (0.033)	3.7e−4 (0.033)
64	4.7e−4 (0.047)	8.9e−4 (0.042)	3.2e−3 (0.036)	4.8e−4 (0.032)	6.0e−4 (0.037)
128	3.1e−4 (0.050)	2.0e−4 (0.044)	1.6e−4 (0.037)	6.4e−4 (0.034)	2.7 (0.344)

we note that similar experiments have been reported in [111]. We repeat them to confirm that the results reported in [111] also hold true for our setup.

Purpose: To show how the presence (or lack thereof) of noise in the training data affects the prediction quality.

Setup: These experiments are performed on a validation set of size 4,000, comparing the following three different noise configurations:

- (0 | 0): Both training and testing data do not contain noise.
- (0 | 1): Only the testing data contain noise.
- (1 | 1): Both training and testing data contain noise.

Aside from these different setups, we also vary the training size. We set the learning rate to 0.002 and train for 8,000 steps (64 epochs). We consider a DNN with 4 layers and $n_u = 64$ and a CNN with 5 layers and $n_f = 8$, which consist of a total 140,739 and 17,223 trainable parameters, respectively. We only estimate the model parameters $\boldsymbol{\theta}_{\text{dyn}}$ and explore how the different settings above affect the performance of our methodology. We only include time series data as features with $n_{\text{feat}} = 2{,}000$. We vary the number of training data.

Results: We report results in Table 17 and Table 18. Here, we show results for two results for the selected DNN ($n_u = 64$, with 4 layers) and CNN ($n_f = 8$, with 5 layers).

Observations: The observations are in line with those reported in [111]. The best performance is obtained for noise free data (both, in testing and training; setting (0 | 0)). Moreover, including noise in the training data is critical if the observations are noise (setting (1 | 1)). The performance deteriorate significantly if we do not include noise in the training data, trying to infer model parameters from noise observations (setting (0 | 1)).

B.3.3. Recovering Noise and Model Parameters: Intrinsic Noise. In this experiment, we estimate the model parameters $\boldsymbol{\theta}_{\text{dyn}} \in \mathbb{R}^3$ along with the noise parameter $\boldsymbol{\theta}_{\text{noise}} = \beta \in \mathbb{R}$ of the intrinsic noise model (see Subsection 2.3.2).

Purpose: To explore the performance of the proposed framework for estimating noise and model parameters in the presence of intrinsic noise (modeled as the solution of an SDE; see Subsection 2.3.2).

Setup: We consider a testing data set of size $n_{\text{test}} = 4{,}000$. No validation data is used. Training and testing data sets both include noise. Like in the former experiment, we consider three different feature configurations, namely, TS, FC, and a concatenation of the two, TS&FC. The TS data has length $n_t = 2{,}000$. We recover $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \beta) \in \mathbb{R}^4$. The learning rate is set to 0.002 and the batch size is 32. We consider

TABLE 17

MdAPE and CDET values (in brackets) of model parameter predictions with noisy observational data. We consider the following setups: no noise in training and testing (label: (0 | 0)); no noise in training, noise in testing (label: (0 | 1)); noise in training and testing (label: (1 | 1)). We consider a DNN with 4 layers with $n_u = 64$.

n_{train}	(0 | 0)	(0 | 1)	(1 | 1)
500	0.079 (0.780)	0.100 (0.743)	0.127 (0.695)
1,000	0.047 (0.888)	0.080 (0.834)	0.096 (0.792)
2,000	0.033 (0.925)	0.080 (0.855)	0.079 (0.853)
4,000	0.046 (0.947)	0.098 (0.838)	0.063 (0.896)
8,000	0.024 (0.976)	0.093 (0.818)	0.060 (0.911)

TABLE 18

MdAPE and CDET values (in brackets) of model parameter predictions with noisy observational data. We consider the following setups: no noise in training and testing (label: (0 | 0)); no noise in training, noise in testing (label: (0 | 1)); noise in training and testing (label: (1 | 1)). We consider a CNN with 5 layers and $n_f = 32$.

n_{train}	(0 | 0)	(0 | 1)	(1 | 1)
500	0.022 (0.957)	0.092 (0.833)	0.068 (0.888)
1,000	0.022 (0.964)	0.086 (0.830)	0.061 (0.899)
2,000	0.020 (0.969)	0.087 (0.839)	0.064 (0.918)
4,000	0.012 (0.991)	0.104 (0.780)	0.049 (0.941)
8,000	0.008 (0.996)	0.126 (0.630)	0.047 (0.947)

different training sizes. We report results after 8,000 training steps (64 epochs).

Results: We report values for the MdAPE and CDET scores for the recovery of the individual parameters in Table 19. We show scatter plots for the prediction of the model parameters in Figure 5 (middle block) for a training size of $n_{\text{train}} = 8{,}000$. The horizontal and vertical axes correspond to the true and predicted values, respectively. Optimal predictions are indicated by a diagonal red line.

Observations: We notice similar patterns as in the previous experiment with additive noise. Time series features alone (with smaller training sizes) yields a poor recovery of the noise model parameter β. Overall, the performance is slightly better than for additive noise; we do no longer observe negative values for $\varepsilon_{\text{CDET}}$. Using Fourier coefficients alone yields a poor recovery of the model parameters $\boldsymbol{\theta}_{\text{dyn}}$ of the dynamical system. Combining both features, time series and Fourier coefficients (TS&FC setting) yields the overall best recovery. Compared with the former experiment, we also require less training data.

B.3.4. Predicted State. Here, we include additional visualizations for the results reported in the main manuscript. In particular, we include plots for the state variable associated with the predicted parameters.

Purpose: We want to explore how the accuracy in the prediction of the parameters of the dynamical system translates relates to errors in the predicted state; this is critical since during inference the NN predicts quantities of interest without considering (i.e., minimizing) the mismatch between predicted and observed state. That is, our predictor is "likelihood free." This might be a significant issue when it comes to actual applications. If we are, for example, solely interested in the underlying model parameters, errors in the parameter predictions for the dynamical system are our primary concern. However, if we are actually interested in the associated state of our system, errors in the predicted state are critical. Since we do not control these errors during inference, it is important to gain an understanding of how well the predicted states match the true state and observed data.

Setup: We jointly estimate model parameters $\boldsymbol{\theta}_{\text{model}}$, noise parameters ($\boldsymbol{\theta}_{\text{noise}} = (\rho, \sigma)$ for additive noise, $\boldsymbol{\theta}_{\text{noise}} = \beta$ for intrinsic noise, and $\boldsymbol{\theta}_{\text{noise}} = (\rho, \sigma, \beta)$ for the combination of these two noise models), as well as entries of the covariance matrix $\boldsymbol{\Gamma}_{\text{post}}$. The results reported below are obtained for a CNN with 5 layers and $n_f = 8$ and a training size of $n_{\text{train}} = 6{,}928$. We consider additive and intrinsic noise, as well as a combination thereof.

Results: We show representative forward simulation results for the membrane potential u in Figure 10. We show a comparison between the predicted membrane potential and the observed membrane potential for additive noise in Figure 11, for intrinsic noise in Figure 12, and for a combination of these two noise models in Figure 13. For each of these plots, we show from top to bottom increasing quantiles of the MSE between true and estimated time series.

Observations: The predictions for the cell membrane potential are in good agreement with the true

TABLE 19

Assessments of joint recovery of ODE model and noise parameters using observational data perturbed by an intrinsic noise model. We report values for ε_{MdAPE} and ε_{CDET} (in parenthesis). We use a CNN architecture with 5 layers and $n_f = 8$. We report results for a varying number of training samples, ranging from 500 to 8,000. We report results for different choices of the features we consider during training (TS: time series data; FC: Fourier coefficients; TS&FC: combination thereof). The remaining parameters are as identified in the text.

n_{train}	**feature**	θ_0	θ_1	θ_2	β
500	TS	0.137 (0.902)	0.206 (0.850)	0.042 (0.643)	0.187 (0.416)
	FC	0.310 (0.364)	0.423 (0.527)	0.040 (0.701)	0.068 (0.921)
	TS&FC	0.125 (0.910)	0.225 (0.817)	0.034 (0.788)	0.069 (0.914)
1,000	TS	0.131 (0.901)	0.217 (0.848)	0.041 (0.655)	0.165 (0.511)
	FC	0.298 (0.315)	0.360 (0.575)	0.037 (0.743)	0.069 (0.925)
	TS&FC	0.127 (0.914)	0.180 (0.897)	0.028 (0.851)	0.065 (0.921)
2,000	TS	0.135 (0.897)	0.210 (0.857)	0.042 (0.652)	0.161 (0.581)
	FC	0.251 (0.283)	0.201 (0.606)	0.036 (0.755)	0.064 (0.935)
	TS&FC	0.110 (0.926)	0.164 (0.906)	0.026 (0.867)	0.062 (0.941)
4,000	TS	0.133 (0.898)	0.200 (0.864)	0.040 (0.688)	0.147 (0.639)
	FC	0.215 (0.321)	0.190 (0.620)	0.034 (0.785)	0.066 (0.935)
	TS&FC	0.109 (0.931)	0.180 (0.904)	0.026 (0.880)	0.056 (0.951)
8,000	TS	0.123 (0.910)	0.179 (0.892)	0.038 (0.706)	0.131 (0.732)
	FC	0.276 (0.312)	0.302 (0.670)	0.033 (0.790)	0.060 (0.944)
	TS&FC	0.109 (0.931)	0.155 (0.919)	0.023 (0.895)	0.055 (0.956)

