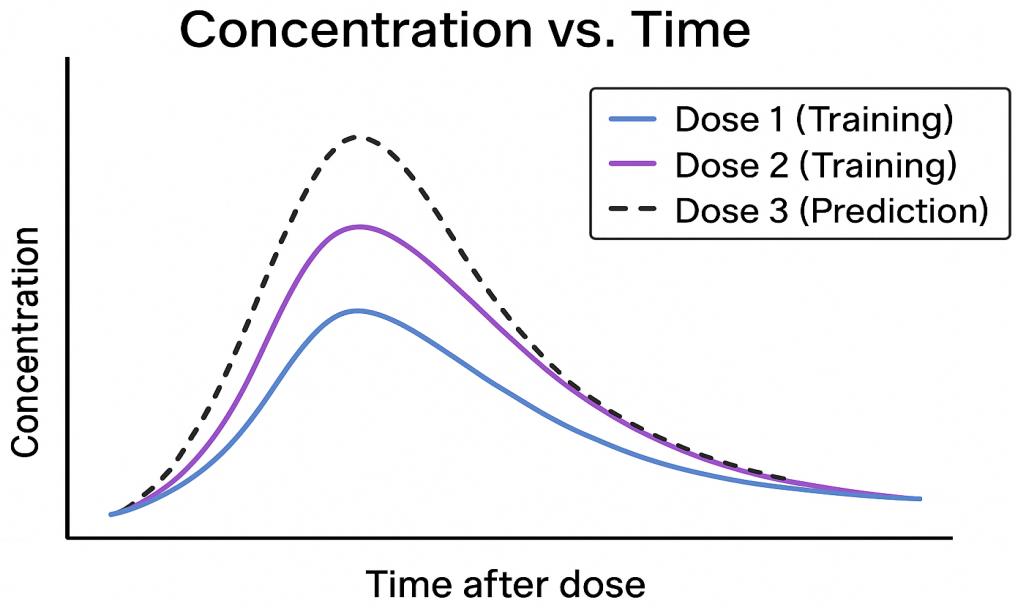



# Designing and Evaluating Neural Ordinary Differential Equation Models for Pharmacokinetic and Pharmacodynamic Time Series

Master's thesis in Computer science and engineering

Benjamin Olsson
Elias Torstensson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

# Designing and Evaluating Neural Ordinary Differential Equation Models for Pharmacokinetic and Pharmacodynamic Time Series

Benjamin Olsson
Elias Torstensson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Designing and Evaluating Neural Ordinary Differential Equation Models for Pharmacokinetic and Pharmacodynamic Time Series

Benjamin Olsson
Elias Torstensson



Supervisor: Morteza Haghir Chehreghani, CSE
Advisor: Mikael Sunnåker, AstraZeneca
Examiner: Simon Olsson, CSE

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover image displays an example of different possible PK curves for different dose sizes, it was generated using ChatGPT.

Typeset in LaTeX
Gothenburg, Sweden 2025

Designing and Evaluating Neural Ordinary Differential Equation Models for Pharmacokinetic and Pharmacodynamic Time Series

Benjamin Olsson & Elias Torstensson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

PKPD modeling is the study of the effect of drugs on living organisms. It combines modeling the concentration (PK) and effect (PD) of a drug over time. This is commonly done through systems of ordinary differential equations (ODEs), although there have been increasing efforts to utilize machine learning methods in the area. In this project, discriminative neural ODE models for PKPD data were developed and compared with other common neural network models. We simulated our train and test data using an underlying PKPD model, resulting in a ground truth set to accurately test the performance of the models. The models aimed to extrapolate as well as possible to dosing regimens not seen in the training data. We found that using neural ODEs, it is possible to accurately predict concentrations and responses resulting from dose sizes that are more than twice the size of those in the training data. This vastly outperforms the other tested models. Furthermore, using the flow matching algorithm, the training time of the neural ODE model was substantially reduced. The project also investigated whether using a neural ODE as a component in a Generative PKPD model yielded any benefits. It was found that this significantly improved the quality of the samples by more closely reflecting the underlying distribution of PKPD time series. The ability of the discriminative neural ODE models to capture information about the PKPD model used to generate the model was then investigated by assessing their ability to predict uncertainty in the underlying data, as well as their capacity to identify which covariates were relevant to the underlying PKPD model. Finally, a neural ODE model was developed for modeling adverse effects, to test how well the neural ODE performed when the data was not generated using an underlying ODE. This model demonstrated excellent performance in predicting the proportion of patients who experience nausea based on the rate of dose increase over time.

Keywords: Neural ODEs, Machine Learning, PKPD Modeling, Modeling, Adverse Effects Modeling

# Acknowledgements

Firstly, we would like to thank our academic supervisor Morteza Haghir Chehreghani for the support and valuable insights throughout the course of the project. Secondly. we want to thank Mikael Sunnåker, our supervisor at AstraZeneca, who has helped us understand PKPD modeling and made us feel at home at AstraZeneca. We would also like to thank the entirety of the CVRM and R&I teams at AstraZeneca, who contributed to a great environment to work on the thesis, along with our examiner Simon Olsson who gave us valuable feedback on how to effectively carry out independent research. Finally, we would like to thank our friends and family who supported us over the duration of the project.

Benjamin Olsson & Elias Torstensson, Gothenburg, 2025-07-02

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

In this master's thesis, we investigate the effectiveness of using neural ordinary differential equations (neural ODEs) [1] for performing pharmacokinetic and pharmacodynamic (PKPD) modeling. The first part of the project compares the performance of neural ODEs to multilayer perceptrons and recurrent neural networks as discriminative models. The MLP was chosen as a baseline for comparison, while the RNN was selected because it is commonly used in sequence processing, including time series analysis. Transformers were also considered, but were ultimately discarded because the complexity of their architecture meant that the number of parameters in the transformer was far larger than in the other models, which complicated making an objective comparison. The second part of the project focuses on building and comparing two diffusion models for PKPD time series, one of which uses a neural ODE in the encoder and the other uses a simpler autoencoder without a neural ODE. The third part of the project involves applying the flow matching technique, which is used for continuous normalizing flows, to our neural ODE time series model, resulting in significantly faster training times. In the fourth part of our project, the neural ODE model is used to recover information from the underlying PKPD model by attempting to quantify the model's uncertainty and using SHAP analysis to determine the impacts of covariates. In the final part of the project, neural ODEs are used to develop a model for the closely related topic of Adverse Effects.

In this project, all machine learning models were developed by the authors. The architectures of the models drew inspiration from works such as [2], yet they contain significant differences. The models were implemented using Python packages such as *PyTorch* and *torchdiffeq*, allowing for easy implementation of neural networks and neural ODEs. The models used to generate the data were taken from literature. The training methods for the models were either already implemented in the libraries, or taken from literature, in the case of the flow matching for trajectories method. Many common machine learning techniques were used throughout the project, such as SHAP-values, DDP, and SWAG, in these cases the authors merely applied the techniques for our particular models, with the help of relevant, preexisting Python libraries. The technique for going to a closed-form ODE was designed and implemented by the authors, with inspiration taken from [3].

## 1.1 Theoretical Background

### 1.1.1 PKPD Modeling

PKPD modeling is a fundamental aspect of pharmacology, which is the study of the effect of drugs on living organisms. It combines pharmacokinetics and pharmacodynamics, each investigating different properties of a drug. In pharmacokinetics, one studies the effects the body has on the concentration of a drug. This is typically summarized as "what the body does to the drug." In pharmacodynamics, one instead looks at what effect the drug has on some *biomarker* in the body, frequently summarized as "what the drug does to the body". A biomarker is a measurable entity in the body, such as a molecule or a cell, that gives information about the state of a process in the body. One can either study just the relationship between biomarker response and concentration, or the relationship between biomarker response and time [4].

In pharmacokinetics, the most widely used type of model is the compartmental model. In this model, the body is represented by a number of different compartments. The model then describes the interaction between these compartments using a system of differential equations. The number of compartments in the model depends on the underlying assumptions that can be made about the drug, which are related to the type of drug being examined. In one-compartment models, there is only the central compartment, in which the drug is assumed to instantly distribute evenly. While not always realistic, this type of model still has numerous practical applications. For more complex interactions, one can add peripheral compartments to create two-compartment or three-compartment models. However, the increased complexity brings along other potential issues, such as a requirement for larger amounts of data or computational difficulties [5].

Pharmacodynamic models, which seek to describe the relationship between drug effect and concentration, come in many forms. Some common examples include the linear model, the log-linear model, and the $E_{max}$ model. In these models, there is usually an underlying assumption that the system is in a steady state, which occurs when the amount of the drug eliminated within each dosing interval equals the increase from the dose. This means that the peak concentrations after each successive dose will be the same from that point forward. Despite this assumption, the models also perform well when modeling a system that is not in a steady state. In the linear model, a direct proportionality between the drug concentration and drug effect is assumed. While this assumption may seem intuitive, it is not always the case in practice. Instead, the log-linear model can sometimes prove more useful, where the effect is instead proportional to the logarithm of the concentration. Lastly, the $E_{max}$-model describes the relationship using the maximal effect $E_{max}$, the concentration value that causes 50% of the maximal effect $EC_{50}$, and some baseline effect $E_0$. This relationship is given by the equation $E = E_0 + \frac{E_{max} \cdot C}{EC_{50}+C}$ [4] [5].

For non-steady-state situations, integrated PKPD models can be used. Such a model has some key characteristics that describe what one assumes about the un-

derlying process. In this thesis, one such characteristic is worth mentioning: Direct response vs Indirect response [5].

*Direct/Indirect response*: Direct response models assume a direct correlation between concentration at the effect site and the observed effect of the drug. This means that there is no delay between the high concentration at the effect site and the observed effect. The indirect response model instead uses an ODE to describe the relationship between the effect and the concentration at the effect site [5].

