NEURAL INTEGRAL EQUATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Integral equations (IEs) are functional equations defined through integral operators, where the unknown function is integrated over a possibly multidimensional space. Important applications of IEs have been found throughout theoretical and applied sciences, including in physics, chemistry, biology, and engineering; often in the form of inverse problems. IEs are especially useful since differential equations, e.g. ordinary differential equations (ODEs), and partial differential equations (PDEs) can be formulated in an integral version which is often more convenient to solve. Moreover, unlike ODEs and PDEs, IEs can model inherently non-local dynamical systems, such as ones with long distance spatiotemporal relations. While efficient algorithms exist for solving given IEs, no method exists that can learn an integral equation and its associated dynamics from data alone. In this article, we introduce Neural Integral Equations (NIE), a method that learns an unknown integral operator from data through a solver. We also introduce an attentional version of NIE, called Attentional Neural Integral Equations (ANIE), where the integral is replaced by self-attention, which improves scalability and provides interpretability. We show that learning dynamics via integral equations is faster than doing so via other continuous methods, such as Neural ODEs. Finally, we show that ANIE outperforms other methods on several benchmark tasks in ODE, PDE, and IE systems of synthetic and real-world data.

1 INTRODUCTION

Integral equations (IEs) are functional equations where the indeterminate function appears under the sign of integration (Stech et al., 1981). The theory of IEs has a long history in pure and applied mathematics, dating back to the 1800's, and it is thought to have started with Fourier's Theorem (Groetsch, 2007); also the Dirichelet's problem (a PDE) was originally solved through its integral formulation. Subsequent studies, carried out by Fredholm, Volterra, and Hilbert and Schmidt, have significantly contributed to the establishment of this theory. IEs appear in many applications ranging from physics and chemistry, to biology and engineering (Wazwaz, 2011; Groetsch, 2007), for instance in potential theory, diffraction and inverse problems such as scattering in quantum mechanics (Wazwaz, 2011; Groetsch, 2007; Lakshmikantham, 1995). Neural field equations, that model brain activity, can be describes using IEs and integro-differential equations (IDEs), due to their highly non-local nature ((Amari, 1977)). IEs are related to the theory of ordinary differential equations (ODEs) and partial differential equations (PDEs), however they possess unique properties. While ODEs and PDEs describe local behavior, IEs model global, long-distance spatiotemporal relations. Moreover, ODEs and PDEs have IE forms that can be solved more effectively and efficiently due to the better stability properties of IE solvers compared to ODE and PDE solvers (Rokhlin, 1985; 1990).

In this article, we introduce and address the outstanding problem of learning dynamics from data through integral equations. Namely, we introduce the *Neural Integral Equation* (NIE). Our setup is that of an operator learning problem, where we learn the integral operator that generates dynamics that fit given data. Often, one has observations of a dynamical system without knowing its analytical form. Our approach permits modeling the system purely from the observations. This model, via the learned integral operator, can be used to generate dynamics, as well as be used to infer the spatiotemporal relations that generated the data. The innovation of our proposed method lies in the fact that we formulate the operator learning problem associated to dynamics in the form of an optimization problem for a solver. Unlike other operator learning methods that learn dynamics

as a mapping between function spaces for fixed time points, i.e. as a mapping $T : \prod_i \mathcal{A}_i \longrightarrow \prod_j \mathcal{B}_j$, where $\mathcal{A}_i$ and $\mathcal{B}_j$ are function spaces each representing a time coordinate, NIE allows to continuously learn dynamics with arbitrary time resolution. Since IEs are functional equations, our solver outputs solutions through an integral version of the Adomian decomposition method Wazwaz (2011), which iteratively converges to a solution of the IE.

1.1 Our Contributions

In this article, we introduce Neural Integral Equations (NIE) and Attentional Neural Integral Equations (ANIE), which are novel neural network based methods for learning dynamics, in the form of IEs, from data. Our architectures allow modeling dynamics with long-distance spatiotemporal relations typical of functional equations. Our main contributions are as follows:

- We introduce the first method for learning dynamics from data through an IE solver.
- We implement a fully differentiable IE solver in PyTorch.
- We also implement a highly scalable version of the solver where integration is done with a self-attention mechanism.
- We use our model to solve ODE, PDE, and IE systems.
- Finally, we use our method to model non-local brain dynamics from fMRI data.

1.2 Background and related work

Due to their wide range of applications, the theory of IEs has attracted the attention of mathematicians, physicists and engineers for a long time. Detailed accounts on integral equations can be found in (Zemyan, 2012; Wazwaz, 2011; Bôcher, 1926). Along with their theoretical properties, much attention has been devoted to the development of efficient integral equation solvers mainly for the solution of certain PDE systems, such as in (Rokhlin, 1985; 1990). In fact, it is known that integral equation solvers obtain more accurate solutions than differential solvers when dealing with a variety of PDEs.

However, deep learning on IE models has not been treated so far, in the approach considered in this article, and it differs profoundly from the implementation of IE solvers. In fact, solving IEs requires the knowledge of a given equation (i.e. an integral operator). Deep learning approaches for solving given integral equations have appeared in the literature (Guan et al., 2022; Que, 2016; Keller & Dahm, 2019; Guo et al., 2021; Effati & Buzhabadi, 2012), and known methods require that the integral operator that determines the IE is known, and given. Instead, in our approach we learn the integral operator itself, and therefore an integral equation, from observations alone. As a consequence, we learn dynamics with possibly complex long-distance spatiotemporal relations through IEs. A similar approach has been followed for IDEs in (Zappala et al., 2022). However, in the present work our implementation does not include differential solvers, and the reformulation of such dynamical problems in terms of IEs has great benefits in terms of solver speed and stability. Moreover, our version of an IE solver that approximates integrals via self-attention is unique in its kind, and allows for higher dimensional integrals, which were considered in (Zappala et al., 2022).

Our method is formulated as an operator learning problem, but unlike previous approaches to operator learning we provide a solver that learns continuous non-local space-time dynamics. A general perspective on operator learning can be found in (Kovachki et al., 2021). See also (Lu et al., 2021) for an approach to operator learning inspired by approximation theory of operators. Modeling dynamics through operator learning has previously been considered, see for instance (Li et al., 2020a;b; Cao, 2021). However, the methods employed in this article substantially differ from the traditional operator learning approaches, in that we learn an integral operator as the generator of dynamics from data, instead of learning a mapping between some products of function spaces, where each term represents different temporal slices of the dynamics.

Modeling continuous dynamics from discretely sampled data is a fundamental task in data science. Methods for continuous modeling include those based on ODEs (Chen et al., 2018; 2021). While ODEs are useful for modeling temporal dynamics, they are fundamentally local equations which neither model spatial nor long-range temporal relations. The authors of (Chen et al., 2018) have employed auxiliary tools, such as RNNs, in order to include non-locality. We point out that RNNs

can be seen as performing a temporal integration (in discrete steps), in order to codify some degree of non-local (temporal) dependence in the dynamics. In this work, we introduce a framework that provides a more general and formal solution to this non-local integration problem. Moreover, the dynamics are not produced sequentially with respect to time, as is done by ODE solvers, but are processed in parallel thus providing increased efficiency.

The self-attention mechanism and transformers were introduced in (Vaswani et al., 2017) and applied to machine translation tasks. Thanks to their initial success, they have since been used in many other domains, including operator learning for dynamics (Cao, 2021; Geneva & Zabaras, 2022). Interestingly, the self-attention mechanism can be interpreted as the Nystrom method for approximating integrals (Xiong et al., 2021). Making use of this connection, we approximate the integral kernel of our model using self-attention, allowing efficient integration over higher dimensions.

2 NEURAL INTEGRAL EQUATIONS

We introduce a deep neural network model based on integral equations. An integral equation takes the general form given by

$$\mathbf{y}(t) = f(t) + \int_{\alpha(t)}^{\beta(t)} G(\mathbf{y}, t, s) ds, \tag{1}$$

where the variable s is the local time used for integration for each t, which is the global time. The indeterminate function $\mathbf{y}(t)$ appears both under the sign of integration and outside of it. Neural Integral Equations (NIEs) are integral equations as defined by Equation 1 where G is a neural network. Training a NIE consists of optimizing G in such a way that the corresponding solution $\mathbf{y}$ to Equation 1 fits the given data.