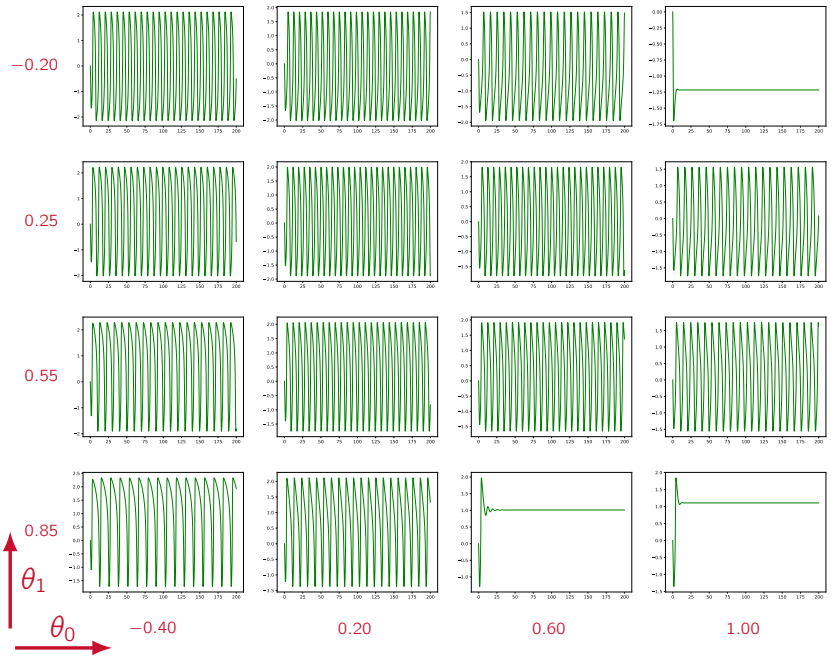

FIG. 10. *Forward simulation results for the membrane potential u for varying parameters $\boldsymbol{\theta}_{dyn}$. We set θ_2 to 0.2 and vary θ_0 and θ_1 as indicated. The simulation time is 200 ms.*

state in most cases, even if the perturbed (noisy) dynamics are quite different. This is particularly true for the additive noise model. For the intrinsic noise we can see larger deviations between the predicted, observed, and true state; small noise perturbation manifest in larger differences in the dynamics.

B.3.5. Non-Positive Definiteness in Predicted Covariance Data. If we use off-the-shelf NNs to predict the covariance matrices we cannot expect the predictions to be positive definite. We take advantage of the (in theory) symmetry of the covariance matrices and reduce complexity of the learning task by recovering only the upper triangular entries. Our testing set for the experiments with covariance data consists of of 3,456 samples. We remove the data that correspond to parameters that yield close to singular (severely ill-conditioned) and negative definite Hessian matrices from our training and testing datasets. We took this approach to remain consistent with our datasets used in prior experiments that did not involve the estimation

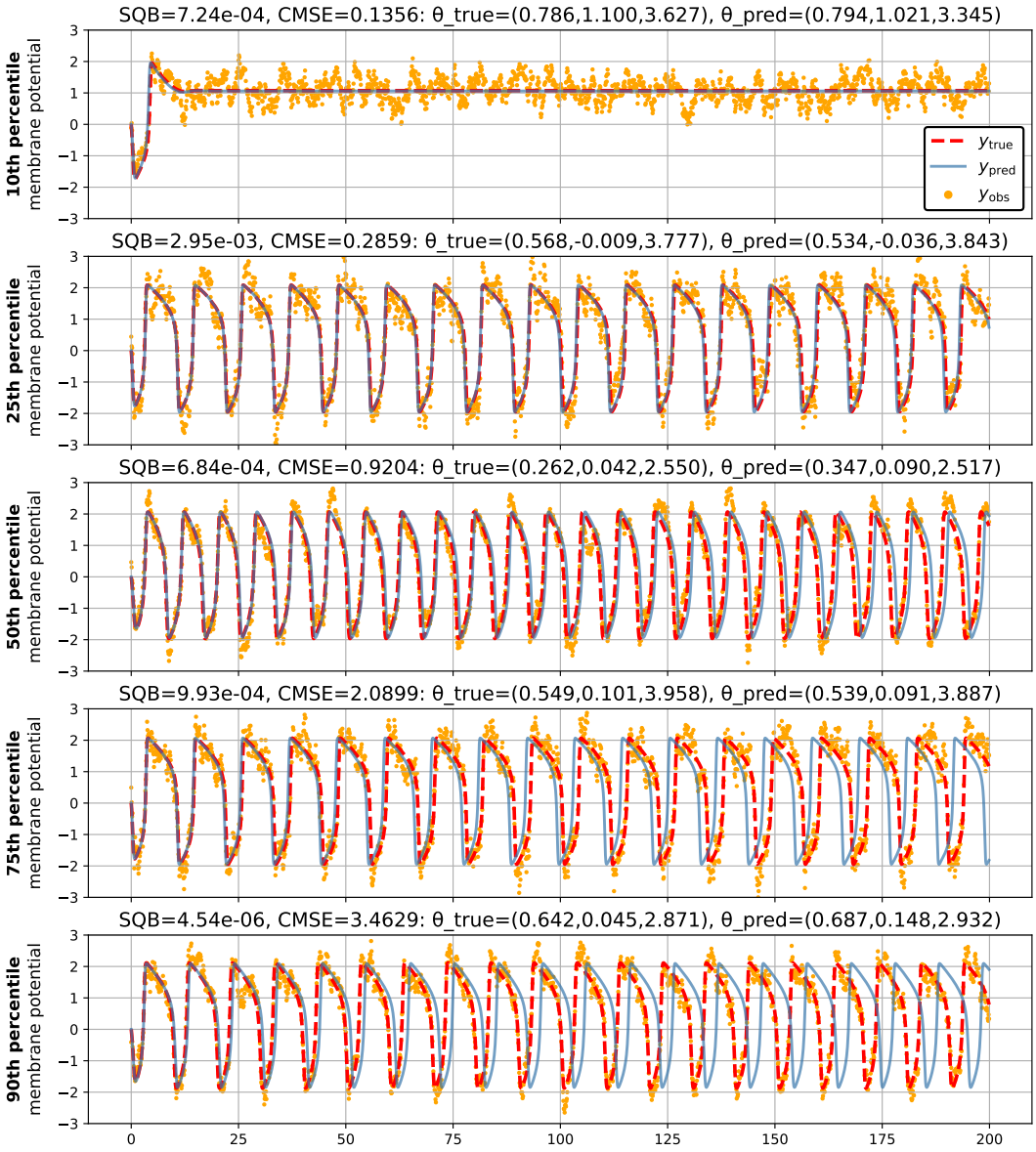

FIG. 11. *Simulation results for the state variable u for representative estimates $\boldsymbol{\theta}_{pred}$ of the model parameters $\boldsymbol{\theta}_{dyn}$. We consider an additive noise model. We show from top to bottom increasing quantiles of the MSE between the observed and estimated time series. The blue line corresponds to the membrane potential u generated using predicted parameters (i.e., solving the forward model). The orange dots represent the observations (perturbed membrane potential using an additive noise model). The red dashed line corresponds to the ground truth membrane potential. We jointly estimate model parameters $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2)$, noise parameters $\boldsymbol{\theta}_{noise} = (\rho, \sigma)$ (additive noise model), as well as entries of the covariance matrix $\boldsymbol{\Gamma}_{post}$. The number of training samples is $n_{train} = 6{,}928$. The number of testing samples is $n_{test} = 3{,}456$. On top of each graph we report the squared bias, the C-MSE, as well as the predicted ($\boldsymbol{\theta}_{pred}$) and true values ($\boldsymbol{\theta}_{true}$) for the model parameters $\boldsymbol{\theta}_{dyn}$.*

of covariance matrices. For the NN predictions we took a different approach.

Exemplary results for NN predictions of the covariance matrix are shown in Table 20. As we can see from these experiments, roughly 30% of the predictions are negative definite or indefinite. To address this issue one can, e.g., leverage the methodology described in [49]; we estimate the nearest positive definite matrix for the predicted covariances. The nearest positive definite matrix is computed by first symmetrizing the predicted matrix, say $\tilde{\boldsymbol{\Gamma}}_{\text{pred}}$, followed by an averaging with a symmetric matrix $\boldsymbol{\Gamma}_{\text{sym}}$ generated from the singular value decomposition of the matrix $\tilde{\boldsymbol{\Gamma}}_{\text{pred}}$. This average is symmetrized and adjusted (iteratively)

