POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

A Comparative Study of Neural Ordinary Differential Equations and Neural Operators for Modeling Temporal Dynamics

Supervisors

Prof. Daniele APILETTI

Prof. Simone MONACO

Candidate

Matteo CELIA

December 2024

Summary

Capturing the intricate three-dimensional (3D) behavior of relational systems is a crucial challenge in natural sciences, with applications spanning from simulating molecular interactions to analyzing particle mechanics. Machine learning approaches have made significant strides in this area by using graph neural networks to learn and represent spatial interactions effectively. Neural Ordinary Differential Equations (NODE) and Neural Operators (NO) represent two powerful, yet distinct, approaches for addressing this challenge. While NODEs excel at continuous-time modeling using differential equation solvers, Neural Operators can handle more complex dynamics across varying initial conditions with a learned mapping of functions. However, it remains unclear which framework is more effective or efficient in general and with respect to specific domains like modeling the temporal evolution of complex systems, such as charged particles systems. By systematically comparing NODE and NO models, this research aims to identify the specific advantages and limitations of each approach, providing insights into their effectiveness on a specific application domain. This exploration can thus contribute to developing a deeper understanding of these emerging methods and help inform future model selection for intricate dynamical systems like the one under analysis. In recent years, different variants of the aforementioned approaches have been proposed to tackle this kind of problem. Among them, there are those on which this work focuses: SEGNO (NODE) and EGNO (NO). Both architectures use graphs to represent the relationships between objects in a system and are equivariant, meaning that their predictions do not change if the input system is rotated, translated, or reflected introducing new mechanisms to improve their generalization capability. Different experiments have been brought out, to show the respective capabilities and downfalls of both architectures. In particular, the main point consists in analyzing the ability of the models to predict multi-step trajectories in a rollout fashion, and how the number of input steps given affect their capability of reconstructing long-range dependency in the data, considering also variable distance between timesteps. The results show the limitations and advantages of the two architectures in handling irregular time series and different number of input snapshots. Overall, standard SEGNO showed to be a more robust architecture to model long multi-steps trajectories, but it seems that the adopted training strategies were not beneficial. EGNO instead, resulted as less robust in maintaining stability after a few rollout iterations but in the portion of the trajectory along which it was reliable, it showed to be more precise than SEGNO, also because it was able to benefit from the training techniques adopted.

Acknowledgements

I would like to express my gratitude to Prof. Apiletti, for giving me the chance to work on this thesis.

I want, then, to thanks Prof. Monaco for his guidance, patience, and invaluable insights throughout my research.

I also wish to thank my cousin Salvo, who helped me find the house in which i stayed for 2 years in Turin and made me feel welcomed when i didn't know anybody else.

And last but not least, I want to thank my family, who always cherished and supported me along the way.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Modeling n-body systems, such as those encountered in molecular dynamics, astrophysics, or fluid simulations, is a longstanding challenge in science and engineering. These systems involve predicting the behavior of multiple interacting entities under forces like gravity, electromagnetism, or intermolecular potentials. While traditional methods for solving these problems have been incredibly successful, they face significant limitations, particularly when dealing with large, complex systems. In recent years, data-driven approaches leveraging techniques from machine learning (ML) and deep learning (DL) have emerged as promising alternatives or complements to these traditional methods. Understanding why this shift is happening requires exploring the limitations of traditional approaches and the strengths of ML-based solutions.

Traditional approaches to modeling n-body systems generally fall into two categories: analytical and numerical methods. Analytical methods aim to solve the governing equations exactly, but this is only feasible for very simple cases, such as systems with two or three bodies. For more complex systems, the equations become intractable due to the vast number of interactions and the nonlinear nature of the forces involved. As a result, most real-world problems rely on numerical methods, which approximate solutions iteratively over time.

Take molecular dynamics (MD) as an example. In MD, the equations of motion for particles are solved step by step to simulate how atoms and molecules move and interact. This involves calculating forces between every pair of particles, which scales poorly with the number of particles typically as $O(N^2)$, though clever algorithms can reduce this. Moreover, MD simulations often require extremely small time steps to ensure accuracy and stability, making long simulations computationally expensive. Similar challenges arise in astrophysics, where simulating galaxy evolution requires tracking gravitational interactions among billions of stars over millions of years.

These traditional methods, while powerful, have clear limitations. The computational demands increase dramatically with the number of bodies and the required resolution in time and space. Additionally, modeling these systems often relies on detailed parameterizations of forces (e.g., Lennard-Jones potentials in MD or gravitational potentials in astrophysics), either approximations or empirically derived. Such parameterizations may not generalize well to new systems. Further, these simulations struggle to bridge vastly different scales, such as atomic interactions and macroscopic behavior. Finally, the deterministic nature of these methods makes them sensitive to initial conditions, which can propagate errors and lead to unreliable outcomes in chaotic systems.

This is where data-driven approaches, including machine learning and deep learning, offer a compelling alternative. Instead of solving the equations of motion directly, ML models learn patterns in the data, bypassing the need for explicit force calculations or iterative time stepping. For instance, neural networks can approximate complex, nonlinear relationships, making them well-suited for predicting the dynamics of n-body systems. Once trained, these models can perform predictions far faster than traditional methods, as they replace computationally expensive iterative processes with efficient inference steps.

One of the key reasons ML and DL are particularly relevant now is the abundance of high-quality data. Advances in both computational simulations and experimental techniques have generated large datasets that can be used to train models. For example, in molecular dynamics, quantum mechanical simulations provide detailed force and energy data that ML models can learn to replicate. Similarly, astrophysical datasets, including those from telescopes and numerical simulations, offer opportunities for training models to predict galaxy or star system dynamics.

The advantages of data-driven methods are numerous. First, they are computationally efficient once trained. For example, instead of calculating pairwise forces for millions of particles at every time step, a neural network can predict the dynamics directly, saving considerable time and resources. Second, these models are flexible and can generalize to systems beyond those they were trained on, provided the underlying physics is similar. Third, ML models excel at capturing nonlinear interactions and complex dependencies, often surpassing the approximations used in traditional methods. Lastly, they provide a means to integrate data from multiple sources, such as experimental results and simulations, creating hybrid models that are both accurate and efficient.

Molecular dynamics is a particularly illustrative example of the potential for data-driven methods. In MD, simulating systems at the atomic scale involves solving equations for millions of atoms over millions of time steps. By training ML models on simulation data, it is possible to predict forces or even entire trajectories without repeatedly solving these equations. This can dramatically accelerate tasks like predicting equilibrium states, studying folding pathways of proteins, or computing free energy landscapes. Moreover, ML models can be used to develop surrogate models for complex quantum mechanical calculations, enabling hybrid quantum-classical simulations that are both accurate and computationally feasible.

However, data-driven approaches are not without their challenges. They are highly dependent on the quality and diversity of training data, and their ability to generalize to unseen scenarios is limited by the scope of the data. Unlike traditional methods, which are built on well-understood physical principles, ML models often function as black boxes, making their predictions harder to interpret. Stability is another concern: while traditional methods are designed to respect conservation laws (e.g., energy and momentum), ML models can sometimes violate these, leading to unphysical results.

In summary, the use of machine learning and deep learning for modeling n-body systems represents a shift in how these complex problems are tackled. While traditional methods rely on direct computation of interactions based on well-defined rules, ML models learn these interactions from data, offering significant advantages in speed and scalability. For tasks like molecular dynamics, where simulations are computationally expensive and involve intricate interactions, ML approaches can dramatically reduce computational costs while maintaining or even improving accuracy. As data-driven techniques continue to evolve, they are poised to become indispensable tools for tackling the many challenges associated with n-body systems. Among the deep learning approaches to tackle this kind of problems, Graph Neural Networks (GNNs) have become one of the main used tools.

1.2 Related work

1.2.1 Graph Neural Networks

Graph Neural Networks (GNNs [1, 2]) have emerged as powerful data-driven tools for modeling the dynamics of n-body systems, where interactions between entities whether particles, stars, or molecules—play a central role in the system's evolution. By representing these systems as graphs, GNNs offer a natural and highly flexible framework for capturing complex, multi-body interactions in an efficient, interpretable, and scalable way.

At their core, n-body systems involve a set of entities (bodies) that interact through forces, leading to changes in their states (e.g., positions, velocities). In the graph paradigm nodes represent individual bodies or particles while edges capture the interactions or relationships between these bodies, such as gravitational attraction, electrostatic forces, or intermolecular potentials. Node attributes might include properties like mass, charge, position, and velocity, while edge attributes might encode pairwise distances, interaction strengths, or other relevant quantities. This graph-based representation aligns well with the mathematical structure of n-body systems, where interactions between entities depend on pairwise or local properties, and the dynamics are driven by aggregating these interactions for each body.

GNNs are specifically designed to operate on graph-structured data, making them ideally suited to learn and model the dynamics of n-body systems. Their key strength lies in their ability to capture relational information and aggregate interactions from connected nodes and edges in a graph. This makes them a natural choice for tasks like predicting forces, energies, or trajectories in systems where interactions are complex and interdependent. The process used by GNNs is called message passing and it generally works in the following way:
For each node:

1. Send Messages: The node gathers information from its neighbors. Each neighbor sends a "message", which is computed using its own features and possibly the edge connecting them.

2. Aggregate Messages: The node collects all the messages from its neighbors and combines them (e.g., by summing or averaging the messages).

3. Update the Node: The node updates its own features based on the aggregated messages and its current state.

This process captures how each node interacts with its local neighborhood. The message-passing step is repeated for several rounds (or "layers"). In each round, the information spreads further in the graph.

Modeling the dynamics of n-body systems involves predicting the evolution of entities under mutual interactions, which are governed by fundamental physical laws. These laws, such as Newton's laws of motion or Coulomb's law for electric forces, exhibit specific symmetries:

- Translation Invariance: The dynamics depend only on relative positions, not absolute coordinates.
- Rotation and Reflection Invariance: The forces and resulting dynamics should not change if the system is rotated or mirrored.
- Permutation Invariance: Swapping the labels of two indistinguishable particles should not alter the dynamics.

Traditional GNNs can approximate these relationships, but without explicit enforcement of these symmetries, they require large amounts of data to learn them implicitly. This inefficiency can lead to suboptimal generalization and predictions that violate physical principles. For instance, a model might predict forces that are not rotation-invariant or trajectories that fail to conserve energy or momentum. Such violations are especially problematic in long-term simulations of n-body systems, where small errors can compound over time.

Equivariant GNNs ([3]) arose to address these limitations by explicitly embedding geometric symmetries into their architecture. They are designed to ensure that their outputs transform in predictable ways when the inputs undergo certain transformations. By enforcing such equivariance, these models align naturally with the structure of physical laws, reducing the need for the model to "learn" these symmetries from data and ensuring that predictions respect these kind of fundamental principles. EGNNs possess equivariance to roto-translational transformations in the Euclidean space, which has been demonstrated as a vital inductive bias to improve generalization ([4]; [5]; [6]; [7]).

Traditionally, GNNs like EGNN (Equivariant Graph Neural Networks) have been employed to predict the state of a system at a specific time step, relying on discrete state transformation layers to learn direct mappings between adjacent states. This approach, however, overlooks the inherent continuity of system trajectories. Two prominent EGNN architectures, SEGNO (Second-order Equivariant Graph Neural Ordinary Differential Equation [8]) and EGNO (Equivariant Graph Neural Operator [9]), have recently shown promising results in this context.

SEGNO addresses this limitations by incorporating Neural Ordinary Differential Equations (Neural ODEs) to approximate the continuous trajectory between observed states. By parameterizing the acceleration function using a GNN, SEGNO leverages second-order motion equations to update the system's position and velocity, thereby capturing the underlying dynamics more accurately.

EGNO, on the other hand, tackles the limitations of traditional GNNs that focus solely on next-step predictions and neglect temporal correlations. It directly models the dynamics as trajectories using neural operators, specifically Fourier Neural Operators (FNOs). EGNO learns the temporal evolution of the system by formulating the dynamics as a function over time. To ensure SE(3)-equivariance, EGNO employs equivariant temporal convolutions in the Fourier space, allowing it to effectively capture temporal correlations while preserving the desired symmetries.

However, it still remains unclear which framework is better overall and with respect to specific aspects taken into consideration, particularly in modeling the temporal evolution of complex systems such as charged particle systems.
The families of models from which SEGNO and EGNO stem, NODE (Neural Ordinary Differential Equations [10]) and NO (Neural Operators [11]) respectively, are gaining more and more interest, becoming the two most valid approaches in this scenario.

1.2.2 Neural Ordinary Differential Equations

Neural Ordinary Differential Equations (Neural ODEs [10]) represent a significant innovation in deep learning, providing a continuous-time perspective on how neural networks process information. Unlike traditional neural networks, which use a series of discrete layers to transform inputs into outputs, Neural ODEs model these transformations as solutions to differential equations. This allows the network's behavior to be described by a continuous, dynamic system.

In standard neural networks, data is passed through a fixed number of discrete layers, each performing a transformation. In Neural ODEs, the depth of the network is treated as a continuous variable, with transformations described by an ODE solver. Instead of pre-defining how many "layers" the model has, the ODE solver computes the necessary transformations over a continuous interval.

Neural ODEs require fewer parameters compared to deep, discrete-layer networks because the transformation is determined by the underlying dynamics of the system rather than a series of separate layers. This makes them especially useful in situations where parameter efficiency is critical. The use of ODE solvers allows Neural ODEs to adaptively allocate computational resources. Complex regions of the input space can be explored with finer precision (by using smaller time steps), while simpler regions require fewer computations. This adaptability makes them both efficient and versatile. Moreover, they use a technique called the adjoint method for backpropagation, which computes gradients by solving a second ODE backward in time. This approach requires less memory than storing all intermediate states, making Neural ODEs suitable for memory-constrained environments.

They are naturally suited for time-series data and dynamical systems because they inherently model data as evolving over continuous time. This makes them highly effective in applications like physics simulations, trajectory prediction, and modeling biological or financial systems. By modeling transformations continuously, Neural ODEs produce smooth, well-behaved functions that often generalize better to unseen data, especially in tasks where continuity and smooth transitions are important.

SEGNO

In SEGNO, it has been taken a deep insight into the continuity and second-order inductive bias in Equiv-GNNs and proposed a framework named Second-order Equivariant Graph Neural Ordinary Differential Equation (SEGNO). Differently from previous models that use Equiv-GNNs to fit discrete kinematic states, SEGNO introduces Neural Ordinary Differential Equations (Neural ODE) to approximate a continuous trajectory between two observed states. Furthermore, to better estimate the underlying dynamics, SEGNO is built upon second-order motion equations to update the position and velocity of the physical systems. Theoretically, it has been proven the uniqueness of the learned latent trajectory of SEGNO and further provided an upper bound on the discrepancy between the learned and the actual latent trajectory.

Meanwhile, it can be proven that SEGNO can maintain equivariance properties identical to the backbone Equiv-GNNs. This property offers the flexibility to adapt various backbones in SEGNO to suit different downstream tasks in plug-and-play manner.

1.2.3 Neural Operators

Neural Operators ([11]) are a groundbreaking approach in machine learning designed to approximate mappings between infinite-dimensional spaces, such as those encountered in solving partial differential equations (PDEs) or modeling complex physical systems. Unlike traditional neural networks, which map finite-dimensional inputs to outputs, neural operators generalize this idea to handle functions as both inputs and outputs. This capability makes them a powerful tool for learning and solving problems governed by complex, multiscale dynamics.