Creating these types of models is a crucial part of the drug development process, as they are used to understand how the body processes the drug and how effective it is at causing a response in the body. One must balance the efficacy of the drug with its safety, both of which may heavily depend on the concentration of the drug. Using limited information from strictly controlled medical trials, pharmacometricians must piece together a mechanistic understanding of how exactly the body processes the drug. These findings can then be used to find dosing regimens that are as safe and effective as possible. This is a difficult, expensive, and time-consuming task. Recently, considerable effort has been devoted to accelerating this process through machine learning. [6] [7] [8] [9] [10]. In particular, neural networks have emerged as a promising candidate to model this data due to their flexibility as universal function approximators [11] [12] [13].

### 1.1.2 Adverse Effects modeling

The topic of modeling adverse effects, such as Nausea and Vomiting, has seen a great increase in popularity recently. These models are crucial in ensuring that patients receive the correct dose amount at the proper time. Typically, the drug of interest follows a titration scheme, wherein the dose size starts relatively small and then increases over time. The ultimate goal is to design a titration scheme that achieves the targeted effect as quickly as possible, while minimizing the risk of adverse effects. By starting with a small dose that increases over time, the patient can build up some resistance to the drug. This results in a lower risk of adverse effects compared to if the patient had taken the final target dose immediately.

The goal of the models is to predict the proportion of patients who experience nausea each week, given a specified titration scheme. If accurate, these models can provide valuable insights into selecting effective titration schemes. Currently, commonly used models include the Proportional Odds Models and different types of Discrete Markov Chain Models [14]. While these models are effective, they lack the same mechanistic understanding as the PKPD models. This could mean that more bias is introduced than for the PKPD models.

This means using a neural ODE model for this type of problem could be of significant interest from a drug development point of view. From a theoretical point of view, it is interesting to compare the challenges of using neural ODEs on problems with limited mechanistic understanding to those with a thorough understanding. By exploring this, we can determine whether the lack of knowledge makes modeling more challenging, and if the development of a Neural ODE model could provide some assistance in understanding the mechanics where knowledge is lacking. Additionally, we are curious to see if the performance of the neural ODE is affected by using a Markov model for data generation, compared to the systems of ODEs used in generating PKPD data.

### 1.1.3 Artificial Neural Networks

Artificial neural networks (ANNs) are a class of mathematical models that are used to approximate complex, nonlinear functions. Although their theoretical foundations are quite old, it is only in the past couple of decades that they have become ubiquitous in technology and science, as we now have the computing power and data requirements necessary to train them quickly and effectively. We begin by describing the simplest type of artificial neural network (ANN), known as a feedforward neural network, also referred to as a multilayer perceptron (MLP).

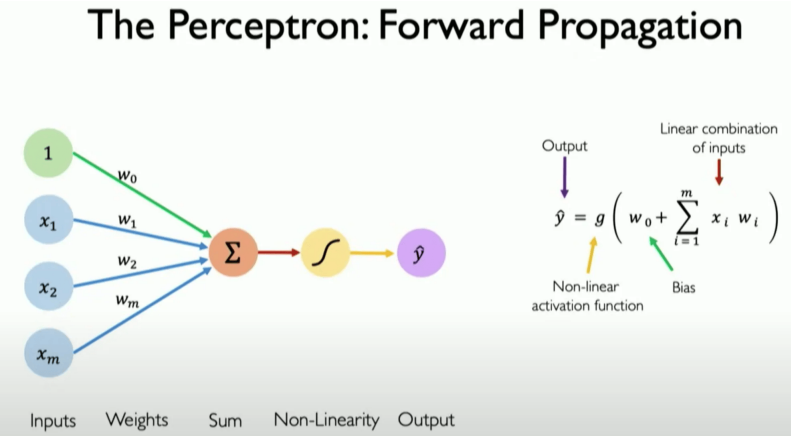



Figure 1.1: Schematic of a perceptron, taken from the MIT Deep Learning Course [15].

#### 1.1.3.1 Multilayer Perceptrons

The building block of an MLP is the neuron, a simple unit that takes one or more inputs, each with a corresponding weight, and sums the product of these weights and inputs together, along with a scalar that is not multiplied by any input, known as the bias. This sum is then put through an activation function, which is typically non-linear, allowing the neural network to approximate non-linear functions. A layer consists of a parallel group of one or more neurons, each receiving the same inputs but each having unique weights corresponding to each input. Finally, the MLP

itself consists of at least three layers, a single input layer taking the original inputs, one or more hidden layers through which the inputs are transformed, and a single output layer giving a final prediction, which is often either a point estimate of some numerical value or the probabilities of classification into a number of classes [16]. Although the MLP is conceptually quite simple, when used in practice, with many thousands, millions, or even billions of parameters, it becomes a universal function approximator. This means it can mimic any function to an arbitrary degree of precision, provided a sufficiently large network is used.

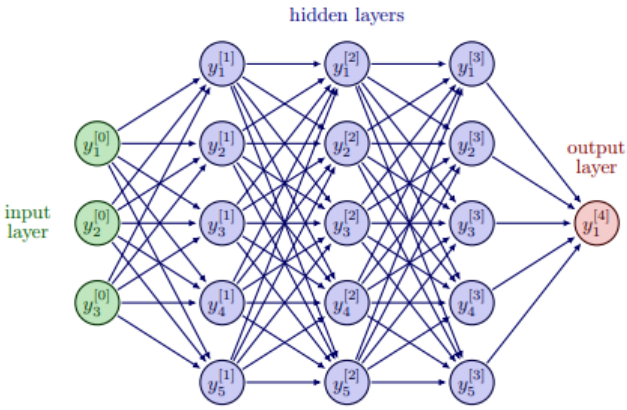



Figure 1.2: Schematic of a MLP [16]

#### 1.1.3.2 Recurrent Neural Networks

Recurrent neural networks are among the most commonly used neural network architectures for sequence prediction. The main difference between a recurrent neural network and an MLP is that there is a hidden state that changes as each data point in the time series is processed. This hidden state encodes information from previous points in the time series, allowing the network to have a "memory" [17]. However, it is important to note that in standard RNNs this memory is only updated discretely, at each time that a data point is observed.

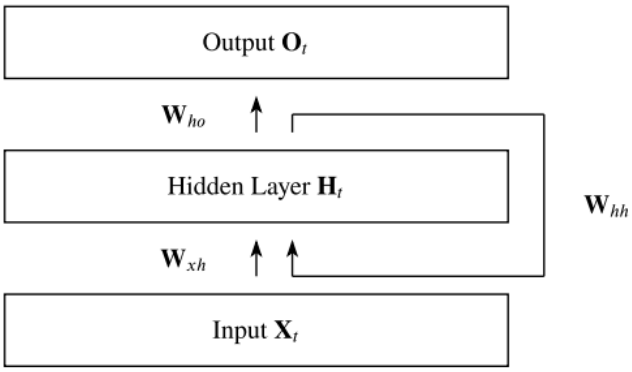



Figure 1.3: Schematic of a RNN [17]

#### 1.1.3.3 Neural ODEs

Neural ODEs represent a novel class of neural networks that utilize a neural network to produce derivatives and then employ an ODE solver to integrate and generate the final predictions [1].

Recall that the general form of a first-order ordinary differential equation is given by $\frac{dy}{dt} = f(y(t), t)$. This equation describes a function $y(t)$ by defining the slope of the function at all points. Differential equations are currently used to model many natural phenomena, from the decay of radioactive atoms to the mechanics of solids. Generally, this approach to modeling is effective, but it has a drawback. How do you "know" what the function $f(y, t)$ looks like? This is something that, in general, is not at all trivial. It is indeed a challenging task for a scientist to transition from experiments to a concise, closed-form differential equation, especially without introducing human bias. This is where neural networks and machine learning can be helpful. Given that neural networks are universal function approximators, it can be reasoned that given enough data, a neural network could approximate the function $f(y, t)$ reasonably well. And so the idea behind neural ODEs is to let the right-hand side be a neural network instead of some explicit function. This can be written as

$$\frac{dy}{dt} = f_\theta(y(t), t),$$

where the function $f_\theta$ is a neural network with parameters $\theta$.

Once the neural network is trained, the function $y(t)$ can be solved for by using a numerical method. Note that we will see the neural network as a complete black-box for the derivative of our function, and therefore there won't be an analytical expression for the function $y(t)$. This is one of the downsides of using neural ODEs, and we discuss it further in later parts of the thesis.

The neural ODE model has demonstrated advantages in memory efficiency and the ability to adapt computation based on problem complexity, making it suitable for time-series modeling, such as PKPD data in clinical trials. Another important advantage of neural ODEs is that they can model irregularly-sampled time series data much more effectively than a traditional recurrent neural network (RNN) can [18]. This is because the neural ODE has a hidden state that can be updated "continuously" by an ODE solver, which leads to more accurate predictions when the time series is irregularly-sampled, i.e, the time between each observation is not always the same. Strictly speaking, the updates are still discrete, but we can specify our step size to be as small or large as we want, making the updates to our hidden state as smooth as we want. Irregularly sampled time series are common in medical data, which is why the neural ODE will be applied in this case.

Another advantage of neural ODEs is that they closely resemble a method used by pharmacometricians, which involves developing PKPD models as systems of differential equations. This method allows the pharmacometricians to incorporate their mechanistic understanding of the system into the model. These systems are then

solved numerically, rather than trying to find closed-form solutions for the concentration or effect directly. There have already been several promising attempts at applying neural ODEs to pharmacokinetic data [2] [19][20]. In Lu et al., the model is developed for use in oncology, where PKPD data from a specific patient are available after the first treatment. This data is then used to predict how that specific patient will respond to a different dosing regimen. We seek to train a model that can generate the mean response of a population of patients to a certain dosing regimen given only informative covariates such as weight, age, and sex. A lot of focus will be on the ability of the model to extrapolate well to dosing regimens that it has not seen in the training data. In this way, our model can be used to identify the optimal dosing regimens for testing in Phase II.

