Sequential Models: A Comparative Study of Neural Differential Equations and Modern Approaches

Tanaka Akiyama[1] **Gian Favero**[1]
Maxime Favreau-Vachon[1] **Mohamed Mohamed**[1]
[1] McGill University
tanaka.akiyama@mail.mcgill.ca
gian.favero@mail.mcgill.ca
maxime.favreau-vachon@mail.mcgill.ca
mohamed.mohamed5@mail.mcgill.ca

Abstract

There is a necessity for longitudinal data modeling techniques capable of effectively handling irregularly sampled or partially observed data, motivating the exploration of methodologies like Neural Differential Equations (nDE). In this study, we conduct a comparative analysis of sequential models for time-series data, focusing on nDEs (both Neural Controlled Differential Equations (nCDE) and Neural Ordinary Differential Equations (nODE)) alongside modern discrete methods like LSTM networks, RNNs, and Transformers. Specifically, we compare their performance in tasks involving the classification of Latin characters (Encoder Task), forecasting of Latin characters (Sequence-to-Sequence Task), and forecasting of weather (Decoder Task). The nCDE model outperforms Transformer, LSTM, and RNN baselines in Latin character classification, with fewer parameters, while in sequence-to-sequence tasks, the Transformer model excels due to self-attention, and the nCDE-nODE model achieves comparable performance with fewer parameters, although self-attention-based models better capture intricate details. In the decoding task, the nODE model performs slightly below RNN and LSTM models in terms of MSE but uses significantly fewer parameters. Our investigation revealed nDE's remarkable ability to handle irregularly sampled data with superior memory efficiency. However, their slower computation speed and limitations in real-time settings pose challenges. Nonetheless, nDEs offer continuous representations of time series, excel on irregularly sampled data, and exhibit memory efficiency, making them compelling options in resource-constrained scenarios.

1 Introduction

In the sphere of data analysis, the attempt to model longitudinal data has evolved into a multifaceted exploration, incorporating a spectrum of methodologies. Spanning from mechanistic theory-driven differential equation models to the field of machine learning (ML), this diverse array of approaches emphasise the complexity of longitudinal modelling. With the growing of popularity surrounding ML techniques, discrete models have emerged as powerful tools capable of capturing intricate patterns within longitudinal data. As such, exploring this domain requires a comprehensive understanding of both traditional theory-driven frameworks and the cutting-edge methodologies of ML.

Transformers and Recurrent Neural Networks (RNNs) and derivatives [1] have become a preferred model for processing sequential data, especially in the context of time series datasets.

These datasets, typically seen as sequences of observations originating from a dynamical system, find representation through such models as discrete approximations capturing these fundamental dynamics. However, challenges arise when dealing with irregularly sampled or partially observed data, necessitating the utilization of data pre-processing techniques such as binning or imputation[2]. Furthermore, in real-world longitudinal data analysis, the discrepancy between theoretical models and realities presents a great challenge to overcome. Both physically informed networks (PINNs) and those reliant on discrete representations, such as Long Short-Term Memory (LSTM) networks [3], often struggle to capture the full nuanced intricacies inherent in datasets derived from natural and dynamic systems.

Ideally, longitudinal models should demonstrate adaptability to the complexities of real-world data without solely relying on perfectly curated datasets. There is a pressing need to model continuous processes more accurately, especially in scenarios with irregularly sampled or partially observed data. Models reliant on discrete representations, such as LSTM networks, struggle to comprehensively adapt to real-world complexities, highlighting their inadequacy. Additionally, the aim extends to capturing the diverse and theory-averse components of dynamical systems. This underscores the necessity for advanced methodologies that can effectively handle the continuous-time dynamics inherent in longitudinal data, enabling a more nuanced analysis of real-world phenomena and enhancing precision and efficacy in longitudinal data analysis.

In response to the limitations of discrete representations, various sophisticated methodologies have emerged, including direct approximations of continuous-time dynamics. A notable strand of research focuses on adapting RNNs to accommodate continuous-time evolution by allowing certain hidden states to evolve according to Ordinary Differential Equations (ODEs)[4, 5]. This approach holds particular interest, serving as a pivotal focus of our investigation. Our methodology involves exploring the Neural Differential Equations (nDE) approach to model longitudinal data, aiming to harness the strengths of classical differential equations and discrete neural networks [2]. The neural network architecture facilitates high-capacity estimation and efficient optimization, while insights from differential equations provide priors on the model space, enhance memory efficiency, and support a continuous-time approach to handling irregular data. In our study, we will conduct a comparative analysis of various machine learning models across diverse tasks involving sequential data. It is crucial to acknowledge that not all algorithms exhibit equal efficacy in handling this type of data. For example, RNNs and LSTMs networks are adept at capturing temporal dependencies, whereas other models such as attention-based models [6] native to natural language processing (NLP) may not perform as effectively.

2 Contribution

Our contribution to the existing body of work is twofold. Firstly, we undertake a meticulous examination of longitudinal models applied to irregularly sampled and partially observed data. Our primary focus centers on conducting a comprehensive comparative analysis of nDEs, comprising Neural ODE (nODE) and Neural Controlled Differential Equations (nCDE) alongside modern discrete approaches like LSTM networks, RNNs, and Transformers. Through this comparative study, we aim to discern the unique strengths and limitations of nDEs within the contemporary landscape of neural network architectures, thereby providing valuable insights into their practical applicability and potential.

Secondly, we delve into the assessment of performance benefits associated with employing neural differential equations for modeling continuous dynamic processes. This endeavor involves a detailed comparison aimed at elucidating the performance attributes, intricacies in training methodologies, and constraints across diverse application scenarios. Our evaluation encompasses multiple tasks in sequential modelling, namely classification and forecasting, formulated as encoder, sequence-to-sequence, and decoder problems. Through these tasks, we aim to comprehensively evaluate the applicability and performance of nDEs in comparison to baseline models, thus contributing insights to the field.

Our code is available at `https://github.com/faverogian/nDEs`.

3 Related Work

3.1 Neural Ordinary Differential Equations

Neural ODEs offer a novel approach to approximating a mapping from input x to output y. This is achieved by learning a function f_θ and linear mappings ℓ_θ^1 and ℓ_θ^2. The approach revolves around the evolution of a latent variable z_t over a hypothetical time dimension t, where z_t continuously evolves from an initial state z_0, determined by $\ell_\theta^2(x)$, through an integral process influenced by f_θ. The output is approximated using a linear combination of z_t weighted by ℓ_θ^1. Notably, f_θ does not explicitly depend on t, though t can be included as an additional dimension in the integral formulation:

$$z_t = z_0 + \int_0^t f_\theta(z_s)ds \tag{1}$$

This novel approach delves into the dynamic nature of hidden layers within neural networks, enhancing data modeling capabilities [2]. Traditionally, hidden layers are static representations of intermediate computations, but in nODEs, they evolve continuously across layer depth t. At each depth t, the hidden state z_t encapsulates the learned information up to that point, influenced by input data, network parameters, and derivative function dynamics. This continuous evolution allows Neural ODEs to capture intricate data patterns and dynamics more effectively than traditional networks.

Moreover, the continuous nature of hidden layer evolution facilitates gradient flow throughout the network [7], enabling efficient parameter optimization through techniques like backpropagation through time. This optimization process involves computing gradients with respect to f parameters and integration start and end points, enabling iterative learning from data.