Integral equations are non-local equations (Stech et al., 1981), unlike ODEs for example, since the integral operator $\int_{\alpha(t)}^{\beta(t)} G(\bullet, t, s) ds : \mathcal{A} \longrightarrow \mathcal{A}$ maps a space of integrable functions $\mathcal{A}$ into itself. Therefore, to evaluate the RHS of Equation 1 at an arbitrary time point t the function $\mathbf{y}(s)$ between $\alpha(t)$ and $\beta(t)$ is needed. Here, α and β are arbitrary functions and common choices include $\alpha(t) = a$ and $\beta(t) = b$ (Fredholm eqautions), or $\alpha(t) = 0$ and $\alpha(t) = t$ (Volterra equations). Consequently, solving an integral equation requires an iterative procedure where the solution is obtained as a sequence of approximations that converge to the solution. We refer the reader to (Wazwaz, 2011) for the theory behind the methods that motivate this procedure.

Interestingly, utilizing NIEs to model NODEs allows to bypass the use of the ODE solver introduced in (Chen et al., 2018; 2021). The convenience in this approach is that the integral equation solver is more stable than the ODE solver. These ODE solvers instabilities, induced by equation stiffness, have been previously considered in (Ghosh et al., 2020; Finlay et al., 2020). The IE solver presented in this work thus does not suffer from these problems, and is also significantly faster.

It is often useful to consider a more specific form for the IE, where the function G factors in the product of a *kernel K* and a generally non-linear function F as $G = K(t, s)F(\mathbf{y})$. Here, K is matrix valued and it carries the dependence on the time (both global and local), while F depends only on the indeterminate function $\mathbf{y}$. The form of this IE is therefore:

$$\mathbf{y}(t) = f(t) + \int_{\alpha(t)}^{\beta(t)} K(t, s)F(\mathbf{y}(s)) ds. \tag{2}$$

NIEs in this form comprise two neural networks, namely K and F. We observe that in IEs, the initial condition is embedded in the equation itself, and it is not an arbitrary value to be specified as an extra condition. The general algorithm for training NIE is given in Alrorithm 1.

2.1 SPACE TIME AND HIGHER DIMENSIONAL INTEGRATION

Partial integral equations (PIEs), are IEs that have one or more integration dimensions. They are formulated as

$$\mathbf{y}(\mathbf{x}, t) = f(\mathbf{x}, t) + \int_{\Omega} \int_{\alpha(t)}^{\beta(t)} G(\mathbf{y}, \mathbf{x}, t, s) d\mathbf{x} ds, \tag{3}$$

where $\Omega \subset \mathbb{R}^n$ is a domain in $\mathbb{R}^n$, and $\mathbf{y} : \Omega \times \mathbb{R} \longrightarrow \mathbb{R}^m$. These equations are integral counterpart of PDEs, similarly to the relation between IEs and ODEs.

Algorithm 1 NIE method training step. Integration is performed using the module torch.quad, with the Monte Carlo method.

Require: $\mathbf{y}_0(t)$ ▷ Initialization
Ensure: $\mathbf{y}(t)$ ▷ Solution to IE with initial $\mathbf{y}_0(t)$
 1: $\mathbf{y}^0(t) := \mathbf{y}_0(t)$ ▷ Initial solution guess
 2: **while** iter $\leq$ maxiter and error $>$ tolerance **do**
 3:      Require gradients on y^i
 4:      Evaluate: $\mathbf{y}^{i+1}(t) = f(\mathbf{y}^i, t) + \int_{\alpha(t)}^{\beta(t)} K(t, s) F(\mathbf{y}^i(s)) ds$
 5:      Set solution to be: $\mathbf{y}^{i+1}$
 6:      New error: error $=$ metric$(\mathbf{y}^{i+1}, \mathbf{y}^i)$
 7: **end while**
 8: Output of solver: $\mathbf{y}(t)$
 9: Compute loss wrt observations: loss$(\mathbf{y}(t), \text{obs})$
10: Backpropagation: loss.backward()

2.2 ATTENTIONAL NEURAL INTEGRAL EQUATIONS

Training of NIE requires an integration step at each time point, incurring a potentially high computational cost. This integration step is implemented using the torchquad package (Gómez et al., 2021), a high performance numerical Monte Carlo integration method, resulting in fast integration and high scalability of NIE. For example, solving ODEs using NIE is significantly faster than using traditional ODE solvers (Table 1). However, several limitations are associated with the torchquad integration method. First, torchquad is optimized for integration in time and scales poorly to arbitrary space dimensions - i.e. "curse of dimensionality" - which requires exponentially increasing numbers of sampled points with increasing numbers of dimensions. To use NIE for solving PDEs and (P)IEs we require efficient spatial integration in high dimensions. Second, while the kernel and the G function that NIE learns can be inspected, their interpretability is not trivial. Ideally, we would like a model in which we can interpret and explain the underlying dynamics. Such explainability would especially be useful when modeling biological systems where the goal is to understand the underlying dynamics rather than just fitting data. For example, when modeling brain activity recordings we would like to use the model to understand which brain regions and neural connectivity patterns drive certain brain activity dynamics.

To address these challenges, we have developed a novel integration method for NIE that is based on a self-attention mechanism. In fact, self-attention can be viewed as an approximation of an integration procedure (Tsai et al., 2019; Xiong et al., 2021), where the product of queries and keys coincides with the notion of a kernel discussed in Section 2. In (Cao, 2021) the parallelism between self-attention and integration of kernels was further exploited to interpret transformers as Galerkin projections in operator learning tasks.

We have replaced the analytical integral $\int_{\alpha(t)}^{\beta(t)} K(t, s) F(\mathbf{y}(s)) ds$ in Equation 2 with a self-attention procedure. The resulting model, which we call *Attentional Neural Integral Equation* (ANIE), follows the same principle of iterative IE solving presented in Section 2 but where the neural networks K and F are replaced by attention matrices. Observe, following the comparison between integration and self-attention, that K is decomposed in the product of queries and keys, as described for instance in (Cao, 2021). The interval of integration $[\alpha(t), \beta(t)]$ is determined, in the attentional approximation, by means of the mask. In particular, if there is no mask we have a Fredholm IE, while the causal attention mask (Yang et al., 2021) corresponds to a Volterra type of IE.

2.3 THE INITIAL CONDITION FOR IEs

NIE does not learn a dynamical system via the derivative of a function $\mathbf{y}$, as is the case for ODEs and IDEs. Therefore, we do not need to specify an initial condition in the solver during training and evaluation. In fact, the initial condition for IEs is encoded in the equation itself. For instance, taking $t = 0$ in a Volterra or a Fredholm equation uniquely fixes $\mathbf{y}(\mathbf{x}, 0)$ for all x.

Therefore, we can specify a condition for IEs by constraining the free function $f(\mathbf{y}, t)$. In what follows, we will make use of this paradigm several times. There are two immediate ways one could

---

Algorithm 2 ANIE method training step. Integration here is replaced by a transformer employing self-attention.