### 1.1.4 Flow Matching

Neural ODEs are still not widely used in practice, largely because they require an extremely long training time. In recent years they are now being used more frequently, and this is mainly because of a new paradigm for training a group of generative models called Continuous Normalizing Flows (CNFs), this paradigm is known as flow matching [1] [21].

CNFs are a group of generative models that attempt to model a target probability distribution by transforming a simple starting probability distribution, such as a Gaussian distribution, into an approximation of the target distribution. The CNF models a vector field, which can be solved using an ODE solver. This results in a probability path that transforms a sample from the simple probability distribution into an approximation of a sample from the complex target distribution. Prior to the introduction of flow matching, CNFs were trained using maximum likelihood, which required expensive numerical ODE simulations at training time. Not only must one solve the entire trajectory for each batch, but one must also perform backpropagation through the entire trajectory. This results in lengthy training times that do not scale well as the size of the training data increases. However, flow matching is a simulation-free training technique that avoids the cost of solving an ODE at training time. Instead, one regresses directly over the vector field on a per-training-sample basis, which eliminates the need for ODE simulations. Training CNFs with flow matching has now been applied in multiple different fields, e.g, for generating 3D molecules [22].

One of the key problems of our thesis was applying this idea in our deterministic time series setting in a way that resulted in predictive performance comparable to when the neural ODE is trained through simulations.

### 1.1.5 Diffusion Models

In recent years, the area of generative models has been dominated by diffusion models. This is in particular due to the introduction of the Denoising Diffusion Probabilistic Model (DDPM) [23]. While this model has been most effectively applied in the domain of image generation, it has also achieved significant success in

time series synthesis [24][25]. We will therefore explore the use of this state-of-the-art model in our specific domain of PKPD time series.

A diffusion model consists of two parts: The forward process and the backward process. In the forward process, noise is sequentially added to the training samples according to a predefined schedule of variances $\{\beta_h\}_1^H$. One refers to each step of adding noise as a state, and we let H denote the final state. Ultimately, given a sufficiently large number of states and noise, this will result in Gaussian noise with a mean of 0 and a variance of 1. Let $\mathbf{e}_0$ denote our original sample, and let $\mathbf{e}_h$ and $\mathbf{e}_{h-1}$ be the noised sample at state $h$ and state $h-1$, respectively. Then the forward process can be written as

$$p(\mathbf{e}_h \mid \mathbf{e}_{h-1}) = \mathcal{N}(\mathbf{e}_h; \sqrt{1-\beta_h}\mathbf{e}_h, \beta_h\mathbf{I}).$$

Because the added noise is Gaussian, we can get a closed-form expression for the distribution of the forward process at any state h given the original state $\mathbf{e}_0$. This significantly simplifies the implementation of the forward process. To perform the forward process until state $h$, we do not have to sample once for every state. Instead, we can sample just a single time, using the distribution given by

$$p(\mathbf{e}_h \mid \mathbf{e}_0) = \mathcal{N}(\mathbf{e}_h; \sqrt{\bar{\alpha}_h}\mathbf{e}_h, (1-\bar{\alpha}_h)\mathbf{I}),$$

where $\bar{\alpha}_h = \prod_{s=1}^{h}(1-\beta_s)$ and $\mathbf{I}$ is an identity matrix of the same dimension as the sample. One can easily verify that as $h$ increases, the probability distribution indeed converges to a multivariate normal distribution with mean 0 and an identity covariance matrix.

The backward process does the opposite, going from pure noise to a realistic sample. Unlike the forward process, the backward process is not predetermined. It is instead parameterized by a neural network that estimates the amount of noise in a sample at any state $h$. We can then sequentially subtract the estimated noise from the sample at each step in the backward process, thereby going from pure noise to a denoised sample from the process. The neural network is trained by first sampling the forward process at a uniformly distributed state, providing this as input to the neural network, and then performing gradient descent with respect to the mean squared error between the predicted noise level and the actual noise added in the forward process.

## 1.2 Constraints

Due to the confidentiality of AstraZeneca's actual PKPD patient data, it is out of the scope of the project to evaluate the performance of the model on real-world data. We instead use *mrgsolve* to simulate data in pharmacology. Having access to the underlying model allows us to easily test how well our different models can extrapolate to new dosing regimes.

## 1.3 Research Questions and Hypotheses

The research component of the project involves exploring the viability of neural ODE-based discriminative and generative models in comparison to other common models used for similar tasks. In this case, it will be tested on PKPD time series data, where it is hypothesized that the structure of neural ODEs will allow for better performance compared to other models. The central question in the project is whether the properties of neural ODEs actually help in practice, and whether it is worth the increased computational cost associated with neural ODEs.

Another important research question we are interested in exploring is whether we can improve the computational efficiency of our neural ODE models without losing predictive performance. Here we hypothesize that we will be able to improve the efficiency of the neural ODEs using the flow matching technique, which has recently been shown to be applicable to training neural ODEs for deterministically mapping paired data as well as for predicting trajectories in clinical data [26], [27]. It allows for the training of a neural ODE without requiring computationally expensive backpropagation through the ODE solver. We investigate whether these techniques can be applied to our task to train the neural ODEs in a manner that is approximately as fast as standard neural networks of similar complexity. We provide a succinct list of the research questions below:

1. How does the performance of neural ODEs compare to multilayer perceptrons and RNNs for discriminative modeling of PKPD data?
2. Does the inclusion of a neural ODE component as part of a generative diffusion model increase the quality of the generated PKPD samples?
3. Can flow matching reduce the training time of the neural ODE model without significantly reducing its performance?
4. Can our trained neural ODE model extract accurate information about the underlying PKPD model used to generate the data?
5. Can neural ODEs predict time series accurately when the data is generated by a Markov model instead of a system of ODEs?

## 1.4 Ethics

We have the following ethical concerns to consider. Great care must be taken to ensure that the data used for training the model is used with the consent of the patients, as medical records are sensitive data. Initially, these models will be trained and evaluated solely on synthetic data. As such, there is no risk here of using any person's data in a manner they did not consent to. However, at a later stage, these models could be used on real clinical data. At that point, AstraZeneca's strict policies in the handling of personal data would be followed, ensuring that the data is used responsibly.

Whenever one develops a model, it is important to think about the biases introduced by the dataset. If the model is to be applied to real-world data, it is crucial to ensure that the underlying data accurately represents the population for which the model will be used. This is not unique to our model, instead it is common practice in the development of PKPD models. Therefore, the underlying data can be expected to represent the target population, and as such, the model does not present a risk of inherent bias from the training data.

# 2

# Methods

## 2.1 The Data Generating Process

This thesis primarily focuses on comparing the performance of neural ODEs with other common deep learning models in terms of PKPD data. One way to perform such a comparison would be to use real-life PKPD data, as even the best simulations cannot capture the complexity of a real-world data-generating process. Unfortunately, due to patient confidentiality concerns, such a dataset is not readily available for use in comparing the performance of these models. Thus, we resorted to simulating PKPD data using *mrgsolve*, an R package designed for simulating hierarchical ODE systems. However, in order to make this simulated data similar to what is commonly found in the real world, we sampled from it relatively sparsely, taking only 35 points from a time series consisting of over 4000 points. Furthermore, the data is sampled in a somewhat irregular fashion, with the time between points being either four hours or twenty hours. This was done to capture the peaks and troughs of drug concentration, and it aligns with how this data is collected in the real world, as this sampling provides the most information about the system's dynamics. Furthermore, it is most realistic to sample at these points, as they occur immediately before and immediately after a dose has been administered. The data was also generated with inter-individual variability, so that no individuals metabolize the drugs in the same way, even when all meaningful covariates are identical.

Our method of generating the data raises some concerns from a scientific standpoint in relation to our project. Most importantly, we have to consider whether generating the data in this way inherently biases our experiment in favor of neural ODEs, as they generate their predictions with an ODE solver in a very similar way to how we simulate our data. We believe this is mitigated by the fact that our data is quite sparsely sampled, and thus the smoothness of the solution resulting from solving the ODE is not present in the data when trained upon. Furthermore, the project investigates the performance of a model that utilizes neural ODEs but has not been tested on data generated by ODEs.

Another simplification in our data is that we have the same times sampled in every time series, which is unrealistic. When real clinical studies are performed, there is naturally a window of 1-2 hours during which the measurements can be performed. However, this simplification does not matter for drugs that are not fast-acting, as the

concentration can be expected to be about the same within this window. This is because the samples are taken right before and right after a dose is administered, which results in the relatively flat peaks and troughs of the concentration and effect curves.

There is also a benefit to using simulated data, as we will be able to generate as much testing data as we want in order to compare the performance of our models. Furthermore, we can see how well the models capture the underlying dynamics of the system by training them on data where only a certain subset of the covariates matter, and then investigating if the model picks up on this.