Although the discussion so far has not addressed sequential data like time series, the introduction of the dimension t in Equation (1), integrated over time, serves as an internal aspect of the model. However, this inclusion prompts an exploration of whether nODEs can be extended to handle sequential data effectively. The challenge lies in aligning the introduced temporal dimension with the natural ordering of sequential data, as Equation 1 inherently defines an ordinary differential equation. Once the parameters θ are learned, the solution to Equation (1) is solely determined by the initial condition at z_0, posing a limitation on directly incorporating subsequent data.

However, this obstacle finds resolution through existing mathematical frameworks, particularly in rough analysis, a field dedicated to the study of controlled differential equations. By leveraging insights from rough analysis, solutions for incorporating incoming data into nODEs can be developed. Neural Controlled Differential Equations [2] illustrate the expansion of the nODE model using controlled differential equations, leading to the development of the nCDE model. Similar to how nODEs serve as the continuous counterpart of a ResNet, the nCDE can be understood as the continuous counterpart of an RNN.

3.2 Neural Controlled Differential Equations

The nCDE model encompasses three fundamental aspects. Firstly, it possesses the capability to handle incoming data, even when it is irregularly sampled or partially observed. Secondly, in contrast to previous approaches, the model can be trained efficiently using memory-saving adjoint-based backpropagation, extending across observations. Thirdly, empirical studies conducted on various datasets, including CharacterTrajectories and Speech Commands, showcase its superior performance compared to similar models based on ODEs or RNNs.

Additionally, the authors provide theoretical insights demonstrating the universality of their model as an approximator and its subsuming nature over apparently similar ODE models where the vector field directly depends on continuous data. To formalize the model, let τ and T be real numbers with τ less than T, and let v and w be natural numbers. Consider X, a function mapping the interval $[\tau, T]$ to a real space of dimension v, which is continuous and of bounded variation, implying X to be Lipschitz. Lipschitz continuity ensures that the function X varies smoothly, without sudden changes or spikes, which is essential for stability and convergence in mathematical models. Next, the authors define X to be a natural cubic spline with knots at $t_0, ..., t_n$, approximating the underlying process observed through x. A natural cubic spline is a piecewise cubic polynomial function defined by its values and first derivatives at the knots, ensuring smoothness and flexibility in capturing the underlying data trends. Knots are points where the cubic polynomials connect, ensuring smooth transitions between segments of the spline curve[8].

Moreover, let f be a continuous function mapping a space of dimension w to a space of dimension w times v. With these definitions, we can express a continuous path z over the interval $[\tau, T]$ in terms of a controlled differential equation (CDE) driven by X:

$$z_t = z_\tau + \int_\tau^t f(z_s) dX_s$$

This equation guarantees global existence and uniqueness under global Lipschitz conditions on f. Here, z is controlled or driven by the data process X. Now, considering a fully observed, potentially irregularly sampled time series $x = ((t_0, x_0), (t_1, x_1), ..., (t_n, x_n))$, where each t_i is the timestamp of the observation x_i, and $t_0 < ... < t_n$, they define X to be a natural cubic spline with knots at $t_0, ..., t_n$, approximating the underlying process observed through x.

Next, let f_θ be a neural network model parameterized by θ, and ζ_θ be another neural network model parameterized by θ. They then define the Neural CDE model as the solution to the following equation:

$$z_t = z_{t_0} + \int_{t_0}^t f_\theta(z_s) dX_s \tag{2}$$

Here, z_{t_0} is initialized as $\zeta_\theta(x_0, t_0)$ to avoid translational invariance. Translational invariance implies that the model's behavior remains unchanged under translations or shifts in the input data, which may lead to undesirable effects such as insensitivity to small changes in the data or redundant representations [9]. Similar to RNNs, the model's output can be either the evolving process z or the terminal value z_{t_n}, with the final prediction typically obtained through a linear transformation of this output.

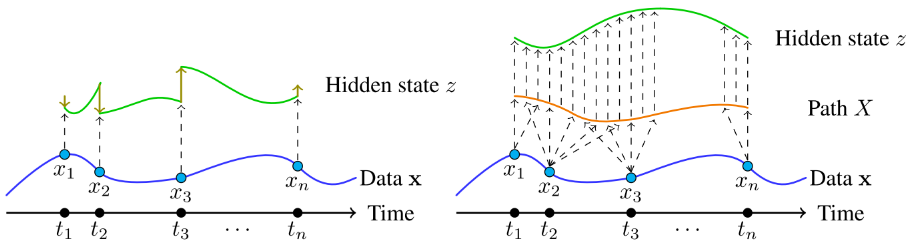


Figure 1: Some data process is observed at times $t_1, ..., t_n$ to give observations $x_1, ..., x_n$. Outside of these intervals, the process remains unobserved. On the left, approaches often adjust the hidden state at each observation and may even continuously evolve it between observations. On the right, the Neural CDE model differs by maintaining continuous dependence on the observed data in its hidden state.[2]

The distinguishing feature of this model is its dependence on the data process X, making it inherently adaptive to incoming data. This adaptability arises from changes in X, which consequently alter the local dynamics of the system. Equation (1) and (2) show this adaptability, where the dynamic evolution of the system is depicted through the integration of neural network function f_θ with respect to the data process, leading to the generation of state variables such as z_t. Equation (2) demonstrates the model's responsiveness to incoming data, as changes in X directly influence the system's dynamics, allowing for a seamless adaptation to evolving conditions. See Figure 1. This is a unique attribute that sets Neural CDE apart from its counterparts, showing it is a powerful tool for modelling complex systems with dynamic data dependencies.

3.3 Recurent Neural Networks

RNNs are a class of artificial neural networks designed to process sequential data by retaining information from previous time steps. They have the advantage of being able to handle input sequences of varying lengths, making them versatile for tasks involving sequences of different lengths, such as natural language processing and time series analysis [10]. However, basic RNNs lack mechanisms to handle irregularly sampled time series data and missing values effectively. While they can adapt to sequences with different lengths, they struggle when faced with irregular time intervals between observations and missing data points.

Models like GRU-D and related approaches attempt to address these limitations by incorporating temporal decay mechanisms to handle irregular sampling and missing values [11]. Therefore, RNN-based approaches offer flexibility in handling sequences of different lengths, their performance on irregularly sampled time series data may be compromised without specialized techniques to address missing values and irregular sampling intervals effectively. Moreover, RNNs are inherently susceptible to the well-established gradient explosion/vanishing problem for long sequences with many forward passes. While many models derived from RNNs have been proposed to mitigate this problem (LSTMs, GRUs), the problem still plagues models in this family more than those introduced in recent years, like the Transformer, or Mamba [12].

3.4 Long Short-Term Memory

LSTM networks are a type of RNN architecture designed to address the limitations of traditional RNNs in capturing long-range dependencies in sequential data. LSTMs achieve this by introducing specialized memory cells and gating mechanisms, including input, forget, and output gates, which control the flow of information within the network. This enables LSTMs to selectively retain or discard information over multiple time steps, allowing them to capture complex temporal patterns and dependencies[13].

LSTMs have demonstrated improved performance over traditional RNNs in modeling sequential data across various domains, including natural language processing, speech recognition, and time series forecasting. Their ability to capture long-term dependencies makes them particularly well-suited for tasks where understanding context over extended sequences is crucial.

When it comes to handling irregularly sampled or partially observed time series data, LSTMs have also shown promise. Their inherent ability to capture temporal dependencies allows them to adapt to varying time intervals between observations. However, while LSTMs can handle missing values to some extent, they may struggle with datasets containing large proportions of missing data or irregular sampling rates. In such cases, simple imputation methods or temporal smoothing techniques may lead to suboptimal results, as the model may not effectively distinguish between true observations and imputed values [14]. Nonetheless, researchers have been exploring innovative approaches to enhance LSTM models for handling irregular time series data, including modifications to the gating mechanisms and incorporating temporal decay factors, aiming to improve their robustness and performance in these challenging scenarios.