Traditional neural networks learn relationships between fixed-size input vectors and output vectors. Neural operators, on the other hand, learn mappings between functions. For example, they can map an input field, such as initial conditions or forcing terms of a PDE, to a solution field, allowing them to model physical systems with high fidelity.

Neural operators decouple the learning process from the discretization of the input or output spaces. This means that once trained, they can generalize across different grid resolutions, enabling predictions on higher-resolution data without retraining. This contrasts with standard neural networks, which are often tied to the resolution of their training data.

Traditional numerical solvers for PDEs, like finite element or finite difference methods, suffer from the curse of dimensionality, where computational costs grow exponentially with the dimensionality of the problem. Neural operators overcome this by learning a compact representation of the operator, allowing for efficient computations even in high-dimensional spaces.

Neural operators leverage a framework where the same set of parameters can handle different input functions or conditions. This generality reduces the need for system-specific training, enabling the model to adapt to variations in boundary conditions, geometries, or forcing terms.

Some neural operator architectures, like the Fourier Neural Operator (FNO), incorporate domain-specific knowledge, such as using Fourier transforms to handle spatial dependencies. This enhances interpretability and aligns the model with the physical nature of the problem, ensuring that it respects inherent properties like smoothness or periodicity.

Neural operators are well-suited for modern computational architectures. By design, they leverage linear-algebra-heavy operations that can be efficiently parallelized on GPUs or other hardware accelerators, making them scalable to large datasets and complex problems.

EGNO

In Equivariant Graph Neural Operator for Modeling 3D Dynamics (EGNO [9]) a novel and principled method to overcome the aforementioned challenge has been presented: by directly modeling the entire trajectory dynamics instead of just the next time-step prediction. Differently from existing approaches, EGNO predicts dynamics as a temporal function that is not limited to a fixed discretization. This framework is based on the Neural Operator (NO [11]), and in particular on the

Fourier neural operator (FNO [12]; [13]) which has shown great effectiveness in learning maps between function spaces with desirable discretization convergence guarantees.

The core idea is to formulate the physical dynamics as a function over time and learn neural operators to approximate it. The main challenge in developing EGNO is to capture the temporal correlations while still keeping the SE(3)-equivariance in the Euclidean space. To this end, it has been developed equivariant temporal convolution layers in the Fourier space and realize EGNO by stacking them with equivariant networks. The key innovation is that was noticed the equivariance property of Fourier and inverse Fourier transforms and that it can be kept in Fourier space with special kernel integral operators. The resulting EGNO architecture is the first efficient operator learning framework that is capable of mapping a current state directly to the solution trajectories, while retaining 3D spatial equivariance.

As mentioned, EGNO explicitly learns to model the trajectory while keeping the intrinsic symmetries in Euclidean space. This, in practice, leads to more expressive modeling of underlying dynamics and achieve higher state prediction accuracy. Moreover, the operator formulation enables efficient parallel decoding of future states (within a time window) with just one model inference, and the model is not limited to one fixed temporal discretization. This allows users to run dynamics inference at any timestep size without switching model parameters. Finally, the employed temporal convolutional layer is general and can be easily combined with any specially designed EGNN layers. This permits EGNO to be easily deployed in a wide range of different physical dynamics scenarios.

1.3 Objectives

This study undertakes a systematic comparison of NODE (SEGNO) and NO (EGNO) models to uncover their respective strengths and weaknesses. By focusing on their performance within a specific application domain, the research seeks to offer valuable insights into the suitability of each approach. Ultimately, this investigation aims to deepen our understanding of these emerging methodologies and guide future model selection for complex dynamical systems like the one under consideration.

Comparing SEGNO and EGNO reveals distinct approaches to modeling n-body system dynamics. While SEGNO emphasizes the continuity of trajectories through Neural ODEs and second-order motion equations, EGNO focuses on capturing temporal correlations using neural operators. This thesis will compare and contrast

SEGNO and EGNO, examining their strengths and limitations in modeling the dynamics of various n-body systems. The analysis will take into account different factors such as prediction accuracy and stability over long trajectories, computational efficiency, and generalization ability.

Chapter 2

Background

This chapter will focus on describing the most influential and fundamental works that constitute the background of the current analysis.

2.1 Graph Neural Networks

Deep learning has revolutionized many machine learning tasks in recent years, ranging from image classification and video processing to speech recognition and natural language understanding. The data in these tasks are typically represented in the Euclidean space. However, there is an increasing number of applications where data are generated from non-Euclidean domains and are represented as graphs with complex relationships and interdependency between objects. The complexity of graph data has imposed significant challenges on existing machine learning algorithms.

Graph neural networks (GNNs explained extensively in [14]) are a category of deep learning models designed for handling graph-structured data. They address tasks related to graphs in an end-to-end manner. GNNs provide a way to apply deep learning principles to graph data by generalizing key operations like convolution to the graph domain. For instance, graph convolution involves aggregating information from a node's neighbors to create a representation of that node.

GNNs excel at capturing complex, non-linear relationships within graph data, which traditional methods often struggle with. They can be adapted to address various graph-related tasks, including node classification, graph classification, link prediction, and graph generation.

2.1.1 Spatial-Temporal Graph

With regard to the analyzed models, the type of graphs that need to be considered are Spatial-Temporal Graphs that, in this case, represent evolving systems of n charged particles. A spatial-temporal graph is an attributed graph where the node attributes change dynamically over time. The spatial-temporal graph is defined as $G^{(t)} = (\mathbf{V}, \mathbf{E}, \mathbf{X}^{(t)}, \mathbf{W}^{(t)})$ where $\mathbf{V}$ is the set of vertices or nodes, $\mathbf{E}$ is the set of edges and $\mathbf{X}^{(t)} \in \mathbf{R}^{n \times d}$ is the node feature matrix considering n nodes and d as the feature vector dimensionality. The edges remain consistent in terms of connectivity (topology), but their weights or features $\mathbf{W}^{(t)}$ change dynamically where usually edge features are computed based on node states (e.g., distance or potential energy).

2.1.2 Equivariant Graph Neural Networks (EGNN)

When talking about Multi-Object Physical Systems, GNNs can learn how these systems evolve by applying message passing layers to the graph representation. This allows information to propagate between nodes, capturing the complex dynamics of the system.

However, traditional GNNs have limitations when dealing with physical systems, mainly due to their inability to naturally incorporate crucial physical inductive biases and to their inability to learn on positional graphs:

Discrete Transformations: Conventional GNNs typically learn a direct mapping between adjacent states, represented by discrete jumps in the system's trajectory. This approach fails to capture the continuous nature of transitions in real-world physical systems.

Limited Order of Information: Most GNNs only consider first-order information like velocity, neglecting the second-order laws (like acceleration in Newton's laws) governing many physical phenomena.

Equivariant Graph Neural Networks (EGNNs [3]) emerged to address some of these limitations, particularly the challenge of incorporating physical symmetries. EGNNs are designed to ensure that the output of the network transforms predictably in response to specific transformations applied to the input. This property is crucial for modeling physical systems, where the underlying laws should hold regardless of the system's position or orientation.

E(3) and SE(3) Equivariance refers to equivariance to transformations like translations, rotations, and reflections in 3D Euclidean space. SE(3)-equivariance is a specific case focusing on rigid body motions, excluding reflections.

By embedding these symmetries directly into their structure, EGNNs can more accurately and reliably learn the dynamics of physical systems. Moreover, by construction, these models are less sensitive to variations caused by transformations of the input data, leading to better performance on unseen data hence better generalization.

2.2 Neural Ordinary Differential Equation (NODE)

Neural Ordinary Differential Equations (Neural ODEs [10]) are a class of machine learning models introduced to model continuous-time dynamics. Unlike traditional neural networks that compute discrete transformations between layers, Neural ODEs represent the forward pass of a network as a continuous process governed by an ordinary differential equation (ODE).

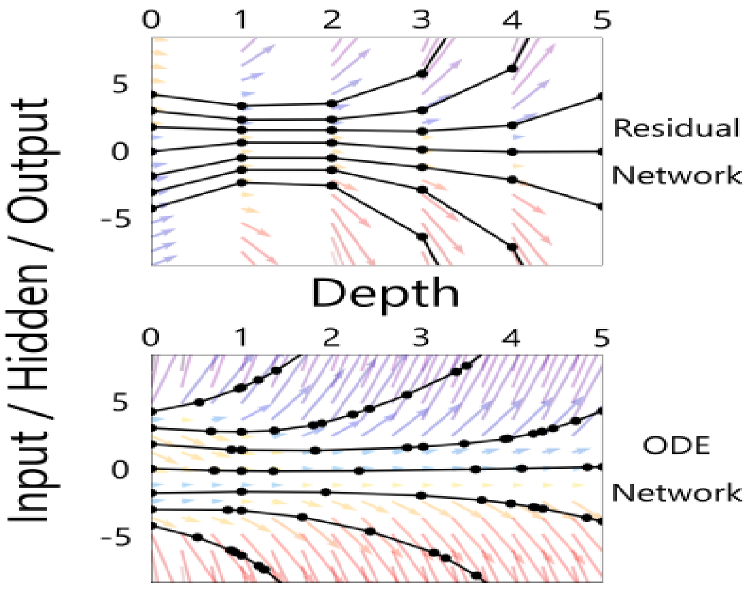


Figure 2.1: top: A Residual network defines a discrete sequence of finite transformations; **bottom:**A ODE network defines a vector field, which continuously transforms the state. **Both:** Circles represent evaluation locations.

Models such as residual networks, recurrent neural network decoders, and normalizing flows build complicated transformations by composing a sequence of transformations to a hidden state:

$$\mathbf{h_{t+1}} = \mathbf{h_t} + f(\mathbf{h_t}, \theta_t) \tag{2.1}$$

where $t \in \{0 \ldots T\}$ and $h_t \in R^D$. These iterative updates can be seen as an Euler discretization of a continuous transformation.

As more layers are added and smaller steps are taken, in the limit, the continuous dynamics of hidden units, is parametrized using an ordinary differential equation (ODE) specified by a neural network:

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h_t}, t, \theta) \tag{2.2}$$

Starting from the input layer $\mathbf{h}(0)$, it can be defined the output layer $\mathbf{h}(T)$ to be the solution to this ODE initial value problem at some time T. This value can be computed by a black-box differential equation solver, which evaluates the hidden unit dynamics f wherever necessary to determine the solution with the desired accuracy. The forward pass involves integrating the ODE from t_0 to t_1:

$$\mathbf{h}(t_1) = \mathbf{h}(t_0) + \int_{t_0}^{t_1} f(\mathbf{h}(t), t, \theta)\, dt \tag{2.3}$$

This integration is performed using numerical solvers such as Runge-Kutta, Euler, etc. The function f represent a neural network parameterized by weights θ which acts as the vector field. Unlike traditional networks, NODEs rely on continuous dynamics, making backpropagation different. The gradients with respect to the parameters θ are computed using the adjoint sensitivity method. This process ensures efficient memory usage and scalability. In practice, learning process in NODEs revolves around treating neural networks as vector fields that define continuous transformations. The use of numerical solvers and the adjoint method for gradient computation makes NODEs a unique and powerful tool for tasks involving continuous-time dynamics. Despite challenges like computational overhead, NODEs offer a flexible and interpretable approach to learning complex dynamical systems. One example of application is learning dynamical systems like n-body problems or fluid dynamics where NODEs can approximate physical laws.

Defining and evaluating models using ODE solvers has several benefits:

Memory efficiency: It can be shown that is possible to compute gradients of a scalar-valued loss with respect to all inputs of any ODE solver, without back-propagating through the operations of the solver by using reverse-mode automatic differentiation.

Adaptive Computation: Euler's method is perhaps the simplest method for solving ODEs. There have since been more than 120 years of development of efficient and accurate ODE solvers. Modern ODE solvers provide guarantees about the growth of approximation error, monitor the level of error, and adapt their evaluation strategy on the fly to achieve the requested level of accuracy. This allows the cost of evaluating a model to scale with problem complexity. After training, accuracy can be reduced for real-time or low-power applications.

Continuous time-series models: Unlike recurrent neural networks, which require discretizing observation and emission intervals, continuously-defined dynamics can naturally incorporate data which arrives at arbitrary times.

One of the possible applications of this type of models can be found in modeling phenomena governed by physical laws, such as fluid dynamics or n-body systems.

2.3 Neural Operators (NO)

Learning mappings between function spaces has widespread applications in science and engineering. For instance, for solving differential equations, the input is a coefficient function and the output is a solution function. A straightforward solution to this problem is to simply discretize the infinite-dimensional input and output function spaces into finite-dimensional grids, and apply standard learning models such as neural networks. However, this limits applicability since the learned neural network model may not generalize well to different discretizations, beyond the discretization grid of the training data. To overcome these limitations of standard neural networks, a new deep-learning framework for learning operators has been formulated, called neural operators, which directly map between function spaces on bounded domains.

Since the neural operator is designed on function spaces, they can be discretized by a variety of different methods, and at different levels of resolution, without the need for re-training. In contrast, standard neural network architectures depend heavily on the discretization of training data: new architectures with new parameters may be needed to achieve the same error for data with varying discretization. It was also proposed the notion of discretization-invariant models and proven that neural operators defined in some way satisfy this property, while standard neural networks do not.

It has been introduced the concept of neural operators for learning operators, that are mappings between infinite-dimensional function spaces. The proposed neural operator architectures are multi-layers, where layers are themselves operators composed with non-linear activations. This ensures that that the overall end-to-end composition is an operator, and thus satisfies the discretization invariance property.

The key design choice for neural operator is the operator layers. Since these layers are composed with non-linear activations, the obtained neural operator models are expressive and able to capture any continuous operator, hence they have the universal approximation property. Neural operators replace finite-dimensional linear layers in neural networks with linear operators in function spaces.

Several design choices have been proposed for the linear operator layers in neural operator such as a parameterized integral operator or through multiplication in the spectral domain as showed in Figure 2.2.

Note: for most of the information in this section refer to [15] as source.

General setting

In this section, the neural operator framework is outlined. We assume that the input functions $a \in A$ are $\mathbb{R}^{d_a}$-valued and defined on the bounded domain $D \subset \mathbb{R}^d$ while the output functions $u \in U$ are $\mathbb{R}^{d_u}$-valued and defined on the bounded domain $D' \subset R^{d'}$. The considered architecture $F_\theta : A \to U$ has the following overall structure:

1. **Lifting**: Using a pointwise function $\mathbb{R}^{d_a} \to \mathbb{R}^{d_{v_0}}$, map the input $\{a : D \to \mathbb{R}^{d_a}\} \mapsto \{v_0 : D \to \mathbb{R}^{d_{v_0}}\}$ to its first hidden representation. Usually, we choose $d_{v_0} > d_a$ and hence this is a lifting operation performed by a fully local operator.