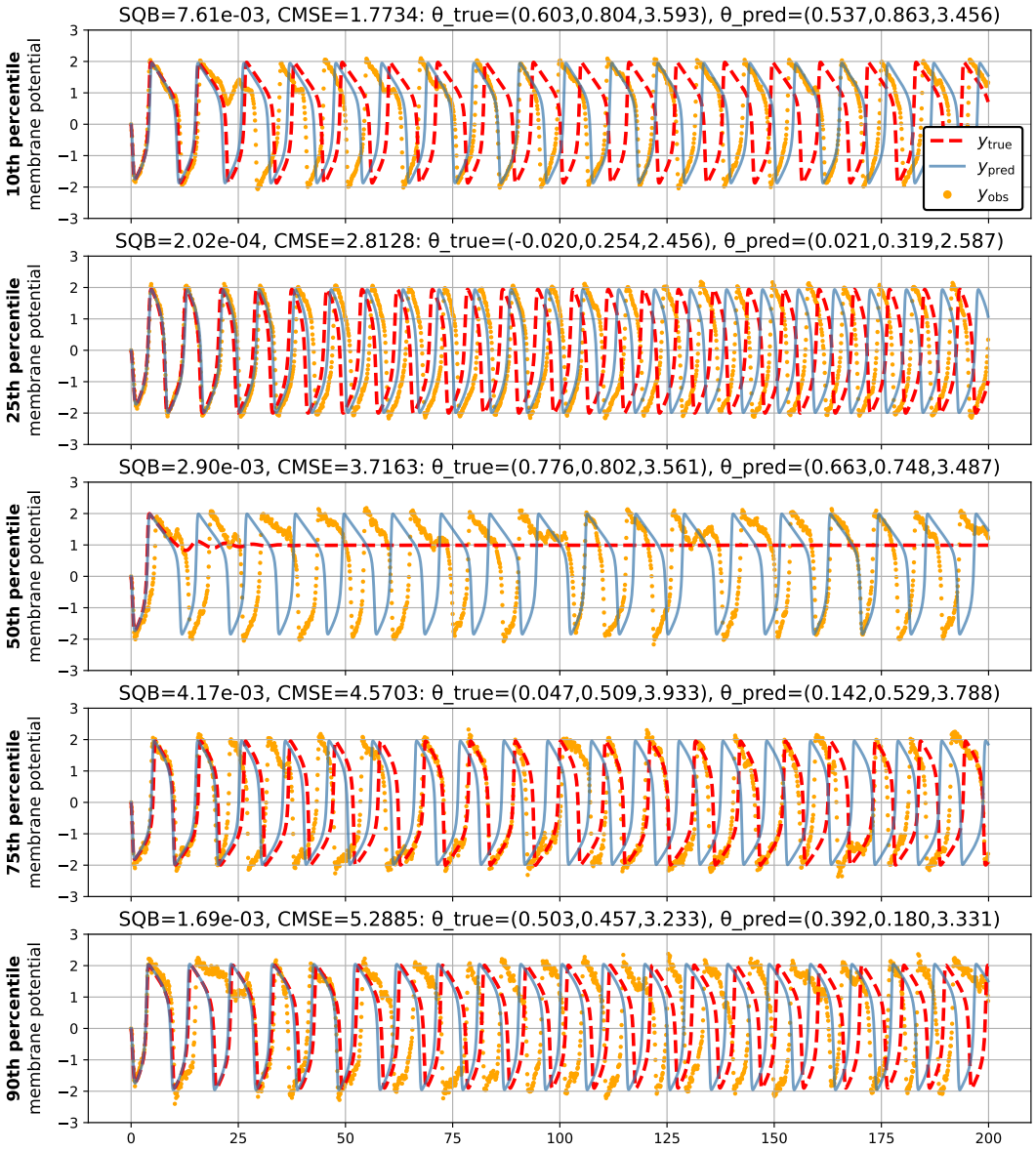

FIG. 12. *Simulation results for the state variable u for representative estimates $\boldsymbol{\theta}_{pred}$ of the model parameters $\boldsymbol{\theta}_{dyn}$. We consider an intrinsic noise model. We show from top to bottom increasing quantiles of the MSE between the observed and estimated time series. The blue line corresponds to the membrane potential u generated using predicted parameters (i.e., solving the forward model). The orange dots represent the observations (perturbed membrane potential using an additive noise model). The red dashed line corresponds to the ground truth membrane potential. We jointly estimate model parameters $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2)$, noise parameters $\boldsymbol{\theta}_{noise} = \beta$ (intrinsic noise model), as well as entries of the covariance matrix $\boldsymbol{\Gamma}_{post}$. The number of training samples is $n_{train} = 6{,}928$. The number of testing samples is $n_{test} = 3{,}456$. On top of each graph we report the squared bias, the C-MSE, as well as the predicted ($\boldsymbol{\theta}_{pred}$) and true values ($\boldsymbol{\theta}_{true}$) for the model parameters $\boldsymbol{\theta}_{dyn}$.*

through the addition of a positive multiple of the identity matrix. Specifically, the identity is scaled by the smallest eigenvalue of the previous iteration until a positive definite matrix is achieved. For details we refer to [65]. In the main manuscript we have developed a different strategy that maps the data to the tangent space.

We report results after mapping the predictions to the closest positive definite matrix as just described in Table 21.

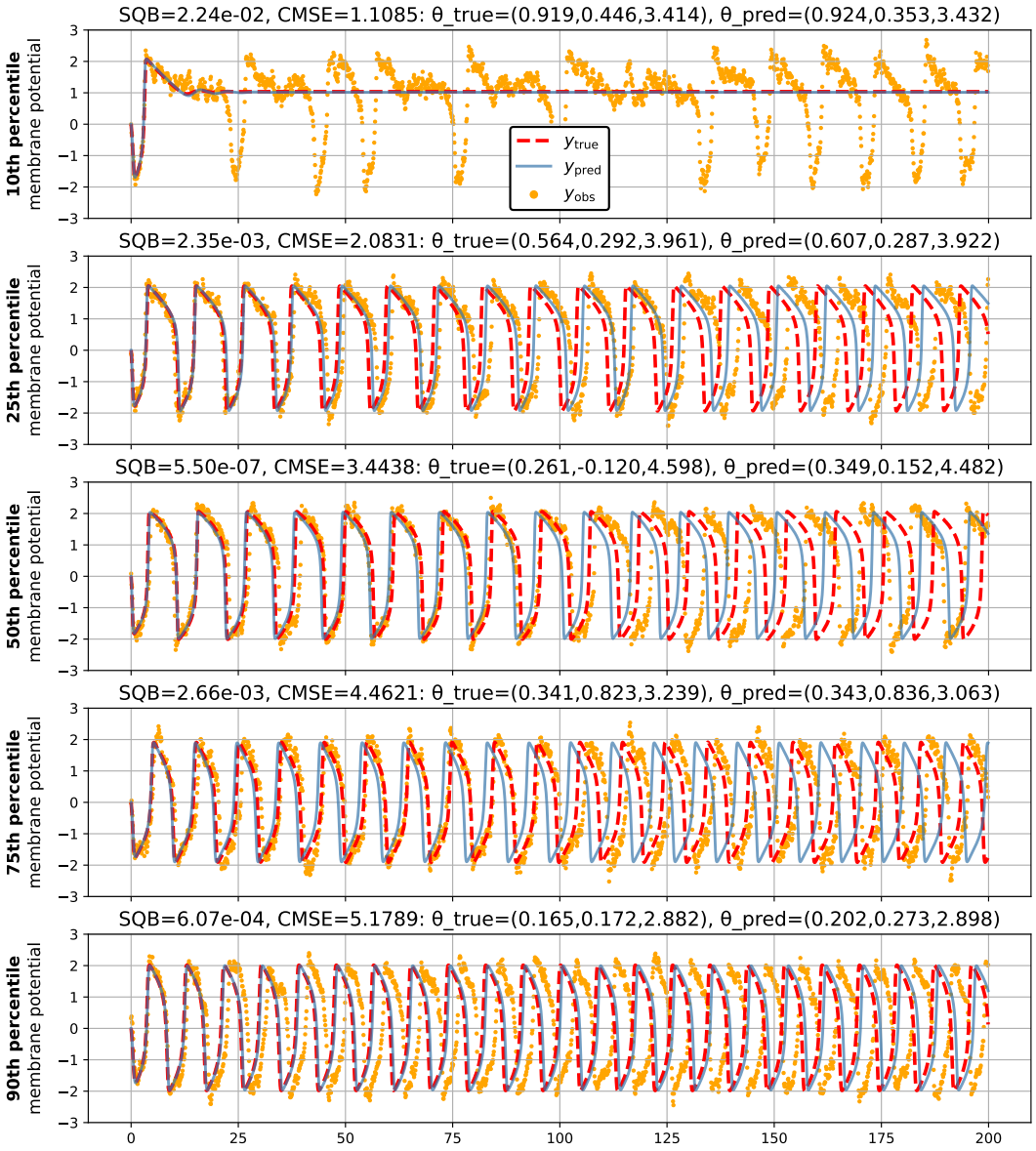

FIG. 13. *Simulation results for the state variable u for representative estimates $\boldsymbol{\theta}_{pred}$ of the model parameters $\boldsymbol{\theta}_{dyn}$. We perturb the simulations with an additive and an intrinsic noise model to obtain the observed membrane potential. We show from top to bottom increasing quantiles of the MSE between the observed and estimated time series. The blue line corresponds to the membrane potential u generated using predicted parameters (i.e., solving the forward model). The orange dots represent the observations (perturbed membrane potential using an additive noise model). The red dashed line corresponds to the ground truth membrane potential. We jointly estimate model parameters $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2)$, noise parameters $\boldsymbol{\theta}_{noise} = (\rho, \sigma, \beta)$ (combined noise model), as well as entries of the covariance matrix $\boldsymbol{\Gamma}_{post}$. The number of training samples is $n_{train} = 6{,}928$. The number of testing samples is $n_{test} = 3{,}456$. On top of each graph we report the squared bias, the C-MSE, as well as the predicted ($\boldsymbol{\theta}_{pred}$) and true values ($\boldsymbol{\theta}_{true}$) for the model parameters $\boldsymbol{\theta}_{dyn}$.*

TABLE 20

Predicted covariance matrices. We report the number of covariance matrices that are positive definite and negative definite/indefinite after feeding them through the trained NN, respectively. The testing data consists of 3,456 samples.

noise type	positive definite	negative definite/indefinite
additive	2399 ($\sim 70\%$)	1057 ($\sim 30\%$)
intrinsic	2573 ($\sim 74\%$)	883 ($\sim 26\%$)
combined	2195 ($\sim 64\%$)	1261 ($\sim 36\%$)

TABLE 21

MdAPE and CDET (in brackets) for the estimation of entries of the covariance matrix of the posterior distributions. Here, we map the predicted covariance matrices (from results in Subsection 3.2) that are negative definite (indefinite) to the nearest positive definite matrix. We apply the same methodology as described in [49].

n_{train}	**noise type**	Γ_{00}	Γ_{01}	Γ_{02}	Γ_{11}	Γ_{12}	Γ_{22}
872	additive	0.134 (0.742)	0.262 (0.770)	0.181 (0.850)	0.160 (0.615)	0.301 (0.788)	0.131 (0.817)
	intrinsic	0.165 (0.583)	0.299 (0.736)	0.204 (0.768)	0.172 (0.555)	0.300 (0.807)	0.145 (0.751)
	combined	0.141 (0.672)	0.298 (0.754)	0.202 (0.807)	0.174 (0.582)	0.324 (0.720)	0.139 (0.784)
1742	additive	0.129 (0.772)	0.254 (0.790)	0.173 (0.864)	0.156 (0.652)	0.264 (0.827)	0.125 (0.827)
	intrinsic	0.158 (0.595)	0.283 (0.759)	0.190 (0.800)	0.170 (0.586)	0.279 (0.824)	0.147 (0.749)
	combined	0.132 (0.716)	0.244 (0.817)	0.169 (0.858)	0.159 (0.648)	0.253 (0.845)	0.128 (0.822)
3480	additive	0.126 (0.782)	0.219 (0.828)	0.170 (0.861)	0.152 (0.659)	0.237 (0.859)	0.117 (0.841)
	intrinsic	0.150 (0.615)	0.280 (0.768)	0.189 (0.809)	0.157 (0.605)	0.281 (0.831)	0.141 (0.759)
	combined	0.125 (0.708)	0.244 (0.818)	0.165 (0.868)	0.150 (0.651)	0.247 (0.863)	0.124 (0.823)
6928	additive	0.117 (0.798)	0.219 (0.836)	0.154 (0.888)	0.141 (0.673)	0.218 (0.882)	0.111 (0.868)
	intrinsic	0.156 (0.617)	0.257 (0.793)	0.183 (0.808)	0.159 (0.604)	0.261 (0.859)	0.134 (0.774)
	combined	0.129 (0.721)	0.241 (0.815)	0.174 (0.858)	0.154 (0.627)	0.253 (0.851)	0.128 (0.818)