### 2.1.1 The Underlying PKPD Model

The PKPD model used to generate the data is an indirect response model with one-compartment PK. Because there is only one compartment, this is a relatively simple type of PKPD model. This model is made slightly more complex because it is aimed at oral dosage, which requires the addition of a simple gut compartment through which the dose must pass before reaching the central compartment. Note that because the drug can not go from the central compartment to the gut compartment, this still classifies as a one-compartment model. The model originates from [28], with slight modifications to incorporate three covariates: sex, weight, and age into the PK equations. These covariates are not included in the equation for response, in order to simplify the task of the model. A rigorous formulation of the model is available below:

**Definitions and Initializations:**

$$\mathrm{CP} = \frac{\mathrm{CENT}}{\mathrm{VCi}}$$

$$\mathrm{INH} = \frac{\mathrm{CP}}{\mathrm{CP} + \mathrm{IC50}}$$

$$\mathrm{RESPONSE}_0 = \frac{\mathrm{KIN}}{\mathrm{KOUT}}$$

$$\mathrm{CLi} = \mathrm{CL} \cdot e^{\eta_1} \cdot \left(\frac{\mathrm{WT}}{70}\right)^{0.75} \cdot \left(\frac{\mathrm{AGE}}{40}\right)^{0.8}$$

$$\mathrm{VCi} = \mathrm{VC} \cdot e^{\eta_2} \cdot (1 + 0.2 \cdot \mathrm{SEX}) \cdot \frac{\mathrm{WT}}{70} \cdot \frac{\mathrm{AGE}}{40}$$

**Differential Equations:**

$$\frac{dGUT}{dt} = -\text{KA} \cdot \text{GUT}$$

$$\frac{dCENT}{dt} = \text{KA} \cdot \text{GUT} - \frac{\text{CLi}}{\text{VCi}} \cdot \text{CENT}$$

$$\frac{dRESPONSE}{dt} = \text{KIN} \cdot (1 - \text{INH}) - \text{KOUT} \cdot \text{RESPONSE}$$

CP denotes the concentration of the drug in the central compartment, CENT denotes the total amount of the drug in the central compartment, VC is the volume of distribution, the size of the space the drug will expand into. CL is the clearance, which refers to how quickly the drug is removed from the body. $\text{CL}_i$ and $\text{VC}_i$ denote an individual's clearance and volume of distribution, each of which is determined by the covariates and a random variable. GUT denotes the amount of the drug present in the gut. KA is the reaction rate coefficient, which describes how quickly the drug leaves the gut and enters the body. RESPONSE shows the decrease in the amount of some unnamed biomarker, which is the value studied in PD. $\text{RESPONSE}_0$ is the baseline level of the biomarker. KIN is the baseline rate at which the biomarker increases. KOUT is the base rate at which the biomarker decreases. INH is the inhibition variable, which determines how the drug affects the biomarker.

A dosing regimen defines the size of the dose, the number of doses to be taken, and the time between each dose. The dosing regimens in the current dataset consist of 5 doses of 100 mg, 10 doses of 50 mg, and no doses at all. This is consistent with the variability that would be found in a real medical trial, where two or three proper dosing regimens are typically tested, along with a placebo group. For each of the dosing regimens, the time between the doses is 24 hours.

Note that because we use a PKPD model, the simulations output two types of output: PK and PD. An example of the generated PK data for the three different dosing regimens is shown in Figure 2.1 and the corresponding PD plots can be seen in Figure 2.2

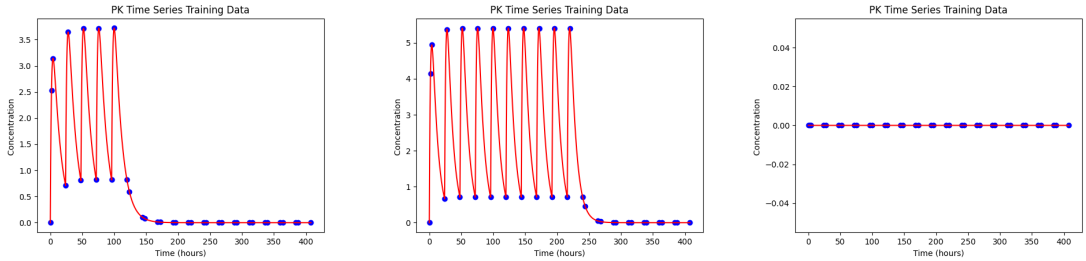



Figure 2.1: PK plots for the three different dosing regimes used as training data. The red lines are the full simulated data while the blue dots are the training points

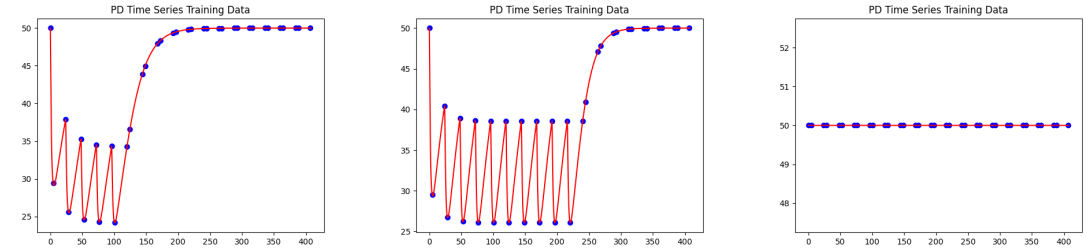



Figure 2.2: PD plots for the three different dosing regimes used as training data. The red lines are the full simulated data while the blue dots are the training points

## 2.2 Discriminative Models

### 2.2.1 Models for PK Time Series

The first architectures tested were discriminative models, i.e., models that only seek to predict the expected value of the time series based on the dosing regime and patient covariates. The input into the model is the dosing regimen and information about the patient, and then it is asked to predict the concentration of the drug in the blood at certain times. In this project, three models have been tested: A neural ODE model, a multilayer perceptron, and an RNN model. Details for the architectures of these three models is available in Appendix D.

The most significant difference between the models, and what is central to our research question, is not their architectures but rather how they build their solutions, which determines what exactly they are modeling. The neural ODE builds its solutions using an ODE solver, so that it outputs a prediction for the derivative at a certain time, which is then used by an ODE solver to build a trajectory. The concentration is updated iteratively, with this updated concentration being used for predicting the derivative at the next time step. Importantly, the neural ODE does not require sampled data points in order to update its trajectory, thus it can build its trajectory as smoothly as needed. The RNN builds its solutions by modeling the change in concentrations between sampled data points, thus it also iterative, but updates its trajectory much less smoothly. Finally the MLP does not build its solution iteratively, but only outputs a predicted concentration given certain inputs. The difference in what the different models are trying to solve is demonstrated in

the following equations:

$$C_{t+1} = f_\theta(C_t, X, t+1)$$
$$C_{t+1} = C_t + f_\theta(C_t, X, t+1)$$
$$\frac{dC}{dt} = f_\theta(C(t), X, t)$$

From these equations we expect that the neural ODE will perform best, followed by the RNN and then the MLP. This is because the underlying PKPD model has relatively simple dynamics that can be written in closed form, but when integrated these dynamics become very complex and can only be calculated using an ODE solver. Thus a model that tries to approximate the dynamics, rather than the concentrations directly, will be more effective.

The models are trained on the three dosing regimens described in 2.1.1, with the size of the dataset consisting of 100 patients for each dosing regimen. They are trained on 75 patients from each regimen, and validated on 25 patients from each regimen.

### 2.2.2 Models for PD Time Series

The architecture and inputs for our PD models are identical to those of our PK models; the only difference between them is that we train using the PD data rather than the PK data. Additionally, the hypothesis is also the same for the PD models, i.e. that the neural ODE will perform best, followed by the RNN and then the MLP, for the same reasons explained above.

## 2.3 Generative Models

We have also developed generative models for the same dataset. The goal of such models is to sample realistic PKPD time series for a given dosing regimen from a specified population. As it stands, the current diffusion models could have one specific use case in particular, namely, uncertainty quantification. Because generative modeling, at its core, is about modeling a probability distribution, one can use this approach to model uncertainty and perhaps find out the underlying randomness in the data. In this project, however, the focus is merely on the theoretical aspect of comparing a neural ODE-based model to a model without a neural ODE component. The general idea behind the architectures of the models comes from [24], although their specific implementation was done manually.

The architecture of the first model, called "the neural ODE model", can be summarized into four parts: 1). An encoder which encodes the covariates into a fixed-dimensioned representation. 2) A neural ODE that takes the information from the encoding and simulates a trajectory in a latent space. 3) A decoder that translates the trajectory into a PKPD time series. 4) A diffusion model that can sample encodings.

The second model, referred to as "the Autoencoder", has a similar structure, but instead of creating a latent trajectory using a neural ODE, the encoded covariates go straight into the decoder. To compensate for this loss of complexity and expressiveness, the encoder and decoder are slightly more complex. The complexity was not increased by a lot, as that turned out to make the resulting sample quality much worse. An overview of the model is presented in Figure 8.3. Further details on the architectures of both models are available in Appendix D.

It is hypothesized that using the neural ODE to solve a trajectory in the latent space will give the latent space a smoother structure that leads to higher quality samples after being decoded.

These two models are trained by first viewing the non-diffusion components as autoencoders and training them on the dataset as normal. Once the networks have learned a good mapping between encodings and time series, the diffusion models are trained on an encoded version of the dataset. The goal of the diffusion model is to transform a pure noise encoding into a more reasonable time series representation. This is achieved by training a noise estimator network, which, for a given encoding, returns a predicted amount of noise in each dimension. The noise estimator is trained by adding noise to examples of real encodings and then allowing the network to predict the amount of noise present. Using the mean squared error between the predicted noise and the actual noise gives us a loss function for which we can train the network.

Once the models are fully trained, a sample is generated by first sampling an encoding that is pure noise, then denoising the sample using the noise estimator (by subtracting the estimated noise from the sample), and finally decoding the result into a time series.

The dataset used to train the models contains data from two different patient cohorts, which received different dose sizes and varying numbers of doses. In the first cohort, patients receive 5 doses of 100mg. In the second cohort, patients receive 10 doses of 50mg. The time between each dose is 24 hours in both cohorts. Note that these models are not trained on the 0-dose dosing regimen like the discriminative models were.