Require: $\mathbf{y}_0(\mathbf{x}, t)$ ▷ Initialization
Ensure: $\mathbf{y}(\mathbf{x}, t)$ ▷ Solution to IE with initial $\mathbf{y}_0(\mathbf{x}, t)$
 1: $\mathbf{y}^0(t) := \mathbf{y}_0(t)$ ▷ Initial solution guess
 2: **while** iter $\leq$ maxiter and error $>$ tolerance **do**
 3:      Concatenate space and time tokens to $\mathbf{y}^i(\mathbf{x}, t)$: $\tilde{\mathbf{y}}^i(\mathbf{x}, t) = \text{concat}(\mathbf{y}^i(\mathbf{x}, t), s, t)$
 4:      Require gradients on $\tilde{\mathbf{y}}^i$
 5:      Evaluate with self-attentionr: $y^{i+1}(t) = f(\tilde{\mathbf{y}}^i, t) + \mathfrak{Att}(\tilde{\mathbf{y}}^i(\mathbf{x}, t), \tilde{\mathbf{y}}^i(\mathbf{x}, t))$
 6:      Set solution to be: $\mathbf{y}^{i+1}$
 7:      New error: error $= \text{metric}(\mathbf{y}^{i+1}, \mathbf{y}^i)$
 8: **end while**
 9: Output of solver: $\mathbf{y}(t)$
10: Compute loss wrt observations: $\text{loss}(\mathbf{y}(t), \text{obs})$
11: Backpropagation: loss.backward()

---

impose constraints on the free function. The simplest is to fix a value $\mathbf{y}_0$ and let $f(\mathbf{y}, t)$ be fixed to be $\mathbf{y}_0$ for all t. Alternatively, one could choose an arbitrary function f and keep this function fixed. In practice, the latter is conceptually more meaningful. For instance, in theoretical physics, when transforming the Schrödinger equation into an integral equation, on the RHS one can choose an arbitrary function $\psi(\mathbf{y}, t)$ which corresponds to the wave function of free particles, i.e. without potential V. Applications of this procedure are found below in the experiments.

3   EXPERIMENTS

3.1   BENCHMARK OF (A)NIE TRAINING SPEED

Neural ODEs (NODEs) can be slow and have poor scalability (Kelly et al., 2020). As such, several methods have been introduced to improve their performance (Kelly et al., 2020; Kidger et al., 2021; Daulbaev et al., 2020; Poli et al., 2020; Pal et al., 2021). Despite these improvements, NODE is still significantly slower than discrete methods such as LSTMs. We hypothesize that (A)NIE has significantly better scalability than NODE, comparable to fast but discrete LSTMs. To test this, we compare (A)NIE to the latest optimized version of (latent) NODE Rubanova et al. (2019) and to LSTM on three different dynamical systems: Lotka-Volterra equations, Lorenz system, and IE generated 2D spirals (see Appendix C for data generation details). During training, models were shown (as initialization) the first half of the data and were tasked to predict the second half. The wall times in seconds per iterations are reported in Table 1. While all models achieve comparable (good) fits to the data, we find that NIE and ANIE, like LSTM, drastically outperform NODE. In fact, NIE and ANIE have similar performance to LSTM. However, like NODE, they are continuous models. Thus (A)NIE has the capabilities of NODE (and more) with the speed of LSTMs.

Table 1: Wall time, in seconds per training iteration, for different models fitted to multiple dynamical systems. All models were trained on an RTX 3090 NVIDIA GPU for 3000 epochs or until convergence. (A)NIE and LSTM are significantly faster than NeuralODE for all systems. Note that while LSTM is fast, it is not a continuous time model like (A)NIE and NeuralODE. Thus, ANIE has the advantages of a true continuous model with the speed of an LSTM. All values are computed over 5 repeats with random initialization of the models.

Models	Dynamical Systems		
	Lotka-Volterra	Lorenz	2D-Spirals)
LSTM	0.0116 ± 0.0054	0.012 ± 0.005	0.02 ± 0.0031
Latent NeuralODE	2.88 ± 1.096	0.845 ± 0.515	0.494 ± 0.109
NIE (ours)	0.0312 ± 0.0068	0.38 ± 0.22	0.154 ± 0.037
ANIE (ours)	0.0115 ± 0.0204	0.252 ± 0.019	0.116 ± 0.028

3.2 HYPERPARAMETER SENSITIVITY BENCHMARK

For most deep learning models, including NODE, finding numerically stable solutions usually requires an extensive hyperparameter search. Since IE solvers are known to be more stable than ODE solvers, we hypothesize that (A)NIE is less sensitive to hyperparameter changes. To test this, we quantify model fit, for the Lotka-Volterra dynamical system, as a function of two different hyperparameters: learning rate and L2 norm weight regularization. We perform this experiment for three different models: LSTM, Latent NODE, and ANIE. As shown in Supp. Figure 9, we find that ANIE generally has lower validation error as well as more consistent errors across hyperparameter values, compared to LSTM and NODE, therefore validating our hypothesis.

3.3 MODELING PDEs WITH IEs: BURGERS' AND NAVIER-STOKES EQUATIONS

PDEs can be reformulated as IEs, and dynamics generated by differential operators can therefore be modeled through ANIE as a PIE, where integration is performed in space and time. We consider two types of PDEs, namely the Burgers' equation and the Navier-Stokes equation.

Table 2: Burgers' equation test with different time intervals $t = 10, 15, 25$ and different space resolution $s = 256, 512$. During evaluation, the time interval for the dynamics is changed to a re-scaled size ($1/4$ of the total interval for training). The dynamics used for testing are unseen during training, so that the models are interpolating with respect to time, and extrapolating with respect to the initial condition. ANIE (ours) shows better evaluation, implying that it more accurately infers the temporal relations and initial condition dependence.

	t=10		t=15		t=25	
	s = 256	s = 512	s = 256	s = 512	s = 256	s = 512
Conv1DLSTM	.0101	.0094	.0094	.0086	.0124	.0093
FNO1D	.0761	.0760	.0716	.0718	.0686	.0693
Galerkin	.0459	NA	.0528	NA	.688	NA
ANIE (ours)	.0015	.0022	.0094	.0024	.0082	.0103

Table 3: Benchmark on Navier-Stokes equation. We evaluate the models on predicting dynamics of different lengths ($t = 3, 5, 10, 20$) for unseen initial conditions. The models that use a single time point are ANIE (ours) and FNO2D, while the convolutional LSTM and FNO3D both use more time points (2 and 10 respectively) to predict the rest of the dynamics. Although ANIE has less information than other methods, it still outperforms them.

	t = 3	t = 5	t = 10	t = 20
Conv2DLSTM	.4935	.4393	.3931	.2999
FNO2D	.2795	.2724	NA	NA
FNO3D	NA	NA	.1751	.0701
ANIE (ours)	.0194	.0220	.0183	.0117

For the Burgers' equation, we focus on the ability of ANIE to model both space and time. In fact, as opposed for instance to the setting of (Cao, 2021; Li et al., 2020a), where it was considered a "static" Burgers' equation and the learned operator mapped the initial condition at time $t = 0$ to the final time of the solution at $t = 1$, we model the solution continuously over a time interval. We then evaluate on a different interval to show that ANIE has learned the temporal component, while the other models treat the learning process as a sequence of initial to finial point pairs. The Galerkin model (Cao, 2021), has not been used on higher spacial resolution, as the amount of memory required exceeded what was available to us during the experiments. The results are reported in Table 2, and an example of dynamics is given in Figure 1.

For the Navier-Stokes equation, we consider an extrapolation task where we evaluate the model on unseen initial conditions. Previous works have shown high performance in predicting dynamics
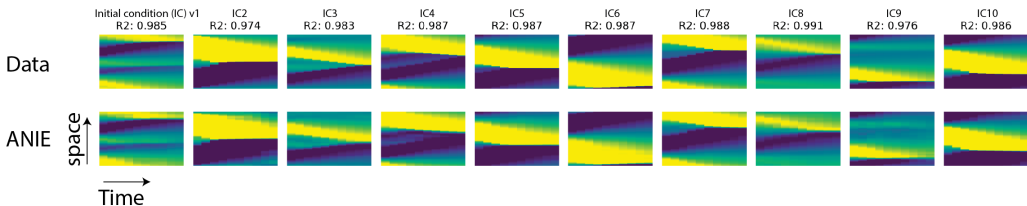
Figure 1: Example dynamics of (1+1)D Burgers' equation. Each frame represents the full dynamics where the x axis shows time and the y axis shows space. Top row is data, and bottom row is ANIE prediction. Columns represent different dynamics resulting from different initial conditions.