TABLE 22

Performance for the recovery of the model parameters $\boldsymbol{\theta}_{dyn}$ (left block) and covariance entries (right block) using a CNN across different random seeds. We report values for CDET, MdAPE, CMSE, and SQB. The training time is reported in seconds.

	parameter recovery				covariance entries				
id	**CDET**	**MdAPE**	**CMSE**	**SQB**	**CDET**	**MdAPE**	**CMSE**	**SQB**	**time**
1	0.858	0.071	3.0e−2	3.9e−4	0.798	0.170	1.2e−3	4.3e−6	72.532
2	0.850	0.073	3.3e−2	1.9e−4	0.791	0.176	1.3e−3	7.2e−6	73.382
3	0.858	0.071	3.0e−2	4.3e−4	0.801	0.173	1.2e−3	1.3e−5	73.975
4	0.861	0.069	2.9e−2	3.7e−4	0.805	0.171	1.2e−3	6.4e−6	73.575
5	0.849	0.074	3.1e−2	6.4e−5	0.780	0.178	1.4e−3	8.4e−6	73.611
6	0.865	0.069	2.8e−2	1.1e−3	0.797	0.173	1.2e−3	5.5e−6	74.776
7	0.845	0.075	3.3e−2	3.3e−4	0.777	0.179	1.4e−3	6.1e−6	72.970
8	0.861	0.071	2.8e−2	9.5e−4	0.793	0.174	1.3e−3	8.7e−6	73.525
9	0.856	0.071	3.1e−2	1.9e−4	0.792	0.171	1.3e−3	2.3e−6	72.808
10	0.853	0.073	3.2e−2	1.7e−4	0.778	0.177	1.4e−3	4.8e−6	73.168

TABLE 23

Performance for the recovery of the model parameters $\boldsymbol{\theta}_{dyn}$ (left block) and covariance entries (right block) using a DNN across different random seeds. We report values for CDET, MdAPE, CMSE, and SQB. The training time is reported in seconds.

	parameter recovery				covariance entries				
id	**CDET**	**MdAPE**	**CMSE**	**SQB**	**CDET**	**MdAPE**	**CMSE**	**SQB**	**time**
1	0.774	0.105	7.3e−2	6.5e−4	0.6	0.230	2.2e−3	2.2e−5	47.309
2	0.794	0.102	6.8e−2	2.2e−4	0.7	0.226	2.0e−3	1.4e−5	50.216
3	0.771	0.104	7.2e−2	1.2e−3	0.6	0.227	2.1e−3	3.1e−5	48.249
4	0.787	0.103	6.9e−2	4.3e−4	0.6	0.223	2.0e−3	2.9e−5	49.185
5	0.780	0.104	7.0e−2	4.9e−3	0.6	0.232	2.1e−3	4.7e−5	49.740
6	0.785	0.106	6.9e−2	1.3e−3	0.6	0.232	2.1e−3	3.0e−5	49.205
7	0.787	0.102	6.8e−2	2.4e−5	0.6	0.227	2.1e−3	2.2e−5	46.982
8	0.778	0.102	6.8e−2	1.9e−3	0.7	0.224	2.0e−3	1.8e−5	47.425
9	0.783	0.107	6.9e−2	1.6e−4	0.6	0.231	2.1e−3	1.5e−5	47.394
10	0.788	0.102	6.6e−2	1.4e−3	0.6	0.227	2.1e−3	2.5e−5	48.734

B.3.6. Sensitivity to Initialization. Here, we report more detailed results for the sensitivity of the considered NN architectures to the initialization of the NN weights. We consider 10 different random initialization.

Purpose: Assess the sensitivity of the NN predictions with respect to the initialization of the NN weights.

Setup: These results complement the summary results reported in Subsection 3.3. We refer to the main manuscript for details about the setup.

Results: We report results for CNN architecture in Table 22 for the CNN and Table 23 for the DNN, respectively. The training time for the CNN architecture is roughly 73.43 seconds total; the training time for the DNN averages 48.44 seconds total.

Observations: Since we report summary results in the main manuscript, the findings in this supplement are in line with the observations we made in the main manuscript: The NNs considered in this work are not sensitive to the initialization in the context we employ them.

B.3.7. Sensitivity to Initialization — Cross Validation. We additionally evaluate the quality of the selected NN models and the impact of different weight initializations through k-fold cross-validation with $k = 6$.

Purpose: To assess generalizability of the considered NN models independent of the dataset.

Setup: The dataset is partitioned into six equally sized subsets; in each fold, five subsets are used for training and one for testing. Evaluation metrics are recorded on the held-out test subset for each fold. These experiments follow the same configuration described in Subsection 3.2, using TS&FC as NN inputs, with observations generated from a combined noise model and using the full set of available samples. The setup and hyperparameters are held constant across runs, except for the random seed controlling initialization. We train for 100 epochs per fold to standardize across experiments.

Results: The aggregated results are presented in Table 24 and Table 25, respectively.

Observations: Similar to the results reported in the prior section, we do not observe a significant deterioration with respect to the initialization. Our cross validation results indicate that we have good generalization performance.

TABLE 24

6-fold cross-validation with (noisy) training and testing sets of sizes $n_{train} = 8653$ and $n_{test} = 1731$, respectively; random seeds for initializing NN weights vary per row. We consider a CNN with 5 layers, with a filter setting of $n_f = 8$. This table summarizes inference for all parameters, $\boldsymbol{\theta}_{dyn} = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$ and $\boldsymbol{\theta}_{noise} = (\beta, \rho, \sigma) \in \mathbb{R}^3$.

	CDET		MdAPE		SQB		CMSE	
id	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
1	0.839	0.005	0.077	0.002	1.1e−3	1.0e−3	3.4e−2	1.1e−3
2	0.834	0.007	0.078	0.004	5.4e−4	4.3e−4	3.5e−2	1.8e−3
3	0.841	0.009	0.078	0.003	9.4e−4	1.1e−3	3.5e−2	1.1e−3
4	0.837	0.005	0.080	0.002	6.1e−4	8.6e−4	3.4e−2	1.1e−3
5	0.838	0.007	0.078	0.002	2.1e−4	1.4e−4	3.4e−2	2.2e−3
6	0.837	0.007	0.078	0.002	3.3e−4	1.5e−4	3.4e−2	1.7e−3
7	0.833	0.007	0.079	0.001	9.1e−4	7.8e−4	3.6e−2	1.2e−3
8	0.837	0.005	0.078	0.002	5.2e−4	6.1e−4	3.4e−2	1.4e−3
9	0.840	0.006	0.077	0.002	5.1e−4	4.3e−4	3.4e−2	1.8e−3
10	0.838	0.005	0.078	0.001	4.5e−4	7.6e−4	3.4e−2	1.4e−3

TABLE 25

6-fold cross-validation with (noisy) training and testing sets of sizes $n_{train} = 8653$ and $n_{test} = 1731$, respectively. The random seeds used for initializing NN weights vary per row. We consider a CNN with 5 layers and filter setting of $n_f = 8$. This table summarizes inference for all entries of the covariance matrix.

	CDET		MdAPE		SQB		CMSE	
id	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
1	0.745	0.003	0.194	0.002	1.1e−5	7.0e−6	1.6e−3	3.4e−5
2	0.745	0.007	0.190	0.004	7.0e−6	4.0e−6	1.6e−3	4.8e−5
3	0.743	0.014	0.194	0.005	1.1e−5	1.1e−5	1.6e−3	6.1e−5
4	0.734	0.005	0.198	0.001	1.4e−5	1.3e−5	1.7e−3	2.9e−5
5	0.745	0.008	0.194	0.002	6.0e−6	3.0e−6	1.6e−3	5.6e−5
6	0.743	0.006	0.193	0.006	6.0e−6	5.0e−6	1.6e−3	6.4e−5
7	0.733	0.012	0.198	0.005	1.5e−5	1.2e−5	1.7e−3	9.0e−5
8	0.732	0.010	0.198	0.004	1.1e−5	1.1e−5	1.7e−3	6.4e−5
9	0.744	0.008	0.195	0.004	8.0e−6	8.0e−6	1.6e−3	4.7e−5
10	0.742	0.002	0.195	0.004	5.0e−6	4.0e−6	1.6e−3	1.4e−5

B.3.8. Performance as a Function of the Noise Perturbation. We study the prediction performance as a function of increasing noise perturbations. The results reported in this section complement those reported in Subsection 3.4.

Purpose: We explore the performance of the proposed framework as a function of increasing noise perturbations. We expect the prediction accuracy to deteriorate as the noise level increases. We note that these results are backed into those reported in prior experiments. We attempt to disentangle the errors and report prediction performance as a function of increasing noise.

Results: We provide 2D maps for the errors in the prediction of the parameters of the dynamical system and the covariance entries for the additive noise model in Figure 14. We report errors as a function of the noise parameters $\boldsymbol{\theta}_{\text{noise}} = (\sigma, \rho)$. We report results for the prediction of the parameters of the dynamical system (top row) and the covariance entries (bottom row). We include (left column to right column) the smallest error, the mean error, and the largest error across multiple datasets that share the same noise perturbations. We also include some visualization for the prediction of covariance matrices to provide some intuition. Results for increasing noise levels for the additive noise model are shown in Figure 15. Similar results are shown in Figure 16 for the intrinsic noise model.

