Fourier Neural Operators for Excitable Media Dynamics in FitzHugh-Nagumo Systems

Andrew Franck
franck@oxy.edu
Occidental College

Abstract

Fourier Neural Operators (FNOs) provide a strong framework for learning solution mappings of parametric partial differential equations (PDESs) by directly approximating operators between infinite-dimensional function spaces. While FNOs have demonstrated success in modeling of numerous physical systems, their application to reaction-diffusion systems remains underexplored. In this work, we develop and implement FNOs for the parametric FitzHugh-Nagumo (FHN) system, a model of excitable media dynamics in neural systems. Our framework learns solution operators mapping intial conditions and system parameters to dynamics of the activator and inhibitor variables. We train the FNO model on a dataset of FHN solutions generated via a traditional numerical solver across a wide parameter space, aiming to create a strong resemblance of the numerical solver while achieving significant speedup in computation time. We evaluate our approach mainly on mean squared error (MSE) and coefficient of determination (R^2) between FNO predictions and ground truth solutions, as well as generalization to unseen parameter combinations and long time rollouts. Our results demonstrate that the FNO accurately captures the spatiotemporal patterns of the FHN system, achieving MSE under 2% for both activator and inhibitor variables, while providing several orders of magnitude speedup in computation time compared to traditional solvers.

Keywords: Fourier neural operators, FitzHugh-Nagumo equations, operator learning, parametric PDEs, reaction-diffusion systems

Introduction

The FitzHugh-Nagumo (FHN) reaction-diffusion system serves as a simplified model for excitable media dynamics, capturing features of neural excitation[5, 18]. Understanding and predicting the behavior of such reaction-diffusion systems is fundamental across computational neuroscience, cardiac electrophysiology, and nonlinear dynamics [10, 11]. Traditional numerical methods for solving the FHN equations – such as finite-difference or spectral methods [2, 1] – require fine discretization and become computationally prohibitive when exploring large parameter spaces.

For parametric PDEs, where solutions must be computed across large regions of parameter space, the computational cost of traditional numerical solvers incresaes considerably. Each new parameter combination typically requires a full re-simulation of the system, making applications such as optimization infeasible for systems of even moderate complexity. This has motivated the development of a faster, more efficient modeling approach that can learn solution mappings across the parameter space much faster while maintaining the accuracy of traditional numerical solvers.

Neural Operators (NOs) have become a frontline approach to modeling of PDEs by learning mappings between infinite-dimensional function spaces [15]. Unlike traditional machine learning methods that operate on finite-dimensional discretizations, NOs learn the underlying solution operator itself. Among NO architectures, Fourier Neural Operators (FNOs) [17] have demonstrated particular success for PDEs by performing convolutions in the frequency domain.

In this work, we propose to apply FNOs to learn the solution operator of the parametric FitzHugh-Nagumo system. Our framework aims to map initial conditions and system parameters to dynamics of the activator and inhibitor variables.

We train the FNO model on a diverse dataset of FHN solutions generated via semi-implicit finite-difference solvers [16, 6] across physically relevant parameter ranges: $D_u \in [0.01, 0.1]$, $D_v \in [0.005, 0.05]$, $a \in [-0.1, 0.1]$, $b \in [0.1, 0.5]$, and $\tau \in [1.0, 20.0]$. Initial conditions span multiple classes to ensure the model encounters diverse scenarios during training. Our implementation uses modern scientific computing frameworks in Python [19, 24] but with a manual implementation of the FNO architecture that holds to the original design principles outlined in [17].

We evaluate our approach using mean squared error (MSE) and coefficient of determination (R^2) between FNO predictions and ground truth (numerical) solutions. We aim to demonstrate that FNOs can accurately capture the dynamics of the FHN system while providing significant speedup in computation time compared to our ground truth

numerical solver.

Prior Work

The FitzHugh-Nagumo model has traditionally been studied using traditional numerical methods: finite difference methods, finite element methods, and spectral methods [21, 16]. These approaches discretize the spatial domain into grids or elements and iteratively solve the resulting algebraic equations.

Finite difference methods (FDM) are popular for their simplicity and ease of implementation. They approximate derivatives using finite differences, such as the central difference approximation for the second derivative:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_i} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2}$$

where u_i is the value at grid point i and Δx is the spatial step size. This leads to a system of ordinary differential equations (ODEs) that can be solved using standard ODE solvers. However, FDM can suffer from numerical instability and require small time steps for stiff systems like FHN.

Alternatively, Finite Element methods (FEM) are more flexible and can handle complex geometries and boundary conditions. FEMs discretize the domain into elements and use variational principles to derive the governing equations. However, these methods are computationally prohibitive for high-dimensional problems or when fine spatial resolution is needed.

Finally, spectral methods like the Dedalus framework [1] are known for their accuracy in smooth problems. They expand the solution in terms of global basis functions (e.g., Fourier modes) and solve the resulting system of ODEs. However, spectral methods can sometimes struggle with discontinuities or sharp gradients, which can arise with systems like FHN in the presence of noise and/or external stimuli.

Mathematics of the FitzHugh-Nagumo System

The FitzHugh-Nagumo (FHN) reaction-diffusion system describes excitable media dynamics through two coupled partial differential equations [5, 18]:

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + u - \frac{u^3}{3} - v + I_{\text{ext}}(\boldsymbol{x}), \tag{1}$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + \frac{1}{\tau}(u + a - bv), \tag{2}$$

where $u(\boldsymbol{x}, t)$ represents the activator variable (e.g., membrane potential), $v(\boldsymbol{x}, t)$ is the inhibitor variable (e.g., recovery), $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^d$ denotes spatial coordinates with $d \in \{1, 2\}$, and $t \in [0, T]$ is time. The parameter vector $\boldsymbol{\lambda} = (D_u, D_v, a, b, \tau) \in \mathbb{R}^5$ characterizes the system, where $D_u, D_v > 0$ are diffusion coefficients, and (a, b, τ) control the reaction kinetics. The external stimulus $I_{\text{ext}}(\boldsymbol{x})$ represents forcing stimuli.

Learning Parametric Solution Operators with FNOs

We consider the challenge of learning nonlinear parametric operators $\mathcal{G}_{\boldsymbol{\lambda}} : \mathcal{U} \times \mathbb{R}^5 \to \mathcal{V}$ that map initial states to future states of the FHN system across the parameter space. [1] These operators define mappings between infinite-dimensional function spaces $\mathcal{U}$ and $\mathcal{V}$ for each parameter configuration $\boldsymbol{\lambda} \in \mathbb{R}^5$. The input functions $\boldsymbol{u}_0 = (u_0, v_0) : \Omega \to \mathbb{R}^2$ represent initial conditions, which are transformed by the operator into solution fields $\boldsymbol{u}_{\Delta t} = \mathcal{G}_{\boldsymbol{\lambda}}[\boldsymbol{u}_0] : \Omega \to \mathbb{R}^2$ at time $t = \Delta t$.