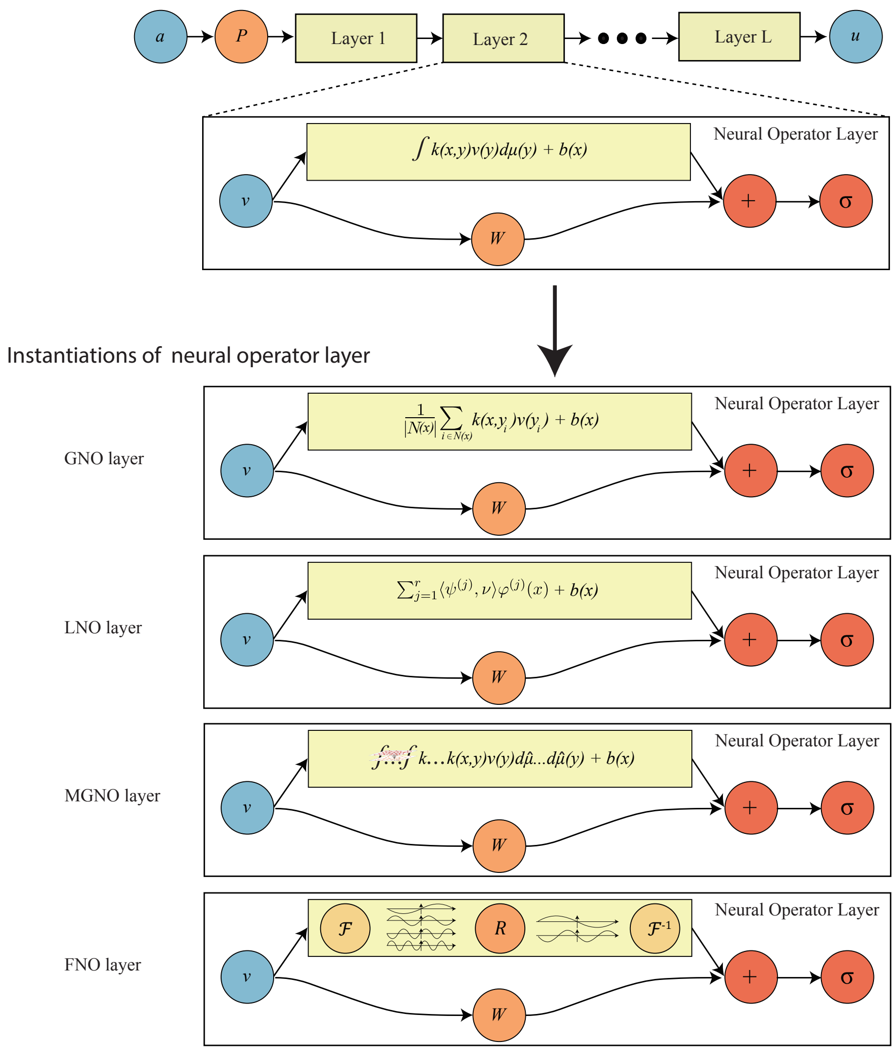
2. **Iterative Kernel Integration**: For $t = 0, \dots, T-1$, map each hidden representation to the next $\{v_t : D_t \to \mathbb{R}^{d_{v_t}}\} \mapsto \{v_{t+1} : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}\}$ via the action of the sum of a local linear operator, a non-local integral kernel operator, and a bias function, composing the sum with a fixed, pointwise nonlinearity. Here we set $D_0 = D$ and $D_T = D'$ and impose that $D_t \subset \mathbb{R}^{d_t}$ is a bounded domain.[1]

3. **Projection**: Using a pointwise function $\mathbb{R}^{d_{v_T}} \to \mathbb{R}^{d_u}$, map the last hidden representation $\{v_T : D' \to \mathbb{R}^{d_{v_T}}\} \mapsto \{u : D' \to \mathbb{R}^{d_u}\}$ to the output function. Analogously to the first step, we usually pick $d_{v_T} > d_u$ and hence this is a projection step performed by a fully local operator.

The outlined structure mimics that of a finite dimensional neural network where hidden representations are successively mapped to produce the final output. In particular, we have

$$\mathcal{G}_\theta \coloneqq \mathcal{Q} \circ \sigma_T(W_{T-1} + \mathcal{K}_{T-1} + b_{T-1}) \circ \cdots \circ \sigma_1(W_0 + \mathcal{K}_0 + b_0) \circ \mathcal{P} \tag{2.4}$$

[1]The index t is not the physical time, but the iteration (layer) in the model architecture.



a
P
Layer 1
Layer 2
Layer L
u
Neural Operator Layer
$\int k(x,y)v(y)d\mu(y) + b(x)$
v
W
+
σ
Instantiations of neural operator layer
GNO layer
$\frac{1}{|N(x)|}\sum_{i\in N(x)} k(x,y_i)v(y_i) + b(x)$
LNO layer
$\sum_{j=1}^{r}\langle\psi^{(j)},v\rangle\varphi^{(j)}(x) + b(x)$
MGNO layer
$\int\ldots\int k\ldots k(x,y)v(y)d\hat{\mu}\ldots d\hat{\mu}(y) + b(x)$
FNO layer
$\mathcal{F}$
R
$\mathcal{F}^{-1}$

Figure 2.2: Neural operator architecture schematic The input function a is passed to a pointwise lifting operator P that is followed by T layers of integral operators and pointwise non-linearity operations σ. In the end, the pointwise projection operator Q outputs the function u. Three instantiation of neural operator layers, GNO, LNO, and FNO are provided.

where $\mathcal{P} : \mathbb{R}^{d_a} \to \mathbb{R}^{d_{v_0}}$, $\mathcal{Q} : \mathbb{R}^{d_{v_T}} \to \mathbb{R}^{d_u}$ are the local lifting and projection mappings respectively, $W_t \in \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}}$ are local linear operators (matrices),

$\mathcal{K}_t : \{v_t : D_t \to \mathbb{R}^{d_{v_t}}\} \to \{v_{t+1} : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}\}$ are integral kernel operators, $b_t : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}$ are bias functions, and σ_t are fixed activation functions acting locally as maps $\mathbb{R}^{v_{t+1}} \to \mathbb{R}^{v_{t+1}}$ in each layer. The output dimensions $d_{v_0}, \ldots, d_{v_T}$ as well as the input dimensions $d_1, \ldots, d_{T-1}$ and domains of definition $D_1, \ldots, D_{T-1}$ are hyperparameters of the architecture. By local maps, we mean that the action is pointwise, in particular, for the lifting and projection maps, we have $(\mathcal{P}(a))(x) = \mathcal{P}(a(x))$ for any $x \in D$ and $(\mathcal{Q}(v_T))(x) = \mathcal{Q}(v_T(x))$ for any $x \in D'$ and similarly, for the activation, $(\sigma(v_{t+1}))(x) = \sigma(v_{t+1}(x))$ for any $x \in D_{t+1}$. The maps, $\mathcal{P}$, $\mathcal{Q}$, and σ_t can thus be thought of as defining Nemitskiy operators when each of their components are assumed to be Borel measurable. This interpretation allows us to define the general neural operator architecture when pointwise evaluation is not well-defined in the spaces $\mathcal{A}$ or $\mathcal{U}$ e.g. when they are Lebesgue, Sobolev, or Besov spaces.

The crucial difference between this architecture and a standard feed-forward neural network is that all operations are directly defined in function space (noting that the activation funtions, $\mathcal{P}$ and $\mathcal{Q}$ are all interpreted through their extension to Nemitskiy operators) and therefore do not depend on any discretization of the data. Intuitively, the lifting step locally maps the data to a space where the non-local part of $\mathcal{G}^\dagger$ is easier to capture. The non-local part of $\mathcal{G}^\dagger$ is then learned by successively approximating using integral kernel operators composed with a local nonlinearity. Each integral kernel operator is the function space analog of the weight matrix in a standard feed-forward network since they are infinite-dimensional linear operators mapping one function space to another.

The final projection step simply gets us back to the space of the output function. We consider the concatenation $\theta \in \mathbb{R}^p$ of the parameters of $\mathcal{P}$, $\mathcal{Q}$, $\{b_t\}$ which are usually themselves shallow neural networks, the parameters of the kernels representing $\{\mathcal{K}_t\}$ which are again usually shallow neural networks, and the matrices $\{W_t\}$. Note, however, that this framework is general and other parameterizations such as polynomials may also be employed.

Neural operators excel at approximating solutions to PDEs, which are central in modeling physical phenomena. Therefore, they are suitable for various application domains:

- Fluid Dynamics: Solving Navier-Stokes equations for turbulent flows, weather modeling, and ocean currents.

- Electromagnetics: Modeling electromagnetic fields for antenna design or wireless communication.

- Structural Mechanics: Predicting stress-strain responses in materials. Quantum Physics: Simulating quantum systems or solving Schrödinger's equations.

One of the most prominent NO that arose in recent times is the Fourier Neural Operator (FNO).

2.3.1 Fourier Neural Operator (FNO)

Instead of working with a kernel directly on the domain D, we may consider its representation in Fourier space and directly parameterize it there. This allows us to utilize Fast Fourier Transform (FFT) methods in order to compute the action of the kernel integral operator with almost linear complexity. The outlined method was first described in [16] and is termed the Fourier Neural Operator (FNO). For simplicity, we will assume that $D = \mathbb{T}^d$ is the unit torus and all functions are complex-valued. Let $\mathcal{F} : L^2(D; \mathbb{C}^n) \to \ell^2(\mathbb{Z}^d; \mathbb{C}^n)$ denote the Fourier transform of a function $v : D \to \mathbb{C}^n$ and $\mathcal{F}^{-1}$ its inverse. For $v \in L^2(D; \mathbb{C}^n)$ and $w \in \ell^2(\mathbb{Z}^d; \mathbb{C}^n)$, we have

$$\begin{aligned}
(\mathcal{F}v)_j(k) &= \langle v_j, \psi_k \rangle_{L^2(D;\mathbb{C})}, && j \in \{1, \dots, n\}, \quad k \in \mathbb{Z}^d, \\
(\mathcal{F}^{-1}w)_j(x) &= \sum_{k \in \mathbb{Z}^d} w_j(k)\psi_k(x), && j \in \{1, \dots, n\}, \quad x \in D
\end{aligned}$$

where, for each $k \in \mathbb{Z}^d$, we define

$$\psi_k(x) = e^{2\pi i k_1 x_1} \cdots e^{2\pi i k_d x_d}, \qquad x \in D$$

with $i = \sqrt{-1}$ the imaginary unit. By letting $\kappa(x, y) = \kappa(x - y)$ for some $\kappa : D \to \mathbb{C}^{m\times n}$ and applying the convolution theorem, we find that

$$u(x) = \mathcal{F}^{-1}\Big(\mathcal{F}(\kappa) \cdot \mathcal{F}(v)\Big)(x) \qquad \forall x \in D.$$

Therefore it has been proposed to directly parameterize κ by its Fourier coefficients. We can write

$$u(x) = \mathcal{F}^{-1}\Big(R_\phi \cdot \mathcal{F}(v)\Big)(x) \qquad \forall x \in D \tag{2.5}$$

where R_ϕ is the Fourier transform of a periodic function $\kappa : D \to \mathbb{C}^{m\times n}$ parameterized by some $\phi \in \mathbb{R}^p$.

For frequency mode $k \in \mathbb{Z}^d$, we have $(\mathcal{F}v)(k) \in \mathbb{C}^n$ and $R_\phi(k) \in \mathbb{C}^{m\times n}$. We pick a finite-dimensional parameterization by truncating the Fourier series at a maximal number of modes $k_{\max} = |Z_{k_{\max}}| = |\{k \in \mathbb{Z}^d : |k_j| \le k_{\max,j}, \text{ for } j = 1, \dots, d\}|$. This choice improves the empirical performance and sensitivity of the resulting model with respect to the choices of discretization. We thus parameterize R_ϕ directly as complex-valued $(k_{\max} \times m \times n)$-tensor comprising a collection of truncated

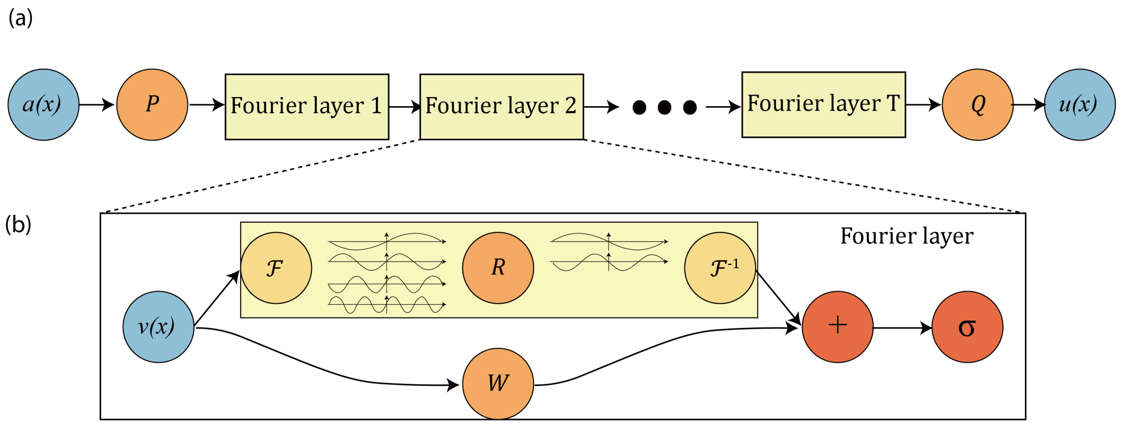


Figure 2.3: The full architecture of neural operator showing the main steps that comprehend: lift to higher dimension, apply T layers of integral operators and then project back to the target dimension.

Fourier modes and therefore drop ϕ from our notation. In the case where we have real-valued v and we want u to also be real-valued, we impose that κ is real-valued by enforcing conjugate symmetry in the parameterization i.e.

$$R(-k)_{j,l} = R^*(k)_{j,l} \qquad \forall k \in Z_{k_{\max}}, \quad j = 1, \ldots, m, \;\; l = 1, \ldots, n.$$

Note that the set $Z_{k_{\max}}$ is not the canonical choice for the low frequency modes of v_t. Indeed, the low frequency modes are usually defined by placing an upper-bound on the ℓ_1-norm of $k \in \mathbb{Z}^d$. $Z_{k_{\max}}$ is chosen as above since it allows for an efficient implementation. Figure 2.3 gives a pictorial representation of an entire Neural Operator architecture employing Fourier layers. The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network $\mathcal{P}$. 2. Apply T (typically $T = 4$) layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. (b) Fourier layers: Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes which also filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

The Discrete Case and the FFT. Assuming the domain D is discretized with $J \in \mathbb{N}$ points, we can treat $v \in \mathbb{C}^{J \times n}$ and $\mathcal{F}(v) \in \mathbb{C}^{J \times n}$. Since we convolve v with a function which only has $k_{\max}$ Fourier modes, we may simply truncate the higher modes to obtain $\mathcal{F}(v) \in \mathbb{C}^{k_{\max} \times n}$. Multiplication by the weight tensor $R \in \mathbb{C}^{k_{\max} \times m \times n}$ is then

$$\big(R \cdot (\mathcal{F}v_t)\big)_{k,l} = \sum_{j=1}^{n} R_{k,l,j} (\mathcal{F}v)_{k,j}, \qquad k = 1, \ldots, k_{\max}, \quad l = 1, \ldots, m. \tag{2.6}$$

When the discretization is uniform with resolution $s_1 \times \cdots \times s_d = J$, $\mathcal{F}$ can be replaced by the Fast Fourier Transform. For $v \in \mathbb{C}^{J \times n}$, $k = (k_1, \ldots, k_d) \in \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_d}$, and $x = (x_1, \ldots, x_d) \in D$, the FFT $\hat{\mathcal{F}}$ and its inverse $\hat{\mathcal{F}}^{-1}$ are defined as

$$(\hat{\mathcal{F}}v)_l(k) = \sum_{x_1=0}^{s_1-1} \cdots \sum_{x_d=0}^{s_d-1} v_l(x_1, \ldots, x_d) e^{-2i\pi \sum_{j=1}^{d} \frac{x_j k_j}{s_j}},$$
$$(\hat{\mathcal{F}}^{-1}v)_l(x) = \sum_{k_1=0}^{s_1-1} \cdots \sum_{k_d=0}^{s_d-1} v_l(k_1, \ldots, k_d) e^{2i\pi \sum_{j=1}^{d} \frac{x_j k_j}{s_j}}$$

for $l = 1, \ldots, n$. In this case, the set of truncated modes becomes

$$Z_{k_{\max}} = \{(k_1, \ldots, k_d) \in \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_d} \mid k_j \leq k_{\max,j} \text{ or } s_j - k_j \leq k_{\max,j}, \text{ for } j = 1, \ldots, d\}.$$

When implemented, R is treated as a $(s_1 \times \cdots \times s_d \times m \times n)$-tensor and the above definition of $Z_{k_{\max}}$ corresponds to the "corners" of R, which allows for a straight-forward parallel implementation of (2.6) via matrix-vector multiplication. In practice, it has been found that the choice of $k_{\max,j}$ roughly around $\frac{1}{3}$ to $\frac{2}{3}$ of the maximum number of Fourier modes in the Fast Fourier Transform of the grid valuation of the input function provides desirable performance.