Observations: The results included in this section again demonstrate that our framework remains relatively stable as the noise level increases. Compared to the box-whisker plots for the additive noise included in the main manuscript, the 2D maps shown in Figure 14 seem to hint at a trend that the error increases as both noise parameters increases (especially for the mean values shown in the middle columns). However, overall the trend is not as striking as for the intrinsic noise model (see Subsection 3.4 in the main manuscript).

REFERENCES

[1] A. M. Abdelaty, M. E. Fouda, and A. Eltawil, *Parameter estimation of two spiking neuron models with meta-heuristic optimization algorithms*, Frontiers in Neuroinformatics, 16 (2022), p. 771730.

[2] J. Adler and O. Öktem, *Solving ill-posed inverse problems using iterative deep neural networks*, Inverse Problems, 33 (2017), p. 124007.

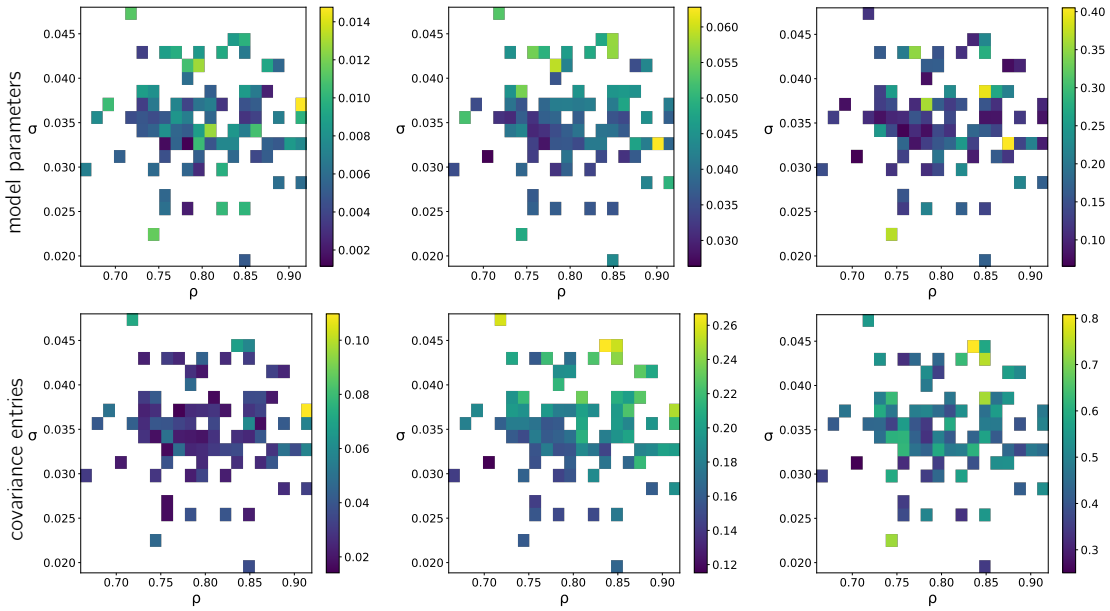

FIG. 14. *2D maps of the relative ℓ^2-error between NN predictions and true values. We bin the errors as a function of ρ and σ, respectively. The amplitude of the error is shown in color. The top row corresponds to the errors for the prediction of the parameters of the dynamical system. The bottom row shows the errors for the prediction of the covariance entries. We show (left column to right column) the minimum, mean, and maximum error.*

[3] J. Adler and O. Öktem, *Deep Bayesian inversion*, Datadriven Models in Inverse Problems, 31 (2024), pp. 359–412.

[4] B. M. Afkham, J. Chung, and M. Chung, *Learning regularization parameters of inverse problems via deep neural networks*, Inverse Problems, 37 (2021), p. 105017.

[5] G. S. Alberti, E. De Vito, M. Lassas, L. Ratti, and M. Santacesaria, *Learning the optimal Tikhonov regularizer for inverse problems*, Advances in Neural Information Processing Systems, 34 (2021), pp. 25205–25216.

[6] F. Altekrüger, A. Denker, P. Hagemann, J. Hertrich, P. Maass, and G. Steidl, *Patchnr: learning from very few images by patch normalizing flow regularization*, Inverse Problems, 39 (2023), p. 064006.

[7] S. Antholzer, M. Haltmeier, and J. Schwab, *Deep learning for photoacoustic tomography from sparse data*, Inverse Problems in Science and Engineering, 27 (2019), pp. 987–1005.

[8] H. Antil, H. C. Elman, A. Onwunta, and D. Verma, *A deep neural network approach for parameterized PDEs and Bayesian inverse problems*, Machine Learning: Science and Technology, 4 (2023), p. 035015.

[9] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, *fast and simple computations on tensors with log-Euclidean metrics*, 2005.

[10] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, *Log-Euclidean metrics for fast and simple calculus on diffusion tensors*, Magnetic Resonance in Medicine, 56 (2006), pp. 411–421.

[11] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, *Geometric means in a novel vector space structure on symmetric positive-definite matrices*, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 328–347.

[12] E. Bach, R. Baptista, D. Sanz-Alonso, and A. Stuart, *Inverse problems and data assimilation: A machine learning approach*, arXiv preprint arXiv:2410.10523, (2024).

[13] H. Bahl, V. Bresó, G. De Crescenzo, and T. Plehn, *Advancing tools for simulation-based inference*, arXiv preprint arXiv:2410.07315, (2024).

[14] G. Batzolis, J. Stanczuk, C.-B. Schönlieb, and C. Etmann, *Conditional image generation with score-based diffusion models*, arXiv preprint arXiv:2111.13606, (2021).

[15] M. Belkin, D. Hsu, S. Ma, and S. Mandal, *Reconciling modern machine-learning practice and the classical bias–variance trade-off*, Proceedings of the National Academy of Sciences, 116 (2019), pp. 15849–15854.

[16] G. Biros and O. Ghattas, *Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization—Part I: The Krylov-Schur solver*, SIAM Journal on Scientific Computing, 27 (2005), pp. 687–713.

[17] G. Biros and O. Ghattas, *Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization—Part II: The Lagrange-Newton solver and its application to optimal control of steady viscous flows*, SIAM Journal on Scientific Computing, 27 (2005), pp. 714–739.

[18] Y. E. Boink, M. Haltmeier, S. Holman, and J. Schwab, *Data-consistent neural networks for solving nonlinear inverse problems*, Inverse Problems and Imaging, 17 (2023), pp. 203–229.

[19] E. Buckwar, A. Samson, M. Tamborrino, and I. Tubikanec, *A splitting method for SDEs with locally Lipschitz drift: Illustration on the FitzHugh–Nagumo model*, Applied Numerical Mathematics, 179 (2022), pp. 191–220.

[20] L. Buhry, F. Grassia, A. Giremus, E. Grivel, S. Renaud, and S. Saïghi, *Automated parameter estimation of the Hodgkin–Huxley model using the differential evolution algorithm: Application to neuromimetic analog integrated*