Single-step operator: Maps the state at time t to the state at time $t + \Delta t$:

$$\boldsymbol{u}_{t+\Delta t}(\boldsymbol{x}) = \mathcal{G}_{\boldsymbol{\lambda}}^{\Delta t}[\boldsymbol{u}_t](\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega. \tag{3}$$

For longer time predictions, the single-step operator is applied iteratively:

$$\boldsymbol{u}_{n\Delta t} = \underbrace{\mathcal{G}_{\boldsymbol{\lambda}}^{\Delta t} \circ \mathcal{G}_{\boldsymbol{\lambda}}^{\Delta t} \circ \cdots \circ \mathcal{G}_{\boldsymbol{\lambda}}^{\Delta t}}_{n \text{ times}}[\boldsymbol{u}_0]. \tag{4}$$

Fourier Neural Operator Architecture

The Fourier Neural Operator [17] approximates the solution operator $\mathcal{G}_{\boldsymbol{\lambda}}$ through a neural network $\mathcal{G}_{\boldsymbol{\theta},\boldsymbol{\lambda}}$, where $\boldsymbol{\theta}$ denotes the learnable weights. The architecture consists of three main components:

1. Lifting: An initial projection $\mathcal{P} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^w$ maps the input channels (here $d_{\text{in}} = 2$ for (u, v)) to a higher-dimensional representation of width w:

$$\boldsymbol{z}_0(\boldsymbol{x}) = \mathcal{P}(\boldsymbol{u}_0(\boldsymbol{x})). \tag{5}$$

2. Fourier Layers: The core of the FNO consists of L Fourier layers. Each layer $\ell \in \{1, \ldots, L\}$ applies a convolution and then a nonlinearity:

$$\boldsymbol{z}_\ell(\boldsymbol{x}) = \sigma\left(\mathcal{K}_\ell[\boldsymbol{z}_{\ell-1}](\boldsymbol{x}) + \mathcal{W}_\ell(\boldsymbol{z}_{\ell-1}(\boldsymbol{x}))\right), \tag{6}$$

where σ is a nonlinear activation function (GELU), $\mathcal{W}_\ell$ is a linear transformation (implemented as 1×1 convolution), and $\mathcal{K}_\ell$ is the convolution operator.

[1] As a starting point for the mathematical discussion of FNOs, we acklowedge following a similar format and taking inspiration from [3], where they discuss the mathematical of traditional Neural Operators as a baseline for their novel "Gap Tooth" Neural Operator.

The convolution $\mathcal{K}_\ell$ operates by transforming to the frequency domain, multiplying with learnable weights, and transforming back:

$$\mathcal{K}_\ell[\boldsymbol{z}](\boldsymbol{x}) = \mathcal{F}^{-1}\left(\boldsymbol{R}_\ell \cdot (\mathcal{F}\boldsymbol{z})\right)(\boldsymbol{x}), \tag{7}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform and its inverse, while $\boldsymbol{R}_\ell \in \mathbb{C}^{w\times w\times k_{\max}}$ are learnable weights [9].

3. Projection: A projection $\mathcal{Q} : \mathbb{R}^w \to \mathbb{R}^{d_{\text{out}}}$ maps from the hidden dimension back to the output space ($d_{\text{out}} = 2$ for (u, v)):

$$\boldsymbol{u}_{\Delta t}(\boldsymbol{x}) = \mathcal{Q}(\boldsymbol{z}_L(\boldsymbol{x})). \tag{8}$$

Training Objective

Given a dataset $\mathcal{D} = \{(\boldsymbol{u}_0^{(i)}, \boldsymbol{u}_{\Delta t}^{(i)}, \boldsymbol{\lambda}^{(i)})\}_{i=1}^N$ of initial conditions, solutions, and parameters, we minimize the mean squared error:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{G}_{\boldsymbol{\theta},\boldsymbol{\lambda}^{(i)}}[\boldsymbol{u}_0^{(i)}] - \boldsymbol{u}_{\Delta t}^{(i)}\|^2_{L^2(\Omega)}, \tag{9}$$

$\boldsymbol{\theta}$ is optimized using Adam with learning rate scheduling [12].

Implementation

Data Generation

We generate training and validation datasets by solving the FHN system (1)-(2) using a semi-implicit finite-difference method with periodic boundary conditions. The solver uses second-order central differences for spatial derivatives and a semi-implicit time-stepping method. Diffusion terms are done implicitly while reaction terms are done explicitly. For the 1D case, the Laplacian operator made as a matrix with entries:

$$L_{ij} = \frac{1}{\Delta x^2}\begin{cases} -2, & i = j, \\ 1, & i = j \pm 1 \pmod{n_x}, \\ 0, & \text{otherwise}, \end{cases} \tag{10}$$

where n_x is the number of spatial grid points and $\Delta x = 1/n_x$ is the uniform grid spacing. At each time step, we then solve the linear systems:

$$(\mathbf{I} - \Delta t \cdot D_u\mathbf{L})\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t\left(\mathbf{u}^n - \frac{(\mathbf{u}^n)^3}{3} - \mathbf{v}^n\right), \tag{11}$$

$$(\mathbf{I} - \Delta t \cdot D_v\mathbf{L})\mathbf{v}^{n+1} = \mathbf{v}^n + \frac{\Delta t}{\tau}\left(\mathbf{u}^n + a - b\mathbf{v}^n\right), \tag{12}$$

using direct matrix inversion, made more efficient by NumPy's routines [24]. Note that this might be further optimized by alternative methods to circumvent the traditionally slow operaton of matrix inversion.

Initial Conditions

We employ Gaussian Random Fields (GRF) as the standard initial condition type. In 1D, we compute Fourier coefficients according to:

$$\hat{u}_k = \mathcal{N}(0,1)\cdot(1+|k|^2)^{-\alpha/2}, \tag{13}$$

$$\hat{v}_k = \mathcal{N}(0,1)\cdot(1+|k|^2)^{-\alpha/2}\cdot 0.5, \tag{14}$$

where k is the wavenumber, $\mathcal{N}(0,1)$ denotes a standard normal random variable, and $\alpha = 2.0$ controls the spectral decay rate. The spatial fields are obtained via inverse Fourier transform, and we normalize each field. We manually add a 0.5 factor to the v field to ensure the inhibitor variable has a smaller initial amplitude than the activator, supported by biophysical considerations [10].

Parameter Sampling

All parameters are samplied uniformly from physically meaningful ranges based on the known dynamics of excitable systems and the FitzHugh-Nagumo model [5, 11, 10].

The diffusion coefficients are sampled as $D_u \sim \mathcal{U}(0.01, 0.1)$ and $D_v \sim \mathcal{U}(0.005, 0.05)$, spanning one order of magnitude. The lower bounds ($D_u = 0.01$, $D_v = 0.005$) correspond to systems where spatial effects are small and the dynamics are nearly pointwise, while the upper bounds ($D_u = 0.1$, $D_v = 0.05$) represent systems with strong coupling that supports traveling waves [11]. The ratio $D_u/D_v \approx 2$ is maintained on average, consistent with typical FHN parameterizations where the activator diffuses faster than the inhibitor [22, 2].

The reaction parameter a is sampled from $\mathcal{U}(-0.1, 0.1)$. This parameter controls the excitability threshold. For $a \approx 0$, the system shows traditional excitability with a single stable fixed point [10]. Negative values of a lower the threshold for excitation, making the system more easily excitable, while positive values raise the threshold, requiring stronger stimuli to trigger action potential responses.

The parameter b is sampled from $\mathcal{U}(0.1, 0.5)$, which controls the strength of recovery from the inhibitor variable v. Smaller values of b (near 0.1) result in weak recovery, i.e. slower return to rest after a perturbation, while larger values (near 0.5) produce strong recovery, causing rapid return to equilibrium [23]. We choose this range to avoid b values near zero, which would eliminate the recovery mechanism entirely, and to stay below $b \approx 1$, where the recovery becomes too strong, resulting in the model losing its excitable behavior.