3.5 Transformers

Transformers [6] are a type of deep learning model that has gained significant attention for its success in various natural language processing tasks. Unlike RNNs, which process input data sequentially, Transformers utilize self-attention mechanisms to weigh the importance of different input tokens simultaneously. This parallel processing capability allows Transformers to capture long-range dependencies more effectively, making them well-suited for tasks involving sequential data, such as language translation and text generation [15].

Transformers have gained prominence in recent years, particularly in NLP tasks, due to their ability to capture long-range dependencies more effectively than traditional RNN-based architectures. They operate in parallel, allowing them to process sequences more efficiently. When it comes to irregularly sampled or partially observed data, Transformers may face challenges. Since Transformers typically operate on fixed-length sequences, handling sequences with missing values or irregular sampling intervals can be problematic without additional preprocessing or modifications.

Despite these challenges, researchers have been exploring various techniques to adapt Transformers for irregularly sampled or partially observed data, such as incorporating masking mechanisms or designing specialized architectures [16, 17]. They can perform well on irregularly sampled or partially observed data by learning contextual representations from the available data points, although their performance may depend on the specific architecture and pre-training strategy used. Their effectiveness on irregularly sampled or partially observed data remains an active area of research.

4 Methodology

A prominent challenge in longitudinal data analysis lies in managing irregularly sampled data. Consider, for instance, weather monitoring datasets, where recording intervals may not always provide sufficient data due to acquisition errors, equipment malfunctioning, or unexpected variations in weather patterns at frequencies higher than expected. Such irregular sampling introduces significant complexities in analysis and interpretation, demanding advanced techniques for data handling and processing.

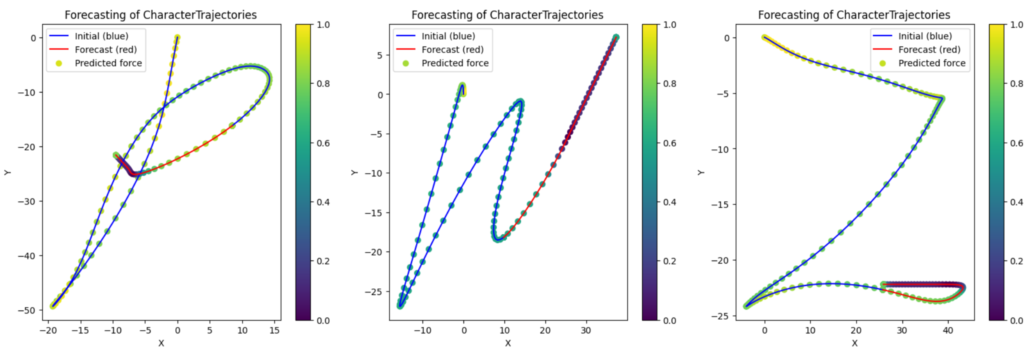
While RNNs and LSTM networks remain favored choices for sequential data analysis, especially in time series datasets, they encounter limitations, especially in scenarios with large gaps between consecutive data points. These models heavily rely on the sequential structure of data, making it challenging to capture temporal intricacies effectively. Moreover, RNNs and LSTMs typically are designed for fixed-size input sequences, complicating the handling of irregularly sampled data and necessitating techniques like padding or interpolation, which may introduce noise and distort temporal relationships [18].

Similarly, Transformers face limitations with irregularly sampled or partially observed data due to their reliance on fixed-length input sequences in addition to an extensive parameter set needed to model regular data without added noise. Moreover, fine-tuning transformers for optimal performance demands extensive parameter tuning, which can be challenging, especially with limited data availability [19]. Despite efforts to mitigate these challenges through techniques like interpolation or imputation to address missing values, these models may still struggle to learn from incomplete information, resulting in suboptimal performance or instability.

Thus, we design three experiments to bring to light a relatively quiet, but promising field in sequential modelling - nDEs.

4.1 Resources and Datasets

4.1.1 CharacterTrajectories Dataset

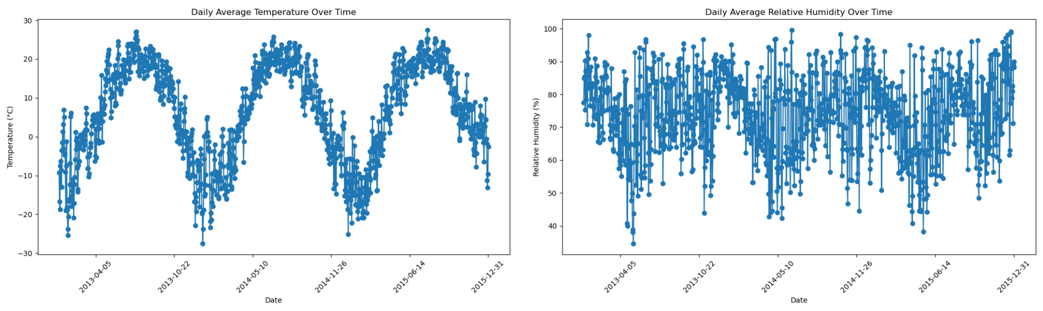


(a) “P” (b) “W” (c) “Z”

Figure 2: Samples from the CharacterTrajectories dataset primed for the forecasting task. Dot colour is the predicted force, the blue line is the provided trajectory and the red line is the target trajectory.

The CharacterTrajectories dataset, sourced from the UEA Time Series Classification Archive, encompasses a collection of meticulously labeled pen tip trajectories captured during the act of writing individual characters. Each trajectory is attributed to the same writer, ensuring consistency for primitive extraction purposes. Notably, only characters characterized by a single pen-down segment were included in this dataset, of which there are 20. Comprising 2858 time series entries, the dataset spans varying lengths and encapsulates the x and y velocities alongside the pen tip force during the composition of one of the 20 Latin alphabet characters, all accomplished in a single stroke. Each entry undergoes normalization by feature and is uniformly sampled at a frequency of 200 Hz. This dataset claims to have regular and fully observed sampling [20].

4.1.2 ERA5 Dataset



(a) Daily Average Temperature (°C) (b) Daily Average Humidity (%)

Figure 3: Visualization of the daily weather patterns in Montreal over one year. Data is taken from the ERA5 dataset and is unprocessed.

The ERA5 dataset [21], also known as the ECMWF Reanalysis v5, provides hourly estimates of various atmospheric, land, and oceanic climate variables. From this dataset, we will utilize extensive measurements of fundamental meteorological variables, including temperature and relative humidity acquired from the centre of downtown Montreal from the years 2013-2022. Represented in three-dimensional arrays with dimensions of time, latitude, and longitude, the dataset offers a detailed snapshot of atmospheric conditions over a specified region and time period. Temperature data, provides insights into air temperature variations across the spatial grid and temporal domain. Concurrently, relative humidity elucidates moisture levels in the air, expressed as a percentage, aiding in understanding atmospheric moisture dynamics.

4.1.3 Other Resources

Table 1: Summary of model implementations with a rough estimate of the number of trainable parameters per encoder/decoder block.

Model	**Python Libraries**	**Module Parameters**
Neural CDE	torch, torchcde, torchdiffeq	9,300
Neural ODE	torch, torchdiffeq	2,575
Transformer	torch	413,364
LSTM	torch	203,284
RNN	torch	52,756

For this project, we leveraged PyTorch's dynamic computational graph and efficient backpropagation to implement baseline models such as LSTM, RNN, and Transformers. To implement the nDE models, we use torchcde and torchdiffeq, which are also based in PyTorch.