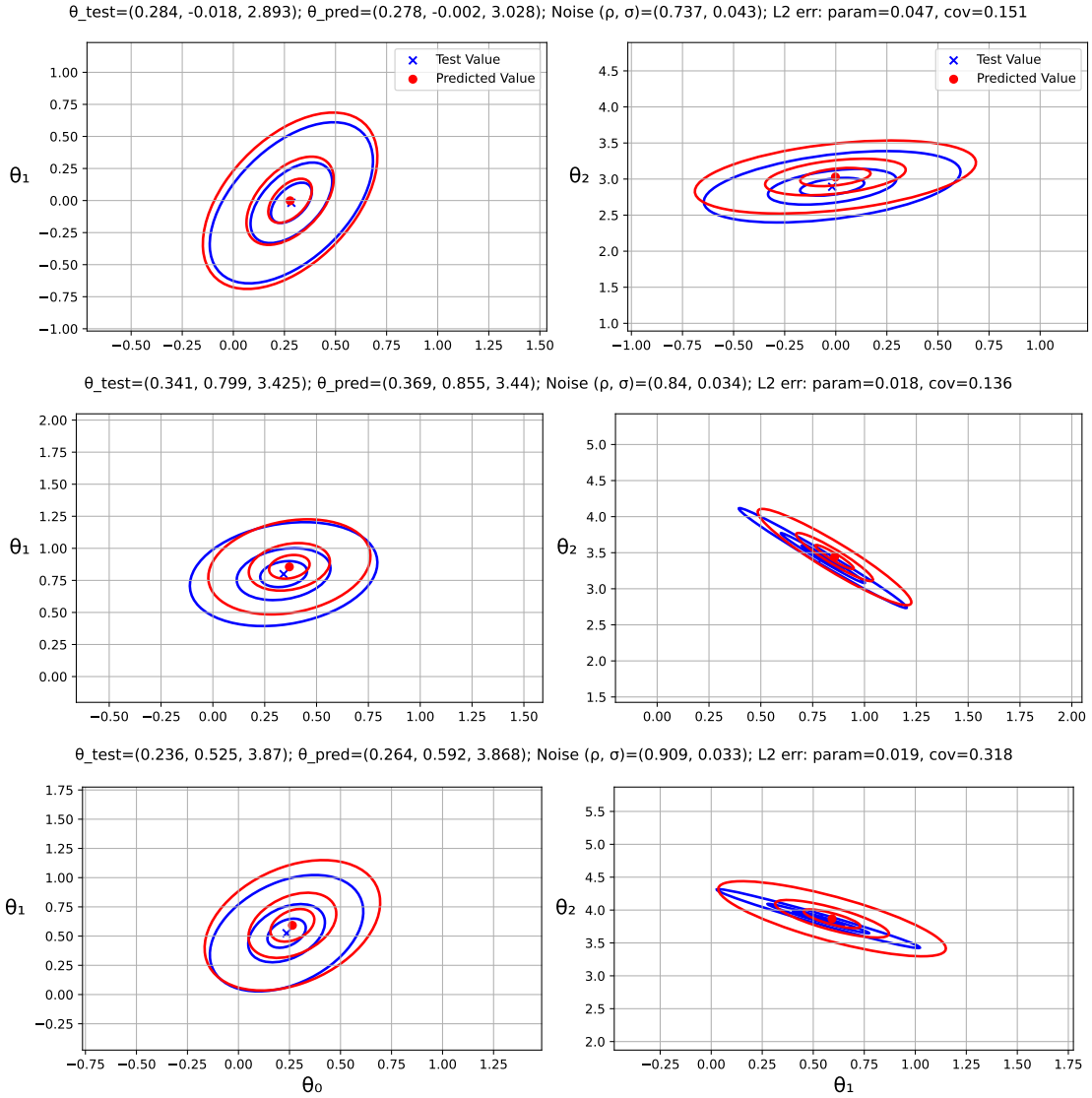

FIG. 15. *2D projections of covariance pairs for increasing noise levels (from top to bottom). We compare the NN prediction to the test data. The predictions are based on observations generated under the additive noise model. Each subplot shows 2D uncertainty ellipses computed from the underlying covariance matrices. Each row corresponds to the 10th, 50th, and 90th percentile noise levels (based on the ℓ-norm of the noise vector $\boldsymbol{\theta}_{noise} = (\sigma, \rho)$). The left column corresponds to the projection into the θ_1-θ_0 plane. The right column corresponds to the projection onto the θ_1-θ_2 plane. In the title of each pair of the plots (i.e., for each row), we include values for the predicted and true parameters of the dynamical system, the noise parameters, and the relative ℓ^2-norm for the prediction of the parameters of the dynamical system and the covariance matrix.*

circuits, Neural Computation, 23 (2011), pp. 2599–2625.

[21] L. Buhry, S. Saighi, A. Giremus, E. Grivel, and S. Renaud, *Parameter estimation of the Hodgkin–Huxley model using metaheuristics: Application to neuromimetic analog integrated circuits*, in 2008 IEEE Biomedical Circuits and Systems Conference, 2008, pp. 173–176.

[22] T. Bui-Thanh, O. Ghattas, J. Martin, and G. Stadler, *A computational framework for infinite-dimensional Bayesian inverse problems—Part I: The linearized case, with application to global seismic inversion*, SIAM Journal on Scientific Computing, 35 (2013), pp. A2494–A2523.

[23] M. J. Byrne and R. A. Renaut, *Learning spectral windowing parameters for regularization using unbiased predictive risk and generalized cross validation techniques for multiple data sets*, Inverse Problems and Imaging, 17 (2023), pp. 841–869.

[24] D. Calvetti and E. Somersalo, *Bayesian scientific computing*, vol. 215, Springer Nature, 2023.

[25] K. Campbell, L. Staugler, and A. Arnold, *Estimating time-varying applied current in the Hodgkin–Huxley model*, Applied Sciences, 10 (2020), p. 550.

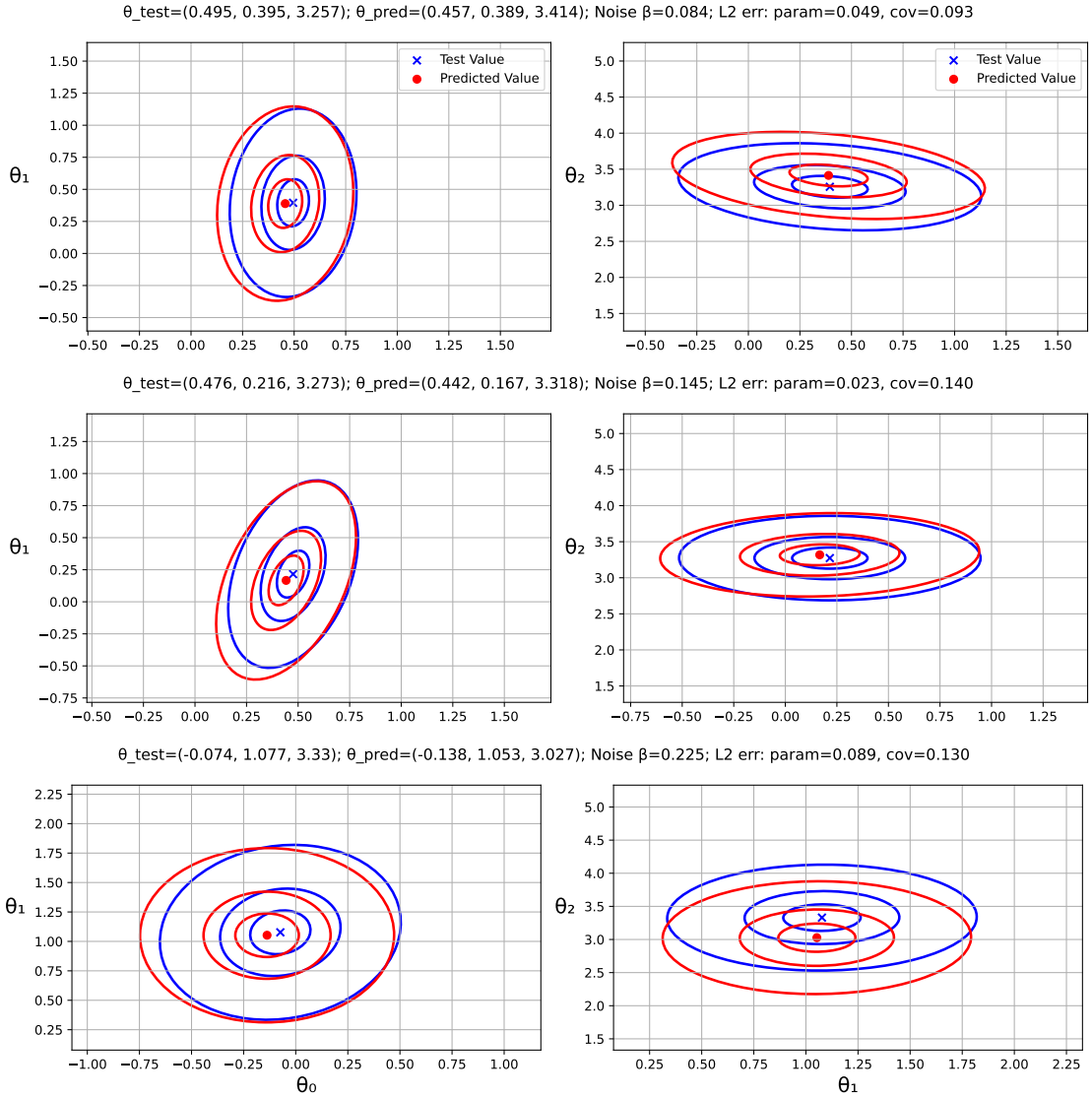

FIG. 16. *2D projections of covariance pairs for increasing noise levels (from top to bottom). We compare the NN prediction to the test data. The predictions are based on observations generated under the intrinsic noise model. Each subplot shows 2D uncertainty ellipses computed from the underlying covariance matrices. Each row corresponds to the 10th, 50th, and 90th percentile noise levels (for the absolute value of the noise parameter* $\boldsymbol{\theta}_{noise} = \beta$*). The left column corresponds to the projection into the* θ_1*-*θ_0 *plane. The right column corresponds to the projection onto the* θ_1*-*θ_2 *plane. In the title of each pair of the plots (i.e., for each row), we include values for the predicted and true parameters of the dynamical system, the noise parameters, and the relative* ℓ^2*-norm for the prediction of the parameters of the dynamical system and the covariance matrix.*

[26] H. CAO, C. TAN, Z. GAO, Y. XU, G. CHEN, P.-A. HENG, AND S. Z. LI, *A survey on generative diffusion models*, IEEE Transactions on Knowledge and Data Engineering, (2024).
[27] J. CASTRO, C. MUÑOZ, AND N. VALENZUELA, *The calderon's problem via deeponets*, arXiv preprint arXiv:2212.08941, (2022).
[28] D. CEBRIÁN-LACASA, P. PARRA-RIVAS, D. RUIZ-REYNÉS, AND L. GELENS, *Six decades of the Fitzhugh–Nagumo model: A guide through its spatio-temporal dynamics and influence across disciplines*, Physics Reports, 1096 (2024), pp. 1–39.
[29] R. T. Q. CHEN, Y. RUBANOVA, J. BETTENCOURT, AND D. K. DUVENAUD, *Neural ordinary differential equations*, Advances in Neural Information Processing Systems, 31 (2018).
[30] T. CHEN, X. CHEN, W. CHEN, Z. WANG, H. HEATON, J. LIU, AND W. YIN, *Learning to optimize: A primer and a benchmark*, The Journal of Machine Learning Research, 23 (2022), pp. 8562–8620.
[31] U. CHIALVA, V. GONZÁLEZ BOSCÁ, AND H. G. ROTSTEIN, *Low-dimensional models of single neurons: a review*, Biological Cybernetics, (2023), pp. 1–21.
[32] J. S. CHLEMPER, J. CABALLERO, J. HAJNAL, A. N. PRICE, AND D. RUECKERT, *A deep cascade of convolutional neural*