of Navier-Stokes from new initial conditions, but they do require several frames (i.e. several time points) to be fed into the model in order to achieve such performance. We see that since ANIE learns the full dynamics from arbitrarily chosen initial conditions, we achieve good performance even when a single initial condition is used to initialize the system. We train FNO2D (Li et al., 2020a) with initialization on a single time point, while convolutional LSTM and FNO3D are trained with 2 and 10 times for initialization, respectively. The results are given in Table 3. Interestingly, the model that performed closer to ANIE, i.e. with error approximately 7 times larger than ANIE and in the same order of magnitude, is FNO3D (Li et al., 2020a), where we have used 10 time frames for training. This is 10 times the amount of information that is used to initialize ANIE. However, we have encountered a situation similar to the case seen in (Li et al., 2020a), where due to instability with lower viscosities the dynamics were shortened to 20 and 30 time points, and the precision of FNO3D was significantly decreased (see Table 1 in (Li et al., 2020a)). In our case, where we have considered 10 and 20 time points sampled from the full dynamics, ANIE is unaffected and it is able to obtain the same high accuracy for all the chosen time points. FNO2D did not converge for higher number of points, and therefore results for time points $t = 10, 20$ have not been reported, while for FNO3D, we have conducted the experiments only for $t = 10, 20$ since using fewer points for the time dimension would have essentially reduced FNO3D to FNO2D.
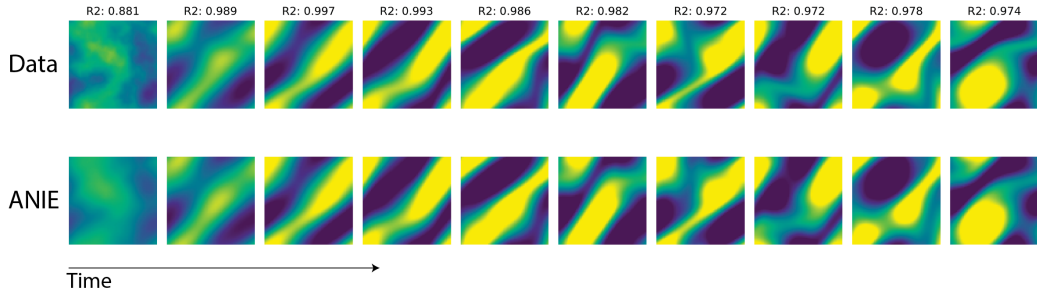

Figure 2: Example dynamics of (2+1)D Navier-Stokes system. Ground truth data (top) and prediction using ANIE (bottom) are shown. Prediction was generated using an initial condition that was not seen during training. R2 values quantify the model fit.

3.4 MODELING BRAIN DYNAMICS USING ANIE

Functional magnetic resonance imaging (fMRI) is the most common technique for studying whole-brain activity in humans. In fMRI, the blood-oxygen-level dependent (BOLD) signal is used as a proxy for brain activity (Akiki & Abdallah, 2019). fMRI data is intrinsically noisy and limited with respect to duration of an experiment for a single subject, which makes the task of modeling fMRI dynamics challenging. The use of fMRI simulators such as the *neurolib* (Cakan et al., 2021) has become a viable alternative to the real fMRI recordings. *neurolib* allows the user to load structural and functional datasets, change parameters that coordinate the whole network behavior and simulate it. The model created by *neurolib* to simulate the whole-brain is implemented as a system of delay differential equations, making it a dynamical system that simulates whole-brain activity. Here we

Table 4: Error per extrapolated time point for unseen initial conditions of the Generated fMRI data. The models that use a single time points to predict the whole dynamics are ANIE (ours), NODE, and Residual Network, while the LSTM uses 2 time points. ANIE shows better extrapolation performance, as shown by lower absolute error.

	t = 3	t = 5	t = 10	t = 20
NODE	0.5295 ± 0.04417	0.98 ± 0.07831	1.759 ± 0.1407	2.361 ± 0.2227
LSTM	1.797 ± 0.1704	2.004 ± 0.1856	2.182 ± 0.195	2.47 ± 0.1993
Residual Network	2.293 ± 0.1764	2.396 ± 0.1705	2.535 ± 0.1706	2.742 ± 0.2
ANIE (ours)	0.6201 ± 0.06875	0.7974 ± 0.08118	1.086 ± 0.112	1.242 ± 0.1256

Table 5: R-squared between model predictions, of ANIE, NODE, and LSTM, and ground truth for the 2D IE spirals. ANIE has the best performance.

NODE	LSTM	**ANIE (ours)**
0.1778 ± 0.06932	0.3410 ± 0.1132	0.7366 ± 0.1440

show the performance of ANIE and other models in modelling data generated by *neurolib*. Details about the data generations and pre-processing can be found in the Appendix.

The generated fMRI data consists of neural activity generated for 80 nodes localized across the cortex. The first half of the data is used for training and the second half is used for test. For training, the data is divided in segments of 20 time points where each time point characterize the initial condition of a 20 time points curve for the 80 nodes. As such, the models are trained on this data as an initial condition problem. During inference, the models are given points from the test set as new initial conditions and asked to extrapolate for the following 19 points. The mean error per point for 500 new initial conditions is shown in Figure 5 and summarized in Table 4. Despite all models effectively learning the dynamics of the training set, our results show that ANIE is the model with best performance in extrapolation for unseen new initial conditions. Figure 8 shows examples of the first 2 principal components of the generated fMRI data with the different model outputs. Figure 4 shows the output of the tested models per fMRI recording node over time with their respective correlation coefficient with the ground truth.

3.5 MODELING 2D IE SPIRALS

Here we test ANIE, NODE and LSTM in modelling 2D spirals generated by integral equations. This data consists of 500 2D curves of 100 time points each. The data was split in half for training and validation. During training, the initial 20 points were given as initial conditions for the different models. Details about the data generation are described in Appendix C. For ANIE, the initialization is given via the free function f, which assumes the values of the first 20 points and sets the remaining 80 points to be equal to the value of the 20th point. For NODE, the initialization is given as the reverse RNN on the first 20 points, which outputs a distribution corresponding to the first time point (see Chen et al. (2018) for more details on their Latent ODE experiments). The configuration used for NODE was the same used for their experiments on 2D spirals. For LSTM, we input the data in segments of 20 points to predict the consecutive point of the sequence. The process is repeated with the output of the previous step until all points of the curve are predicted. During inference we test the models performance to a new initialization. For that, we provide the segment from the 20th to the 40th point of the curve as the initialization and predict the remaining points of the curve. In other words, we provide a shifted sequence of 20 points. Table 5 shows the correlation between the ground truth curve and the extrapolation obtained with each model. Figure 7 shows the correlation coefficient for each of the 500 curves. In summary, ANIE significantly outperforms the other tested models when tested on new initial conditions.

3.6 INTERPRETABILITY OF ANIE

The attentional version of the IE solver provides attention weights that we can interpret, potentially giving insight into the underlying dynamics that generate the system. To inspect this, we extract the learned attention weights of ANIE trained on the Navier-Stokes data. Figure 3 shows the dynamics with its associated attention weights. We find that the attention values have a smooth and periodic pattern with respect to the 2D spatial coordinates and input dynamics.
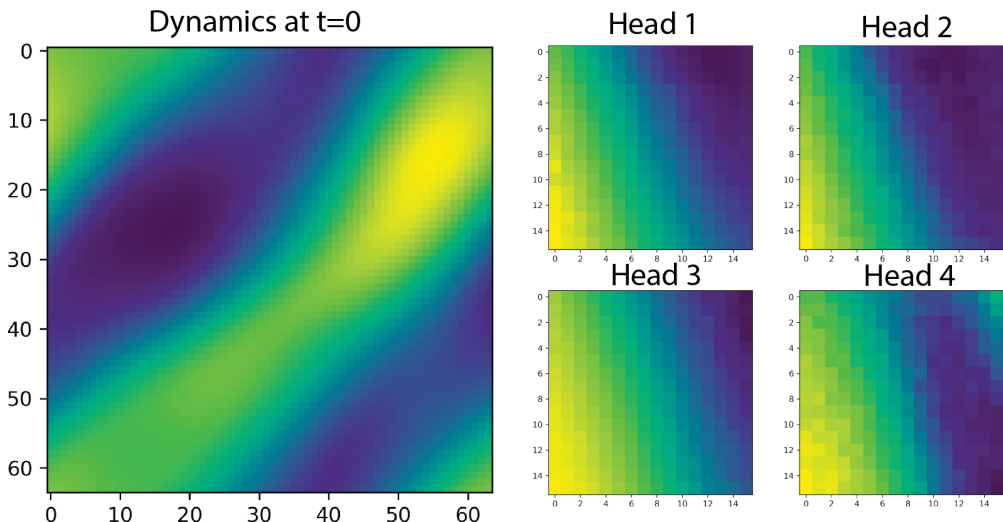