Our primary focus lay in developing a flexible and modular code framework to facilitate seamless modifications in model structures, dataset variations, and hyperparameter tuning. Each model had the same NVIDIA V100 16 GB GPU at its disposal for training. Collaborative efforts were coordinated through GitHub. To ensure the reliability of our models and mitigate overfitting risks, we implemented validation techniques and incorporated early stopping with a carefully defined patience parameter to enhance model generalizability. Recognizing potential risks associated with dataset errors, missing values, or inconsistencies, we conducted thorough data pre-processing and cleaning procedures.

4.2 Designed Solution and Implementation

In our solution, we design three tasks to evaluate the effectiveness of DE networks in modeling time-series data in comparison to other well-known models.

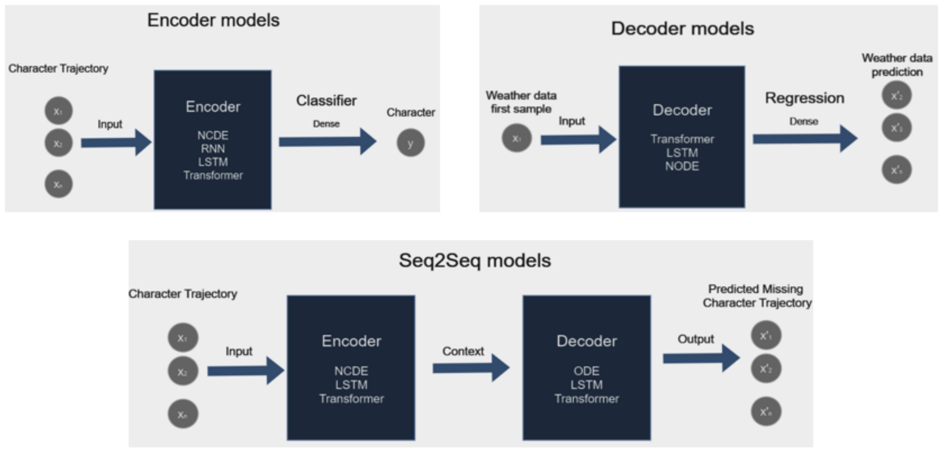


Figure 4: High-level view of the architectures implemented for each style of task.

4.2.1 Classification of Latin Characters - Encoder Task

Pre-Processing and Implementation: The pre-processing pipeline for the CharacterTrajectories dataset begins with loading the raw data and concatenating the training and test sets, which are provided in an unorthodox 50-50 split. Labels are then encoded to start from 0 and converted to integers for consistency. Sequences are padded to the same length using zeros, followed by normalization to standardize the data. The pre-processed data is split into training and testing sets, with an 80-20 ratio, and saved for future use.

To evaluate the effectiveness of sequential models on irregularly sampled time series, we conducted three experiments where 25%, 50%, or 75% of the data was randomly and independently dropped for each sample. The data dropout occurred uniformly across channels to simulate full observation but irregular sampling. Additionally, two extra channels were introduced in the dataset for this purpose: time and cumulative observations. The former represents a linearly spaced vector of the sequence length, while the latter corresponds to the applied masking for dropping the data. For this task, an nCDE model was compared against RNN, LSTM, and Transformer encoder baselines.

For the nCDE model, each sample was extended using a fill-forward method to match the length of the longest sample, which was 182 as documented in [2]. To handle this discrete irregularly sampled time series, we employed a natural cubic spline to approximate the data as a continuous process. This choice offers specific advantages for differentiation by ensuring that the data remains twice differentiable, a requirement for the efficient use of differential equation solvers. The coefficients of the fitted spline were then utilized as the training data for the models. In the case of the baseline models, each sample was padded with zeros instead of employing a fill-forward technique. Additionally, an attention mask was introduced to indicate the locations of missing data and padding for the Transformer.

The data was partitioned into a 65/15/20 split for training, validation, and testing, respectively. Each model underwent tuning on the validation set and was subsequently trained for 500 epochs, with the best parameters determined based on validation accuracy. Upon completion of training, the models were evaluated based on test accuracy and model size, measured in terms of trainable parameters. Hyperparameters were chosen based on the experiment where 25% of the data was dropped and were retained for the subsequent experiments. The experiments were run 5 times with the test accuracy reported as the mean and the standard deviation of the accuracy.

Training Details: We trained the models on cross-entropy loss (for 20 class classification) and then reported on accuracy and model size.

The RNN and LSTM models shared similar configurations, featuring hidden dimensions of 128, and layer dimensions of 2 for 52,756 and 203,284 trainable parameters respectively. The default tanh activation function was used. Trained with a learning rate of 1E-3 and a batch size of 32, these models employed early stopping with a patience of 100 to prevent overfitting.

The nCDE model integrates a differential equation, f_θ, which was approximated by a three-layer feedforward neural network (FFNN) with one final linear mapping layer. The FFNN comprised 3 hidden layers, each consisting of 32 units with ReLU activations, except for the last layer, which employed a tanh activation. This model contained a total of 9,300 trainable parameters. Employing a fourth-order Runge-Kutta with 3/8 rule solver for the differential equation, the nCDE model utilized a step size of 1, representing the minimum distance between observations. Construction of the nCDE model utilized the torchcde, torchdiffeq, and PyTorch libraries for model representation, differential equation solver, and FFNN construction, respectively. With a learning rate of 1E-3 and a batch size of 32, the nCDE model underwent training for 500 epochs, utilizing the best validation parameters for test metrics.

Similarly, the Transformer model was implemented as an encoder with an embedding dimension of 32, generated through a single-layer neural network. This architecture included 4 attention heads and 3 attention layers, with a dropout rate of 0.1. The output was computed using a single-layer neural network. The Transformer model comprised a total of 413,364 trainable parameters and was implemented using the PyTorch library. Similar to the Neural CDE model, the Transformer model underwent training with a learning rate of 1E-4 and a batch size of 32, over the course of 500 epochs, employing the best validation parameters for test metrics.

4.2.2 Forecasting of Latin Characters - Sequence-to-Sequence Task

Pre-Processing and Implementation: The same CharacterTrajectories dataset was used as in the previous task. To assess the efficacy of the sequential models on forecasting irregularly sampled time series, we run an experiment in which only the first 70% of the data is provided to the model, and the rest must be predicted. A time channel being a linear spaced vector of the sequence length was added as a channel and each sample was padded to the length of the longest sample.

This is an example of a problem that can be solved in an encoder-decoder structure. For the nDE based model, we use an nCDE encoder with an nODE decoder. For the baseline models, we use a pure LSTM encoder-decoder, and a pure Transformer encoder-decoder. An RNN based model was not used for this experiment due to gradient propagation issues seen in the encoder task. We additionally run ablation experiments using an nCDE encoder alongside a Transformer decoder and an LSTM decoder to assess the effectiveness on a component level. For the nDE model, we use the same natural cubic spline to approximate the data as a continuous and process. To handle padding, the same techniques outlined in the encoder-only experiment were held. For the Transformer model, start and end tokens were appended to the source and target sequences for more stability in training.

The data was separated into a 65/15/20 split. Each model was tuned on the validation set then was trained for 500 epochs with its best parameters saved based on validation mean-squared error (MSE). Once training was complete, the models were reported based on average test MSE and model size in terms of trainable parameters.

Training Details: We trained the models on MSE loss of the forecasted sequence (with loss from the padded areas zeroed out) and then reported on test metrics and model size.