Invariance to Discretization. The Fourier layers are discretization-invariant because they can learn from and evaluate functions which are discretized in an arbitrary way. Since parameters are learned directly in Fourier space, resolving the functions in physical space simply amounts to projecting on the basis elements $e^{2\pi i \langle x,k \rangle}$; these are well-defined everywhere on $\mathbb{C}^d$.

Chapter 3

Methods

In this chapter, the analyzed architectures are going to be presented, alongside with the qualitative and quantitative experimental setting adopted.

3.1 Architectures

This section will briefly present the SEGNO and EGNO architectures, outlining their most peculiar and defining characteristics in view of the experimental comparison that will follow next.

3.1.1 SEGNO

SEGNO's Framework

SEGNO integrates Neural ODEs with Equiv-GNNs to model the latent continuous trajectory of a system. Given initial system states (position and velocity), SEGNO calculates the position at any future time by integrating the velocity, which is itself determined by integrating the acceleration. The acceleration is parameterized by an Equiv-GNN that takes the system's trajectory and node features as input.

To approximate the continuous trajectory, SEGNO utilizes a numerical integrator, such as the Euler integrator, which divides the time interval into smaller steps and iteratively updates the position and velocity. This allows SEGNO to reuse existing Equiv-GNNs as building blocks.

Equivariant Graph Neural Networks (Equiv-GNNs) have emerged as essential tools for simulating the multi-object physical system, i.e., N-body systems. In particular, given the input state, they learn to predict the output state after a specific timestep. To achieve these, Equiv-GNNs model the whole system as a geometric graph, which treats physical objects as nodes, and physical relations as edges, and encode the symmetry into a message-passing network to ensure their

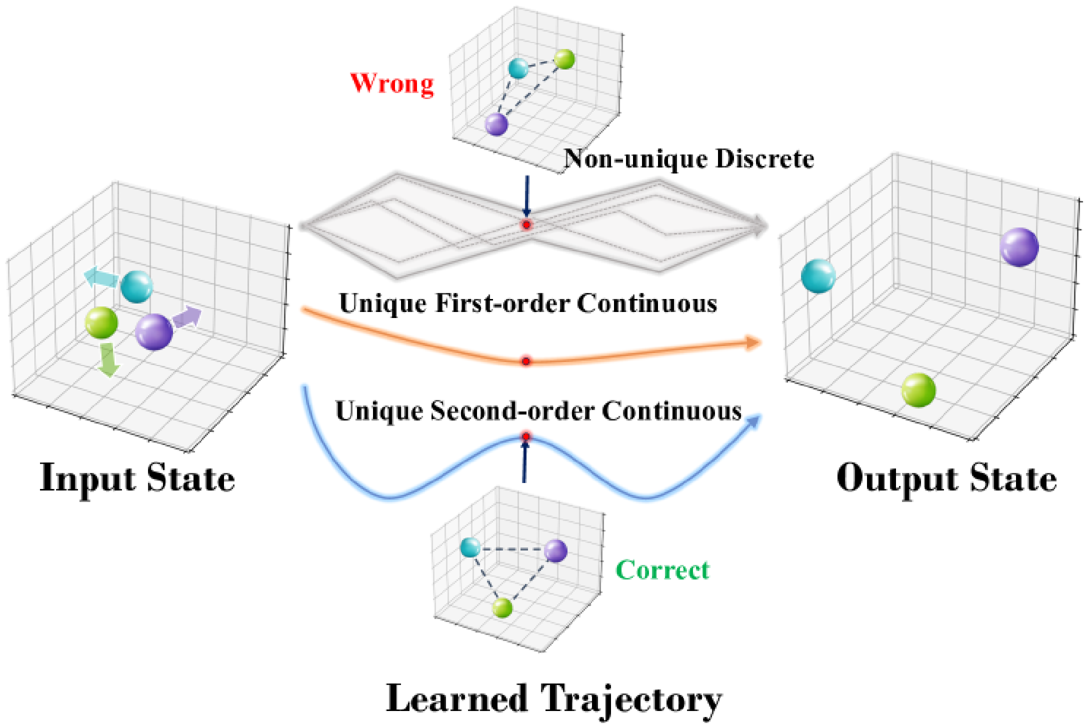


Figure 3.1: Learned trajectories of models with different inductive bias. All models can map input to output. However, discrete and first-order continuous models fail to learn the true intermediate states due to the lack of considering continuity and second-order laws. For the image refer to [8].

outputs are equivariant with respect to any translation/orientation/reflection of the inputs. This property makes them well-suited for learning the unknown dynamics of complex physical systems that cannot be described analytically.

Even though Equiv-GNNs have partially addressed the problem of learning solutions in the vast parameter space of GNNs to model such interacting systems, they had yet to incorporate sufficient physical inductive bias to model the physical dynamics. In particular, two essential inductive biases have not been well investigated in this field.

First, existing models are composed of a sequence of discrete state transformation layers, which learn direct mapping between adjacent states with discrete trajectories. They have been referred to as discrete models. They are inconsistent with the continuous nature of system trajectories and fail to learn correct intermediate states.

Second, most models only account for first-order information. Many physical dynamical systems, such as Newton's equations of motion, are governed by second-order laws. Therefore, these methods learn incomplete representations of the system's state and fail to capture the underlying dynamics of the physical systems. Figure 3.1 illustrates the comparison of learned trajectories of models with different types of inductive bias.

In SEGNO, it has been taken a deep insight into the continuity and second-order inductive bias in Equiv-GNNs and proposed a framework named Second-order Equivariant Graph Neural Ordinary Differential Equation (SEGNO). Differently from previous models that use Equiv-GNNs to fit discrete kinematic states, SEGNO introduces Neural Ordinary Differential Equations (Neural ODE) to approximate a continuous trajectory between two observed states. Furthermore, to better estimate the underlying dynamics, SEGNO is built upon second-order motion equations to update the position and velocity of the physical systems. Theoretically, it has been proven the uniqueness of the learned latent trajectory of SEGNO and further provided an upper bound on the discrepancy between the learned and the actual latent trajectory.

Meanwhile, SEGNO can be proven to maintain equivariance properties identical to the backbone Equiv-GNNs. This property offers the flexibility to adapt various backbones in SEGNO to suit different downstream tasks in a plug-and-play manner.

Advantages Over Past Models

SEGNO offers several advantages over previous models in this domain:

Unique and Bounded Trajectories: SEGNO learns a unique trajectory between observed states, unlike discrete models that can have multiple possible trajectories. Additionally, the discrepancy between the learned trajectory and the true trajectory is bounded, ensuring accurate and stable predictions.

Preservation of Equivariance: SEGNO maintains the equivariance properties of the underlying Equiv-GNN, ensuring that its outputs are consistent with rotations, translations, and reflections of the inputs. This allows for flexible adaptation of various Equiv-GNN backbones for different tasks.

Improved Generalization Ability: By incorporating continuity and second-order information, SEGNO captures the underlying dynamics of physical systems more effectively, leading to improved generalization performance compared to state-of-the-art baselines.

Pseudocode

The algorithm represented in 1 shows the main steps that compose its architecture. In the pseudocode, f_θ represents an Equiv-GNN with parameters θ which computes the message passing operations. G1 and G2 are the increment functions that approximate the increment of a continuous integral given the initial value of the integrand and the integration width Δt. For instance, with the increment functions $G1(x, y) = G2(x, y) = x \times y$, the numerical integrators become the Euler integrators. It is worth noting that the f_θ weights are shared among all the iteration steps of the forward step, as in the neural ODE standard procedure.

Algorithm 1 SEGNO Algorithm represented at high level

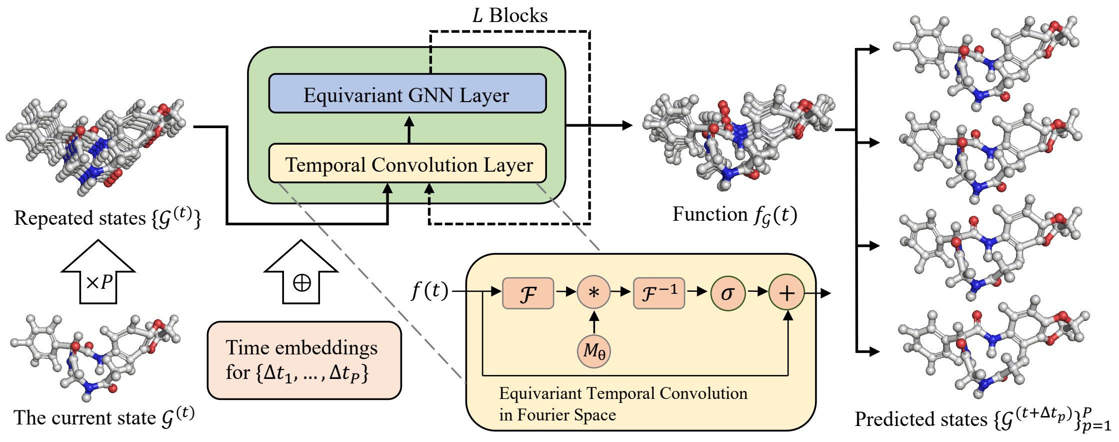
```
function SEGNO(graph, initial_positions, initial_velocities, $\Delta t$,
num_iterations)
    positions ← initial_positions
    velocities ← initial_velocities
    for i = 1 to num_iterations do
        accelerations ← $f_\theta$(graph, positions, velocities)
        velocities ← velocities + G1(accelerations, $\Delta t$)
        positions ← positions + G2(velocities, $\Delta t$)
    end for
    return positions, velocities
end function
```

3.1.2 EGNO

The EGNO (Equivariant Graph Neural Operator) model is a novel approach to modeling the 3D dynamics of relational systems, addressing the limitations of previous Equivariant Graph Neural Networks (EGNNs) in accurately capturing temporal correlations along trajectories. EGNO moves beyond next-step predictions by directly modeling dynamics as trajectories, predicting dynamics as a temporal function. The model is inspired by the Fourier Neural Operator (FNO), inheriting its desirable discretization convergence guarantees.

EGNO learns the temporal evolution of 3D dynamics while maintaining SE(3)-equivariance (3 dimensional Special Euclidean group), crucial for generalization in Euclidean space. Unlike conventional EGNNs that focus on next-step predictions, EGNO directly models the entire trajectory dynamics, capturing temporal correlations more effectively. Moreover, EGNO incorporates equivariant temporal convolutions parameterized in the Fourier space to ensure SE(3)-equivariance while learning temporal correlations.

The temporal convolutional layer is compatible with various EGNN layers, making EGNO adaptable to diverse physical dynamics scenarios. It offers efficient parallel decoding of future states within a time window and is not restricted to a fixed temporal discretization, allowing for dynamics inference at any timestep size.



L Blocks
Equivariant GNN Layer
Temporal Convolution Layer
Repeated states $\{\mathcal{G}^{(t)}\}$
$\times P$
$\oplus$
The current state $\mathcal{G}^{(t)}$
Time embeddings for $\{\Delta t_1, \dots, \Delta t_P\}$
$f(t)$
$\mathcal{F}$
$*$
$\mathcal{F}^{-1}$
σ
$+$
M_θ
Equivariant Temporal Convolution in Fourier Space
Function $f_\mathcal{G}(t)$
Predicted states $\{\mathcal{G}^{(t+\Delta t_p)}\}_{p=1}^{P}$

Figure 3.2: EGNO blocks (green) can be built with any EGNN layers (blue) and the equivariant temporal convolution layers (yellow). Consider discretizing the time window ΔT into P points $\{\ \Delta t_1, \dots, \Delta t_P\}$. Given a current state $G(t)$, we will first repeat its features by P times, concatenate the repeated features with time embeddings, and feed them into L EGNO blocks. Within each block, the temporal layers operate on temporal and channel dimensions while the EGNN layers operate on node and channel dimensions. Finally, EGNO can predict future dynamics as a function $f_G(t)$ and decode a trajectory of states $\{G^{(t+\Delta t_p)}\}_{p=1}^{P}$ in parallel. For the image refer to [9].

EGNO's Framework

EGNO is built by stacking equivariant temporal convolution layers with equivariant networks. Given the current state $G(t)$, EGNO first replicates its features and concatenates them with time embeddings. These features are then fed into L EGNO blocks as shown in Figure 3.2, each composed of:

- Temporal Convolution Layers: Operating on temporal and channel dimensions to capture temporal correlations.
- EGNN Layers: Operating on node and channel dimensions to model spatial interactions within each graph.

- The output is a temporal function that predicts future dynamics, allowing parallel decoding of a trajectory of states.

Advantages over Past Models

Thanks to EGNO's features, like those presented earlier, the model provides different advantages over past models developed in this kind of scenarios.

- Explicit Trajectory Modeling: EGNO explicitly models the trajectory dynamics as a temporal function, capturing temporal correlations more effectively than next-step prediction models like traditional EGNNs.
- Parallel Decoding and Timestep Flexibility: EGNO's operator formulation allows parallel decoding of future states and is not limited to a fixed discretization, enabling efficient inference at varying timestep sizes.
- Enhanced Performance: EGNO demonstrates significantly superior performance compared to existing methods, as shown in experiments on particle simulations, motion capture, and molecular dynamics.
- Data Efficiency: Compared to EGNNs, EGNO proves to be considerably more data-efficient in simulations.

Pseudocode

The algorithm 2, shows, at high level, the prodecure adopted by EGNO. The graph features represent one single snapshot of the whole trajectory, which is given as input to the model. The input features are repeated ΔT times and then concatenated with the time embeddings. Then they are forwarded to a specific number of blocks each applying a temporal convolution layer, followed by an Equiv-GNN layer which applies the message passing operation. The temporal convolution layer applies the FFT (Fast Fourier Transform) to the input features. Then they are given as input to a temporal integration layer implemented directly in the frequency domain. Afterwards the IFFT (Inverse Fast Fourier Transform) is applied to bring the data again in the time domain. In the end, the model is trained to produce ΔT snapshots in parallel.

3.2 Experimental setting

The following section will qualitatively present the experimental setting, outlining the techniques under analysis and their primary purposes.