Figure 3: Visualization of attention of a ANIE model trained on Navier-Stokes data. The dynamics at time $t = 0$ is shown on the left, and the corresponding attention weights for each of the 4 attention heads are shown on the right.
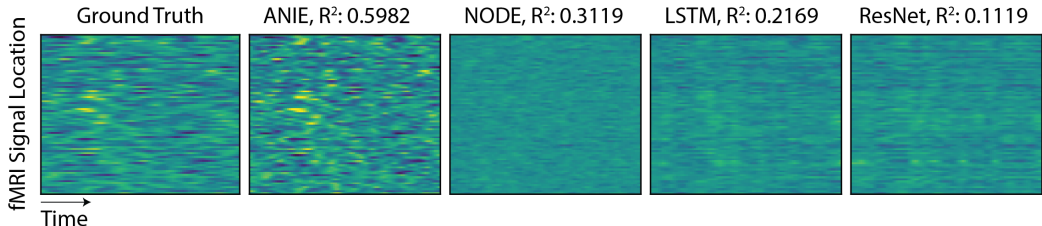
Figure 4: Example dynamics of fMRI data. Each image represents time on the x axis and 80 nodes (brain locations) on the y axis. R-squared of fit to ground truth data is shown for each model. ANIE has the best fit to the data.

4 CONCLUSIONS

We have presented the first neural network based integral equation solver that can learn an integral operator and its associated dynamics from data. A self-attention based version of the solver provides high scalability as well as interpretability of the dynamics. We have demonstrated the ability of our method to learn ODE, PDE, and IE systems, better than other dynamics models, in terms of better predictability as well as scalability. Since IEs can be used to model many real-world systems, we anticipate that our framework will have broad applications in science and engineering.

REPRODUCIBILITY STATEMENT

In order to allow the replication of ours results, we have provided all key components, including code, for the training and evaluation of all experiments performed in this manuscript:

- Pseudo Code for Neural Integral Equations is provided in Section 2, Algorithms 1, and for Attention Neural Integral Equations in Section 2.2, Algorithms 2.
- Hyperparameters for model training are defined in Appendix F.
- For artificial datasets, the code for generation has been included in Appendix C, E, D.

.

REFERENCES

Teddy J Akiki and Chadi G Abdallah. Determining the hierarchical architecture of the human brain using subject-level clustering of functional networks. *Scientific reports*, 9(1):1–15, 2019.

Shun-ichi Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological cybernetics*, 27(2):77–87, 1977.

Edward R Benton and George W Platzman. A table of solutions of the one-dimensional burgers equation. *Quarterly of Applied Mathematics*, 30(2):195–212, 1972.

Maxime Bôcher. *An introduction to the study of integral equations*. Number 10. University Press, 1926.

Caglar Cakan, Nikola Jajcay, and Klaus Obermayer. neurolib: a simulation framework for whole-brain neural mass modeling. *Cognitive Computation*, pp. 1–21, 2021.

Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in Neural Information Processing Systems*, 34, 2021.

Ricky T. Q. Chen, Brandon Amos, and Maximilian Nickel. Learning neural event functions for ordinary differential equations. *International Conference on Learning Representations*, 2021.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Alexandre Joel Chorin. Numerical solution of the navier-stokes equations. *Mathematics of computation*, 22(104):745–762, 1968.

Talgat Daulbaev, Alexandr Katrutsa, Larisa Markeeva, Julia Gusak, Andrzej Cichocki, and Ivan Oseledets. Interpolation technique to speed up gradients propagation in neural odes. *Advances in Neural Information Processing Systems*, 33:16689–16700, 2020.

Sohrab Effati and Reza Buzhabadi. A neural network approach for solving fredholm integral equations of the second kind. *Neural Computing and Applications*, 21(5):843–852, 2012.

Charles L Fefferman. Existence and smoothness of the navier-stokes equation. *The millennium prize problems*, 57:67, 2000.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *International conference on machine learning*, pp. 3154–3164. PMLR, 2020.

Nicholas Geneva and Nicholas Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022.

Arnab Ghosh, Harkirat Behl, Emilien Dupont, Philip Torr, and Vinay Namboodiri. Steer: Simple temporal regularization for neural ode. *Advances in Neural Information Processing Systems*, 33: 14831–14843, 2020.

Pablo Gómez, Håvard Hem Toftevaag, and Gabriele Meoni. torchquad: Numerical integration in arbitrary dimensions with pytorch. *Journal of Open Source Software*, 6(64):3439, 2021.

Yurii N Grigoriev, Nail H Ibragimov, Vladimir F Kovalev, and Sergey V Meleshko. *Symmetries of integro-differential equations: with applications in mechanics and plasma physics*, volume 806. Springer, 2010.

Charles W Groetsch. Integral equations of the first kind, inverse problems and regularization: a crash course. In *Journal of Physics: Conference Series*, volume 73, pp. 012001. IOP Publishing, 2007.

Yu Guan, Tingting Fang, Diankun Zhang, and Congming Jin. Solving fredholm integral equations using deep learning. *International Journal of Applied and Computational Mathematics*, 8(2): 1–10, 2022.

Rui Guo, Zhichao Lin, Tao Shan, Maokun Li, Fan Yang, Shenheng Xu, and Aria Abubakar. Solving combined field integral equation with deep neural network for 2-d conducting object. *IEEE Antennas and Wireless Propagation Letters*, 20(4):538–542, 2021.

Alexander Keller and Ken Dahm. Integral equations and machine learning. *Mathematics and Computers in Simulation*, 161:2–12, 2019.

Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. *Advances in Neural Information Processing Systems*, 33:4370–4380, 2020.

Patrick Kidger, Ricky TQ Chen, and Terry Lyons. " hey, that's not an ode": faster ode adjoints with 12 lines of code. 2021.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Vangipuram Lakshmikantham. *Theory of integro-differential equations*, volume 1. CRC press, 1995.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations, 2020b.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Avik Pal, Yingbo Ma, Viral Shah, and Christopher V Rackauckas. Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In *International Conference on Machine Learning*, pp. 8325–8335. PMLR, 2021.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models. *Advances in Neural Information Processing Systems*, 33:21105–21117, 2020.

Qichao Que. *Integral Equations For Machine Learning Problems*. PhD thesis, The Ohio State University, 2016.

Vladimir Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of computational physics*, 60(2):187–207, 1985.

Vladimir Rokhlin. Rapid solution of integral equations of scattering theory in two dimensions. *Journal of Computational Physics*, 86(2):414–439, 1990.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Edwin E Salpeter and Hans Albrecht Bethe. A relativistic equation for bound-state problems. *Physical Review*, 84(6):1232, 1951.

Harlan W Stech, Samuel M Rankin, Terry L Herdman, et al. *Integral and functional differential equations*, volume 67. CRC Press, 1981.

W Tobocman and LL Foldy. Integral equations for the schrödinger wave function. *American Journal of Physics*, 27(7):483–490, 1959.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: A unified understanding of transformer's attention via the lens of kernel. In *Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4344—-4353, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Abdul-Majid Wazwaz. *Linear and nonlinear integral equations*, volume 639. Springer, 2011.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nystöm-based algorithm for approximating self-attention. In *Proceedings of the... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, volume 35, pp. 14138. NIH Public Access, 2021.

Xu Yang, Hanwang Zhang, Guojun Qi, and Jianfei Cai. Causal attention for vision-language tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9847–9857, 2021.

Emanuele Zappala, Antonio Fonseca, Andrew Moberly, Michael Higley, Chadi Abdallah, Jessica Cardin, and David van Dijk. Neural integro-differential equations. *arXiv preprint arXiv:2206.14282*, 2022.