networks for dynamic mr image reconstruction, IEEE Transactions on Medical Imaging, 37 (2017), pp. 491–503.
[33] T. Cho, J. Chung, and J. Jiang, *Hybrid projection methods for large-scale inverse problems with mixed Gaussian priors*, Inverse Problems, 37 (2021), p. 044002.
[34] H. Chung, B. Sim, D. Ryu, and J. C. Ye, *Improving diffusion models for inverse problems using manifold constraints*, Advances in Neural Information Processing Systems, 35 (2022), pp. 25683–25696.
[35] M. Chung, E. Hart, J. Chung, B. Peters, and E. Haber, *Paired autoencoders for inverse problems*, arXiv preprint arXiv:2405.13220, (2024).
[36] S. Court, *Relaxation approach for learning neural network regularizers for a class of identification problems*, ArXiv Preprint, arXiv:2109.11658v2 (2023).
[37] K. Cranmer, J. Brehmer, and G. Louppe, *The frontier of simulation-based inference*, Proceedings of the National Academy of Sciences, 117 (2020), pp. 30055–30062.
[38] K. Csilléry, M. G. B. Blum, O. E. Gaggiotti, and O. François, *Approximate Bayesian computation (ABC) in practice*, Trends in Ecology & Evolution, 25 (2010), pp. 410–418.
[39] G. Daras, H. Chung, C.-H. Lai, Y. Mitsufuji, J. C. Ye, P. Milanfar, A. G. Dimakis, and M. Delbracio, *A survey on diffusion models for inverse problems*, arXiv preprint arXiv:2410.00083, (2024).
[40] A. Dasgupta, D. V. Patel, D. Ray, E. A. Johnson, and A. A. Oberai, *A dimension-reduced variational approach for solving physics-based inverse problems using generative adversarial network priors and normalizing flows*, Computer Methods in Applied Mechanics and Engineering, 420 (2024), p. 116682.
[41] A. Dasgupta, D. V. Patel, D. Ray, E. A. Johnson, and A. A. Oberai, *A dimension-reduced variational approach for solving physics-based inverse problems using generative adversarial network priors and normalizing flows*, Computer Methods in Applied Mechanics and Engineering, 420 (2024), p. 116682.
[42] A. Dasgupta, H. Ramaswamy, J. Murgoitio-Esandi, K. Y. Foo, R. Li, Q. Zhou, B. F. Kennedy, and A. A. Oberai, *Conditional score-based diffusion models for solving inverse elasticity problems*, Computer Methods in Applied Mechanics and Engineering, 433 (2025), p. 117425.
[43] A. Dasgupta, H. Ramaswamy, J. Murgoitio-Esandi, K. Y. Foo, R. Li, Q. Zhou, B. F. Kennedy, and A. A. Oberai, *Conditional score-based diffusion models for solving inverse elasticity problems*, Computer Methods in Applied Mechanics and Engineering, 433 (2025), p. 117425.
[44] S. Doi, Y. Onoda, and S. Kumagai, *Parameter estimation of various Hodgkin–Huxley-type neuronal models using a gradient-descent learning method*, in Proceedings of the 41st SICE Annual Conference, vol. 3, 2002, pp. 1685–1688.
[45] S. Druckmann, Y. Banitt, A. A. Gidon, F. Schürmann, H. Markram, and I. Segev, *A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data*, Frontiers in Neuroscience, 1 (2007), p. 56.
[46] C. Durkan, I. Murray, and G. Papamakarios, *On contrastive learning for likelihood-free inference*, in International Conference on Machine Learning, PMLR, 2020, pp. 2771–2781.
[47] H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of inverse problems*, vol. 375, Springer Science & Business Media, 1996.
[48] O. G. Ernst, B. Sprungk, and H.-J. Starkloff, *Bayesian inverse problems and Kalman filters*, Extraction of Quantifiable Information from Complex Systems, (2014), pp. 133–159.
[49] A. Fasih, *A Python/Numpy port of john d'errico's `nearestSPD` MATLAB code*, 2017, https://gist.github.com/fasiha/fdb5cec2054e6f1c6ae35476045a0bbd.
[50] Y. Feng, Y. Chen, P. Jin, S. Feng, Z. Liu, and Y. Lin, *Simplifying full waveform inversion via domain-independent self-supervised learning*, arXiv preprint arXiv:2305.13314, (2023).
[51] R. FitzHugh, *Impulses and physiological states in theoretical models of nerve membrane*, Biophysical Journal, 1 (1961), pp. 445–466.
[52] M. Gabrié, G. M. Rotskoff, and E. Vanden-Eijnden, *Adaptive Monte Carlo augmented with normalizing flows*, Proceedings of the National Academy of Sciences, 119 (2022), p. e2109420119.
[53] S. Geman, E. Bienenstock, and R. Doursat, *Neural networks and the bias/variance dilemma*, Neural Computation, 4 (1992), pp. 1–58.
[54] M. Genzel, J. Macdonald, and M. März, *Solving inverse problems with deep neural networks–robustness included?*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 45 (2022), pp. 1119–1134.
[55] M. Glöckler, M. Deistler, and J. H. Macke, *Variational methods for simulation-based inference*, in ICLR 2022, 2022.
[56] H. Goh, S. Sheriffdeen, J. Wittmer, and T. Bui-Thanh, *Solving Bayesian inverse problems via variational autoencoders*, in Mathematical and Scientific Machine Learning, 2022, pp. 386–425.
[57] P. J. Gonçalves, J.-M. Lueckmann, M. Deistler, M. Nonnenmacher, K. Öcal, G. Bassetto, C. Chintaluri, W. F. Podlaski, S. A. Haddad, T. P. Vogels, D. S. Greenberg, and J. H. Macke, *Training deep neural density estimators to identify mechanistic models of neural dynamics*, Elife, 9 (2020), p. e56261.
[58] D. Greenberg, M. Nonnenmacher, and J. Macke, *Automatic posterior transformation for likelihood-free inference*, in International Conference on Machine Learning, PMLR, 2019, pp. 2404–2414.
[59] M. D. Gunzburger, *Perspectives in flow control and optimization*, SIAM, Philadelphia, Pennsylvania, US, 2003.
[60] M. Gurkiewicz and A. Korngreen, *A numerical approach to ion channel modelling using whole-cell voltage-clamp recordings and a genetic algorithm*, PLoS Computational Biology, 3 (2007), p. e169.
[61] R. N. Gutenkunst, J. J. Waterfall, F. P. Casey, K. S. Brown, C. R. Myers, and J. P. Sethna, *Universally sloppy parameter sensitivities in systems biology models*, PLoS Computational Biology, 3 (2007), p. e189.
[62] A. Hartoyo, P. J. Cadusch, D. Liley, and D. G. Hicks, *Parameter estimation and identifiability in a neural population model for electro-cortical activity*, PLoS Computational Biology, 15 (2019), p. e1006694.
[63] J. Hermans, V. Begy, and G. Louppe, *Likelihood-free MCMC with amortized approximate likelihood ratios*, in International Conference on Machine Learning, PMLR, 2020.

[64] D. J. Higham, *Mean-square and asymptotic stability of the stochastic theta method*, SIAM Journal on Numerical Analysis, 38 (2000), pp. 753–769.
[65] N. J. Higham, *Computing a nearest symmetric positive semidefinite matrix*, Linear algebra and its applications, 103 (1988), pp. 103–118.
[66] A. L. Hodgkin and A. F. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve*, The Journal of Physiology, 117 (1952), pp. 500–544.
[67] D. Z. Huang, T. Schneider, and A. M. Stuart, *Iterated Kalman methodology for inverse problems*, Journal of Computational Physics, 463 (2022), p. 111262.
[68] A. C. Jensen, S. Ditlevsen, M. Kessler, and O. Papaspiliopoulos, *Markov chain Monte Carlo approach to parameter estimation in the Fitzhugh–Nagumo model*, Physical Review E—Statistical, Nonlinear, and Soft Matter Physics, 86 (2012), p. 041114.
[69] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, *Deep convolutional neural network for inverse problems in imaging*, IEEE Transactions on Image Processing, 26 (2017), pp. 4509–4522.
[70] J. Kaipio and E. Somersalo, *Statistical and computational inverse problems*, vol. 160, Springer Science & Business Media, 2006.
[71] D. P. Kingma and J. Ba, *ADAM: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).
[72] A. Kofler, F. Altekrüger, F. A. Ba, C. Kolbitsch, E. Papoutsellis, D. Schote, C. Sirotenko, F. F. Zimmermann, and K. Papafitsoros, *Learning regularization parameter-maps for variational image reconstruction using deep neural networks and algorithm unrolling*, arXiv preprint arXiv:2301.05888, (2023).
[73] M. Kostuk, B. A. Toth, C. D. Meliza, D. Margoliash, and H. Abarbanel, *Dynamical estimation of neuron and network properties II: Path integral Monte Carlo methods*, Biological Cybernetics, 106 (2012), pp. 155–167.
[74] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Communications of the ACM, 60 (2017), pp. 84–90.
[75] A. Lenzi, J. Bessac, J. Rudi, and M. L. Stein, *Neural networks for parameter estimation in intractable models*, Computational Statistics & Data Analysis, 185 (2023), p. 107762.
[76] J. R. León and A. Samson, *Hypoelliptic stochastic Fitzhugh–Nagumo neuronal model: Mixing, up-crossing and estimation of the spike rate*, Annals of Applied Probability, 28 (2018), pp. 2243–2274.
[77] H. Li, J. Schwab, S. Antholzer, and M. Haltmeier, *NETT: Solving inverse problems with deep neural networks*, Inverse Problems, 36 (2020), p. 065005.
[78] Y. Lin and Y. Wu, *InversionNet: A real-time and accurate full waveform inversion with convolutional neural network*, The Journal of the Acoustical Society of America, 144 (2018), pp. 1683–1683.
[79] J.-M. Lueckmann, J. Boelts, D. Greenberg, P. Goncalves, and J. Macke, *Benchmarking simulation-based inference*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 343–351.
[80] J.-M. Lueckmann, P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher, and J. H. Macke, *Flexible statistical inference for mechanistic models of neural dynamics*, Advances in Neural Information Processing Systems, 30 (2017).
[81] K. O. Lye, S. Mishra, D. Ray, and P. Chandrashekar, *Iterative surrogate model optimization (ISMO): An active learning algorithm for PDE constrained optimization with deep neural networks*, Computer Methods in Applied Mechanics and Engineering, 374 (2021), p. 113575.
[82] A. Mang and G. Biros, *A semi-Lagrangian two-level preconditioned Newton–Krylov solver for constrained diffeomorphic image registration*, SIAM Journal on Scientific Computing, 39 (2017), pp. B1064–B1101.
[83] A. Mang, A. Gholami, C. Davatzikos, and G. Biros, *CLAIRE: A distributed-memory solver for constrained large deformation diffeomorphic image registration*, SIAM Journal on Scientific Computing, 41 (2019), pp. C548–C584.
[84] A. Mang and L. Ruthotto, *A Lagrangian Gauss–Newton–Krylov solver for mass- and intensity-preserving diffeomorphic image registration*, SIAM Journal on Scientific Computing, 39 (2017), pp. B860–B885.
[85] J. Martin, L. C. Wilcox, C. Burstedde, and O. Ghattas, *A stochastic Newton MCMC method for large-scale statistical inverse problems with application to seismic inversion*, SIAM Journal on Scientific Computing, 34 (2012), pp. A1460–A1487.
[86] C. D. Meliza, M. Kostuk, H. Huang, A. Nogaret, D. Margoliash, and H. Abarbanel, *Estimating parameters and predicting membrane voltages with conductance-based neuron models*, Biological Cybernetics, 108 (2014), pp. 495–516.
[87] T. C. Mills, *Time series techniques for economists*, Cambridge University Press, 1990.
[88] S. Mukherjee, M. Carioni, O. Öktem, and C.-B. Schönlieb, *End-to-end reconstruction meets data-driven regularization for inverse problems*, in Advances in Neural Information Processing Systems, vol. 34, 2021, pp. 21413–21425.
[89] S. Mukherjee, A. Hauptmann, O. Öktem, M. Pereyra, and C.-B. Schönlieb, *Learned reconstruction methods with convergence guarantees: A survey of concepts and applications*, IEEE Signal Processing Magazine, 40 (2023), pp. 164–182.
[90] J. Nagumo, S. Arimoto, and S. Yoshizawa, *An active pulse transmission line simulating nerve axon*, Proceedings of the IRE, 50 (1962), pp. 2061–2070.
[91] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas, *A modern take on the bias-variance tradeoff in neural networks*, arXiv preprint arXiv:1810.08591, (2018).
[92] H. V. Nguyen and T. Bui-Thanh, *TNet: A model-constrained Tikhonov network approach for inverse problems*, SIAM Journal on Scientific Computing, 46 (2024), pp. C77–C100.
[93] J. Nocedal and S. J. Wright, *Numerical optimization*, Springer, 2006.
[94] D. Obmann, J. Schwab, and M. Haltmeier, *Deep synthesis network for regularizing inverse problems*, Inverse Problems, 37 (2020), p. 015005.
[95] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett, *Deep learning techniques for inverse problems in imaging*, IEEE Journal on Selected Areas in Information Theory, 1 (2020), pp. 39–56.
[96] S. Pakravan, P. A. Mistani, M. A. Aragon-Calvo, and F. Gibou, *Solving inverse-PDE problems with physics-aware*