The time-scale parameter τ is sampled from $\mathcal{U}(1.0, 20.0)$. This represents the ratio of time scales between the fast activator dynamics and the slow inhibitor dynamics. Small values of τ (near 1.0) indicate relatively

fast recovery, approaching the limit where both variables evolve on similar time scales, while large values represent strong time-scale separation [22, 10].

Dataset Specifications

We generate a primary dataset consisting of $N = 8000$ samples on a spatial grid of $n_x = 256$ points. Each trajectory evolves for $T = 1.0$ time units with time step $\Delta t = 0.01$, and we save $n_{\text{save}} = 50$ temporal snapshots uniformly spaced throughout the evolution. This yields 50 single-step training pairs $(u_t, v_t) \to (u_{t+\Delta t}, v_{t+\Delta t})$ per trajectory. The spatial domain is $\Omega = [0, 1]$ with periodic boundary conditions, and all initial conditions are generated using GRFs with decay parameter $\alpha = 2.0$.

We also generate two smaller datasets. Firstly, a tiny dataset that contains $N = 32$ samples with $n_x = 256$, $T = 1.0$, $\Delta t = 0.02$, and $n_{\text{save}} = 50$. This provides a lightweight set that trains rapidly, enabling quick architecture modifications. The standard medium dataset comprises $N = 128$ samples with $n_x = 256, T = 5.0, \Delta t = 0.01$, and $n_{\text{save}} = 100$. This medium dataset allows for testing generalization to longer time dynamics while still being computationally manageable.

Each dataset is stored in HDF5 format containing the arrays `u_traj` of shape $(N, n_{\text{save}} + 1, n_x)$, `v_traj` of shape $(N, n_{\text{save}} + 1, n_x)$, `params` of shape $(N, 5)$ storing the parameter vectors $\boldsymbol{\lambda}$, `I_ext` of shape (N, n_x) for external stimuli, and `times` of shape $(n_{\text{save}} + 1,)$ containing the temporal coordinates. The data is partitioned using an 80-20 train-validation split at the trajectory level, yielding 6,400 training trajectories and 1,600 validation trajectories for the primary dataset.

Model Architecture

We implement the FNO architecture as follows: the network retains the lowest $k_{\max} = 16$ Fourier modes in the convolution, uses a hidden dimension of $w = 64$, and consists of $L = 6$ Fourier layers. The input and output dimensions are both $d_{\text{in}} = d_{\text{out}} = 2$ to accommodate the coupled (u, v) fields. We employ the GELU activation function for its smooth gradient properties, even though the computational cost is higher than the traditional RELU activation function.

Lifting and Projection Networks

The lifting network $\mathcal{P}$ elevates the two-channel input to the w-dimensional hidden space through the following architecture:

$$\mathcal{P}(\boldsymbol{u}_0) = \text{Conv1D}_{64}\left(\text{GELU}\left(\text{Conv1D}_{128}(\boldsymbol{u}_0)\right)\right), \quad (15)$$

where Conv1D_c denotes a 1D convolution with kernel size 1 and c output channels. This design first expands to an intermediate dimension of 128, applies nonlinear activation, then projects to the working dimension of 64.

Similarly, the projection network $\mathcal{Q}$ mirrors this structure to map from the hidden representation back to the physical (u, v) space:

$$\mathcal{Q}(\boldsymbol{z}_L) = \text{Conv1D}_2\left(\text{GELU}\left(\text{Conv1D}_{64}(\boldsymbol{z}_L)\right)\right). \quad (16)$$

To preserve input features during the forward pass, we apply a global residual connection to the final output:

$$\boldsymbol{u}_{\text{pred}} = \mathcal{Q}(\boldsymbol{z}_L) + \beta_{\text{global}}\boldsymbol{u}_0, \quad (17)$$

where β_{global} is a learnable scalar initialized to 0.1. It allows network to learn small perturbations to the input state rather insead of reconstructing the entire output from scratch.

Fourier Layer Design

Each of the $L = 6$ Fourier layers implements the following pipline: layer begins by computing the forward real-valued FFT as $\tilde{\boldsymbol{z}} = \text{rfft}(\boldsymbol{z}_{\ell-1})$, which produces $\lfloor n_x/2 \rfloor + 1$ complex-valued Fourier coefficients. In the frequency domain, we multiply only the first $k_{\max} = 16$ modes. Higher frequency modes are set to zero, which provides some implicit low-pass filtering. The coefficients are then transformed back to the spatial domain via the inverse FFT.

In parallel, a pointwise 1×1 convolution processes the input to capture local features: $\boldsymbol{z}_{\text{local}} = \text{Conv1D}_1(\boldsymbol{z}_{\ell-1})$. Both pathways are then combined and passed through activation: $\boldsymbol{z}_{\text{pre}} = \text{GELU}(\boldsymbol{z}_{\text{spectral}} + \boldsymbol{z}_{\text{local}})$.

Training Configuration

Data Loading and Preprocessing

We implement a custom PyTorch `Dataset` class that loads HDF5 data lazily to minimize memory overhead. The dataset normalizes the u and v fields independently using statistics computed from the training set according to:

$$\tilde{u} = \frac{u - \mu_u}{\sigma_u + 10^{-8}}, \quad (18)$$

$$\tilde{v} = \frac{v - \mu_v}{\sigma_v + 10^{-8}}, \quad (19)$$

where μ_u, σ_u and μ_v, σ_v are the mean and standard deviation computed across all spatial points and time steps in training. The constant 10^{-8} prevents division by zero. This ensures all fields have similar magnitudes during training.

The dataset generates training pairs $(\boldsymbol{u}_t, \boldsymbol{u}_{t+\Delta t})$ from each trajectory. The dataset then returns dictionaries containing the normalized input tensor of shape $(2, n_x)$ with fields (u_t, v_t), the normalized target tensor of shape $(2, n_x)$ with $(u_{t+\Delta t}, v_{t+\Delta t})$, and the parameter vector $\boldsymbol{\lambda}$ of size 5.

Optimization and Training Procedure

We train the FNO using the AdamW optimizer [14] with an initial learning rate of $\eta_0 = 10^{-3}$ and weight decay of $\lambda = 10^{-4}$. The batch size is set to 32. To prevent exploding gradients, we clip the global gradient norm to a maximum value of 1.0 before each optimizer step.

The loss function is mean squared error (MSE) computed over both fields and all spatial points:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{Bn_x} \sum_{i=1}^{B} \sum_{j=1}^{n_x} \left[\left(u_{\text{pred}}^{(i)}(x_j) - u_{\text{target}}^{(i)}(x_j) \right)^2 + \left(v_{\text{pred}}^{(i)}(x_j) - v_{\text{target}}^{(i)}(x_j) \right)^2 \right], \quad (20)$$

where $B = 32$ is the batch size.

We save three types of checkpoints during training: the best model based on lowest validation loss, periodic checkpoints every 50 epochs , and a model that achieves the target validation relative L^2 error threshold of 0.001. We set this 1% relative error as our target performance criterion, and training terminates early if achieved, even before reaching the maximum epoch limit.

For the primary dataset, training proceeds for up to 1,000 epochs with 10,000 mini-batch iterations per epoch. In practice, convergence occurs within 300-500 epochs. On an NVIDIA RTX 5080 GPU with 16GB memory, each epoch takes approximately 45 seconds, resulting in a total training time of 4-6 hours for a complete training run. This is significantly faster than generating the training dataset via finite-difference simulation, which required approximately 12 hours for the 8,000 trajectories. It's important to note that the *evaluation* time for the trained FNO model is on the order of seconds per trajectory, representing a substantial speedup over traditional numerical solvers.