The encoder was represented as a nCDE model with its differential equation, approximated by a three-layer FFNN. The FFNN consisted of 3 hidden layers each of size 32 units with ReLU activations at each layer except the last, which used a tanh activation, as recommended by [13]. An nODE was used as the decoder based on the final hidden state of the nCDE. The nODE used an identical vector field in hidden layer size, hidden layers, activation, and FFNN. A final linear mapping layer was used to map the sequence of predicted hidden states to (x, y, pressure) values corresponding to the forecasted sequence. A fourth-order Runge-Kutta with 3/8 rule solver was used as the differential equation solver for both models, with the step size set at 1, which was the minimum distance between observations. The torchcde, torchdiffeq, and PyTorch libraries were used to construct the nCDE model, the differential equation solver and nODE model, and the FFNNs, respectively. The learning rate was set to 0.001, the batch size used was 32, and the model was trained for 150 epochs, with the best validation metric parameters being used for the test metrics. To optimize the training of the nODE decoder, the target label was incrementally unmasked throughout training upon convergence (ie. at the start only 10% of the label was to be predicted, then 20% etc.).

The Transformer model was constructed with an encoder featuring an embedding dimension of 32 (generated via a single layer NN), utilizing 4 attention heads across 3 attention layers, a dropout rate of 0.1 to enhance generalization, and a single mapping layer from the hidden state to the output state. It had 838,307 trainable parameters and was trained for 500 epochs using a learning rate of 1E-3 and a batch size of 32. The model's architecture was implemented with the PyTorch library, and it employed early stopping based on validation performance (with a patience of 100).

The LSTM Encoder-Decoder model was constructed with blocks featuring 128 hidden channels and two hidden layers. It had 401,283 trainable parameters total. The model was trained for 500 epochs using a learning rate of 1E-3 and a batch size of 32. The model's architecture was implemented with the PyTorch library, and it employed early stopping based on validation performance (with a patience of 100).

For the ablations, equivalent nCDE encoder modules were used as in the nCDE-nODE model. An identical Transformer decoder block was used, but a slightly boosted LSTM decoder block (1 extra hidden layer) was used due to instability with the original block. Each model was trained using the same strategy as outlined above.

For this task, we utilized MSE as the performance metric for forecasting character trajectory on the validation set. MSE quantifies the average squared difference between the predicted trajectory values and the actual trajectory values not including the padding sequences for each sample. The results were reported based on a single run for each experiment due to the time cost of training the nCDE-nODE model.

4.2.3 Forecasting of Weather - Decoder Task

Pre-Processing and Implementation: Initially, the ERA5 data is loaded from multiple NetCDF files containing temperature and humidity data for the latitude and logitude coordinates of downtown Montreal for the years 2013-2022. Subsequently, temperature and humidity values are extracted and processed, including handling missing values, calculating bi-monthly averages (twice per month), and normalization. The time dimension is also considered, with numeric time values converted to linear spaced vectors corresponding to the 26 bi-monthly intervals throughout one year.

We formulate this problem as an autoregressive decoder style. Effectively, each model must learn the dynamics of downtown Montreal's temperature and humidity over the course of one year span given only an initial observation. For the nDE based model, we use an nODE. This is compared against three baselines: an autoregressive Transformer decoder, RNN decoder, and LSTM decoder.

The data over 10 years was separated into an 70/20/10 split. Each model was tuned on the validation set and was trained for a maximum of 250 epochs with its best parameters saved based on MSE between the predicted and actual 12 month forecast. Once training was complete, the models were reported based on average test MSE, R2 correlation, and model size in terms of trainable parameters. The experiments were run 5 times for each model and the mean and standard deviation were reported.

Training Details: We employed MSE and R2 correlation between the predicted and actual weather forecast (bi-weekly temperature and humidity) as the performance metric. A lower MSE implies a higher degree of closeness in predicting temperature, humidity variables, while the R2 correlation indicates the effectiveness of the forecasting model in capturing the complex dynamics of weather patterns over time.

The nODE decoder integrated over a differential equation approximated by a two layer FFNN with 16 hidden channels with ReLU activations at each layer except the last, which used a tanh function. A final linear mapping layer was used to map the sequence of predicted hidden states to (temperature, humidity) values corresponding to the forecasted sequence - 26 data points corresponding to a one-year bi-weekly span. A fourth-order Runge-Kutta with 3/8 rule solver was used as the differential equation solver for both models, with the step size set at 1, which was the minimum distance between observations. The torchdiffeq and PyTorch libraries were used to construct the the differential equation solver and nODE model, and the FFNNs, respectively. The learning rate was set to 1E-3, the batch size used was 32, and the model was trained for 50 epochs, with the best validation metric parameters being used for the test metrics. To optimize the training of the nODE decoder, the target label was incrementally unmasked throughout training upon convergence (ie. at the start only 10% of the label was to be predicted, then 20% etc.).

The Transformer decoder was implemented as an autoregressive self-attention module that takes in one initial (temperature, humidity) value and outputs the remainder of the sequence one value at a time. It consisted of an embedding dimension of 32 (generated via a single layer NN), utilizing 4 attention heads across 3 attention layers, a dropout rate of 0.1 to enhance generalization, and a single mapping layer from the hidden state to the output state, all installed via the PyTorch library. It was trained for a maximum of 250 epochs with a learning rate of 1E-3, and a batch size of 32, with validation metrics being used to select the best model parameters.

The LSTM and RNN models were implemented in a very similar manner to the Transformer model. Autoregressive predictions were made on standalone modules each with 32 hidden channels and two hidden layers and constructed using PyTorch. The models were each trained for a maximum of 250 epochs with a learning rate of 1E-3 and a batch size of 32.

4.3 Alternative Solutions

Although we emphasize the models mentioned above, it is important to acknowledge alternative machine learning approaches such as Convolutional Neural Networks (CNN) or Multi-layer Fully Connected Networks (MLP). However, existing literature, such as Yin et al.[22], suggests that these alternatives may be less effective in capturing the structural and semantic complexities of the entire input necessary for comprehensive sequence modeling, in comparison to the models selected for our analysis.

In the field of nDEs, several authors have introduced Neural SDEs as a stochastic variant of nODEs that can be used for generative modelling for time series [23]. These models are closely related but can suffer from poorer convergence rates and require adjustments in the adjoint backpropagation to perform adequately. For these reasons the more well-established nODE and nCDE were evaluated in this study.

There are additionally special cases of nCDEs that have been proposed in which the hidden state of an RNN is updated in continous time via an ODE between observations [24]. Examples of such models include the GRU-D, ODE-RNN, and ODE-LSTM. These models are considered discretisations of nCDEs and are not researched in this study.

5 Results

5.1 Classification of Latin Characters - Encoder Task

We begin by demonstrating the effectiveness of nCDEs on irregularly sampled data via an encoder multi-class classification task. The results are shown in Table 2. The nCDE outperforms each of the baselines considered in mean test accuracy as well as limiting its standard deviation over five runs. Its performance remains consistent across the experiments with 25% to 75% of the sequences dropped, demonstrating a further superiority in nCDE models over the baselines. Moreover, the performance is achieved using an order of magnitude less trainable parameters from the next highest baseline.

We see that the Transformer and LSTM baselines perform comparably across the experiments, with very similar decreases in performance at the 75% dropped level as well as similar variances across the runs. Compared to the nCDE, the performance did not reach the same level of accuracy nor did it remain consistent as sampling became more irregular. Furthermore, the memory cost of each of Transformer and LSTM encoder models was vastly greater than that of the nCDE. The RNN was not able to converge or remain stable in virtually any of the experiments across a variety of hyperparameter combinations, potentially highlighting a gradient propagation issue for sequences at lengths present in this task.