neural networks, Journal of Computational Physics, 440 (2021), p. 110414.
[97] G. Papamakarios and I. Murray, *Fast ε-free inference of simulation models with Bayesian conditional density estimation*, Advances in Neural Information Processing Systems, 29 (2016).
[98] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, *Normalizing flows for probabilistic modeling and inference*, Journal of Machine Learning Research, 22 (2021), pp. 1–64.
[99] G. Papamakarios, D. Sterratt, and I. Murray, *Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows*, in The 22nd International Conference on Artificial Intelligence and Statistics, PMLR, 2019, pp. 837–848.
[100] G. Papamakarios, D. Sterratt, and I. Murray, *Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows*, in International Conference on Artificial Intelligence and Statistics, 2019, pp. 837–848.
[101] D. V. Patel, D. Ray, and A. A. Oberai, *Solution of physics-based Bayesian inverse problems with deep generative priors*, Computer Methods in Applied Mechanics and Engineering, 400 (2022), p. 115428.
[102] X. Pennec, P. Fillard, and N. Ayache, *A Riemannian framework for tensor computing*, International Journal of Computer Vision, 66 (2006), pp. 41–66.
[103] N. Petra, J. Martin, G. Stadler, and O. Ghattas, *A computational framework for infinite-dimensional Bayesian inverse problems, Part II: Stochastic Newton MCMC with application to ice sheet flow inverse problems*, SIAM Journal on Scientific Computing, 36 (2014), pp. A1525–A1555.
[104] A. A. Prinz, D. Bucher, and E. Marder, *Similar network activity from disparate circuit parameters*, Nature Neuroscience, 7 (2004), pp. 1345–1352.
[105] M. Puthawala, K. Kothari, M. Lassas, I. Dokmanić, and M. De Hoop, *Globally injective ReLu networks*, The Journal of Machine Learning Research, 23 (2022), pp. 4544–4598.
[106] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational physics, 378 (2019), pp. 686–707.
[107] J. O. Ramsay, G. Hooker, D. Campbell, and J. Cao, *Parameter estimation for differential equations: A generalized smoothing approach*, Journal of the Royal Statistical Society Series B: Statistical Methodology, 69 (2007), pp. 741–796.
[108] D. Ray, J. Murgoitio-Esandi, A. Dasgupta, and A. A. Oberai, *Solution of physics-based inverse problems using conditional generative adversarial networks with full gradient penalty*, Computer Methods in Applied Mechanics and Engineering, 417 (2023), p. 116338.
[109] D. Rezende and S. Mohamed, *Variational inference with normalizing flows*, in International Conference on Machine Learning, PMLR, 2015, pp. 1530–1538, https://proceedings.mlr.press/v37/rezende15.
[110] J. H. Rick C., C.-L. Li, B. Poczos, B. V. K. Vijaya K., and A. C. Sankaranarayanan, *One network to solve them all–solving linear inverse problems using deep projection models*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5888–5897.
[111] J. Rudi, J. Bessac, and A. Lenzi, *Parameter estimation with dense and convolutional neural networks applied to the Fitzhugh–Nagumo ODE*, in Mathematical and Scientific Machine Learning, 2022, pp. 781–808.
[112] M. Sainsbury-Dale, A. Zammit-Mangion, and R. Huser, *Likelihood-free parameter estimation with neural bayes estimators*, The American Statistician, 78 (2024), pp. 1–14.
[113] A. Samson, M. Tamborrino, and I. Tubikanec, *Inference for the stochastic FitzHugh–Nagumo model from real action potential data via approximate bayesian computation*, Computational Statistics & Data Analysis, 204 (2025), p. 108095.
[114] J. Schwab, S. Antholzer, and M. Haltmeier, *Deep null space learning for inverse problems: convergence analysis and rates*, Inverse Problems, 35 (2019), p. 025008.
[115] V. Shah and C. Hegde, *Solving linear inverse problems using GAN priors: An algorithm with provable guarantees*, in IEEE International Conference on Acoustics, Speech and Signal Processing, 2018, pp. 4609–4613.
[116] S. A. Sisson, Y. Fan, and M. Beaumont, *Handbook of approximate Bayesian computation*, CRC Press, 2018.
[117] S. A. Sisson, Y. Fan, and M. M. Tanaka, *Sequential Monte Carlo without likelihoods*, Proceedings of the National Academy of Sciences, 104 (2007), pp. 1760–1765.
[118] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, The Journal of Machine Learning Research, 15 (2014), pp. 1929–1958.
[119] A. M. Stuart, *Inverse problems: A Bayesian perspective*, Acta Numerica, 19 (2010), pp. 451–559.
[120] A. Tarantola, *Inverse problem theory and methods for model parameter estimation*, SIAM, 2005.
[121] K. E. Taylor, *Summarizing multiple aspects of model performance in a single diagram*, Journal of Geophysical Research: Atmospheres, 106 (2001), pp. 7183–7192.
[122] S. Tennøe, G. Halnes, and G. T. Einevoll, *Uncertainpy: a Python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience*, Frontiers in Neuroinformatics, 12 (2018), p. 49.
[123] L. Tenorio, *An introduction to data analysis and uncertainty quantification for inverse problems*, SIAM, 2017.
[124] B. A. Toth, M. Kostuk, C. D. Meliza, D. Margoliash, and H. Abarbanel, *Dynamical estimation of neuron and network properties I: Variational methods*, Biological Cybernetics, 105 (2011), pp. 217–237.
[125] F. Tröltzsch, *Optimal control of partial differential equations: Theory, methods, and applications*, vol. 112, American Mathematical Society, 2010.
[126] W. Van Geit, E. De Schutter, and P. Achard, *Automated neuron model optimization techniques: A review*, Biological Cybernetics, 99 (2008), pp. 241–251.
[127] D. V. Vavoulis, V. A. Straub, J. A. D. Aston, and J. Feng, *A self-organizing state-space-model approach for parameter estimation in hodgkin-huxley-type models of single neurons*, PLoS Computational Biology, 8 (2012), p. e1002401.
[128] G. Villalobos, *Scientific machine learning for Bayesian inverse problems governed by the Fitzhugh–Nagumo model*,

ph.d. dissertation, Department of Mathematics, University of Houston, 2023.
[129] S. Wang, M. A. Bhouri, and P. Perdikaris, *Fast PDE-constrained optimization via self-supervised operator learning*, arXiv preprint arXiv:2110.13297, (2021).
[130] Y. C. Wang, J. Rudi, J. Velasco, N. Sinha, G. Idumah, R. K. Powers, C. J. Heckman, and M. K. Chardon, *Multimodal parameter spaces of a complex multi-channel neuron model*, Frontiers in Systems Neuroscience, 16 (2022).
[131] T. Würfl, F. C. Ghesu, V. Christlein, and A. Maier, *Deep learning computed tomography*, in Medical Image Computing and Computer-Assisted Intervention, 2016, pp. 432–440.
[132] Z. Yang, Y. Yu, C. You, J. Steinhardt, and Y. Ma, *Rethinking bias-variance trade-off for generalization of neural networks*, in International Conference on Machine Learning, PMLR, 2020, pp. 10767–10777.
[133] H. Yu and X. Ji, *A near complete nonasymptotic generalization theory for multilayer neural networks: Beyond the bias-variance tradeoff*, arXiv preprint arXiv:2503.02129, (2025).
[134] K. Zeng, J. Yu, R. Wang, C. Li, and D. Tao, *Coupled deep autoencoder for single image super-resolution*, IEEE Transactions on Cybernetics, 47 (2015), pp. 27–37.