Algorithm 2 EGNO Algorithm represented at high level

```
function EGNO(graph, features, time_embeddings)
    input_features ← REPEAT_FEATURES(features, len(time_embeddings))
    input_features ← CONCATENATE(input_features, time_embeddings)
    for i = 1 to num_blocks do
        input_features ← TEMPORAL_CONVOLUTION(input_features)
        input_features ← EQUIVARIANT_GNN(graph, input_features)
    end for
    future_states ← DECODE(input_features)
    return future_states
end function
```

3.2.1 Rollout

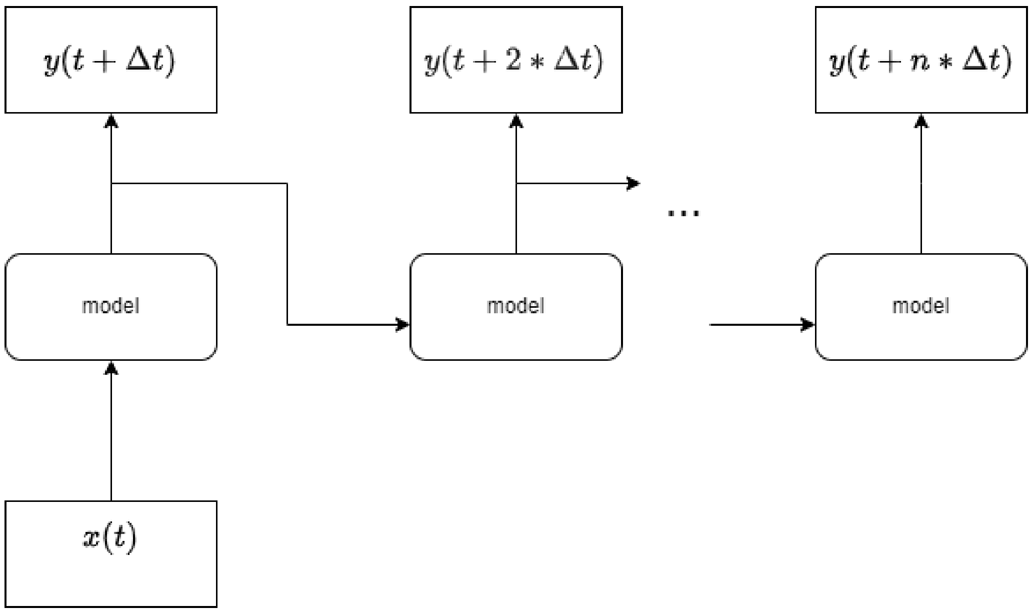


Figure 3.3: Schematic representing the rollout technique

The rollout technique [17] consists of the process of using a model's predictions as inputs for the next steps to generate a sequence of future predictions as explained in Figure 3.3. It is a way to evaluate how well a model performs over extended sequences or time horizons by simulating its behavior over multiple prediction steps, often revealing how errors or instabilities might accumulate.

When a model is evaluated using a rollout technique, several characteristics are typically assessed to understand its performance over extended sequences or time horizons. The primary characteristics that are tested when applying rollout are:

- Stability: Rollouts test the model's ability to maintain stable predictions over long sequences. Instability might cause errors to accumulate, leading to diverging or implausible outputs. This is especially relevant in dynamical systems, where small prediction errors can cascade over time.

- Generalization: Rollouts reveal how well the model can handle inputs outside its training distribution. The technique evaluates the model's robustness to scenarios it hasn't seen before, particularly if the rollout introduces data points progressively further from the initial training data distribution.

- Error Accumulation: Rollout techniques test whether prediction errors grow or dampen as the rollout progresses. Ideally, a model should be resilient to error compounding, where inaccuracies from one step feed into the next, amplifying the deviation from the true sequence.
- Long-term Consistency: This evaluates if the model can produce predictions that remain consistent with underlying system constraints or invariants over extended sequences. For physical systems, this might mean preserving energy, momentum, or other conserved quantities, ensuring that long-term predictions are physically plausible.
- Accuracy in Sequential Prediction: Rollouts directly assess the model's accuracy in sequential steps, helping to measure how well the model captures temporal dependencies and dynamics over time. This characteristic is crucial for models of time series, control systems, and other temporally evolving phenomena.
- Adaptability: Some rollout techniques evaluate how a model adapts or recalibrates when subjected to unexpected changes in the input sequence. For instance, a model might need to react to new states or altered dynamics without drastically losing accuracy.
- Computational Efficiency: When models are evaluated with rollouts over extended time horizons, computational efficiency (in both memory and speed) becomes important. Rollouts can highlight models that require excessive computational resources to maintain performance over long sequences, which is particularly relevant in real-time or resource-constrained applications.

The rollout technique is going to applied in all the presented experiments, considering some variations in order to further analyze the models' capabilities. For the considered experiments, trajectories with 100 steps are considered. In practice though, each macro step comprise of 10 "inner"steps ($\Delta T = 10$), meaning that the first step is given as inputs and the 10th step is the output (EGNO actually outputs the whole 10 step trajectory of which the last step is selected as the final prediction) at each rollout iteration. In the end the total lenght of the considered trajectory is 100 (10 prediction steps each with $\Delta T = 10$ time window).

3.2.2 Multiple inputs

In this experiment, the goal was to get an insight into the capabilities of the models to maintain stability in their predictions when receiving more inputs than just the initial one. This variant is applied differently to the two models based on their specific way of handling input timesteps.

SEGNO

In this case, when using multiple inputs, the prediction from last step is aggregated in some way (usually just summed) with the observation of the current step. This process is applied until the specified number of inputs is reached. After that point the rollout will be completed aas per usual: using the prediction from the last step as input for the next step. For this experiment, the way that multiple inputs are combined took inspiration from [18]: the prediction from last step is summed with the ground truth of the current step and then given as the new input of the model: $\hat{G}^{(t+i\Delta T)} = F_\theta(G^{(t+(i-1)\Delta T)} + \hat{G}^{(t+(i-1)\Delta T)})$ where $\hat{G}^{(t+i\Delta T)}$ is the prediction at the i-th step. This technique is applied both at training and validation/test time.

In Figure 3.4 it is represented the way multiple input works in this case. The goal of this strategy is to take advantage of the residual behaviour in SEGNO, by adding information from last step prediction to the current processing state.

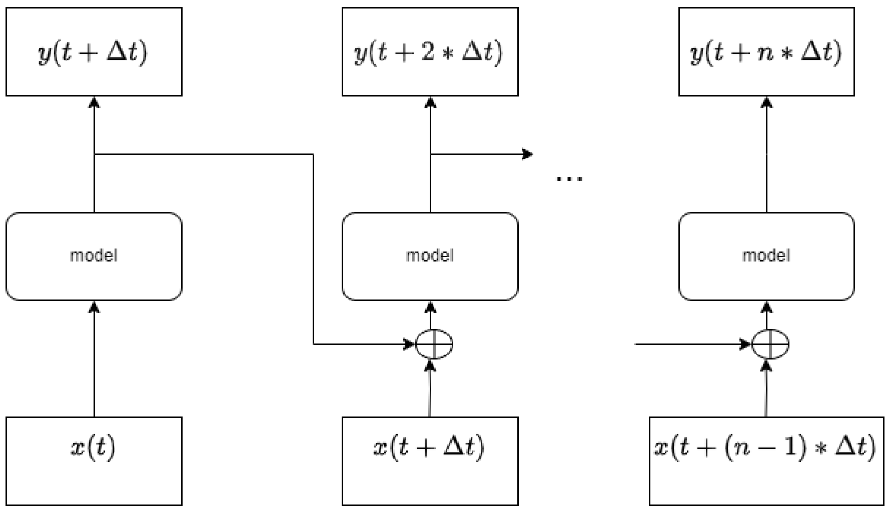


Figure 3.4: Schematic representing the multiple inputs procedure applied to SEGNO

EGNO

For this architecture the procedure applied in order to use multiple inputs is different than for the previous one. Indeed, EGNO directly models the entire trajectory dynamics, which in the standard usage of the model means that: the time window ΔT is discretized into P points; given a current state $G(t)$, first repeat its features by P times, concatenate the repeated features with time embeddings, and feed them into L EGNO blocks.

When adopting multiple inputs, the time window ΔT, is equally (whenever possible) divided between the input values (e.g. if two inputs are used, the first $\frac{\Delta T}{2}$ values ar equal to the first input and the last $\frac{\Delta T}{2}$ values are equal to the second input), and then concatenated with time embeddings accordingly. The first input to EGNO during the rollout procedure with multiple inputs is obtained as explained earlier. The time embedding of the given input timesteps are also added in this case (when not using multiple inputs, the initial input is repeated ΔT times so there is no need for the time embedding of the input) to help to model in tracking these new information to improve predictions accuracy. Again, this methodology is applied both at training and validation/test time.

3.2.3 Variable Δt

When using multiple inputs, the time window Δt (different from ΔT which is the entire time window of the single prediction step) is kept constant, meaning that the time distance between two consecutive inputs is always the same (Note: in the case of SEGNO with multiple inputs $\Delta t = \Delta T$, while in EGNO with multiple inputs Δt is the time distance between steps given as inputs inside the $\Delta T = 10$ window). Everything else said in the multiple inputs section remains true, but in addition to that, in this experiment, variable Δt are also considered for both models.

Varying the time interval between inputs can reveal how the system's state changes over different timescales, highlighting the dynamics of slow versus fast-changing components. This can lead to insights into how sensitive the system is to changes over time, potentially identifying which factors or events cause significant shifts.

Moreover, introducing variability in time steps allows the model to handle situations where inputs come at irregular intervals, which is often the case in real-world scenarios (e.g., irregular sampling in sensor data). This can make the model more robust and generalizable, as it learns to adapt to a variety of temporal patterns rather than only fixed intervals.

Variable Δt can be incorporated in the two models in a different way. Figure 3.5 shows a possible approach for EGNO, in which the input P snaphots are not constant repetitions of one single timestamps, but can be any combination of the desired input graphs. The model can understand the temporal meaning of each input snapshot thanks to the use of input time embeddings.

In contrast, SEGNO can manage variable time ranges between inputs by taking different numbers of integration steps to build each prediction of the subsequent snapshot. At the same time, the number of steps, in the end, will be discrete.

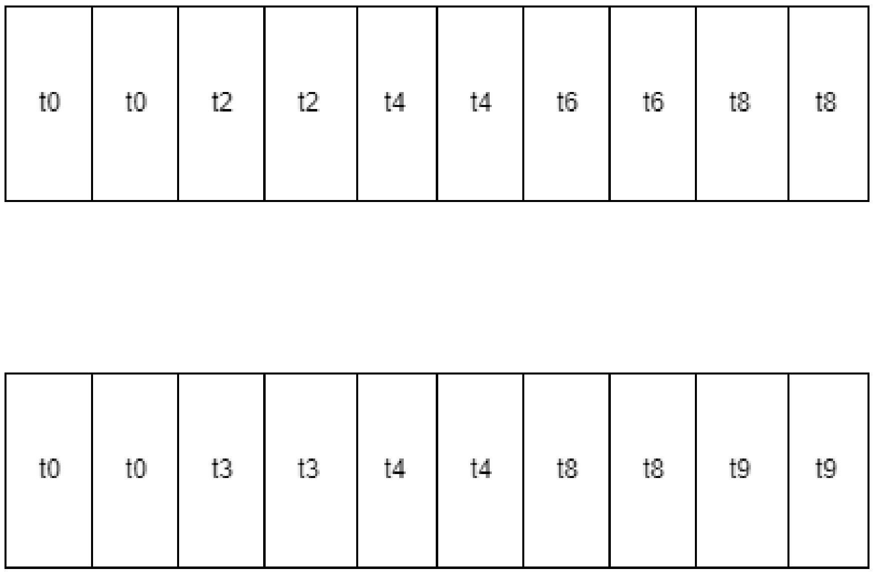


Figure 3.5: Schematic representing an example of input given to EGNO when using 5 different inputs (figure on top) and when using multiple inputs with variable Δt (figure on the bottom)

SEGNO

As explained before, SEGNO considers just one step as input for each model call, and return just the final prediction. To consider non uniform sampling, in this experiment, the time window between input and output prediction is not fixed but can change along the trajectory (the total length of the trajectory though stays the same, as the number of prediction steps).

EGNO

In the case of EGNO, the model usually takes as input one single step that then gets duplicated ΔT times. When using multiple inputs, different steps are given as input to the model. These steps are equispaced meaning the time distance between two next step is always the same (we call this Δt) as showed in the upper part of Figure 3.5 . For this experiment instead, Δt is allowed to change thus having as an example input what is showed in the lower part of Figure 3.5). The time embedding of the input is calculated accordingly to the selected timesteps as done for the "basic" multiple inputs experiment.

3.2.4 N-body System Simulations

The dataset adopted is the 3D N-body simulation dataset [3] which comprises multiple trajectories depicting the dynamical system formed by N charged particles, with movements driven by Coulomb force. The experimental setup considers different number of particles N, while the time window remains the same as $\Delta T = 10$, and 3000/2000/2000 trajectories for training/validation/testing. For EGNO uniform discretization is used, with P = 10 points in each time window (meaning every timestep inside the ΔT time window is predicted by the model).

In practice, we want to model the dynamics of multi-body systems, i.e., a sequence of geometric graphs $G^{(t)}$ indexed by time t. It can also be viewed as a function of 3D states over time $f_G : t \to G^{(t)}$.

Chapter 4

Results

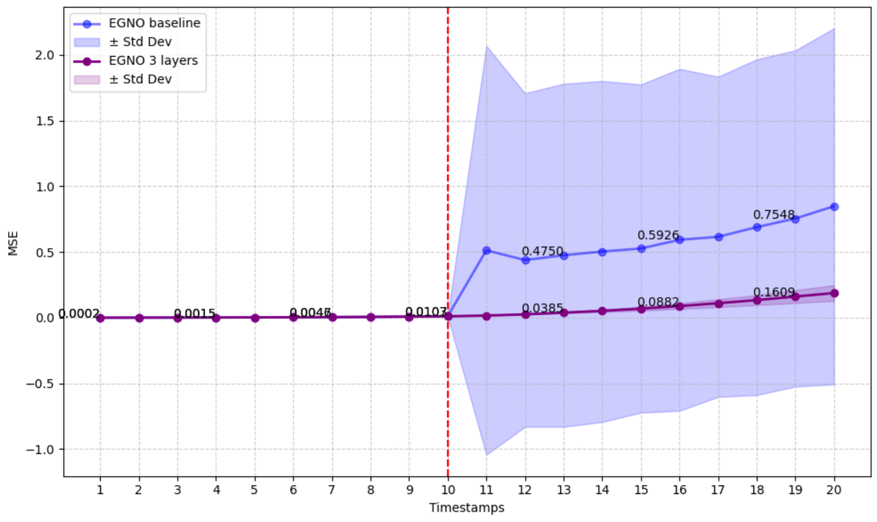
In this chapter, the experimental setting will be further and more accurately described along with the most relevant results.

In this section the results of the aforementioned experiments are goingto be reported and analyzed alongside graphs that display the metrics of interest in studying the properties of the two models and in general of the two approaches that they respectively represent.