As noted earlier, the number of parameters and training times of the two models are quite similar. More importantly, when training the autoencoder, their final losses are close, with a less than 10%

## 2.4 Flow Matching for PKPD Time Series

The architecture for the flow matching models was modified from the first neural ODE model, see Appendix D for further details. In order to use the flow matching techniques on our data, we used results from [27]. In particular, we modified Algorithm 1 in this paper, which describes how to apply flow matching to time series.

With flow matching, backpropagating through the solver is entirely removed, although to validate the models performance the full trajectories do have to be solved for at certain intervals, in this case every 100 epochs. However, solving this is not very time consuming, as the entire batch can be solved for at once, as the results will not be used for training. The modified flow matching training algorithm is defined as follows:

**Algorithm 1** Flow Matching for One-Dimensional Time Series

**Require:** a set of time series spanning the same time period, a chosen step size for the ODE solver, $s$, a chosen number of epochs $n$, a neural ODE model $M$

1: **for** $i \leftarrow 1$ to $n$ **do**
2: Create a set $\{t_{start}, t_{start} + s, ..., t_{end} - s, t_{end}\}$, with $t_{start}$ being the earliest time point and $t_{end}$ being the latest time point in your time series data.
3: Sample a time $t_0$ uniformly from this set, $t \sim U\{t_{start}, ..., t_{end}\}$
4: Using the first data point before the sampled time $t_0$, $(t_{before}, x_{before})$, and the first data point after this time $(t_{after}, x_{after})$, approximate the value of your time series at $t_0$ by interpolating between these two points, call this approximated value $x_{approx}$.
5: Sample a value $x_0$ from a normally distributed random variable centered around $x_{approx}$, $x_0 \sim N(x_{approx}, \sigma^2)$, with the value of $\sigma^2 \approx (x_{max} - x_{min})$, the difference between the largest and smallest values in your time series
6: Use $(x_0, t_0)$ as input for your neural ODE, along with any other relevant inputs, to predict the value of the derivative at $t$.
7: Calculate loss from your predicted derivative $\frac{dx}{dt}_{predicted}$ with the value of the vector field at time $t$, which can be defined in a variety of different ways (see below), using MSE, $Loss = (\frac{dx}{dt}_{predicted} - \frac{dx}{dt}_{training})^2$
8: Update the weights of $M$ using a standard neural network optimizer, e.g. Adam or stochastic gradient descent
9: **end for**
10: **return** $M$

One important detail of the algorithm above is that we do not use a continuous uniform distribution to sample the time $t$, but instead we use a discrete uniform distribution. We can do this because we can specify an acceptable step size for our ODE solver beforehand, which allows us to know exactly when the solver will need to predict the derivative. Knowing this, we do not have to make a prediction for the derivative at every possible time, but instead only at the time points that the solver will use. This ultimately led to faster convergence during training when compared to using a continuous uniform distribution.

The algorithm below contains two important but closely related ambiguities, how we interpolate between our two data points and how we calculate $\frac{dX}{dt}_{training}$, we will refer to the vector field defined by our calculation of $\frac{dX}{dt}_{training}$ as our training vector field (TVF). How we calculate the TVF determines the shape and behavior of our curve, and also the type of interpolation we use.

### 2.4.1 Flow Matching for PK Time Series

To begin with, we tested out our flow matching algorithm on PK data. Although most implementations of flow matching use the slope between the interpolated data point and the training data, the best performing models for PKPD time series instead used what will be referred to as a nonlinear TVF, rather than a linear TVF, where the slope is used. This means that instead of using the slope to define our TVF, we instead use a nonlinear function to calculate the vector field at our interpolated data points.

For modeling the decay in concentration after peak concentration was reached, the flow is determined by an exponentially decaying function. This forces the flow to approach zero by default as the concentration goes to zero. Furthermore, it is often assumed in PKPD modeling that the rate of clearance of the drug from the body is proportional to the current concentration of the drug in the body, so we are not making an unrealistic assumption. However, it is possible there are cases where exponential decay is not a good fit for the data, and if there is good reason to believe this, then defining the TVF using exponential decay would be unwise.

For the increase in concentration after a dose is taken, we expect to have a large increase shortly after the dose is taken and then smoothly approach zero as we reach four hours since the last dose, four hours being the time taken to reach maximum concentration. The simplest way to enforce this behavior was to define a quadratic polynomial whose derivative approaches zero as we approach the peak dose size. This polynomial can be determined entirely from the data points available to us.

Using TVFs defined from these nonlinear equations ultimately gave us the best results, and is what we use for comparison with the neural ODE model trained with backpropagation through the solver. For more details on the all of the TVFs that were implemented and the results of the models trained on them, see Appendix E.

### 2.4.2 Flow Matching for PD Time Series

In order to model the decrease in the biomarker after a dose was taken we used a linear TVF, and to model the increase in the biomarker back to baseline we used saturating exponential growth, i.e. an exponential growth curve with a horizontal tangent at the baseline value of the biomarker. This is similar to exponential decay but flipped upside down. The inputs we used for this model were the same as those used for the PK flow matching model.

However, the results for this model were unsatisfactory, so instead of using the dosing regime as input, we used our trained PK model to predict concentrations to be used as input for the PD model. This was done because in PKPD modeling it is not the dose that directly affects the response in the patient, but rather the concentration in the body. Thus, if we have a PK model that can accurately predict the concentrations of the drug in the body, we can use these predicted concentrations as input for the PD model. Because the current concentration directly determines the change in the response, we believed that this would improve the capabilities of our PD model.

The PK model's predictions are used rather than the true concentrations because the true concentrations are sampled very sparsely. From the results obtained in the comparison of the neural ODE with the RNN and MLP, it was seen that using a small step size is important for the PD model. Thus in order to have a smooth curve of concentration values, we need to use the predictions of the model. This can be done by first solving for smooth PK trajectories using the dosing regime as input, and then when training using the corresponding PK value at the randomly sampled time. During inference we simply solve a two differential equations at once, one for the PK values and one for the PD values.

## 2.5 Extracting information about the underlying PKPD model

A goal of the project is to extract information about the underlying PKPD model from the trained neural ODE model. The trained neural ODE model we extracted our information from was the neural ODE model trained using flow matching. If the model is successful in this, then its predictions would be more trustworthy, as it would demonstrate that the model has learned fundamental information about the underlying data-generating process. Furthermore, it would demonstrate that the model could be used for the creation of a mechanistic PKPD model. We investigate three different methods for information extraction: Uncertainty quantification, SHAP Analysis, and finding a closed-form ODE for the neural ODE.

### 2.5.1 Uncertainty Quantification

Although deep neural networks have been very successful in making accurate predictions in a variety of domains, they can often be unreliable when quantifying how confident they are in their predictions [29]. Quantifying the uncertainty of a prediction is particularly important in medical applications, where observations can vary significantly on a personal level. Furthermore, medications inevitably come with unwanted side effects that can become dangerous at certain doses. Therefore, one cannot rely only on the point estimates of the model. Instead, a trustworthy confidence interval of possible values is needed. This made finding a method of generating a reliable confidence interval for the model an important aspect in improving the interpretability and practicality of the model.

Given that in PKPD modeling it is typically most important to find the mean response to a given dosing regime, it made the most sense for us to quantify the uncertainty of this mean response. When we have PKPD data for a given dosing regime, the uncertainty in this mean response can easily be calculated using the standard error, which is defined as $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$, where $\sigma$ is the sample standard deviation and $n$ is the number of samples. Then, 95% confidence intervals can be generated by taking the average over the time series at each point and then adding and subtracting the standard error. However, without any data, it is unknown what the standard error would be at a different dosing regime. Thus, what we want is a way for our model to estimate what the standard error would be if we had access to

data from dosing regimes not included in our training dataset. We can test this by comparing our model's estimated uncertainty during extrapolation to new dosing regimes with the standard error of datasets from those dosing regimes by generating them with our underlying PKPD model. Of particular interest to us is to see if there is a method of uncertainty quantification that we can use that would change depending on the level of noise in our training data. For example, we would want a method where the magnitude of uncertainty would change depending on whether or not random effects are included, and consequently, the size of the standard error would be smaller or larger. Thus, we would want a method of uncertainty quantification that scales with both the dose size and the size of random effects in the data-generating process.

When discussing the uncertainty of a neural network, or any model fitted to a dataset, one must consider two distinct sources of uncertainty. One source of uncertainty comes from the architecture and parameters of the model itself, which is known as epistemic, or model, uncertainty. This type of uncertainty results from a lack of data and will, in theory, tend to zero as the amount of data the model is trained on increases [29]. The other type of uncertainty is known as aleatoric, or data, uncertainty. This uncertainty is irreducible in the sense that no amount of data will completely remove it, it is inherent to the process given the existing experimental setup. This type of uncertainty may arise from measurement uncertainty or inherent randomness within the process itself; however, the main point is that no model can account for it. Random effects in the context of a PKPD model are somewhat difficult to categorize as either aleatoric or epistemic uncertainty. On the one hand, they are randomly chosen variables inserted into our model, thus making it impossible to model. However, they do not make the model stochastic in the sense that the data can move randomly, the data will always be a smooth curve.

Thus, we decided to focus on quantifying the epistemic uncertainty of our model and comparing it with the noise in our dataset. The epistemic uncertainty of our neural ODE model was quantified using a procedure known as stochastic weight averaging - Gaussian (SWAG) [30] [31]. SWAG is a computationally efficient method for approximating a posterior distribution of your network weights, providing a distribution of optimal network weights rather than a single point estimate. The method is easily applied on top of your standard neural network training, except that after training the network to convergence using a chosen optimizer, SGD is performed with a constant learning rate on the trained model for some specified number of epochs, with the weights being saved every epoch. The results of [30] show that SGD explores a region of optimal weights close to the optimum found during training. These weights are then used to create a Gaussian distribution that effectively approximates the geometry of this region of high-performing models. This Gaussian distribution is an accurate approximation of the geometry of this region of high-performing weights [31].