Table 2: Test accuracy and model size in the classification of CharacterTrajectories (encoder) task, computed over five independent runs.

Model	**Test Accuracy**			**Parameters**
	25% Dropped	50% Dropped	75% Dropped	
Transformer	96.8 ± 1.0	93.9 ± 1.3	89.9 ± 2.2	413,364
LSTM	97.5 ± 2.3	97.2 ± 1.0	92.3 ± 2.2	203,284
RNN	51.1 ± 5.5	44.2 ± 7.6	34.7 ± 6.2	52,756
Neural CDE	**98.1 ± 0.8%**	**97.9 ± 0.5%**	**97.9 ± 0.7%**	**9,300**

5.2 Forecasting of Latin Characters - Sequence-to-Sequence Task

We next observe nDEs in their capability to effectively carry out sequence-to-sequence tasks. Quantitative results are seen in Table 3, while a sample of visual results are seen in Figure 5.

The Transformer model exhibits the lowest average test MSE loss on the forecasted sequence, with the benefits of self-attention in modelling long range dependencies being brought to the foreground in this experiment. However, this good performance comes at a significant expense, as it boasts the largest number of parameters by over double the nearest model. The LSTM model also performed well in this sequence-to-sequence task, achieving an average test MSE loss around one order of magnitude higher than the pure Transformer model, though using half as many parameters.

The nCDE-nODE model achieved a test MSE on the same order of magnitude as the LSTM, though its performance is notably worse both quantitatively and qualitatively. In saying this, though, it used only a marginal fraction of the memory of the baseline models. Anecdotally, the model exhibited much slower training times due to repeated numerical solving and required specific training tricks to improve performance of the nODE decoder, as mentioned in the training details for this experiment.

To observe the performance on a component level, two ablations were run. We paired an nCDE encoder with a Transformers decoder as well as an LSTM decoder. We note that when utilizing nCDE as an encoder in conjunction with the Transformer architecture, there was a slight decrease in performance compared to the pure Transformer model. However, the test MSE remained on the same order of magnitude. Additionally, the nCDE-Transformer model boasted half as many parameters, demonstrating the effectiveness of the nCDE approach in achieving comparable results with fewer parameters. A similar understanding was gained from coupling an nCDE with an LSTM decoder. These results suggest that an nCDE performs on par or better than SOTA encoders while requiring a fraction of the memory, while nODEs are not as capable as LSTM or Transformer decoders in sequence-to-sequence tasks.

Qualitatively, it was found that the Transformer and LSTM based models more accurately reflect the ground truth across the 20 classes. For simpler letters, such as “w” each model holds its own. However, for more difficult letters, such as “z”, in which artistic flair can be added to the end of the stroke, the nDE based decoder suffers, while the intricacies are detected by the Transformer model and neutralized by the LSTM model. Hence, there are clear benefits in self-attention for tasks of this nature. Additional figures are included in the Appendix.

5.3 Forecasting of Weather - Decoder Task

Our third evaluation of nDE models revolves around their ability to serve as decoders in an initial value problem involving forecasting natural dynamic processes. The results are seen in Table 4.

Table 3: Test MSE and model size in the forecasting of CharacterTrajectories (seq-seq) task.

Model	Test MSE	Parameters
Transformer	**3.2E-4**	838,307
LSTM	1.6E-3	401,283
Neural CDE-ODE	8.4E-3	**11,875**
Neural CDE-Transformer	5.7E-4	433,187
Neural CDE-LSTM	1.9E-3	344,451

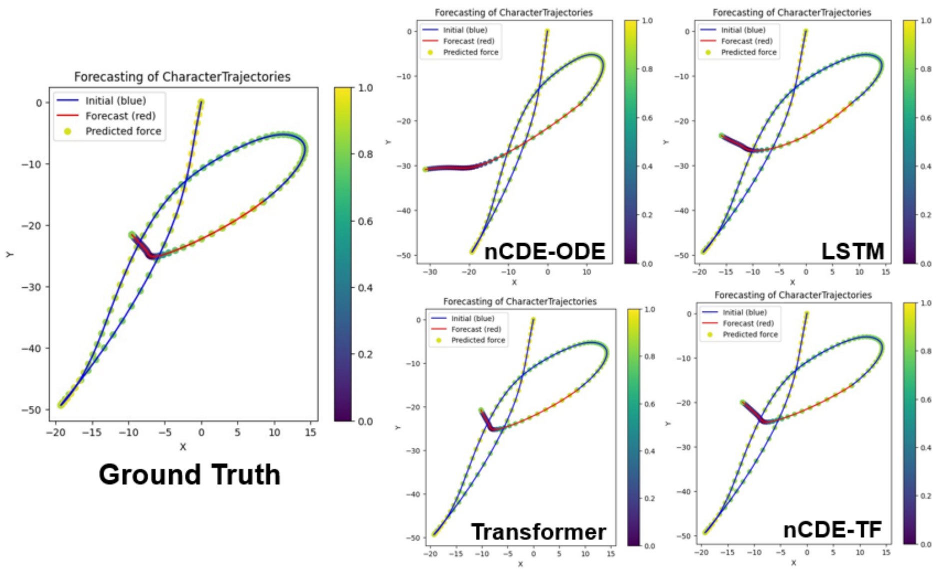


Figure 5: Demonstration of the forecasted trajectories (red) based on the ground truth observation provided (blue) of the letter "p" for the nCDE-ODE, LSTM, Transformer, and nCDE-Transformer models.

This task revolves around a very small dataset (n = 10), with moderate size sequence length (l = 26), and a bias towards continuous modelling since it is based on natural process. The circumstances favour the RNN based models, though all models perform somewhat comparably. The LSTM and RNN models achieve nearly equal and high performance on an unseen sample (2022 forecast), while the Transformer baseline struggles in both MSE and R2 correlation of the predicted temperature and humidity sequences. The nODE model assumes continuous priors on the data, which is actually favourable in this scenario. It achieves performance slightly below the RNN and LSTM models, though with 3x less parameters than the next highest baseline. Predictions from the models can be seen in Figure 6. Visually, the nODE achieves the best harmony between continuous prediction and correlation with the ground truth data. While the LSTM technically has the best MSE, the effects of the discrete representation of the data are apparent.

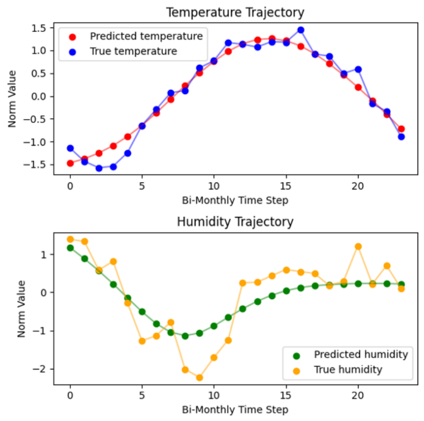
This result reinforces the findings in the sequence-to-sequence task in that nODEs are subpar as decoders compared to the baselines in this study. For long term dependencies, they are outperformed by self-attention based models, and for short term dependencies they seem to be outperformed by RNN derived models.

6 Discussion

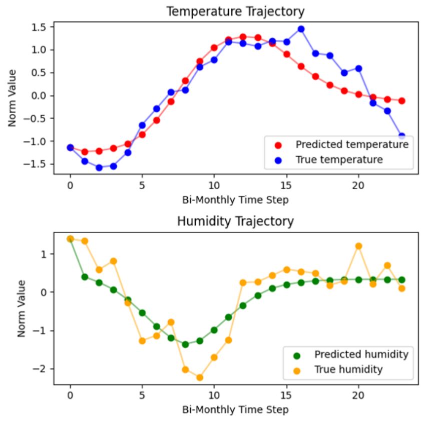
Our comprehensive comparison on nDEs against modern approaches to sequential data modelling is revealing in both the limitations, advantages, and interoperability of model families within the body of research. We outline a series of them below.