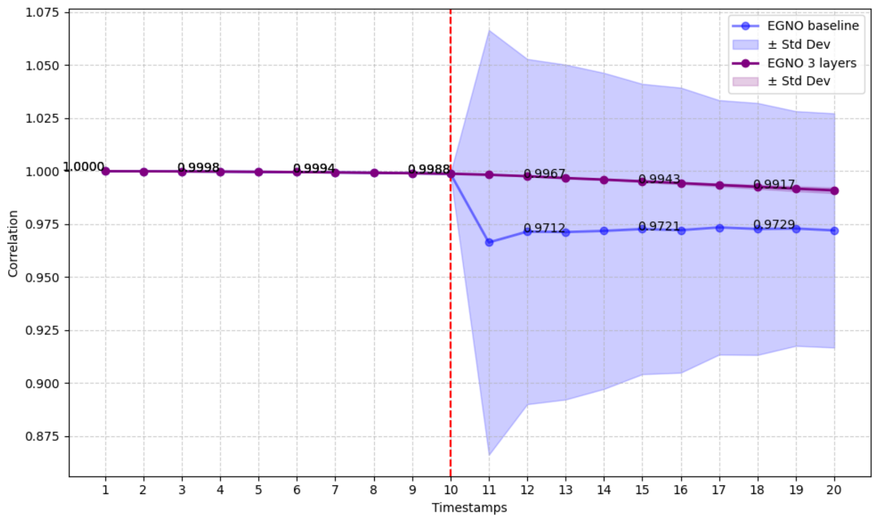
Before conducting a detailed analysis of the results, it's important to note that EGNO demonstrated that after a few iterations of the rollout procedure, the generated trajectories began to diverge, resulting in little to no meaningful outcomes. Therefore, in the following graphs, the lengths of the trajectories produced by EGNO will be significantly shorter than those generated by SEGNO.

4.1 Best EGNO model

During the analysis of the models, it was found that, when using 3 as number of layers (meaning 3 blocks of temporal convolution and EGNN layer, before the considered number of layers was 4), both the MSE and the correlation look more stable, also looking at the standard deviation, as showed in Figure 4.1. The reason for this might be found in the fact that, probably, 3 layers are enough to learn the trajectories dynamics, but when applying rollout, using more layers can lead to some sort of overfitting and thus to less stable predictions. For this reason, from now on the EGNO configuration considered as baseline will be the one with 3 layers differently to the original paper's implementation.



(a) Trajectories MSE for 3 and 4(baseline layers EGNO



(b) Trajectories Correlation for 3 and 4(baseline) layers EGNO

Figure 4.1: Trajectories Mean MSE (top in log scal) and Correlation (bottom) $\pm$ standard deviation (represented by the shaded region) calculated along 10 runs, after applying Rollout to 3 layers EGNO and standard EGNO (4 layers). The thick points represent the models' calculated points while the red dotted lines represent the ΔT intervals, meaning the final prediction after each rollout iteration.

4.2 Standard Rollout

The analysis starts with applying the rollout procedure at test time for the standard models, as presented before, with no modifications to the training procedure. The metrics are the trajectories, per timestamp average MSE (Mean Squared Error) and average correlation with their respective standard deviation, where the average and the standard deviation are computed for each timestep among multiple runs of the same model configuration (10 runs are used for all the experiments). For this experiment, $\Delta T = 10$ is considered, while the number of rollout iterations is 10 and stays the same for all experiments. The results of this experiment are shown in Figure 4.2.

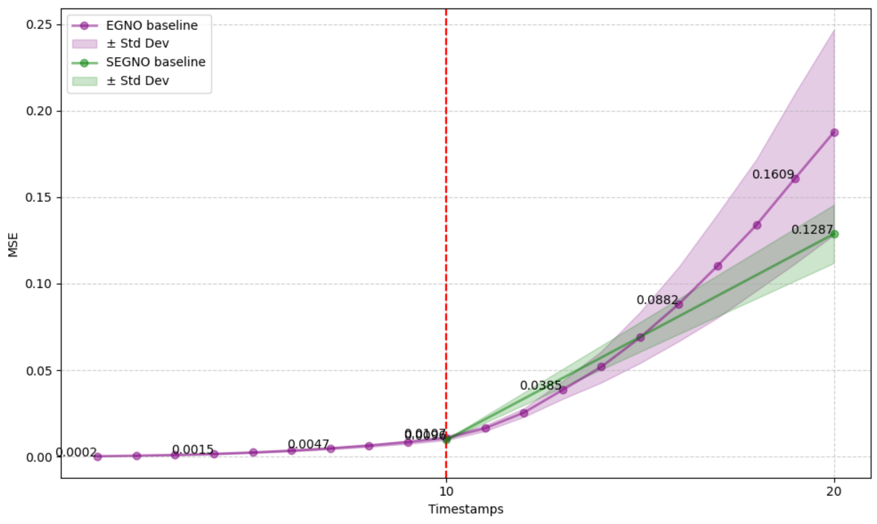
The investigated timesteps, involve only the initial part of the trajectory. That's because outside of this region, EGNO struggles to maintain stability and very quickly diverge. SEGNO, on the other hand, remains stable across the whole trajectory. Of course, it should be kept in mind that EGNO is predicting the whole "inner" trajectory of length ΔT (for now equal to 10), while SEGNO predicts only the last step (e.g. in this case: $t = 10,20,30,\dots$). Moreover, after the first rollout iteration, it can be noticed that the standard deviation in EGNO starts increasing significantly, though a good correlation is maintained until the third iteration when the results cannot be considered anymore.

Regarding MSE, both models start to struggle after the second iteration of the rollout (so after $t = 20$ in this case). For EGNO though, after the second rollout iteration, the results start to be meaningless, with also a clear increase in the standard deviation after the first rollout iteration. SEGNO, on the other hand, still maintain some sort of relevance even though the error starts increasing more and more. This behavior should be expected in this kind of experiments since, at every further iteration, the errors add up, leading to a misalignment with the ground truth.

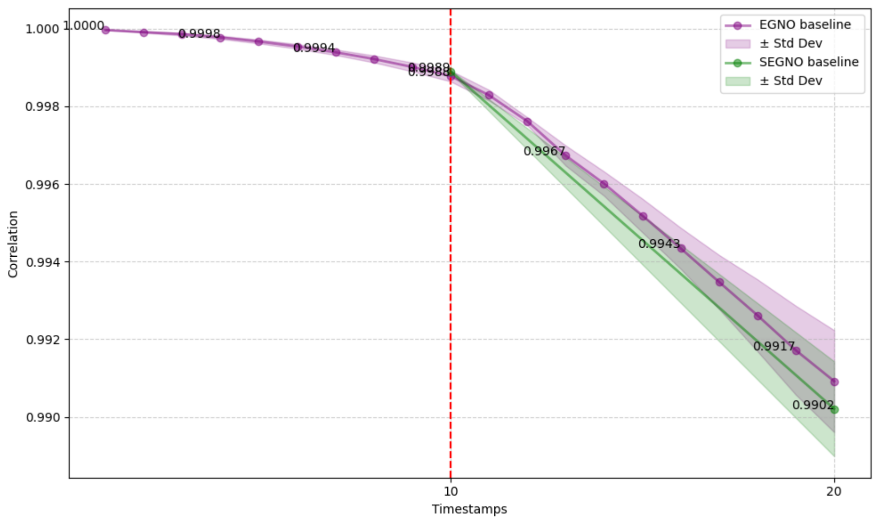
4.3 Multiple Inputs Rollout

Now, we are also going to consider models trained and tested using multiple inputs, as explained in last chapter, to check if this technique can be useful in helping the model maintaining predictions' stability for longer. For this section, it should be kept in mind that the way multiple inputs are introduced differs in the two architectures, hence, this procedure will influence the results differently for the two models.

Starting from considering 2 inputs (represented as MI(2) in the Figure 4.3 and the Table 4.1), it can be seen that, considering the overall trajectory, SEGNO



(a) Trajectories MSE for standard EGNO and SEGNO



(b) Trajectories Correlation for standard SEGNO and EGNO

Figure 4.2: Trajectories Mean MSE (top in log scale) and Correlation (bottom) ± standard deviation (represented by the shaded region) calculated along 10 runs, after applying Rollout to standard EGNO and SEGNO. The thick points represent the models' calculated points while the red dotted lines represent the ΔT intervals, meaning the final prediction after each rollout iteration.

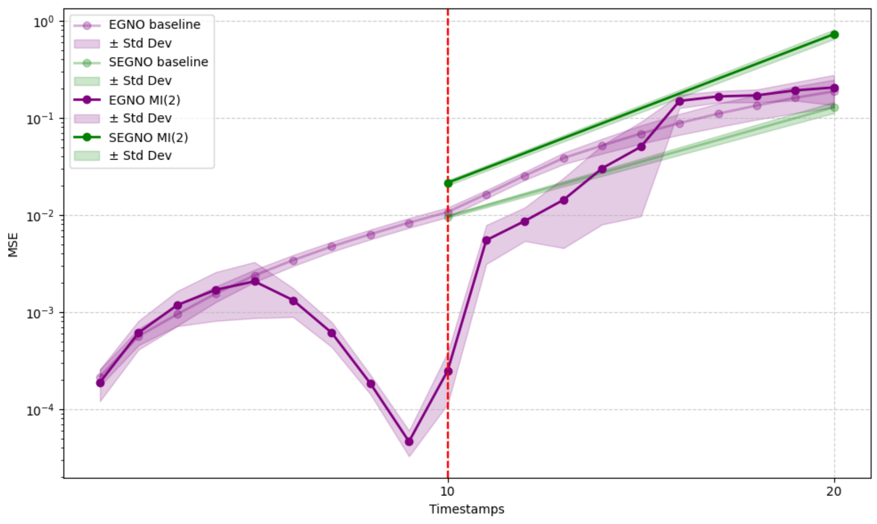
seems to achieve worse performance of both EGNO MI(2) and standard SEGNO. The reason for this might be found in the fact that, when applying multiple inputs on SEGNO, the observation at the current is aggregated (in practice, summed) with the prediction from last step. This technique, borrowed from other works in the field, as referenced previously, might indeed not be a good technique in this case and/or the aggregation strategy adopted might not be the optimal one.

In EGNO, the use of 2 inputs leads to an improved capability in predicting the earlier states of the trajectory, while the overall average MSE shows to be just a little bit higher than the standard case as displayed on the Table 4.1. Indeed, because of the way the inputs are used by the model, in which they are replicated as many times as necessary in the first ΔT interval and used jointly with input time embedding, it seems reasonable that EGNO can better predict the earlier steps in the trajectory (with a clear valley near the additional values provided as inputs, visible in Figure 4.3 (a). At the same time though, this approach doesn't look good enough to help the model in keeping more stable predictions for more timesteps than usual, leading to overall similar performance with respect to standard EGNO, and better performance than MI(2) SEGNO at least up until the point to which EGNO remains relevant (until 20 timestep so before the third rollout iteration).

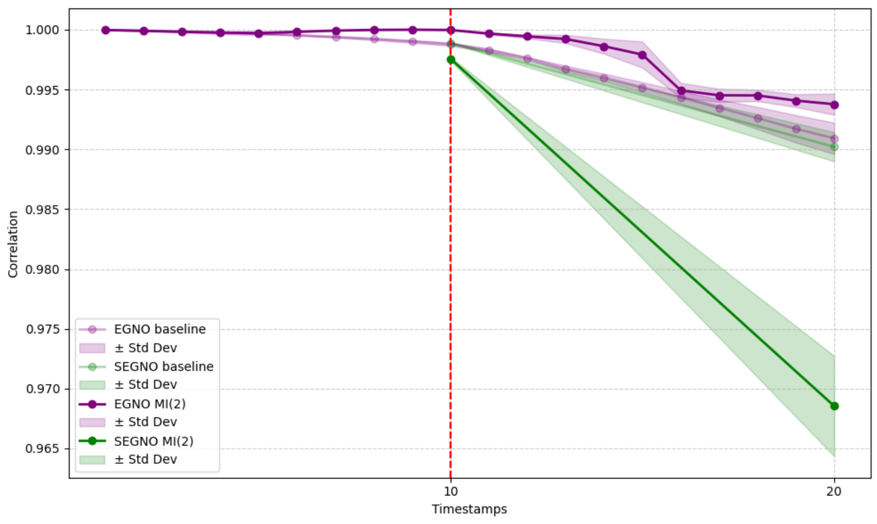
Looking at MI(3) SEGNO plots in Figure 4.4, it can be found the same pattern noticed before, except from the fact that in this case, the correlation seems to increase along the trajectory with respect to MI(2) SEGNO. Overall, this training approach for SEGNO didn't result in better performance or stability of the predictions along the trajectory. Moreover, it might be possible that different strategies or modifications of the used one, can bring a contribution to the goal of improving the stability of this kind of models.

What has just been said for MI(3) SEGNO is true also for MI(3) EGNO, especially with regard to the noticeable recurrent pattern present also in MI(2) EGNO, where, in this case, two clear valleys can be seen at the beginning of the trajectory, again near the additional points provided. Also the standard deviation of the MSE, in this initial part of the trajectory seems thinner than both EGNO and MI(2) EGNO. Moreover, also the correlation looks, even if slightly, more stable along the trajectory, which might prove the contribution of this training strategy. Overall, even if EGNO's analysis is limited to an initial portion of the trajectory, it seems like a more stable model when applying this kind of training strategy like MI(2) and MI(3).

In Table 4.1 is shown the average MSE used as an aggregated metric to get an overall glimpse at the, up until now, analysed models. The first three sections of

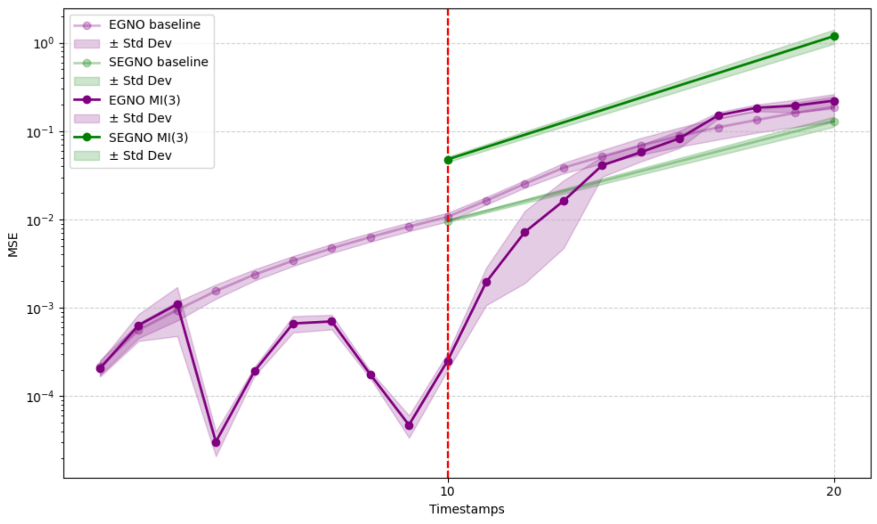


(a) Trajectories MSE for EGNO and SEGNO

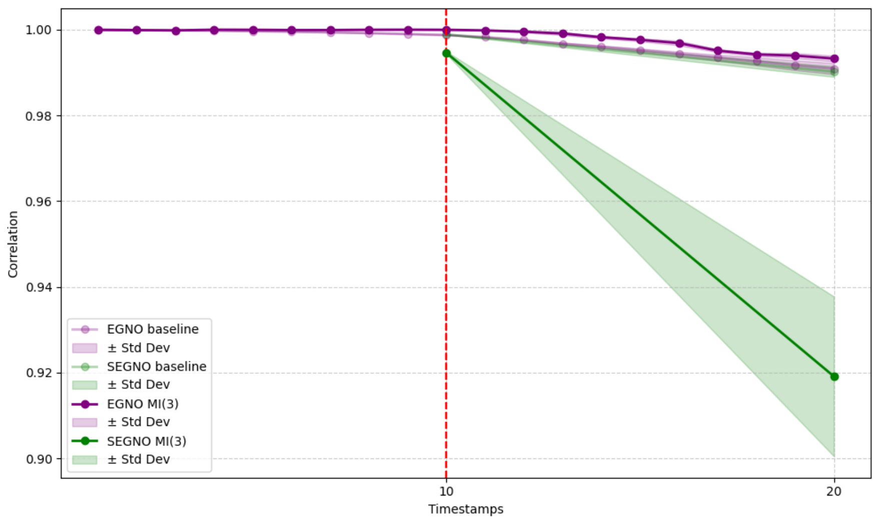


(b) Trajectories Correlation for SEGNO and EGNO

Figure 4.3: Trajectories Mean MSE (top in log scale) and Correlation (bottom) $\pm$ standard deviation (represented by the shaded region) calculated along 10 runs, after applying Rollout to standard and to 2 inputs EGNO and SEGNO. The thick points represent the models' calculated points while the red dotted lines represent the ΔT intervals, meaning the final prediction after each rollout iteration.