Although SWAG can only capture epistemic uncertainty, we still expect this to scale with the size of the random effects in the dataset. This is because we expect that higher levels of random effects will lead to a more entropic Gaussian posterior

of high-performing weights in the loss landscape. Without noise, we would expect this region to be smaller because the relationship between inputs and outputs would be clearer. We devised an experiment to test the ability of SWAG to capture our underlying PKPD model's epistemic uncertainty by training two identical neural ODE models on two different training sets, one with random effects and one without random effects. We would then compare how the magnitude of the uncertainty in our predictions changes with the dose size. We would then compare our neural ODE's uncertainty with the uncertainty of the underlying PKPD model.

The uncertainty is most important with respect to the maximum steady-state concentration, so the confidence intervals were generated from this value.

During the project, we also attempted to tackle uncertainty from another perspective. We wanted to quantify the uncertainty to determine how certain we are that the population mean lies where we predict it to be. The end goal was to use the standard deviation from ensemble predictions to create confidence intervals of where the underlying population mean is. Then, we wanted to empirically prove that when the model extrapolates, the population mean will lie within one standard deviation of our prediction approximately 68% of the time. Despite the substantial time spent on this problem, our approach ultimately did not yield satisfactory results. For more information on our approach and its results, see Appendix B.

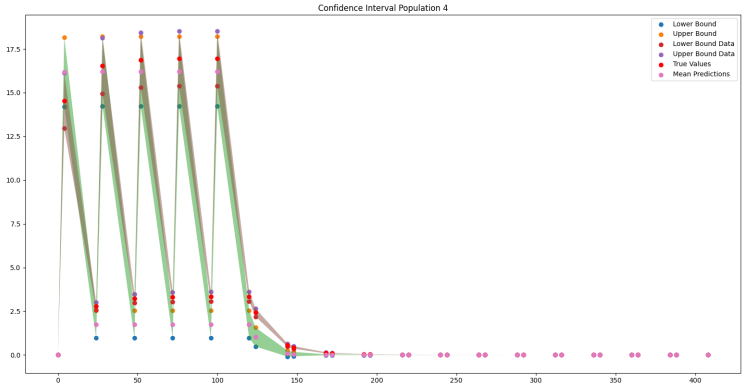



Figure 2.3: Figure showing how confidence intervals were generated. The confidence interval of the 4th peak is used.

### 2.5.2 SHAP Analysis for determining the effects of covariates

One area in which the model could be of great practical use to pharmacometricians attempting to define a mechanistic PKPD model is to identify covariates that impact the concentration and response of our drug of interest. One important tool for analyzing the effect of inputs on the outputs of a machine learning model is SHAP. SHAP (Shapley Additive Explanations) informs us about the positive or negative

impact of different features on the numerical value of our neural network prediction [32]. The SHAP method is inspired by Shapley values, a concept from cooperative game theory, which aims to quantify the effects of different players on the total reward [33]. Shapley values were previously applied to quantify the effects of different variables in a multiple regression model [34]. However, calculating them directly is extremely computationally expensive, as one would have to train the model multiple times on all possible combinations of feature subsets. Thus, SHAP is a method to approximate Shapley values for your machine learning model without having to train the model multiple times. Because SHAP values can indicate the impact of our different covariates on the numerical value of our concentration or response derivative, they reveal which covariates have a positive, neutral, or negative effect on the dynamics of our trajectories. The Python library shap is used to determine the SHAP values of the various inputs. We will only perform SHAP analysis on the trained PK model, as the covariates have no direct effect on the PD ODE.

### 2.5.3 Finding a closed-form equation of a neural ODE

As previously mentioned, a big drawback of neural ODEs is the lack of a closed-form ODE expression. Having an interpretable model is always good, but in the field of PKPD, it is especially important, as faulty prediction may have dire consequences.

In an effort to make our model more trustworthy, we have developed a method of going from a trained neural ODE into a closed-form ODE. Due to time constraints, we have unfortunately not been able to apply this approach to the PKPD model, but the method is verified on toy examples in the results section of the project.

The method looks at a neural ODE and tries to interpret what closed-form ODE it describes. This is not a modification to the neural ODE architecture that makes it inherently more interpretable, but rather a tool that can be used to interpret neural ODEs. While it might in some ways be desirable to make the interpretability inherent in the model, there are benefits to creating an external tool. A huge benefit is that once such an interpreting tool has been developed, it could be used on all existing neural ODE models. If one instead made the interpretability inherent due to changes in architecture, then this would require changing existing neural ODE models for them to be interpreted. Lastly, one could hypothesize that creating inherent interpretability is a necessarily complex problem, requiring careful architecture development and use of complex mathematics. In contrast, the proposed tool in this project is conceptually simple.

To find the ODE that fits the neural ODE the best, we begin by searching for the optimal structure of the ODE. That is, we want to know what terms should be included in the ODE. We do this by the following procedure: 1. Define a language of ODE structures. 2. Define a way of comparing different ODE structures. 3. Define a way to search this language for the optimal structure.

Once this optimal structure is found, the best coefficients can be determined, re-

sulting in a single ODE. We now go through each of the steps in detail.

**Defining a language of ODE structures:** We start by noting that given two different ODE structures, we can create a new ODE structure by combining the original two. For example, the families $\frac{dy}{dt} = c_1 t$ and $\frac{dy}{dt} = c_2 y$ could be combined into either $\frac{dy}{dt} = c_1 t + c_2 y$ or $\frac{dy}{dt} = c_3 t \cdot y$. This means that given some elementary families of ODEs, we can create an infinite language by combining the elementary families through addition and multiplication. Because ODEs consisting of quite simple components can give rise to a wide variety of functions, we will not need particularly many elementary families. In this project, we start with the following elementary families: $\frac{dy}{dt} = c_1$, $\frac{dy}{dt} = c_2 t$, and $\frac{dy}{dt} = c_3 y$. This allows us to interpret a wide range of ODEs, although, should one need to interpret other types of ODEs, it would be easy to add more elementary families.

**A way of comparing ODE families:** Given two different ODE families, how do we decide which one describes our neural ODE the best? There is no objective answer here, but we have developed a way of comparing ODE families against each other.

Because the ultimate goal is to find a singular ODE, comparing two families is equivalent to comparing the best ODE of each of those families. We do this by finding the best coefficients for each particular family according to some metric, and then comparing the metrics of these two ODEs with the determined coefficients.

We define the metric as follows: Fix an integer $n > 0$ and let $\hat{f}(y, t, \mathbf{b})$ be the predicted derivative at point (y,t) by an ODE candidate with coefficients $\mathbf{b}$. Furthermore, let $f_\theta(y, t)$ be the derivative of the neural ODE at a point $(y, t)$ and let the vectors $\mathcal{T}$ and $\mathcal{Y}$ represents the trajectory of the neural ODE with $n$ time points. Then, we define the score of an ODE as

$$score(\hat{f}) = \sum_{i=1}^{n} (\hat{f}(\mathcal{Y}_i, \mathcal{T}_i, \mathbf{b}) - f_\theta(\mathcal{Y}_i, \mathcal{T}_i))^2.$$

That is, the score of an ODE is given by the MSE between the predicted derivative and the derivative of the neural ODE at $n$ points along the trajectory of the neural ODE. We can easily find the optimal coefficients for a particular family of ODEs by the method of least squares. An illustration of this procedure can be seen in Figure 2.4.

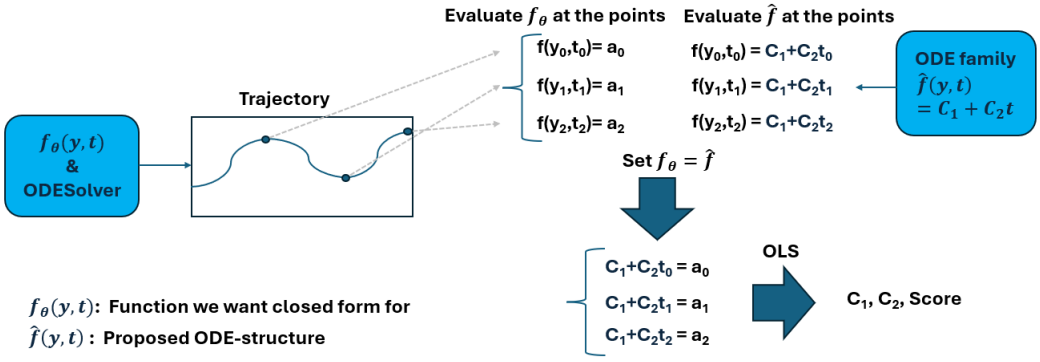



Figure 2.4: Visualization of the scoring mechanism.

**A way to search the language of ODE families:** If one were to manually try to find the correct closed-form ODE, it would make sense to start looking at relatively simple ODEs and successively look at more complex ODEs until one reaches a satisfactory fit. Our tool follows this intuition and makes use of the way we defined our language. We start by trying out all of the elementary families, comparing their scores with each other. The best-performing family of ODEs is then modified to become a more complex expression by combining it with another elementary family of ODEs, either through addition or subtraction. By going through the different families of ODEs that can be combined, we can create a set of new, slightly more complex candidates. Their performance is then evaluated, and the best-performing one gets combined with new elementary families of ODEs. This process is repeated until there is no significant performance increase by increasing the complexity of the ODE, at which point a proposed family of ODEs has been found.

Because of the way the scoring works, this means we have also found the optimal coefficients! Therefore, the tool can now output the interpreted ODE. See Figure 2.5 for a visualization of the search procedure.