Stephen M Zemyan. *The classical theory of integral equations: a concise treatment*. Springer Science & Business Media, 2012.
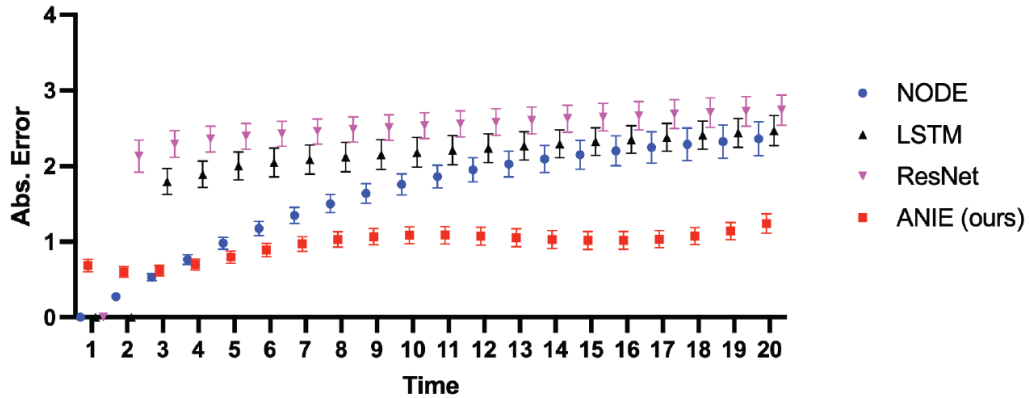
A   ADDITIONAL FIGURES

Figure 5: Quantification, using absolute error per time point, of model fits to simulated fMRI dataset. Models were run during inference on initial conditions not seen during training. ANIE has the best performance (lowest error) in predicting the dynamics.
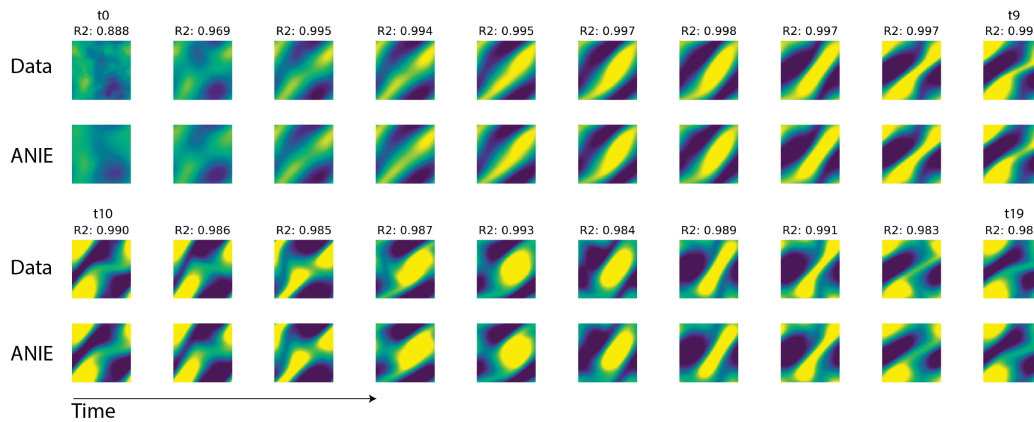

Figure 6: Example dynamics of Navier-Stokes system. Ground truth data (top) and prediction using ANIE (bottom) are shown. Prediction was generated using an initial condition that was not seen during training. R2 values quantify the model fit.
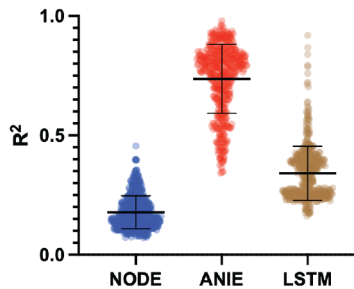

Figure 7: Quantification, using R-squared, of model fits to 2D IE spiral dataset. Models were run during inference on initial conditions not seen during training. ANIE has the best performance (highest R-squared) in predicting the dynamics.
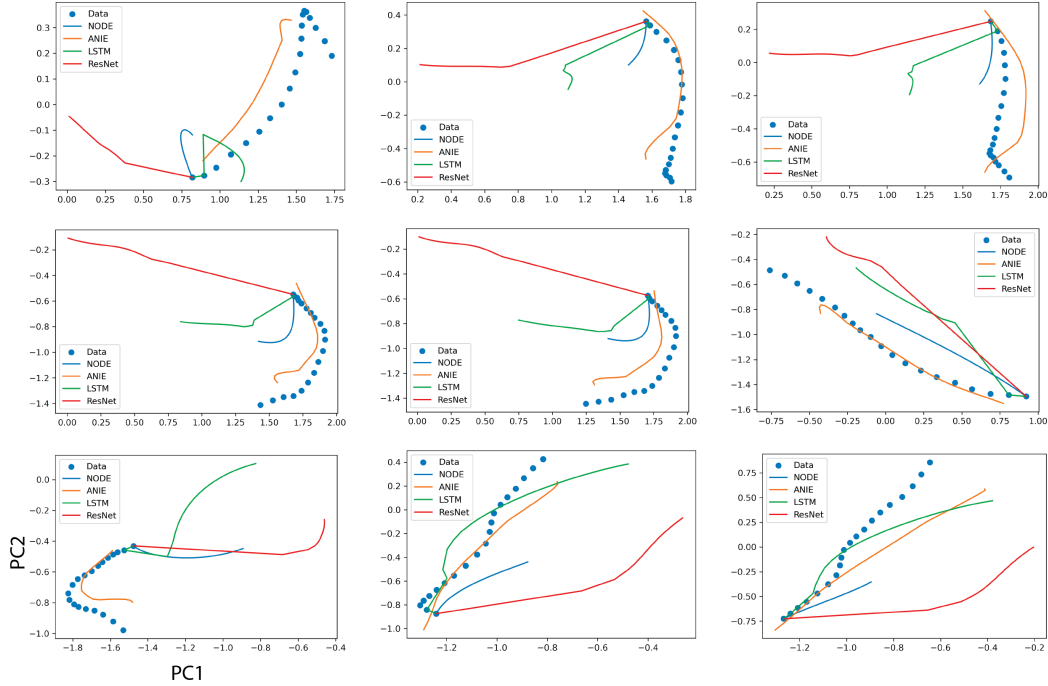
Figure 8: Example curves for simulated fMRI data and their model fits of an extrapolation task from unseen initial condition. ANIE performs best in predicting the dynamics for a new initial condition.
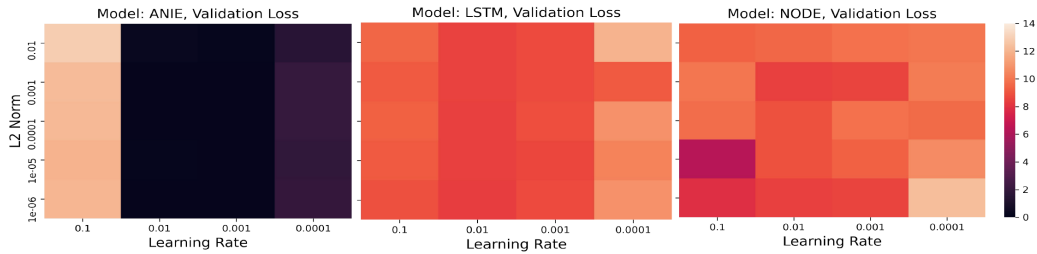

Figure 9: Hyperparameter sensitivity analysis for ANIE, LSTM and Neural ODE. Validation set Mean Squared Error for different hyperparameter combinations for all models trained for 300 epochs.

B   INTEGRAL EQUATIONS

Integral Equations (IEs) are equations where the unknown function appears under the sign of integral. The theories of IEs and Integro-Differential Equations (IDEs) are tightly related, and it is often the case to reduce problems in IEs to problems in IDEs and vice versa, both in practical and theoretical situations. IEs are also related to differential equations, and it is possible to reformulate problems in ODEs in the language of IEs or IDEs. In certain cases, IEs can also be converted to differential equations problems, even though this is not always possible (Zemyan, 2012; Grigoriev et al., 2010). In fact, the theory of IEs is not equivalent to that of differential equations. The most intuitive way of understanding this is by considering the local nature of differential euqations, as opposed to the non-local origin of IEs. By non-locality of IEs it is meant that each spatiotemporal point in an IE depends on an integration over the full domain of the solution function $\mathbf{y}$. In the case of differential equations, each local point depends only on the contiguous points through the local definition of the differential operators.