(a) Trajectories MSE for EGNO and SEGNO



(b) Trajectories Correlation for SEGNO and EGNO

Figure 4.4: Trajectories Mean MSE (top in log scale) and Correlation (bottom) $\pm$ standard deviation (represented by the shaded region) calculated along 10 runs, after applying Rollout to standard and to 3 inputs EGNO and SEGNO. The thick points represent the models' calculated points while the red dotted lines represent the ΔT intervals, meaning the final prediction after each rollout iteration.

the table consider the timestamp t after which the correlation falls under a certain thershold. In the last section, instead, the timestamp considered is the same for all the models. This is done because better models that might maintain a high value of correlation for longer, will consider more values in the calculation of the average, thus leading to a potential higher final loss. On the other hand, less stable models that show lower values of correlation more early on, will consider less values to compute the average and thus reaching lower average MSE than more stable models.

From this last section of the table it can be summarised what was said up until now: EGNO showed to be more stable and precise in generating the inital part of the trajectory with respect to SEGNO. On the other end though, SEGNO is able to maintain more stable and relevant result for a longer number of timesteps. The multiple inputs approached used in SEGNO showed to actually worsen the results in some cases and it surely does when considering the average MSE in the initial part of the trajectory. In EGNO, the multiple inputs help in having more stable predictions early on, but, overall they provide little to no improvement with respect to standard EGNO. So, even though the training approach didn't drastically change the results, it was more useful than in SEGNO.

With regard to the Table 4.1, EGNO performs better than SEGNO with respect to all metrics. In particular, for every metric except average MSE, EGNO MI(3) results as the best model.

model	**avg MSE**	**avg MAE**	**MSE first step**	**MAE first step**
EGNO	**0.0386**	0.0959	0.0164	0.0694
SEGNO	0.0639	0.1121	0.0094	0.0473
EGNO MI(2)	0.0418	0.1005	0.0055	0.0548
SEGNO MI(2)	0.3748	0.3781	0.0214	0.0937
EGNO MI(3)	0.039	**0.0955**	**0.002**	**0.0275**
SEGNO MI(3)	0.6191	0.4891	0.0479	0.152

Table 4.1: The table shows the average MSE, average MAE, MSE and MAE at first rollout iteration. The average is computed until EGNO stays relevant

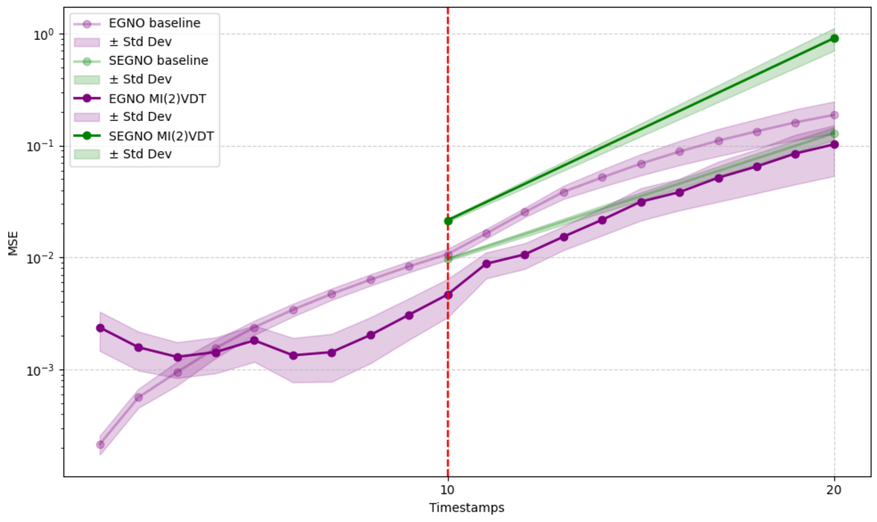
4.4 Variable ΔT

In this section, a variant of the models introduced up until this moment will be considered. In particular, this approach considers variable distance between the timesteps, while before we only considered a fixed and constant distance. Also in this case, the practical adaptation of using variable ΔT differs between the two models, as explained in the Experimental setting described in the last chapter. In practice we want to consider a task which admits non-uniform sampling, in order to test the capabilities of the two approaches presented. The training technique is again modified to better suit this variant of the original task.

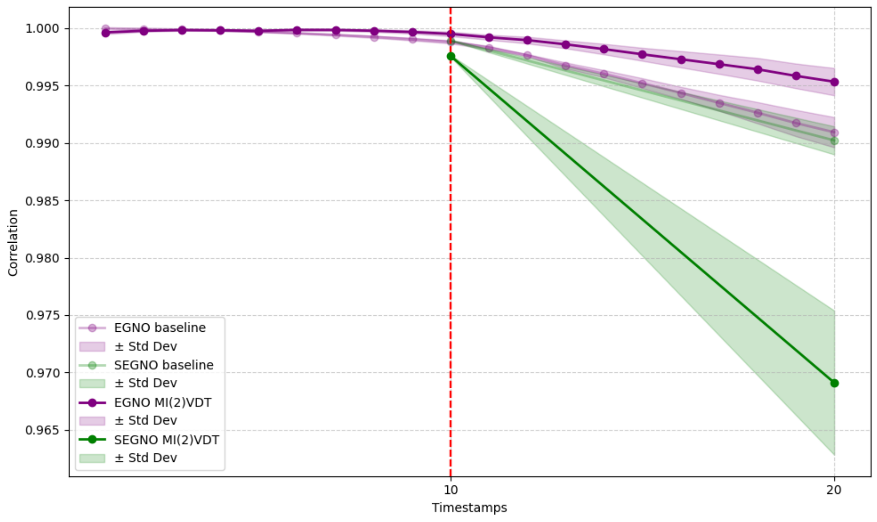
Moreover, in EGNO, the concept of variable ΔT is actually related to the distance between the different inputs provided at the beginning of the rollout more than the actual number of timesteps produced. This, because one of EGNO's features is to be discretization invariant thanks to its operator formulation, thus it could be more interesting to analyse this aspect.

Starting from EGNO, it can be seen from Figure 4.5, 4.6 and Table 4.2, that its performance, even though the standard deviation looks thicker, are actually slightly enanched in terms of MSE by this technique, especially in EGNO MI(2)VDT with respect to standard EGNO and the other variants. This might be caused by the fact that, when using MI technique alone, the inputs used are selected as equispaced timesteps in the first ΔT interval. Hence, this might add some overhead to the model learning without introducing enough helpful information. When applying VDT, on the other hand, the input timesteps used are, indeed, variable (in practice, randomly selected with different seeds for each of the 10 runs). This fact might therefore introduce a good enough variability in the information received by the model to actually help it generalize more in this task. In EGNO MI(3)VDT though, there is a slight decrease in MSE as showed by the last section in the bottom of table 4.2, maybe because if providing too many input values they might overcomplicate the learning process of the model. In SEGNO, on the other hand, the results are relatively aligned with the MI experiments with no VDT. Indeed, the MSE seems to increase again but, this time there is also an additional complication introduced, which forces the model to learn how to better predict non unifrom sampled trajectories. In view of this the results might not look that bad, especially considering that SEGNO MI(3)VDT performs slightly better than the respective model without VDT. Indeed, it might be useful to consider a larger number of inputs for this kind of task.

Taking a look at the Table 4.2, it seems clear that EGNO MI(2)VDT performs better than SEGNO with respect to all metrics except MAE at first step, enforcing the usefulness of tha training strategies applied in EGNO.

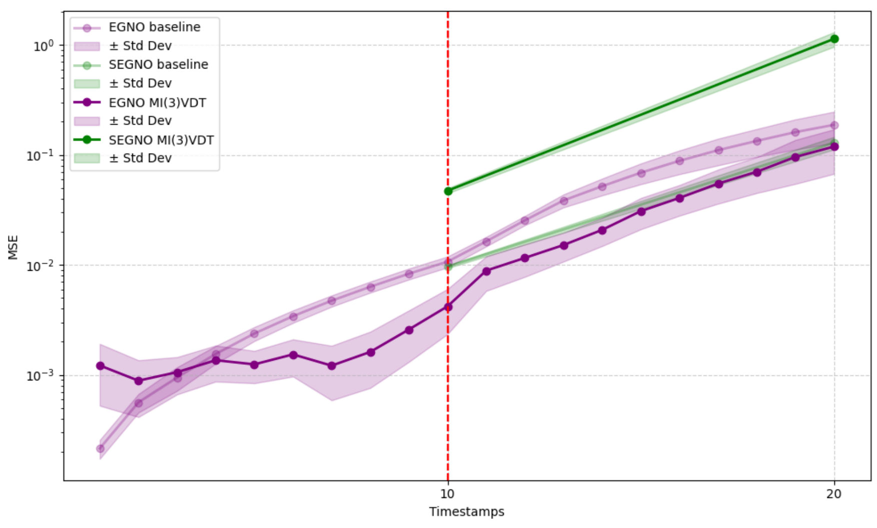


(a) Trajectories MSE for EGNO and SEGNO

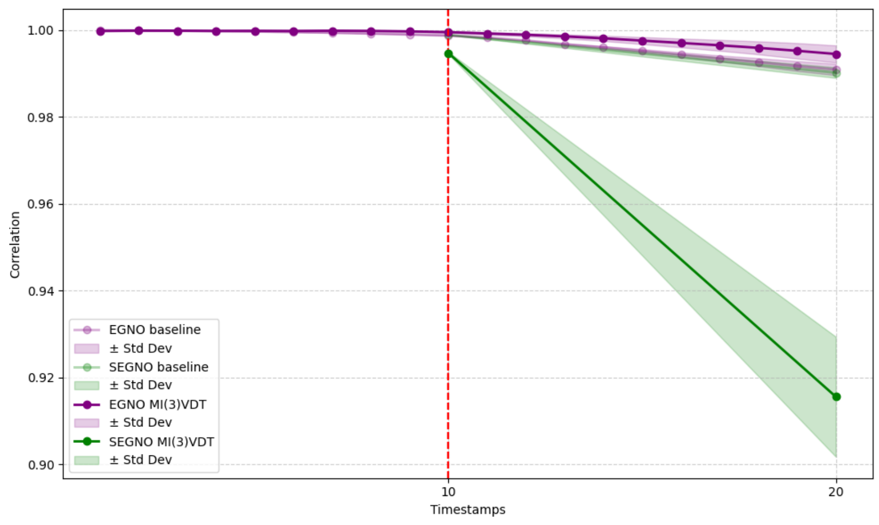


(b) Trajectories Correlation for SEGNO and EGNO

Figure 4.5: Trajectories Mean MSE (top in log scale) and Correlation (bottom) ± standard deviation (represented by the shaded region) calculated along 10 runs, after applying Rollout to standard and to 2 inputs and variable ΔT EGNO and SEGNO. The thick points represent the models' calculated points while the red dotted lines represent the ΔT intervals, meaning the final prediction after each rollout iteration.



(a) Trajectories MSE for EGNO and SEGNO



(b) Trajectories Correlation for SEGNO and EGNO

Figure 4.6: Trajectories Mean MSE (top in log scale) and Correlation (bottom) $\pm$ standard deviation (represented by the shaded region) calculated along 10 runs, after applying Rollout to standard and to 3 inputs and variable ΔT EGNO and SEGNO. The thick points represent the models' calculated points while the red dotted lines represent the ΔT intervals, meaning the final prediction after each rollout iteration.

model	**avg MSE**	**avg MAE**	**MSE first step**	**MAE first step**
EGNO	0.0386	0.0959	0.0164	0.0694
SEGNO	0.0639	0.1121	0.0094	**0.0473**
EGNO MI(2)VDT	**0.0183**	**0.0663**	**0.0088**	0.0591
SEGNO MI(2)VDT	0.4653	0.4141	0.0215	0.093
EGNO MI(3)VDT	0.0192	0.0672	0.0089	0.0598
SEGNO MI(3)VDT	0.5898	0.4779	0.0472	0.1486

Table 4.2: The table shows the average MSE considering the timestep after which the correlation falls under a certain threshold. The last section on the bottom of the table, considers the same timestamp t for all models

4.5 Different values of ΔT

Up until now we considered a $\Delta T = 10$, as interval between two rollout prediction (with the exception of SEGNO VDT in which ΔT is variable but the overall trajectory length remained unchanged). The number of rollouts iteration though, remains the same and equal to 10. For this experiment we consider the starting setup of the models presented in last section to compare the two approaches. That's because the MI(2)-MI(3) models show similar trends than the respective VDT variance, hence only the latter are showed for simplicity. The different interval size tested are: $\Delta T = 5$ and $\Delta T = 2$. Nonetheless, the pattern that will be shown for $\Delta T = 5$ are analogous to those for $\Delta T = 2$, thus, to avoid redundancy, the explicit analysis of the latter will be omitted going forward. One more thing to keep in mind is that the total lenght of the trajectory is influenced by ΔT. In practice it's given by: $tot\_lenght = \Delta T * rollout\_iterations$ where rollout_iteration is always equal to 10.

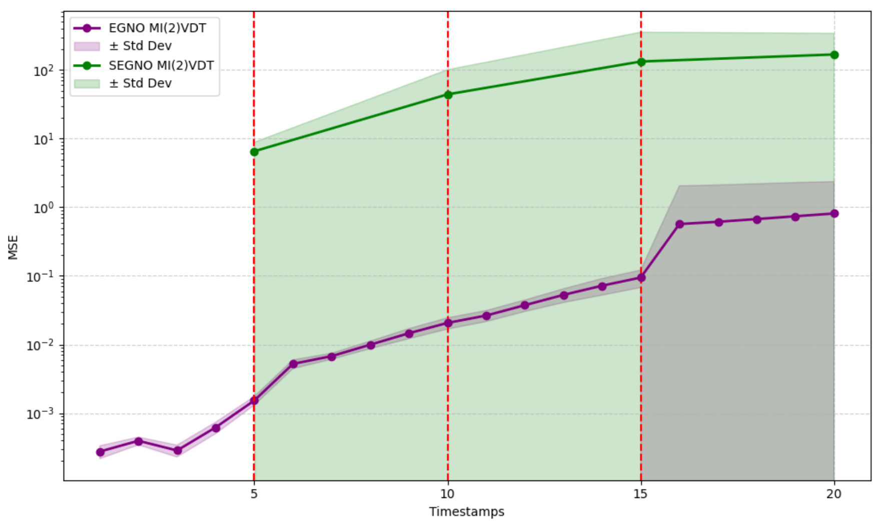
It is clear already from the graphs in Figures 4.7 and 4.8, that EGNO can now be stable for more rollout iterations, Indeed, it provides reliable results until timestep 20 (so for $\Delta T = 5$, it is stable until the 4th iteration), even though at around timestep 14 theere is a clear deterioration in MSE and correlation, visible also through the standard deviation. The reason for this can be seen in the fact that, since each prediction step now consists in predicting a lower number of timesteps, the model is able to be more accurate, thus decreasing the error propagated with rollout onto the next steps of the trajectory.