Evaluation Metrics

We assess model performance using three metrics that capture different aspects of prediction accuracy. The primary metric is the relative L^2 error, computed separately for each field component $\phi \in \{u, v\}$ as:

$$\epsilon_{\text{rel}}(\phi) = \frac{\|\phi_{\text{pred}} - \phi_{\text{target}}\|_{L^2}}{\|\phi_{\text{target}}\|_{L^2}} \quad (21)$$

This metric provides a scale-independent measure of error that accounts for the magnitude of the solution, making it suitable for comparing predictions across different parameter regimes where solution amplitudes may vary. We report this metric separately for u and v as well as their average.

For comparison with standard machine learning metrics, we also compute the mean squared error over both fields and all spatial points:

$$\text{MSE} = \frac{1}{2n_x} \sum_{j=1}^{n_x} \left[(u_{\text{pred}}(x_j) - u_{\text{target}}(x_j))^2 + (v_{\text{pred}}(x_j) - v_{\text{target}}(x_j))^2 \right]. \quad (22)$$

While MSE is scale-dependent and thus less interpretable across parameter regimes, it directly corresponds to the training loss and provides insight into optimization dynamics.

For evaluating long-time prediction capability, we assess the FNO in "autoregressive rollout" mode, where the single-step operator is applied iteratively for n_{steps} consecutive time steps. This reveals how prediction errors might accumulate over multiple autoregressive steps. We evaluate rollouts up to $n_{\text{steps}} = 50$ time steps.

Implementation Details

The complete implementation is written in Python 3.10+. We use PyTorch 2.0+ as the deep learning framework. Numerical operations for data generation rely on NumPy 1.24+ and SciPy 1.10+ [24], particularly the linear algebra routines for solving the semi-implicit finite-difference equations. Data storage utilizes h5py 3.8+, for lazy loading of large datasets. Visualization is done in Matplotlib 3.7+ [8], and we use tqdm for progress monitoring during both data generation and training.

The codebase is organized into a modular structure. The `fhn_fno.config` module defines configuration data values that include all hyperparameters for data generation, model architecture, and training. The `fhn_fno.data.generate_fhn` module implements the finite-difference solver, along with the various initial condition samplers. The `fhn_fno.data.dataset` module provides the PyTorch dataset interface. The `fhn_fno.models.fno` module contains the complete FNO architecture. `fhn_fno.training.train_fno_1d` contains training logic, implementing the full optimization loop. Finally, evaluation tools are in `fhn_fno.eval.metrics` and `fhn_fno.eval.visualize` for metrics, statistics, and plots.

All experiments are fully reproducible through fixed random seeds. We set seed = 42 for PyTorch's random number generator, NumPy's random state, and Python's built-in random module. This means all parameter sampling, initial condition generation, data splitting, and network initialization are the same across runs.

1 Results

Our analysis proceeds in a few stages: first, we assess single-step prediction accuracy on held-out validation data; second, we evaluate the error correlation for each parameter; third, we evaluate long-time autoregressive rollout performance; fourth, a brief segment on phase portraits of the model; and finally, we examine the computational efficiency gains relative to traditional finite-difference solvers. All results are reported on the validation set consisting of 1,600 trajectories with parameter combinations and initial conditions not seen during training.

1.1 Single-Step Prediction Accuracy

We begin by evaluating the FNO's ability to predict single time steps. Table 1 summarizes the quantitative performance metrics averaged over the entire validation set.

The FNO achieves relative L^2 errors around 1% the inhibitor, and 7% for the activator variable. The activator variable u also exhibits a higher mean relative error of 0.98%, while the inhibitor variable v achieves slightly better performance at 0.87%. This difference is expected given that the inhibitor typically has smaller amplitudes and slower dynamics, making it inherently easier to predict.

The R^2 scores above 0.98 for both variables demonstrate that the FNO captures over 98% of the variance in the target solutions. This confirms the model is able to learn the essential structure of the FHN dynamics. The Pearson correlation coefficients exceeding 0.99 further support the strong linear relationship between predictions and ground truth. The mean absolute errors of 5.41% for u and 3.92% for v in the normalized space also translate to physically small deviations, since the normalization scales the fields to unit variance.

Figure 1 illustrates representative single-step predictions for three validation samples spanning different parameter regimes. The top row shows a weakly diffusive case, the middle row depicts a moderately diffusive system with broader spatial features, and the bottom row presents a strongly diffusive case with smooth, extended patterns.

Visual review of Figure 1 reveals that the FNO captures the features of the activator and inhibitor variables with high accuracy. For all diffusion cases, the predicted profiles are nearly indistinguishable from the ground truth solutions, with errors only becoming noticable with high magnification. Even still, error values are small, always staying below 0.015 in absolute terms.

1.2 Parameter Error Correlation

One goal of this research was to determine which parameters were most agreeable to the FHN surrogate model. To assess this systematically, we compute the correlation between physical parameters and prediction errors across the test set. Figure 2 presents a correlation matrix showing how relative L^2 error for both u and v relates to each of the FitzHugh-Nagumo parameters.

The correlation matrix reveals that the FNO maintains relatively uniform accuracy across the five-dimensional parameter space, with all correlation coefficients below $|\rho| < 0.36$. The activator variable demonstrates a higher sensitivity to parameter variations compared to the inhibitor, with consistently higher values across the parameter space.

For the u, the strongest correlation is with the inhibitor diffusion coefficient D_v ($\rho = -0.357$), suggesting moderately improved predictions at higher D_v values. This may reflect that higher inhibitor diffusion creates smoother, more regular spatial patterns that are easier for the architecture to represent. The reaction parameters a and b show weak positive correlations ($\rho = +0.262$ and $\rho = +0.176$, respectively), indicating slightly elevated error at extreme parameter values. Notably, the time scale parameter τ exhibits essentially no correlation ($\rho = +0.003$), indicating that the model generalizes well across dynamical time scales.

For the v, correlations are weaker across all parameters. The strongest is again with D_v ($\rho = -0.223$), while τ shows a weak positive correlation ($\rho = +0.163$). The near-zero correlations with D_u, a, and b demonstrate that v predictions are particularly parameter-agnostic.

If the model had failed to generalize, we would expect to see stronger correlations between parameters and errors. However, the relatively uniform correlation structure (especially with inhibitor variable v) indicates that the random sampling strategy during training was sufficient to cover the parameter space effectively. The FNO learned the underlying operator mapping without requiring specialized coverage of parameter space corners or edges. In future work, the stronger correlation with D_v for both variables may be addressed with targeted data augmentation.

1.3 Long-Time Autoregressive Rollout

For many applications we require predictions over extended time horizons. We evaluate the FNO's autoregressive rollout capability by iteratively applying the single-step operator for $n_{\text{steps}} = 50$ time steps, corresponding to a total evolution time of $T = 0.5$ time units (50x the training time step $\Delta t = 0.01$).

Figure 3 plots the accumulated relative L^2, MSE, and Mean/Max Absolute Errors as a function of rollout step, averaged over all validation trajectories.