B.1   INTEGRAL EQUATIONS (1D)

We first discuss IEs where the integral operator only involves a temporal integration (i.e. 1D), as discussed in Section 2. In analogy with the case of differential equations, this case can be considered as the one corresponding to ODEs.

These IEs are given by an equation of type

$$\mathbf{y}(t) = f(t) + \int_{\alpha(t)}^{\beta(t)} G(\mathbf{y}, t, s) ds, \tag{4}$$

where f is the free term, which does not depend on $\mathbf{y}$, while the unknown function $\mathbf{y}$ appears both in the LHS, and in the RHS under the sign of integral. The term $\int_{\alpha(t)}^{\beta(t)} G(\mathbf{y}, t, s) ds$ is an integral operator $\mathcal{C}(D) \longrightarrow \mathcal{C}(D)$ from the space of integrable functions $\mathcal{C}(D)$ over some domain of $\mathbb{R}$, into itself. We observe that the variables t and s appearing in G are both in D, and they are interpreted as time variables. We refer to them as *global* and *local* times, respectively, following the same convention of (Zappala et al., 2022). The functions α and β determine the extremes of integration for each (global) time t. Common choices for α and β include $\alpha(t) = 0$ and $\beta(t) = t$ (Volterra equations), or $\alpha(t) = a$ and $\beta(t) = b$ (Fredholm equations).

The fundamental question in the theory of IEs is whether solutions exists and are unique. It turns out that under relatively mild assumptions on the regularity of G IEs admit unique solutions (Zemyan, 2012). Furthermore, the proofs in the first chapter of (Lakshmikantham, 1995) show the close relation between IEs and IDEs, as existence an uniqueness problems for IDEs are shown to be equivalent to analogous problems for IEs. Then the fixed point theorems of Schauder and Tychonoff are used to prove the results.

B.2   PARTIAL INTEGRAL EQUATIONS (N+1D)

We now discuss the case of IEs where the integral operator involves integration over a multi-dimensional domain of $\mathbb{R}^n$. This is the IE version of PDEs, and they are commonly referred to as Partial Integral Equations (PIEs). An equation of this type takes the form

$$\mathbf{y}(\mathbf{x}, t) = f(\mathbf{x}, t) + \int_{\Omega} \int_{\alpha(t)}^{\beta(t)} G(\mathbf{y}, \mathbf{x}, t, s) d\mathbf{x} ds, \tag{5}$$

where $\Omega \subset \mathbb{R}^n$ is a domain in $\mathbb{R}^n$, and $\mathbf{y} : \Omega \times \mathbb{R} \longrightarrow \mathbb{R}^m$. Here m does not necessarily coincide with n.

PIEs have been studied in some restricted form since the 1800's, as they have been employed to formulate the laws of electromagnetism before the unified version of Maxwell's equations was published. In addition, early work on the Dirichelet's problem found the IE approach proficuous, and it is well known that several problems in scattering theory (molecular, atomic and nuclear) are formulated in terms of PIEs. In fact, the Schrödinger equation can be recast as a PIE (Tobocman & Foldy, 1959). See also (Salpeter & Bethe, 1951), where bound-state problems are treated with the IE formalism.

C   ARTIFICIAL DATASET GENERATION

C.1   LOTKA-VOLTERRA SYSTEM

Lotka-Volterra is a classic system of nonlinear differential equations that models the interaction between two populations. The equations are described by:

$$\frac{dx}{dt} = \alpha xy - \beta y$$

$$\frac{dy}{dt} = \delta xy - \gamma y$$

where α and δ define the population interaction terms and β and γ the intrinsic population growth for population x and y. For our training selected 100 values of α, β, δ and γ and produce curves with the same initial condition. See code below (was adapted from `https://scipy-cookbook.readthedocs.io/items/LoktaVolterraTutorial.html`), and example visualization:
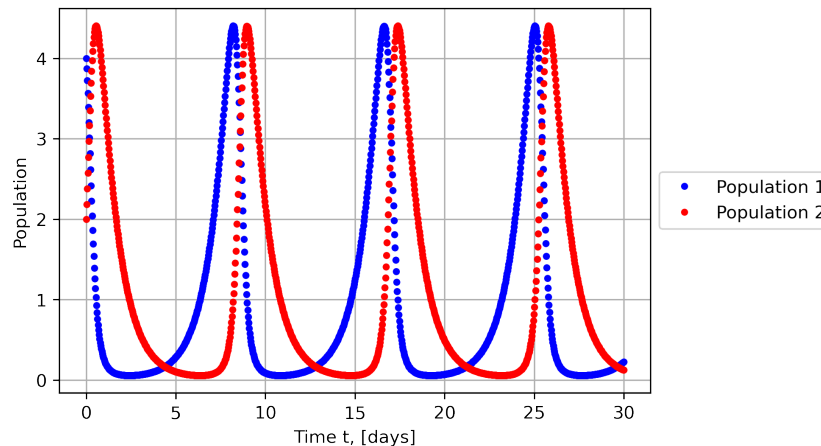

Figure 10: Example visualization of Lotka-Volterra 2D System

```
import numpy as np
from scipy import integrate

a_samples = np.random.uniform(low=0.5, high=1.5, size=100)
b_samples = np.random.uniform(low=0.5, high=1.5, size=100)
c_samples = np.random.uniform(low=0.5, high=2.5, size=100)
d_samples = np.random.uniform(low=0.25, high=1.25, size=100)

def dX_dt(X, t=0):
    """ Return the growth rate of two populations over time """
    return np.array([ a*X[0] -    b*X[0]*X[1] ,
                 -c*X[1] + d*b*X[0]*X[1] ])

def data_generation(a_samples, b_samples, c_samples, d_samples):
    Data = []
    for i in range(100):
        a = a_samples[i]
        b = b_samples[i]
        c = c_samples[i]
        d = d_samples[i]

        t = linspace(0, 15, 100)                    # time
        X0 = array([10, 5])
# initials conditions: 10 rabbits and 5 foxes
```

```
        X, _ = integrate.odeint(dX_dt, X0, t, full_output=True)
        Data.append(X)

    return np.array(Data)
```

C.2 LORENZ SYSTEM

Lorenz is a 3-dimensional system of ordinary differential equations, for modelling atmosferic convection. Furthermore, this system is chaotic which means that small variations of initial conditions can significantly affect the final trajectory. The system is described by:

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = x(\rho - z) - y$$

$$\frac{dz}{dt} = xy - \beta z$$

We decided to sample 100 random initial conditions, and run the system with the same parameters. See code below, which was adapted from `https://github.com/gboeing/lorenz-system`, and example visualization with default hyperparams.
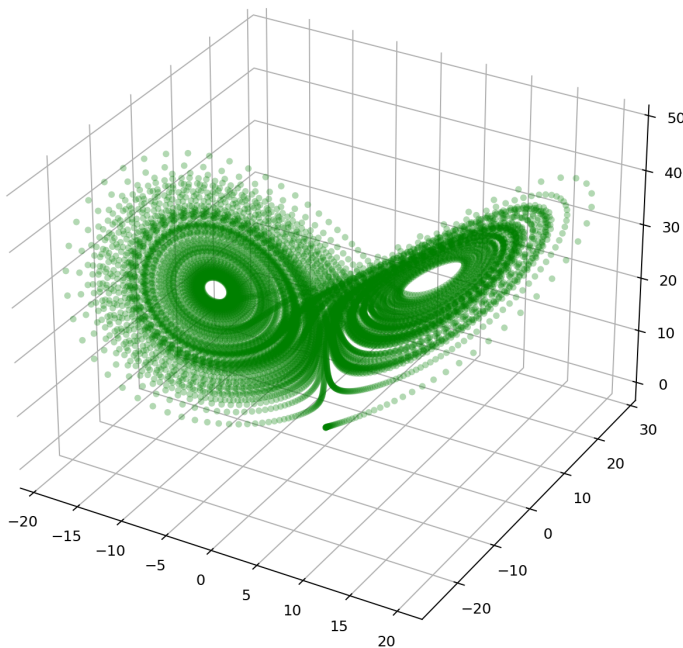