Table 4: Test MSE and model size in the forecasting of ERA5 (decoder) task across five runs.

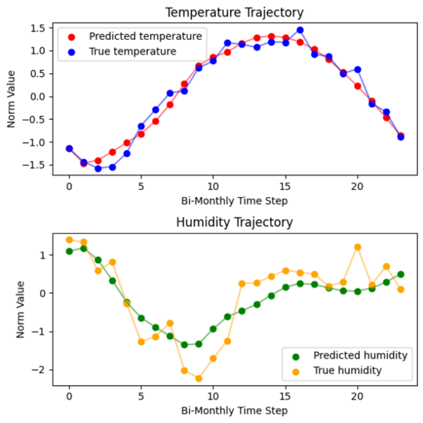
Model	**Test MSE**	**R2**	**Parameters**
Transformer	0.29 ± 0.05	0.71 ± 0.05	425,538
LSTM	$\mathbf{0.19 \pm 0.01}$	$\mathbf{0.81 \pm 0.02}$	13,122
RNN	0.20 ± 0.02	0.80 ± 0.02	3,330
Neural ODE	0.23 ± 0.04	0.76 ± 0.04	**1,186**



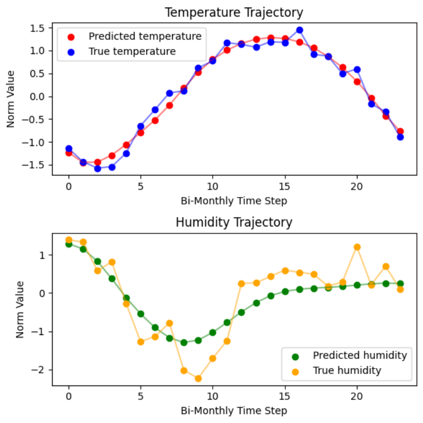
(a) nODE



(b) Transformer



(c) LSTM



(d) RNN

Figure 6: Forecasts predicted for the 2022 calendar year in downtown Montreal.

6.1 Limitations

Speed of Computation: Neural CDE and ODE based models were consistently much slower to train and infer from over the baseline models across all experiments. In training, the nDE models were roughly 4x slower than the LSTM, RNN, and Transformer models. One factor at play is that the implementations of both the nCDE and nODE are through the `torchdiffeq` library, which Kidger et al. claim to be inherently slow due to being constructed in Python, which uses double-precision arithmetic. This has since been updated in the `diffrax` library [24], however this only has compatibility with the JAX framework. Furthermore, a fixed-size step solver was used to speed up computation, though for more precise results a smaller or adaptive step size would be beneficial.

nODEs as Decoders: We find that nODEs in their natural state adapt poorer to the sequence-to-sequence and decoder tasks presented in this study than the baseline models. Uses in other applications such as normalizing flows [25] show that networks of nODE models connected in series is beneficial for modelling more flexibly, however this was computationally unfeasible in this study. We also find that nODEs were anecdotally more intricate to train stably, requiring a convergence-in-steps approach that guided the model away from getting trapped in local minima by only revealing progressive amounts of the full target sequence over time. Perhaps of much use in other applications, for ease of computation and performance in the designed experiments, it is more beneficial to use one of the baseline methods such as an LSTM or Transformer. The ablations in the sequence-to-sequence task show that using a nCDE as an encoder alongside either a Transformer or LSTM decoder results in minimal changes in performance, highlighting an effective path towards memory savings if desired.

nCDEs in Causal Real-Time Settings: The approximation of X to use as input to the nCDE is constructed using a cubic spline to ensure twice differeniability for the integration process. These splines are not causal, restricting the application of the nCDEs implemented in this study to non-real-time settings. While some workarounds exist, this natural limitation pushed us to restrict the use of nCDEs as encoders in our experiments.

6.2 Advantages

Continuous Representations of Time Series: Among all the baseline models considered, the nDEs were unique in that they modeled sequential data in a continuous forms rather than discrete representations. This has benefits across a variety of time series applications: natural physical processes, like our ERA5 forecasting experiment, applications with irregular sampling, like our CharacterTrajectories classification experiment, and applications with partially observed sampling, of which we do not directly study but is noted in [2].

Excel on Irregularly Sampled Data: nCDEs offer a more efficient approach to updating hidden states compared to discrete models when dealing with varying time intervals between observations. While discrete models allocate equal processing power to each time step, which can be inefficient for close intervals and inadequate for distant ones, nCDEs update continuously. The computational workload scales with the gap between observations, aligning more closely with the natural timescale of belief updates about the system. The benefits of the continual updates are seen foremost in our CharacterTrajectories classification task. The nCDE model outperformed each of the Transformer, LSTM, and RNN baselines, while demonstrating no performance decrease as data became more irregularly sampled, which cannot be said about the baseline models.

Memory Efficiency: What the nDE models suffer with in time cost, they make up for in memory cost. Adjoint backpropagation [2] allows each step to be backpropagated through individually only requiring the result and the underlying data to be stored in memory at once. Where an RNN requires $\mathcal{O}(HT)$, a nCDE only requires $\mathcal{O}(H + T)$, where H is the result of a single backpropagation step and T is the length of the time series [24]. This is evidently seen across all experiments.

As presented in the CharacterTrajectories classification experiment, if the task involves irregularly sampled data, nCDEs are without a doubt an excellent option as an encoder, both in performance and memory consumption. For the sequence-to-sequence and decoder evaluations, the argument could be made that in certain circumstances, sacrificing marginal performance in the test metrics is worth the memory savings of the nDE models. In the ablation studies, we show that nearly identical performance can be achieved using an nCDE as an encoder for LSTM and Transformer decoders - this is an insightful result in the case that memory is at a premium and researchers are looking for ways to make more efficient use of their resources.

Compatibility with Baselines: In our experiments we have demonstrated that not only are nDEs high performing and memory efficient, but that they also work well in conjunction with discrete models in an end-to-end configuration. While hybrid models like the ODE-RNN operate in a continuous hidden state, the sequence-to-sequence models we have shown operate in a continuous time state. This has the inherent benefits with irregularly sampled data, while not sacrificing any of the benefits of modelling long term dependencies with gated unit or self-attention based decoders, while cutting down drastically on overall memory consumption.

7 Conclusion

We have conducted a comprehensive comparison on a relatively quiet class of sequential models, nDEs. Over the course of a series of tailored tasks in sequential modelling, we have shown that the approach offered by these models, namely the nODE and nCDE, offers inherent benefits and advantages. Firstly, they can operate directly on irregularly sampled time series, exhibit state-of-the-art performance on classification of irregularly sampled time series, and perform near state-of-the-art on sequence-to-sequence and initial value forecasting tasks. Furthermore, nDEs benefit from a memory-efficient adjoint backpropagation structure that allows for competitive performance with a fraction of the trainable parameters. As a whole, we find that nDEs present a promising and high performing alternative to modern approaches in sequential modelling.

References

[1] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, 1987, pp. 318–362.