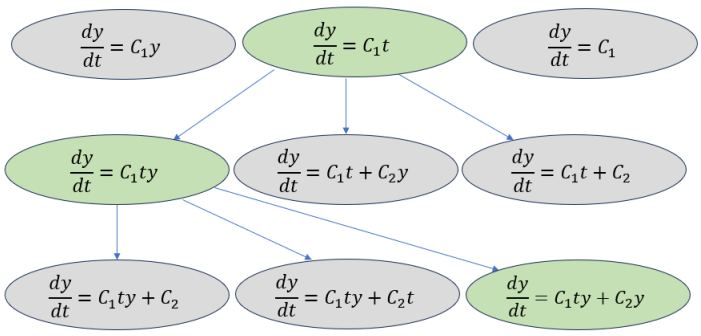



Figure 2.5: Visualization of the ODE search.

To make the interpreter work in our specific application of PKPD modeling, a slight modification would be necessary. As previously mentioned, PKPD modeling is typically performed using systems of ODEs, whereas our tool can currently only handle a single ODE. We can change the search procedure slightly to accommodate this. Previously, we only allowed the addition and multiplication of elementary families to increase complexity. Now, we also allow for the creation of an entirely new compartment, resulting in a system of ODEs instead. Lastly, to successfully apply this interpretation method to PKPD modeling, one also needs to change the set of elementary ODEs slightly.

## 2.6 Model for Adverse Effects

In this project, a Markov model [14] has been used to generate adverse events data on which to train and test a neural ODE model. Due to how the model would be used in practice, the split into train and test data has been predetermined. The training dataset comprises three titration schemes, while the testing dataset consists of eight additional titration schemes. Each titration scheme has been given a name that describes its characteristics. The differences between the titration schemes typically involve the size of the starting dose, the frequency of dose increases, and the rate of dose increase. See Table 2.1 below for descriptions of the titration schemes. After a titration scheme reaches the highest dose, that dose size is used every week until the end of the 36-week period.

Table 2.1: Descriptions of the Titration Schemes. The final dose size is 600mg for all titration schemes except Weekly ST100 ED800, where the final dose size is 800mg.

| **Titration Scheme** | **Description** |
|---|---|
| Weekly | Weekly increase: 50,100,200,300,400,500,600... |
| Biweekly | Biweekly increase: 50,100,200,300,400,500,600... |
| Biweekly 2-fold | Biweekly increase: 50,100,200,400,600... |
| Weekly ST100 | Weekly increase: 100,200,300,400,500,600... |
| Weekly ST100 ED800 | Weekly increase: 100,200,300,400,500,600,700,800... |
| Weekly ST25 | Weekly increase: 25,50,100,200,300,400,500,600... |
| Biweekly 2-Fold ST100 | Biweekly increase: 100,200,400,600... |
| Monthly | Monthly increase: 50,100,200,300,400,500,600... |
| Monthly ST100 | Monthly increase: 100,200,300,400,500,600... |
| Monthly ST25 | Monthly increase: 25,50,100,200,300,400,500,600... |
| Monthly 2-Fold | Monthly increase: 50,100,200,400,600... |

To get a feel for what the data looks like, see Figure 2.6 below for a visualization of the training data set.

Recall that our model for Adverse Effects is designed to predict the proportion of patients who experience mild or moderate nausea each week over a 36-week period,

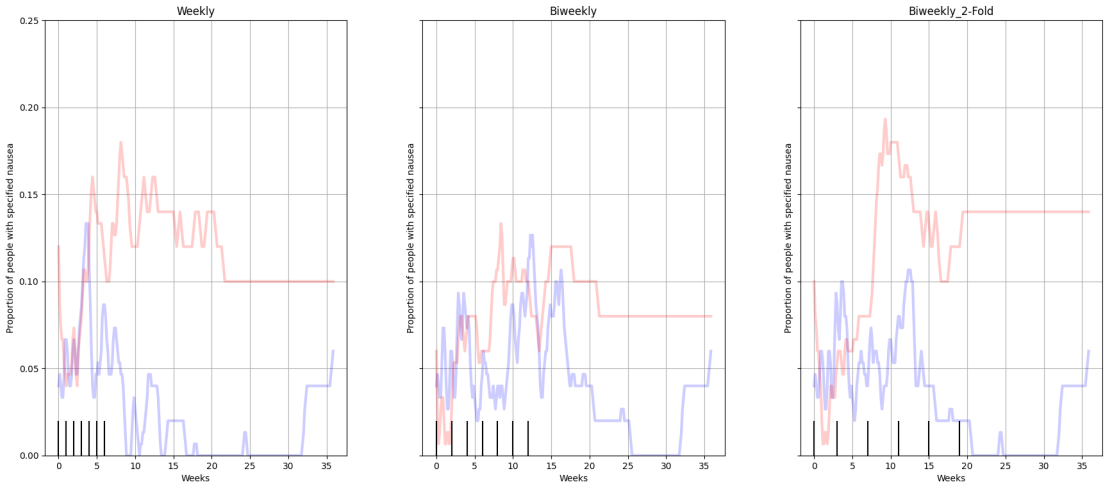



Figure 2.6: Data for the three titration schemes in the training data. The red line represents the proportion of mild nausea, the blue line represents the proportion of moderate nausea. The black lines at the bottom indicate when an increase in dose occurs.

given a specific titration scheme. This means that the input must be a representation of a titration scheme. We settled on a vector with 36 elements, each element representing the cumulative amount of the dose taken at each week in the titration period. This allows the user to input any titration scheme of interest, provided it is conducted over a period of no more than 36 weeks. To help the model understand the problem, another 36-size vector is created, representing the time since the last change in dose size. Note that this vector can be automatically created using the information from the first vector and can therefore be viewed as an internal process of the model rather than an input from the user.

To avoid overfitting and obtain an estimate of the model's uncertainty, we train five copies of the model using early stopping, resulting in an ensemble model. The final prediction is the mean of the five models, and our uncertainty estimation is the standard deviation of the predictions. See Appendix D for the specific architecture of the model.

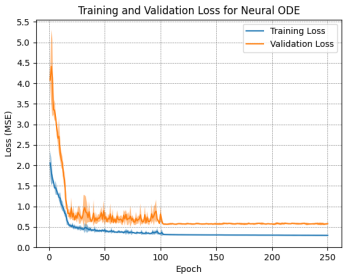
# 3

# Results and Discussion

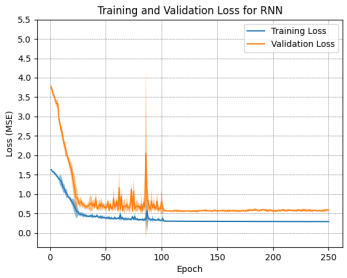
## 3.1 Research Question 1

### 3.1.1 Comparison of Discriminative PK Models

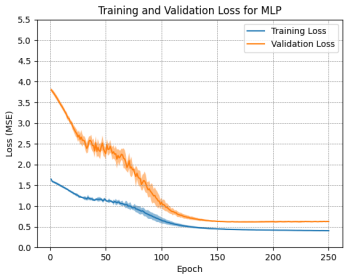
The results below were generated by running each model five times and averaging the results. In the plots, the solid lines are the average result, and the lighter shading indicates a 95% confidence interval. For the three dosing regimens in the training data, the model is trained on 75% of the data, and the remaining 25% becomes the validation dataset. For the test regimens, the model is validated on the mean of the population, i.e., a patient with the average weight (70 kg), age (40 years), and sex (0.5), with no random effects.




(a) ODE Training/Validation Loss

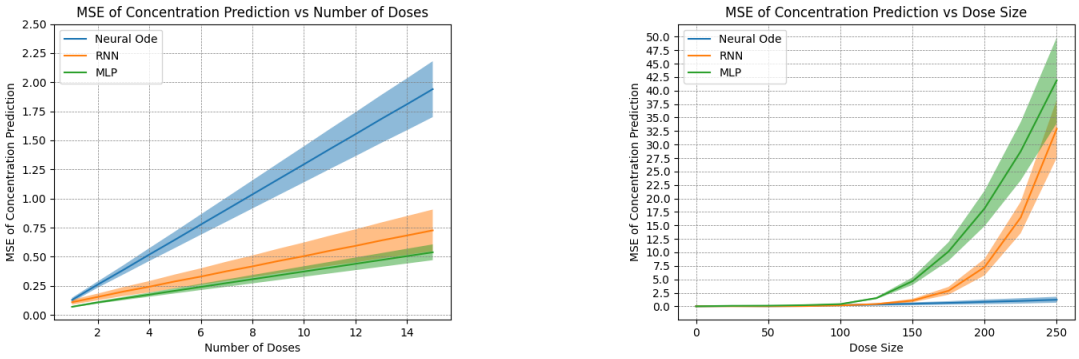



(b) RNN Training/Validation Loss




(c) MLP Training/Validation Loss

Figure 3.1: Training/validation losses




(a) Number of Doses

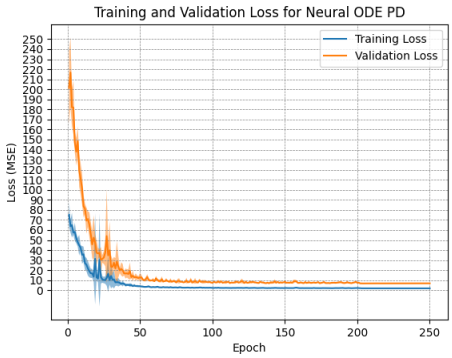
(b) Dose Size

Figure 3.2: Extrapolation across dose size and number of doses

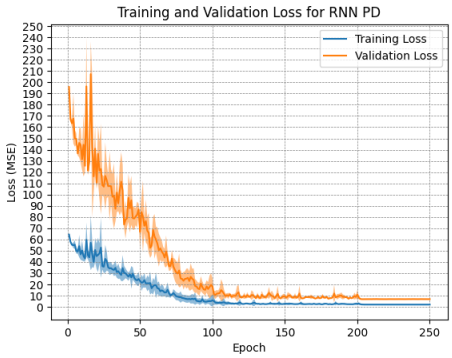
### 3.1.2 Analysis

The results from the comparison supports the hypothesis that neural ODEs will outperform the RNN and MLP models. Not only did the neural ODE converge in fewer epochs, but it also extrapolated to new dosing sizes much more accurately than either the RNN or the MLP. Although the performance of the neural ODE is worse in extrapolating across the number of doses, its performance is still stable and does not exponentially diverge. For our purposes, it is the performance of the models during training and across different dose sizes that is of most interest, and it is here where the neural ODE shows clear advantages. The RNN also performed significantly better than the MLP, both in terms of how quickly it converged and how well it extrapolated to new dosing sizes. This supports our assumption that the modeling problem is easier as we move from attempting to model the concentration directly to modeling the dynamics of the concentration. Learning the dynamics seems to lead to a more generalizable model of the concentration trajectories, which is expected, given that it seeks to approximate a pair of closed-form differential equations whose analytical integration is intractable.