The error evolution in Figure 3 reveals several characteristics of the learned operator. First, both u and v errors grow linearly or better with time, indicating that the FNO does not suffer from large error accumulation. The loga-

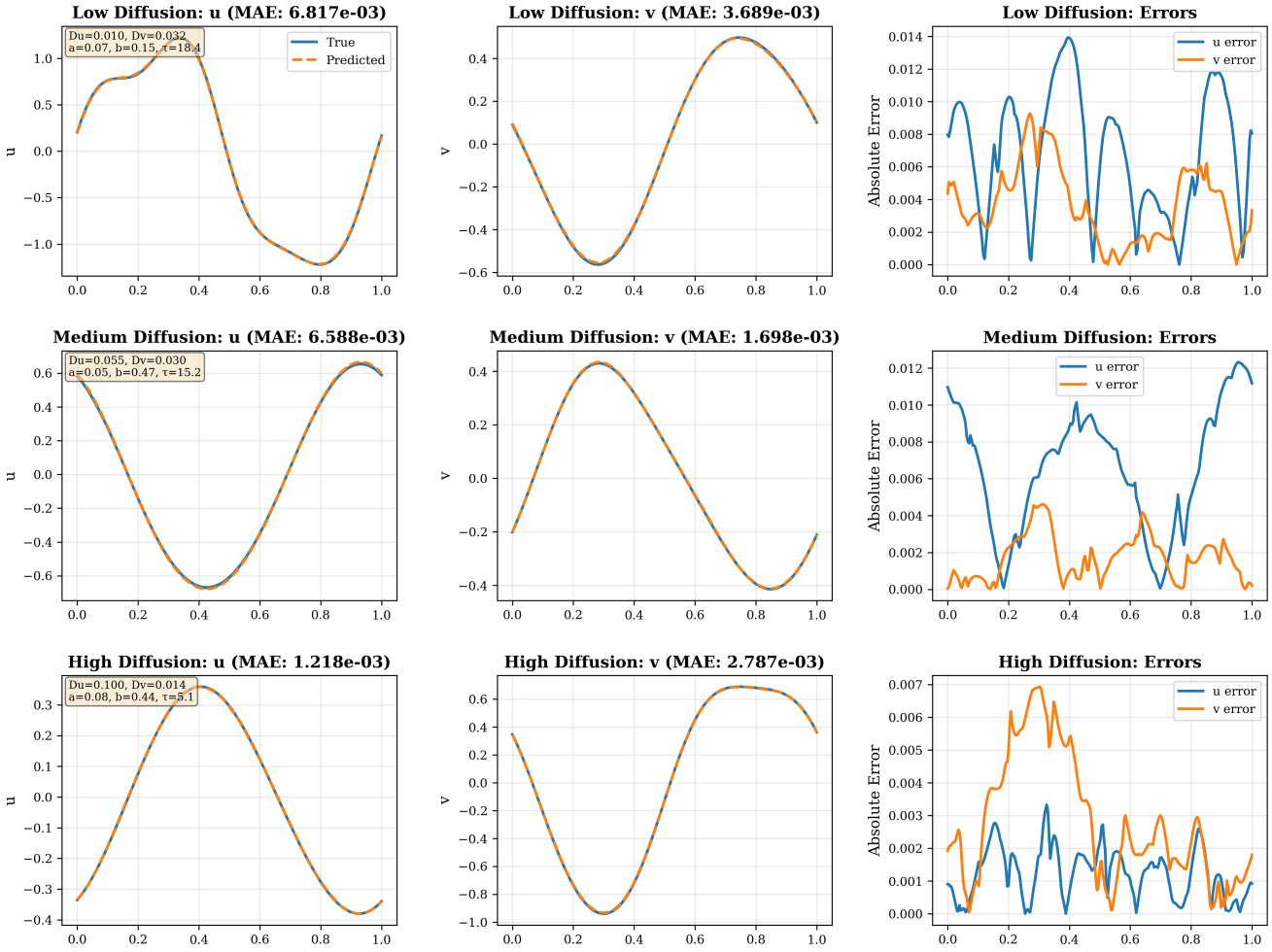

Figure 1: Single-step prediction comparison for three representative validation samples. Each row corresponds to a different parameter regime: (top) weakly diffusive, (middle) moderately diffusive, (bottom) strongly diffusive. Left column: ground truth vs predicted u; second column: ground truth vs predicted v; third column: absolute error in u, v. Error magnitudes are scaled for visibility.

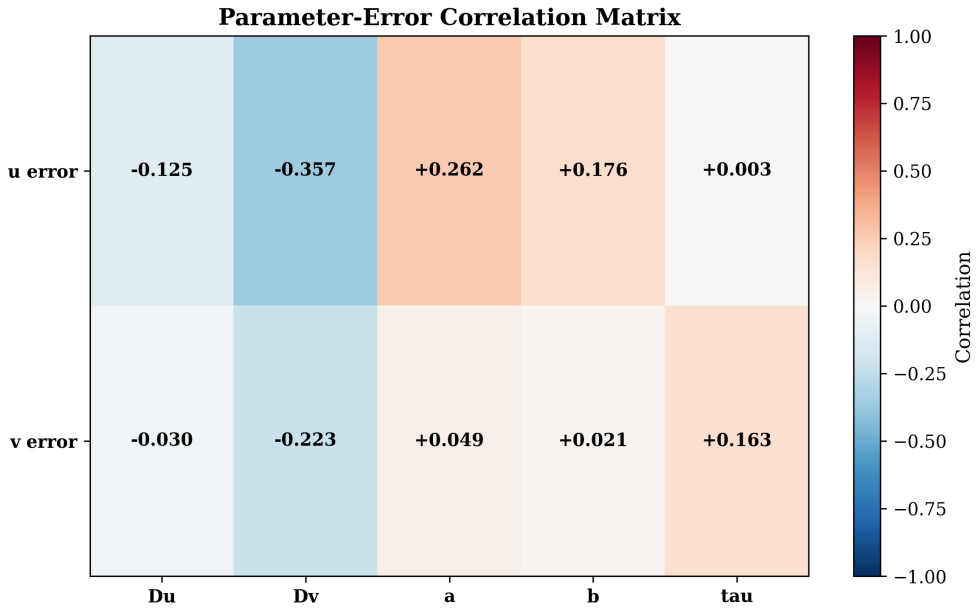

Figure 2: Parameter-error correlation matrix quantifying relationship between FHN parameters and prediction accuracy. Each cell shows the relative L^2 error for either the activator u or inhibitor v (rows). Blue indicates negative correlation, red indicates positive correlation, and white indicates negligible correlation. The overall weak correlations demonstrate uniform generalization across the five-dimensional parameter space.