[2] P. Kidger, J. Morrill, J. Foster, and T. J. Lyons, "Neural controlled differential equations for irregular time series," *CoRR*, vol. abs/2005.08926, 2020. [Online]. Available: https://arxiv.org/abs/2005.08926

[3] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[4] Y. Rubanova, R. T. Q. Chen, and D. Duvenaud, "Latent odes for irregularly-sampled time series," *CoRR*, vol. abs/1907.03907, 2019. [Online]. Available: http://arxiv.org/abs/1907.03907

[5] E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau, "Gru-ode-bayes: Continuous modeling of sporadically-observed time series," *CoRR*, vol. abs/1905.12374, 2019. [Online]. Available: http://arxiv.org/abs/1905.12374

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] J. Sinai, "Understanding neural ode's," Jan 2019. [Online]. Available: https://jontysinai.github.io/jekyll/update/2019/01/18/understanding-neural-odes.html

[8] L. Leal, "Numerical interpolation: Natural cubic spline," Dec 2018. [Online]. Available: https://towardsdatascience.com/numerical-interpolation-natural-cubic-spline-52c1157b98ac

[9] M. ÇELİK, "What is translation invariance?" Dec 2022. [Online]. Available: https://celik-muhammed.medium.com/what-is-translation-invariance-3ec5fc4f3cd4

[10] J. Nabi, "Recurrent neural networks (rnns)," Jul 2019. [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85

[11] S. N. Shukla, "Deep learning models for irregularly sampled and incomplete time series," Oct 2021. [Online]. Available: https://scholarworks.umass.edu/dissertations_2/2349/

[12] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[13] C. Olah, "Understanding lstm networks," Aug 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[14] P. B. Weerakody, K. W. Wong, G. Wang, and W. Ela, "A review of irregular time series data handling with gated recurrent neural networks," *Neurocomputing*, vol. 441, pp. 161–178, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221003003

[15] G. Giacaglia, "Transformers," Mar 2019. [Online]. Available: https://towardsdatascience.com/transformers-141e32e69591

[16] Z. Li, S. Li, and X. Yan, "Time series as images: Vision transformer for irregularly sampled time series," 2023.

[17] S. Tipirneni and C. K. Reddy, "Self-supervised transformer for sparse and irregularly sampled multivariate clinical time-series," 2022.

[18] J. G. Wade, J. M. A. Pomarolli, and L. Poirier, "What are the advantages and disadvantages of using interpolation and extrapolation techniques?" Jan 2024. [Online]. Available: https://www.linkedin.com/advice/0/what-advantages-disadvantages-using-interpolation?lang=en

[19] T. Thangavel, “Limitations of transformer architecture,” Sep 2023. [Online]. Available: https://medium.com/@thirupathi.thangavel/limitations-of-transformer-architecture-4e6118cbf5a4

[20] B. Williams, “Character Trajectories,” UCI Machine Learning Repository, 2008, DOI: https://doi.org/10.24432/C58G7V.

[21] A. Simmons, C. Soci, J. Nicolas, B. Bell, P. Berrisford, R. Dragani, J. Flemming, L. Haimberger, S. Healy, H. Hersbach, A. Horányi, A. Inness, J. Munoz-Sabater, R. Radu, and D. Schepers, “Global stratospheric temperature bias and other stratospheric aspects of era5 and era5.1,” 01/2020 2020. [Online]. Available: https://www.ecmwf.int/node/19362

[22] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative study of cnn and rnn for natural language processing,” 2017.

[23] R. Deng, B. Chang, M. A. Brubaker, G. Mori, and A. M. Lehrmann, “Modeling continuous stochastic processes with dynamic normalizing flows,” *CoRR*, vol. abs/2002.10516, 2020. [Online]. Available: https://arxiv.org/abs/2002.10516

[24] P. Kidger, “On Neural Differential Equations,” Ph.D. dissertation, University of Oxford, 2021.

[25] D. Rezende and S. Mohamed, “Variational inference with normalizing flows,” in *International conference on machine learning.* PMLR, 2015, pp. 1530–1538.

Appendix

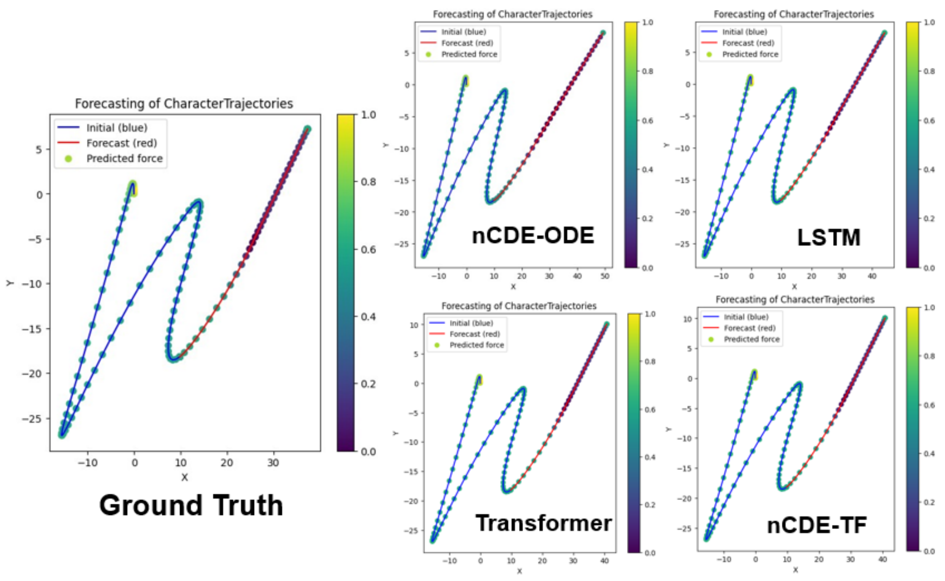


Figure 7: Demonstration of the forecasted trajectories (red) based on the ground truth observation provided (blue) of the letter "w" for the nCDE-ODE, LSTM, Transformer, and nCDE-Transformer models.

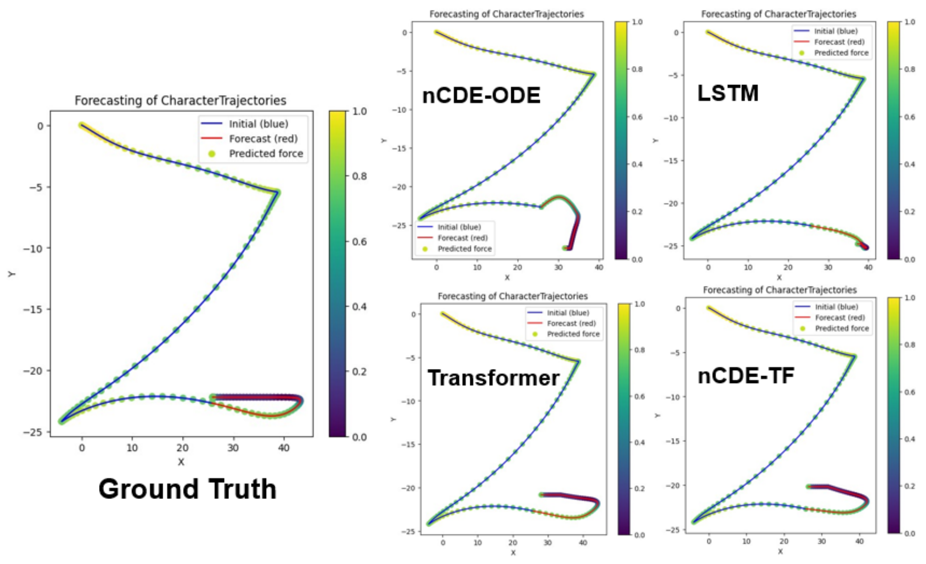


Figure 8: Demonstration of the forecasted trajectories (red) based on the ground truth observation provided (blue) of the letter "z" for the nCDE-ODE, LSTM, Transformer, and nCDE-Transformer models.