From Table 4.3, it can be noticed that, as in the case with $\Delta T = 10$ present previously, EGNO MI(2)/MI(3)VDT variants achieve better performances than the standard model. In this case, the improvement is even more clear than before, and it might be caused by the fact that, as said before, the model remains stable for longer, thus enhancing the performances that were already improved before when considering $\Delta T = 10$. However, with respect to MSE and MAE first step, standard SEGNO performs better than any EGNO variants.

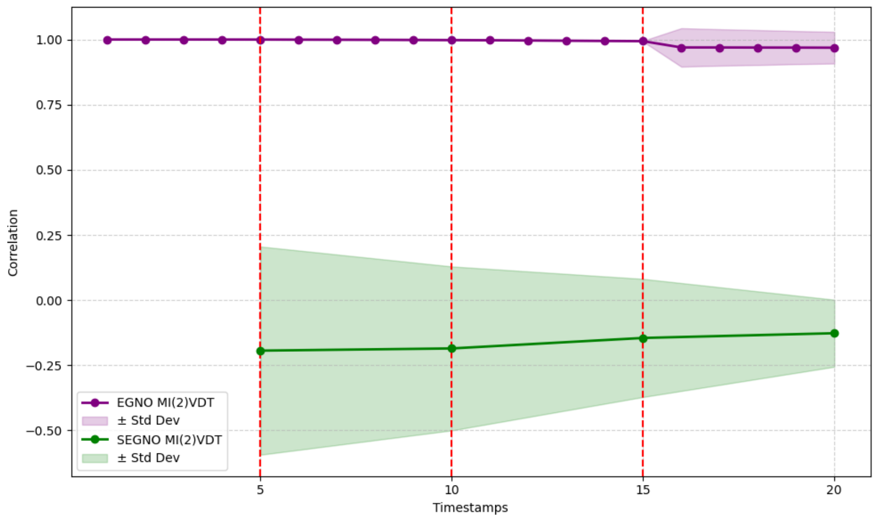
model	**avg MSE**	**avg MAE**	**MSE first step**	**MAE first step**
EGNO	0.3293	0.2897	0.0067	0.048
SEGNO	12.2259	2.2701	**0.0046**	**0.047**
EGNO MI(2)	0.1545	0.1236	0.0052	0.0489
SEGNO MI(2)	87.8297	5.7015	6.5335	1.9428
EGNO MI(3)	**0.053**	**0.1193**	0.0059	0.055
SEGNO MI(3)	134.689	6.846	5.5568	1.7623

Table 4.3: The table shows the average MSE with $\Delta T = 5$, considering the same timestamp t for all models

Talking about SEGNO, both from Figures 4.7, 4.8 and the Table 4.3, it looks like the model is very unstable since the beginning. This might be cause by the fact that it wasn't performed an extensive hyperparameter tuning for different values of ΔT other than ΔT. The different number of timesteps can thus introduce a factor of instability in this model that was not eased by the training approaches. Indeed, differently from EGNO, SEGNO is not discretization invariant, thus this kind of unbalances might cause the model to performs poorly. EGNO instead, even under the same situation regarding the hyperparameter tuning, it was able to adapt to the new prediction interval and actually gain even an advantage in terms of stability thanks to its peculiar properties.

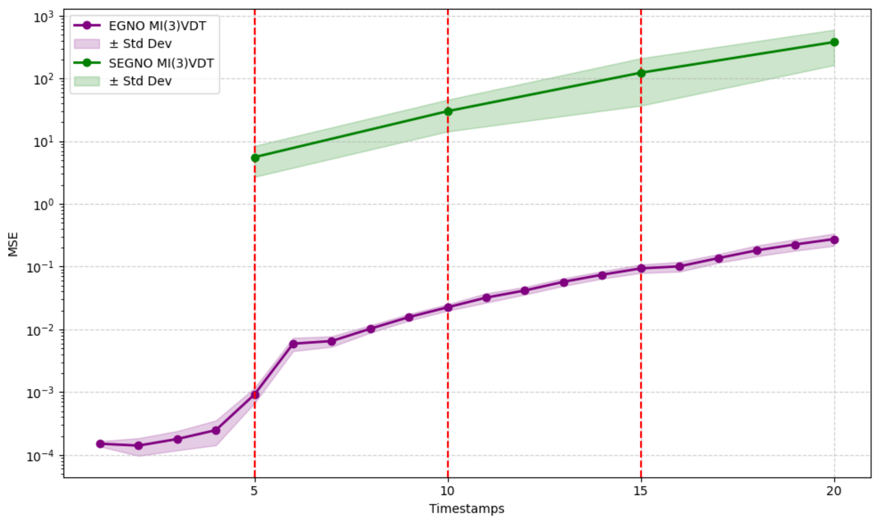


(a) Trajectories MSE for EGNO and SEGNO

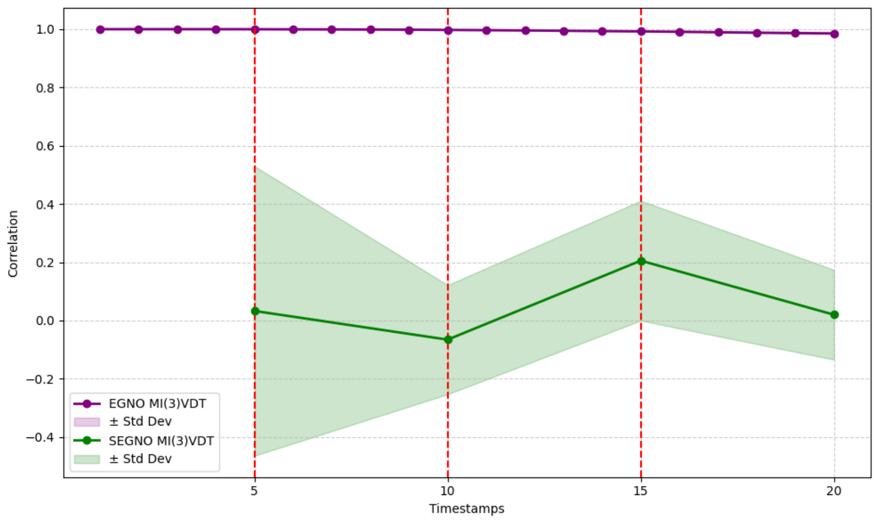


(b) Trajectories Correlation for SEGNO and EGNO

Figure 4.7: Trajectories Mean MSE (top in log scale) and Correlation (bottom) $\pm$ standard deviation (represented by the shaded region) calculated along 10 runs, after applying Rollout to standard and to 2 inputs and variable ΔT EGNO and SEGNO. In this case, $\Delta T = 5$ is used. The thick points represent the models' calculated points while the red dotted lines represent the ΔT intervals, meaning the final prediction after each rollout iteration.



(a) Trajectories MSE for EGNO and SEGNO



(b) Trajectories Correlation for SEGNO and EGNO

Figure 4.8: Trajectories Mean MSE (top in log scale) and Correlation (bottom) $\pm$ standard deviation (represented by the shaded region) calculated along 10 runs, after applying Rollout to standard and to 3 inputs and variable ΔT EGNO and SEGNO. In this case, $\Delta T = 5$ is used. The thick points represent the models' calculated points while the red dotted lines represent the ΔT intervals, meaning the final prediction after each rollout iteration.

Chapter 5

Discussion and Conclusion

The traditional approaches to modeling n-body systems, while powerful, have clear limitations. The computational demands increase dramatically with the number of bodies and the required resolution in time and space. Additionally, modeling these systems often relies on detailed parameterizations of forces, either approximations or empirically derived. Such parameterizations may not generalize well to new systems.

These and other limitations, are the reason why data-driven approaches, including machine learning and deep learning, offer a compelling alternative. Instead of solving the equations of motion directly, ML models learn patterns in the data, bypassing the need for explicit force calculations. For instance, neural networks can approximate complex, nonlinear relationships, making them well-suited for predicting the dynamics of n-body systems. Once trained, these models can perform predictions far faster than traditional methods, as they replace computationally expensive iterative processes with efficient inference steps. This tractation started from analysing the most prominent data-driven approaches to modeling n-body systems. These approaches shpwed to be NODE (Neural Ordinary Differential Equations) and NO (neural Operators). However, it still remains unclear which framework is better overall and with respect to specific aspects taken into consideration, particularly in modeling the temporal evolution of complex systems such as the one under analysis: charged particle systems. In particular one model was selected for each approach: SEGNO (NODE) and EGNO (NO).

Therefore, an experimental setting was built in order to try to analyze the different aspect and capabilities of these two architectures. This experimental setting consisted mainly in applying the rollout technique, with some other variations, to the selected models. In particular the variants consisted in considering multiple inputs to the models in order to check how it can help them in maintaing stability for a longer amount of steps, and adopting variable ΔT. This last variant involves, in the case of SEGNO, to put the model under an uniform sampling prediction task, while for EGNO, to use input timesteps at varying distances inside the first prediction step. These variants consist in both modifying the training precedure and the testing one, accordingly.

From these experiments, it resulted that EGNO has a limited capability of maintaining stable predictions after a few rollout iterations. On the other hand, SEGNO showed to be more stable than EGNO in predicting the overall trajectory. At the time though, considering only the initial part of the trajectory, on which EGNO is more stable, EGNO shows better performances in terms of average MSE. Moreover, when applying multiple inputs at training time, both models didn't seem to improve on the analysed metrics. On the other hand, the multiple inputs plus variable ΔT strategy seemed to improve EGNO performances on most metrics. The same cannot be said for SEGNO, which showed nonetheless to be albe to make long range stable predictions better than EGNO. Furthermore, different values for the size of the prediction step, were tested, and again EGNO obtained better performances in all analysed cases (again, the part of the trajectory used to compare the models at this stage was the initial one) also thanks to its discretization invariant property.

Overall, standard SEGNO showed to be a more robust architecture to model long multi-steps trajectories, but it seems that the adopted training strategies were not beneficial. EGNO instead, resulted as less robust in maintaining stability after a few rollout iterations but in the portion of the trajectory along which it was reliable, it showed to be more precise than SEGNO on most metrics, also because it was able to benefit from the training techniques adopted.

5.1 Future works

The current analysis might have provided a good starting point for further research on this domain. Indeed, one of the possible natural continuation of this work could be to the systems with more particles, and see how the models adapt (a few trials were made in this direction and EGNO proved to be a lot more computationally demanding as the number of particles increased).

Moreover, one way to better exploit the neural operators properties that EGNO holds, could be to increase the ΔT and see if that helps the model in maintaining more stability for long range trajectories. Furthermore, different training strategy than the adopted ones, might be beneficial for the models. One example could be to test other type of aggregation strategies when using multiple inputs in SEGNO.

Bibliography

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. «The Graph Neural Network Model». In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: `10.1109/TNN.2008.2005605` (cit. on p. 3).

[2] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. «Interaction Networks for Learning about Objects, Relations and Physics». In: *CoRR* abs/1612.00222 (2016). arXiv: `1612.00222`. URL: `http://arxiv.org/abs/1612.00222` (cit. on p. 3).

[3] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. *E(n) Equivariant Graph Neural Networks*. 2022. arXiv: `2102.09844 [cs.LG]`. URL: `https://arxiv.org/abs/2102.09844` (cit. on pp. 5, 12, 34).

[4] Jonas Köhler, Leon Klein, and Frank Noé. *Equivariant Flows: sampling configurations for multi-body systems with symmetric energies*. 2019. arXiv: `1910.00753 [stat.ML]`. URL: `https://arxiv.org/abs/1910.00753` (cit. on p. 5).

[5] Fabian B. Fuchs, Daniel E. Worrall, Volker Fischer, and Max Welling. *SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks*. 2020. arXiv: `2006.10503 [cs.LG]`. URL: `https://arxiv.org/abs/2006.10503` (cit. on p. 5).

[6] Jiaqi Han, Wenbing Huang, Tingyang Xu, and Yu Rong. *Equivariant Graph Hierarchy-Based Neural Networks*. 2022. arXiv: `2202.10643 [cs.LG]`. URL: `https://arxiv.org/abs/2202.10643` (cit. on p. 5).

[7] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. *GeoDiff: a Geometric Diffusion Model for Molecular Conformation Generation*. 2022. arXiv: `2203.02923 [cs.LG]`. URL: `https://arxiv.org/abs/2203.02923` (cit. on p. 5).

[8] Yang Liu, Jiashun Cheng, Haihong Zhao, Tingyang Xu, Peilin Zhao, Fugee Tsung, Jia Li, and Yu Rong. *SEGNO: Generalizing Equivariant Graph Neural Networks with Physical Inductive Biases.* 2024. arXiv: `2308.13212 [cs.LG]`. URL: `https://arxiv.org/abs/2308.13212` (cit. on pp. 5, 23).

[9] Minkai Xu, Jiaqi Han, Aaron Lou, Jean Kossaifi, Arvind Ramanathan, Kamyar Azizzadenesheli, Jure Leskovec, Stefano Ermon, and Anima Anandkumar. *Equivariant Graph Neural Operator for Modeling 3D Dynamics.* 2024. arXiv: `2401.11037 [cs.LG]`. URL: `https://arxiv.org/abs/2401.11037` (cit. on pp. 5, 8, 26).

[10] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. «Neural Ordinary Differential Equations». In: *CoRR* abs/1806.07366 (2018). arXiv: `1806.07366`. URL: `http://arxiv.org/abs/1806.07366` (cit. on pp. 6, 13).

[11] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. *Neural Operator: Graph Kernel Network for Partial Differential Equations.* 2020. arXiv: `2003.03485 [cs.LG]`. URL: `https://arxiv.org/abs/2003.03485` (cit. on pp. 6–8).

[12] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. *Fourier Neural Operator for Parametric Partial Differential Equations.* 2021. arXiv: `2010.08895 [cs.LG]`. URL: `https://arxiv.org/abs/2010.08895` (cit. on p. 9).

[13] Hongkai Zheng, Weili Nie, Arash Vahdat, Kamyar Azizzadenesheli, and Anima Anandkumar. *Fast Sampling of Diffusion Models via Operator Learning.* 2023. arXiv: `2211.13449 [cs.LG]`. URL: `https://arxiv.org/abs/2211.13449` (cit. on p. 9).

[14] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. «A Comprehensive Survey on Graph Neural Networks». In: *CoRR* abs/1901.00596 (2019). arXiv: `1901.00596`. URL: `http://arxiv.org/abs/1901.00596` (cit. on p. 11).

[15] Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. «Neural Operator: Learning Maps Between Function Spaces». In: *CoRR* abs/2108.08481 (2021). arXiv: `2108.08481`. URL: `https://arxiv.org/abs/2108.08481` (cit. on p. 16).

[16] Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. «Fourier Neural Operator for Parametric Partial Differential Equations». In: *CoRR* abs/2010.08895 (2020). arXiv: `2010.08895`. URL: `https://arxiv.org/abs/2010.08895` (cit. on p. 19).

[17] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. *Learning to Simulate Complex Physics with Graph Networks*. 2020. arXiv: `2002.09405 [cs.LG]`. URL: `https://arxiv.org/abs/2002.09405` (cit. on p. 29).

[18] Alessio Gravina, Daniele Zambon, Davide Bacciu, and Cesare Alippi. «Temporal Graph ODEs for Irregularly-Sampled Time Series». In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*. Ed. by Kate Larson. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2024, pp. 4025–4034. DOI: `10.24963/ijcai.2024/445`. URL: `https://doi.org/10.24963/ijcai.2024/445` (cit. on p. 31).