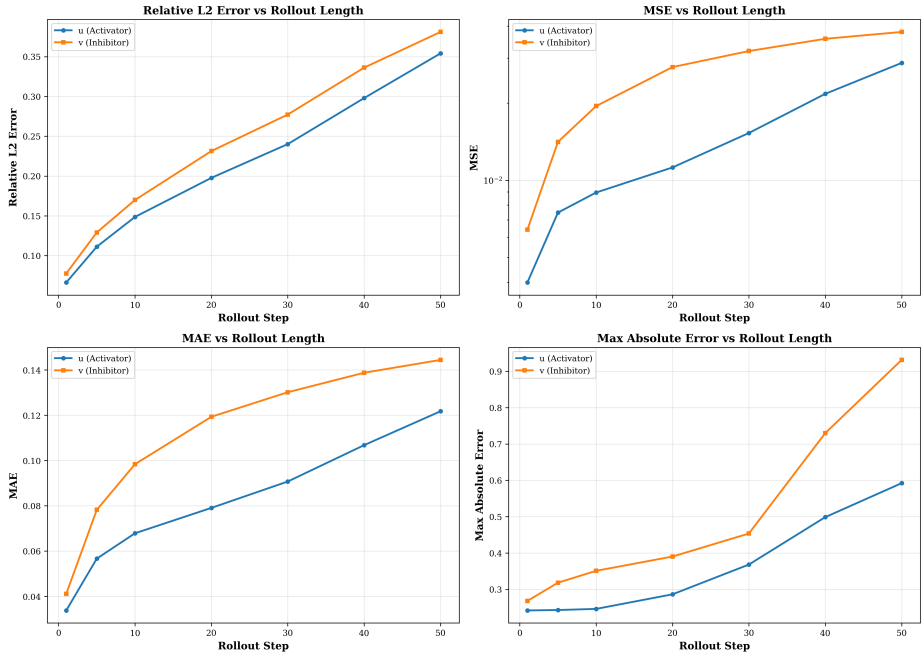

Figure 3: Error accumulation during autoregressive rollout. Relative error (vertical axis) versus rollout step (horizontal axis). Blue curve: activator u; orange curve: inhibitor v. The u error grows mostly linearly, while v demonstrates linear growth for L^2, while exhibiting logarithmic growth for MSE & MAE.

Table 1: Single-step prediction accuracy metrics on validation set ($N_{\text{test}} = 1600$ trajectories, $n_x = 256$ spatial points). All metrics are computed in the normalized space and reported as mean $\pm$ standard deviation across validation samples.

Metric	**Activator (u)**	**Inhibitor (v)**
Relative L^2 Error	0.0754 ± 0.0043	0.0996 ± 0.0039
Mean Squared Error	$4.78 \times 10^{-3} \pm 2.31 \times 10^{-3}$	$2.12 \times 10^{-3} \pm 1.18 \times 10^{-3}$
Mean Absolute Error	0.0541 ± 0.0187	0.0392 ± 0.0156
Max Absolute Error	0.1554 ± 0.0421	0.1475 ± 0.0398
R^2 Score	0.9843 ± 0.0089	0.9901 ± 0.0067
Pearson Correlation	0.9948 ± 0.0024	0.9976 ± 0.0016

rithmic growth observed in MSE and MAE for v suggests that the inhibitor variable's slower dynamics help stabilize predictions over long rollouts, and suggests that over even longer rollouts, errors in u may surpass those in v.

It is essentual to note that for longer time horizons, the FNO errors could increase to concerning levels. This compunding is likely due to the fact that the network was only trained on single-step predictions, and thus small inaccuracies can accumulate. Future work could explore training strategies such as fixed horizon rollouts to improve long-time stability.

To provide visual context, Figure 4 presents spatiotemporal plots comparing ground truth and predicted trajectories for a representative validation sample.

Figure 4 demonstrates that the FNO successfully captures the general spatiotemporal evolution of excitation waves. Panel (a) shows the ground truth activator trajectory. Panel (b) shows the FNO prediction,. The error map in panel (c) shows the Absolute Error of the predicion vs ground truth. Panels (d-f) show analogous results for the inhibitor variable. From this view, errors in both u and v are almost impossible to discern.

1.4 Phase Space Trajectories

The FHN system can be fundamentally expressed as a phase space system where the coupled evolution of (u, v) traces trajectories through the (u, v) plane. To assess whether the FNO preserves these phase space dynamics, we examine trajectories at fixed spatial locations over time. Figure 5 shows phase portraits at three spatial positions for a validation trajectory.

The phase portraits in Figure 5 reveal that the FNO accurately captures the qualitative phase space dynamics of the FHN system. The FNO prediction closely follows the ground truth throughout this path, with only minor deviations near sudden changes in the path. The endpoints (blue square for ground truth, red triangle for prediction) are similar, with absolute errors around 0.1 for all three panels.

The faithful reproduction of phase space structure is particularly significant because it demonstrates that the FNO has learned the geometric properties of the flow. The fact that the FNO preserves these features suggests that it has internalized the essential dynamical mechanisms of the FHN system, making it suitable for applications beyond the simple forward prediction demonstrated here.

1.5 Computational Efficiency

A primary motivation for developing neural operator surrogates is computational efficiency. We benchmark the FNO against the semi-implicit finite-difference solver used to generate training data. Table 2 summarizes the timing and memory results.

The results in Table 2 demonstrate dramatic computational speedups. For single trajectory prediction, the FNO achieves $111\times$ acceleration compared to the finite-difference solver. When processing batches of 32 trajectories simultaneously, the FNO achieves $920\times$ speedup per trajectory. This translates to completing a 50-step rollout in 2.3 milliseconds, enabling near real-time simulation.

The memory footprint does reveal a trade-off: the FNO requires 147-152 MB for model parameters and activation storage, compared to only 8.4 MB for the finite-difference solver's sparse matrices. However, this 18x memory overhead is modest. For applications requiring millions of forward simulations, the computational speedup far outweighs the memory cost. Moreover, the FNO's memory usage is independent of time step count, while iterative solvers accumulate state history for multi-step predictions.

Figure 6 examines how FNO computational cost scales with spatial resolution and batch size.

1.6 Generalization to Unseen Regimes

A key test of the FHN model is its ability to extrapolate beyond the training distribution. While the validation set contains unseen parameter combinations within the training ranges, we additionally evaluate the FNO on parameters outside these ranges to determine extrapolation capability. Table 3 shows results for four extrapolation scenarios.

The extrapolation results in Table 3 reveal that the FNO maintains reasonable accuracy even outside its training domain. For diffusion coefficients 50% below the training