Figure 11: Example visualization of 3D Lorenz Dynamical System

```
from numpy import *
import numpy as np
from scipy.integrate import odeint

# Definition of parameters
```

```
init_x = np.random.uniform(low=0.0, high=0.5, size=100)
init_y  = np.random.uniform(low=0.0, high=0.5, size=100)
init_z  = np.random.uniform(low=0.0, high=0.5, size=100)

# define the lorenz system
# x, y, and z make up the system state, t is time,
#and sigma, rho, beta are the system parameters
def lorenz_system(current_state, t):

    # positions of x, y, z in space at the current time point
    x, y, z = current_state

    # define the 3 ordinary differential equations
    # known as the lorenz equations
    dx_dt = sigma * (y - x)
    dy_dt = x * (rho - z) - y
    dz_dt = x * y - beta * z

    # return a list of the equations that describe the system
    return [dx_dt, dy_dt, dz_dt]

Data = []
for i in range(100):
    initial_state = [init_x[i], init_y[i], init_z[i]]
    # define the initial system state (aka x, y, z
    #positions in space)

    # define the system parameters sigma, rho, and beta
    sigma = 10.
    rho   = 28.
    beta  = 8./3.

    # define the time points to solve for,
    #evenly spaced between the start and end times
    start_time = 0
    end_time = 100
    time_points = np.linspace(start_time, end_time, end_time*1)
    xyz = odeint(lorenz_system, initial_state, time_points)
    Data.append(xyz)
```

C.3 INTEGRAL EQUATION SPIRALS

We defined an Integral Equation system for a 2D spiral. We then solve the system using an integral equation solver in Pytorch analytically, without the need of a neural network for the forward function. See c, d, k and f which defines the dynamical system in IESolver monoidal. See code below and example dynamics:

```
z0 = torch.Tensor([[0.1, 0.7]]).to(device)

t_max = 1
t_min = 0
n_points = 100

index_np = np.arange(0, n_points, 1, dtype=int)
index_np = np.hstack([index_np[:, None]])
times_np = np.linspace(t_min, t_max, num=n_points)
times_np = np.hstack([times_np[:, None]])

########################################################
```
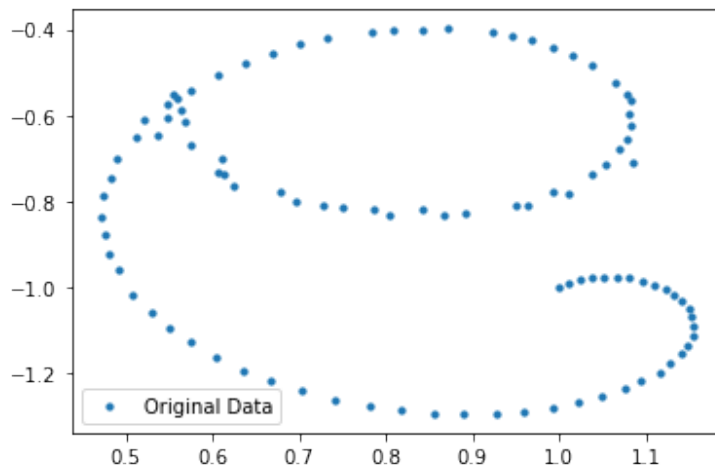
Figure 12: Example visualization of 2D Integral Equation Spiral

```
times = torch.from_numpy(times_np[:, :, None]).to(z0)
times = times.flatten()
#times = times/t_max
##############################################################
start = time.time()
solver = IESolver_monoidal(x = times.to(device),

c = lambda x: torch.Tensor([torch.cos(torch.Tensor([x])),
torch.cos(torch.Tensor(x+np.pi))]).to(device),

d = lambda x,y: torch.Tensor([1]).to(device),
k = lambda t,s: kernels.cos_kernel(2*np.pi*t,-2*np.pi*s),
f = lambda y: torch.tanh(2*np.pi*y).to(device),
lower_bound = lambda x: torch.Tensor([t_min]).to(device),
upper_bound = lambda x: x,
max_iterations = 3,
integration_dim = 0,
mc_samplings = 10000)

Data = solver.solve()
Data = Data.unsqueeze(1)
```

C.4 FMRI DATA GENERATION

The simulated fMRI data was generated using *neurolib* Cakan et al. (2021). The authors of this tool provide code to generate fMRI data for Resting-state with a given structural connectivity matrix and a delay matrix. The code can be found in their GitHub page [1]. We used this code to generate 100000 time points of data for 80 voxels corresponding to regions of the cortex.

The generated data is normalized via computing the z-score of the logarithm of the whole data. This data is then downsampled in time by a factor of 10, thus resulting in 10k time points. In our tests, we use the first 5k points, where the first 2.5k points are used for training and the remaining points are reserved for testing. During batching, each point is taken as the initial condition of a curve of length 20 points.

---

[1] https://github.com/neurolib-dev/neurolib/blob/master/examples/example-0-aln-minimal.ipynb

Table 6: Number of parameters for each experiment

	ANIE	NODE	LSTM	Residual Net
Generated fMRI	321,233	319,280	328,400	319,280
2D curves	18,819	1,654	20,862	-

D BURGERS' EQUATIONS

The Burgers' equation is a quasilinear parabolic partial differential equation that takes the form

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \tag{6}$$

where x is a spatial dimension, while t indicates time, and ν is a diffusion coefficient called *viscosity* see (Benton & Platzman, 1972). A very interesting behavior of the solutions of the Burgers' equation regards the presence of shock waves.

Our dataset is generated using the Matlab code used in Li et al. (2020a), which can be found in their GitHub page [2]. The solution is given on a spacial mesh of 1024 and 400 time points are generated from a random initial condition. We use 1000 curves for training and test on 200 unseen curves, where the interval spans $1/4$ of the original time used for testing.

E NAVIER-STOKES EQUATIONS

The Navier-Stokes equations are partial differential equations that arise in fluid mechanics, where they are used to describe the motion of viscous fluids. They are derived from the conservation laws (for momentum and mass) for Newtonian fluids subject to an external force with the addition of pressure and friction forces, where the unknown function indicates the velocity vector of the fluid (Chorin, 1968; Fefferman, 2000). Their expression is given by the system

$$\frac{\partial}{\partial t}u_i + \sum_j u_j\frac{\partial u_i}{\partial x_j} = \nu\Delta u_i - \frac{\partial p}{\partial x_i} + f_i(\mathbf{x}, t) \tag{7}$$

$$\mathrm{div}u = \sum_i \frac{\partial u_i}{\partial x_i} \tag{8}$$

where Δ is the Laplacian operator, f is the external force, and $\mathbf{u}$ is the unknown velocity function. We experiment on the same data set for $\nu = 1e-3$ of (Li et al., 2020a), which can be found in their GitHub page [3]. We use 4000 instances for training and 1000 for testing. In our tasks, we utilize a single time point to inizialize our model (ANIE) and obtain the full dynamics from a single frame. For comparison, we use the minimal number of time points allowed for the other models for comparison. This is not always possible, for instance, FNO3D cannot be applied on a single time point, or few time points, as the time convolution needs several time points to produce significant results. Despite this significant advantage given to FNO3D, ANIE (ours) still better performs on the prediction of 10 and 20 time points.

F ADDITIONAL DETAILS ON THE EXPERIMENTS AND HYPERPARAMETERS

The number of parameters for the models used in the experiments and are given in Tables 6 and 7. In all cases, the optimizer "Adam" has been employed.

---

[2]https://github.com/zongyi-li/fourier_neural_operator/tree/master/data_generation/burgers
[3]https://github.com/zongyi-li/fourier_neural_operator/tree/master/data_generation/navier_stokes

Table 7: Number of parameters for each experiment

	FNO3D	FNO2D	ANIE	Galerkin	Conv1D LSTM	Conv2DLSTM
Navier Stokes	6,558,537	414,517	1278627	-	-	447,528
Burgers	-	-	161,154	511,329	149,520	-