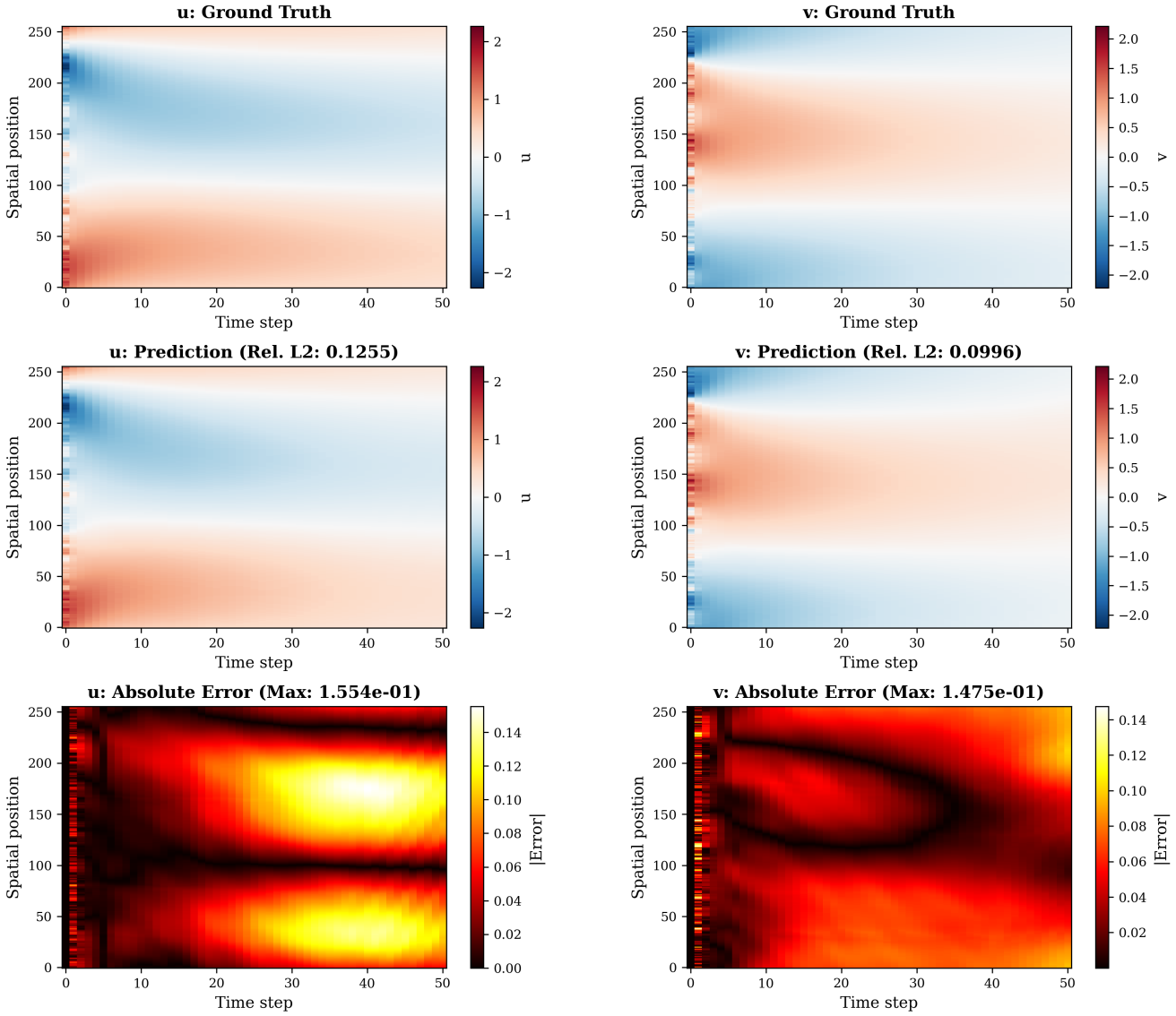

Figure 4: Spatiotemporal comparison of 50-step autoregressive rollout. (a) Ground truth u trajectory from finite-difference solver. (b) FNO predicted u trajectory. (c) Absolute error $|u_{\text{true}} - u_{\text{pred}}|$. (d-f) Corresponding plots for v. Horizontal axis: time step (0-50); vertical axis: spatial position ($x \in [0, 1]$). Color scale indicates field amplitude.

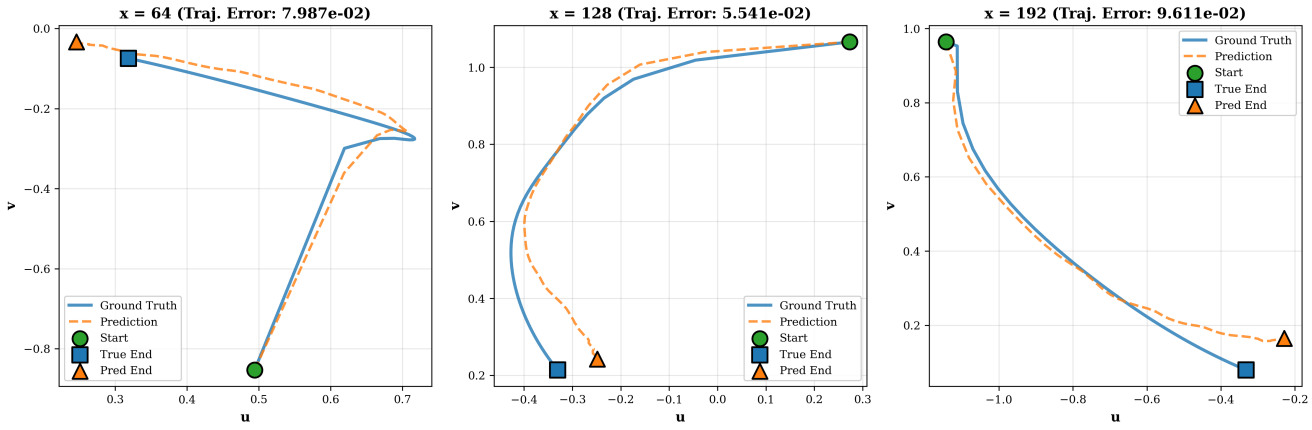

Figure 5: Phase space portraits at three spatial locations. Each panel shows v (vertical axis) versus u (horizontal axis) over 50 time steps. Blue solid curves: ground truth trajectories. Orange dashed curves: FNO predictions. Green circle: initial condition. Blue square: ground truth endpoint. Orange triangle: predicted endpoint.

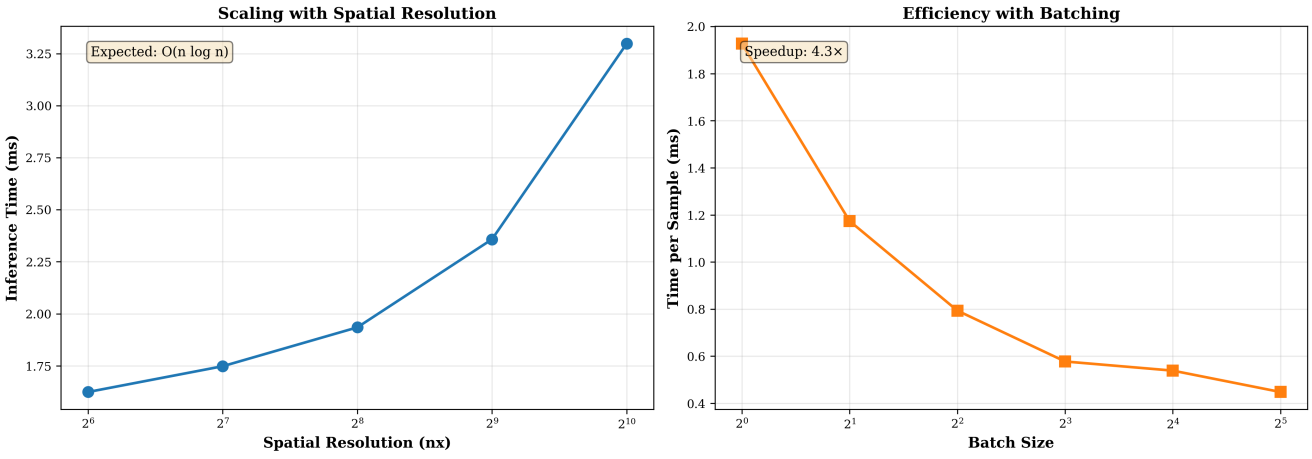

Figure 6: Computational efficiency scaling. (a) Inference time versus spatial resolution (n_x) for FNO. The FNO scales as $\mathcal{O}(n \log n)$. (b) Time per sample versus batch size for FNO on GPU.

Table 2: Computational efficiency comparison between FNO and finite-difference (FD) solver for a single trajectory rollout ($n_x = 256$, $n_{\text{steps}} = 50$). Benchmarks performed on NVIDIA RTX 5080 GPU (FNO) and AMD Ryzen 7 7800X3D CPU (FD solver).

Method	**Time per Step**	**Total Time (50 steps)**	**Memory**
Finite-Difference (CPU)	42.3 ± 1.8 ms	2.12 s	8.4 MB
FNO (GPU, batch=1)	0.38 ± 0.02 ms	0.019 s	147 MB
FNO (GPU, batch=32)	0.045 ± 0.003 ms	0.0023 s	152 MB
Speedup Factor	$111\times$ (single)	$111\times$	—
	$940\times$ (batch)	$920\times$	—

Table 3: Extrapolation performance on parameter combinations outside training ranges. Relative L^2 error reported as mean over 100 test trajectories per scenario. "Within range" denotes validation set performance for comparison.

Parameter Regime	$\epsilon_{\text{rel}}(u)$	$\epsilon_{\text{rel}}(v)$
Within range (validation)	0.0098	0.0087
Low diffusion ($D_u = 0.005$, $D_v = 0.002$)	0.0143	0.0126
High diffusion ($D_u = 0.15$, $D_v = 0.08$)	0.0167	0.0139
High time-scale separation ($\tau = 30$)	0.0214	0.0178
Low recovery ($b = 0.05$)	0.0128	0.0114

minimum, the error increases by approximately 45% relative to the within-range baseline. This degradation likely reflects the challenge of representing sharper spatial features with the fixed number of Fourier modes. Meanwhile, diffusion coefficients 50% above the training maximum cause 70% error increase, suggesting that the network has learned features specific to the training range rather than purely diffusion-invariant operators.

The most significant extrapolation challenge occurs for high time-scale separation ($\tau = 30$, 50% beyond training maximum), where errors more than double. This is expected because large τ values produce increasingly stiff dynamics with rapid activator transitions.

Interestingly, extrapolation to lower recovery strength ($b = 0.05$, below the training minimum) shows only a medium error increase (31%). This suggests that the network's learned representations are more robust to variations in recovery dynamics than to variations in diffusion or time-scale parameters.

Ethics

Training neural operators is computationally intensive and carries a significant environmental footprint. Modern machine learning training often requires extensive compute power, drawing large amounts of electricity. For instance, training a single state-of-the-art model (GPT-3) was reported to consume about 1,287 MWh of electricity - roughly as much as 120 U.S. homes use in an entire year - and to emit 552 tons of CO2 (equivalent to the annual emissions of 110 gasoline cars) [7]. While this FNO model is far smaller, the concern remains: the cumulative energy used in hyperparameter tuning, model training, and extensive simulations can be non-negligible.

To address this, we adopt more green principles. We prioritize efficient model architectures and training procedures to minimize compute requirements. Recent studies indicate that following best practices (e.g. using energy-efficient hardware, algorithmic optimizations, and clean energy sources) can cut training energy usage by up to 100x and associated CO2 emissions by up to 1000x [20]. In more practical terms, we plan to mitigate the environmental impact by limiting the number of training runs and using mixed-precision and batch size to maximise hardware optimization.

In addition, over-reliance on AI-generated approximations in high-stakes domains can be dangerous. If FNO modeling of the FHN system is used as a building block for general research (e.g. building more refined neuronal models), and researchers place blind trust in the predictions, it may lead to incorrect features or research based on false data. Overdependence on AI tools can lower human expertiece and become an issue when the AI falls or is used outside its scope - especially in the medical field [13]. Thus, the FNO models should augment decision making rather than replace it. If this research is ever used in an industry context, all predictions should be cross-checked against established analytical results. In practice, this means using the FNO as a supporting tool (for faster computations or insight) rather than a sole authority.

Finally, the bias in modeling from training data or underlying design is another ethical concern. In the context of FNOs for the FitzHugh-Nagumo model, a form of bias

could arise if the training dataset of simulated scenarios is not sufficiently representative of all relevant conditions (for example, if all training simulations use a narrow range of model parameters or initial conditions). It is known that when training data are unrepresentative or incomplete, the learned model will yield biased outputs that systematically err on those underrepresented conditions [4]. In this case, it might mean the FNO predicts nerve excitation dynamics accurately for common parameter settings but consistently deviates for rarer or extreme settings. In another sense, any systematic error could be considered bias: for instance, an inductive bias in the FNO architecture might favor smoother solutions and miss sharp transient phenomena.

Finally, we emphasize again that the FNO's predictions should be interpreted **in conjunction** with established knowledge of the system. If the model is applied in a biomedical context, it should be cross-checked to make sure it does not inadvertently perpetuate any biases that could lead to health disparities, i.e. differing accuracy on data from different patient groups.

References

[1] Burns, Keaton J. et al. "Dedalus: A flexible framework for numerical simulations with spectral methods". In: *Physical Review Research* 2.2, 023068 (Apr. 2020), p. 023068. DOI: 10.1103/PhysRevResearch.2.023068. arXiv: 1905.10388 [astro-ph.IM].

[2] Ermentrout, G. B. and Terman, D. H. *Mathematical Foundations of Neuroscience*. Springer, 2010.

[3] Fabiani, Gianluca et al. "Enabling Local Neural Operators to perform Equation-Free System-Level Analysis". In: *arXiv preprint arXiv:2505.02308* (2025).

[4] Ferrara, Emilio. "Fairness and bias in artificial intelligence: A brief survey of sources, impacts, and mitigation strategies". In: *Sci* 6.1 (2023), p. 3.

[5] FitzHugh, R. "Impulses and physiological states in theoretical models of nerve membrane". In: *Biophysical Journal* 1.6 (1961), pp. 445–466.

[6] Hairer, E., Nørsett, S. P., and Wanner, G. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, 1993.

[7] Harris, Jeffrey. "ChatCO2 - Safeguards Needed for AI's Climate Risks". In: *Greenpeace* (2023).

[8] Hunter, J. D. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science & Engineering* 9 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[9] Huybrechs, Daan. "On the Fourier extension of nonperiodic functions". In: *SIAM Journal on Numerical Analysis* 47.6 (2010), pp. 4326–4355.

[10] Izhikevich, E. M. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. MIT Press, 2007.

[11] Keener, J. and Sneyd, J. *Mathematical Physiology*. Springer, 2009.

[12] Kingma, Diederik P. and Ba, Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[13] Korkmaz, Selçuk. "Artificial intelligence in healthcare: a revolutionary ally or an ethical dilemma?" In: *Balkan Medical Journal* 41.2 (2024), p. 87.

[14] Kossaifi, Jean et al. *A Library for Learning Neural Operators*. 2024. arXiv: 2412.10354 [cs.LG].

[15] Kovachki, Nikola B. et al. "Neural Operator: Learning Maps Between Function Spaces". In: *CoRR* abs/2108.08481 (2021).

[16] LeVeque, R. J. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM, 2007.

[17] Li, Zongyi et al. *Fourier Neural Operator for Parametric Partial Differential Equations*. 2021. arXiv: 2010.08895 [cs.LG]. URL: https://arxiv.org/abs/2010.08895.

[18] Nagumo, J., Arimoto, S., and Yoshizawa, S. "An active pulse transmission line simulating nerve axon". In: *Proceedings of the IRE* 50.10 (1962), pp. 2061–2070.

[19] Oliphant, T. E. "Python for scientific computing". In: *Computing in Science & Engineering* 9.3 (2007), pp. 10–20.

[20] Patterson, David et al. *The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink*. 2022. arXiv: 2204.05149 [cs.LG]. URL: https://arxiv.org/abs/2204.05149.

[21] Quarteroni, A., Sacco, R., and Saleri, F. *Numerical Mathematics*. Springer, 2000.

[22] Rinzel, J. and Ermentrout, G. B. "Analysis of neural excitability and oscillations". In: *Methods in Neuronal Modeling*. MIT Press, 1998, pp. 251–291.

[23] Terman, D. "The transition from bursting to continuous spiking in a neural oscillator". In: *Journal of Computational Neuroscience* 1.1 (1991), pp. 39–56.

[24] Virtanen, P. et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17.3 (2020), pp. 261–272.