### 3.1.3 Comparison of Discriminative PD Models

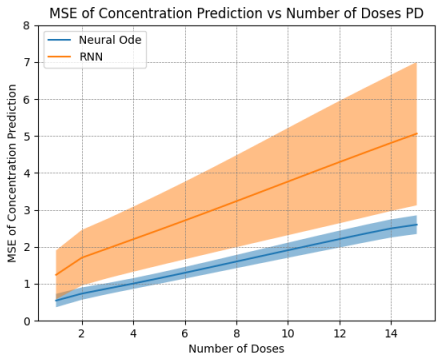



(a) ODE Loss

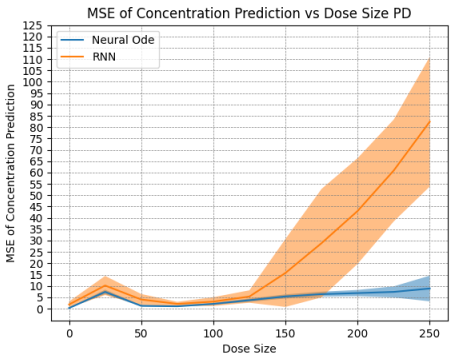



(b) RNN Loss

Figure 3.3: Training/validation Losses for models trained on PD data




(a) Number of Doses




(b) Dose Size

Figure 3.4: Extrapolation across dose size and number of doses for models trained on PD data

### 3.1.4 Analysis

In general, the results for the PD models are very similar to that of the PK models, except the difference in performance is even larger. The MLP model was excluded from comparison, as its performance was so poor that there was little value in including it in the comparison, and the error in the RNN begins to diverge even more quickly than compared to the PK data. We attribute this larger difference in performance to the fact that the integrated dynamics are even more complicated for PD modeling than for PK modeling. Thus we would expect that as the step size becomes smaller and the trajectories become smoother, the model's performance will increase, as it will in effect be modeling a simpler problem.

In order to further test this hypothesis, the neural ODE model was trained with three different step sizes in order to compare the performance as a function of step size. The smaller the step size, the smoother the dynamics of the solved trajectories. As can be seen in the figure on the next page, smaller step sizes improve the accuracy of the neural ODE. This helps strengthen the conclusion that as we force smoother dynamics, we get a more accurate modeling of the system. However, this results in the neural ODE taking increasingly longer to train. A smaller step size results in more computations for both the forward and backward passes. Therefore, a method of training that allows quick training of a neural ODE with a small step size is needed. This problem is resolved by using flow matching to train the neural ODE.

The training and validation plots once again demonstrate a faster convergence for the neural ODE, so it not only outperforms the RNN but also requires fewer epochs to accomplish this. This again points to the fact that the neural ODE is, in essence, solving a simpler problem by learning the underlying dynamics of the system rather than its integrated results.

It is interesting to note that smoother solutions improve performance even though the administration of the dose is a discontinuous event. However, this does not impede performance because time from last dose is included as an input. Once a new dose is taken, the input also shifts discontinuously back to a time since last dose of 0, which allows the model to still perform effectively.

Summarizing the results of both the PK and the PD models, we can say that the neural ODE outperforms the other models when extrapolating to new dose sizes. All models are stable with regard to the number of doses, so there is no difference in performance in this regard. This answers the first research question.

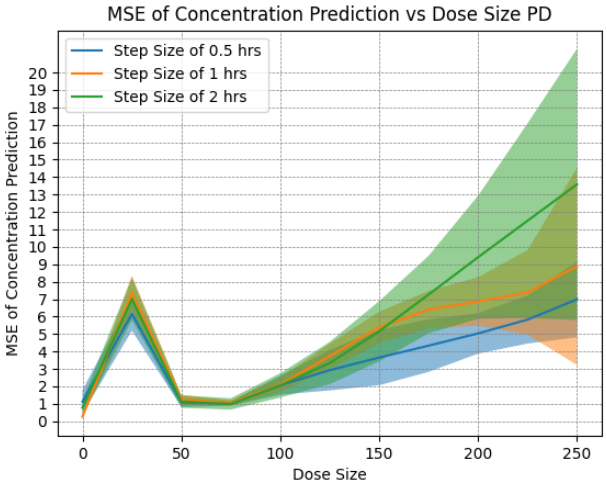



Figure 3.5: Comparison of performance for same neural ODE model with different step sizes.

## 3.2 Research Question 2

### 3.2.1 Visual inspection of sampled time series

The goal of the generative models is to be able to sample time series that follow the distribution of the training data. One way of evaluating the quality of these samples is through visual inspection. This ensures they look like realistic PKPD time series. Plotting the samples together with their collective mean and then comparing this to the true mean of the population gives a rough idea of the quality of the generative model.

Firstly, 100 samples are generated from each model using the dosing regimen of 5 doses of 100mg. The results can be seen in Figure 3.6.

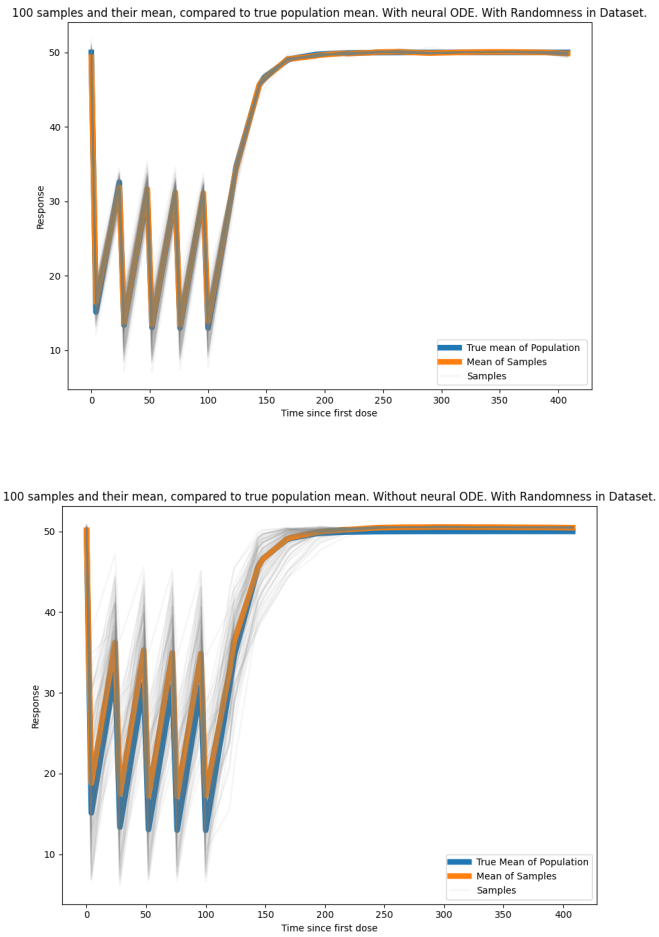



Figure 3.6: Comparison of the model with neural ODE (top) and the model without (bottom). One hundred samples are generated from each, the mean of these samples are calculated and compared to the true population mean.

The result for the second dosing regimen is similar, and the plots can be found in the appendix.

### 3.2.2 Numerical Comparison

While a visual comparison gives a decent idea about the relative performance of the two models, a more rigorous method of comparison is desirable. The goal of the model is that its samples reflect the underlying population. To measure this, we use two metrics: 1. The MSE between the population mean and the mean of the samples, denoted $MSE_\mu$. 2. The MSE between the population standard deviation at each time point and the sample standard deviation at each time point, denoted $MSE_\sigma$. The first metric measures whether the samples are centered correctly around the population mean, and the second metric measures whether the

spread of the samples accurately reflects the spread of the population.

The evaluation procedure is as follows. For each of the models, 100 samples are generated, and from this, the respective metrics are calculated. This process is repeated five times, with the results being summarized in the table below. Because the results for the two dosing regimens are so similar, these tests are only done for the dosing regimen with 5 doses of 100mg.

Table 3.1: Performance metrics of the Generative models

| | Average $MSE_\mu$ | Std of $MSE_\mu$ | Average $MSE_\sigma$ | Std of $MSE_\sigma$ |
|---|---|---|---|---|
| With nODE | 0.2518 | 0.2329 | 1.79e-05 | 5.25e-06 |
| Without nODE | 0.8127 | 0.7899 | 0.0014 | 0.0022 |

### 3.2.3 Visualizing the Diffusion procedure

To gain insight into the inner workings of our model, one can plot the results at different steps in the reverse process. This way, a progressive denoising of a time series should be visible. By taking the encoding at some states in the backward process of the DPPM and decoding them into time series, we get such an illustration, see Figure 3.7 We can compare this to the same type of plot for when we use the

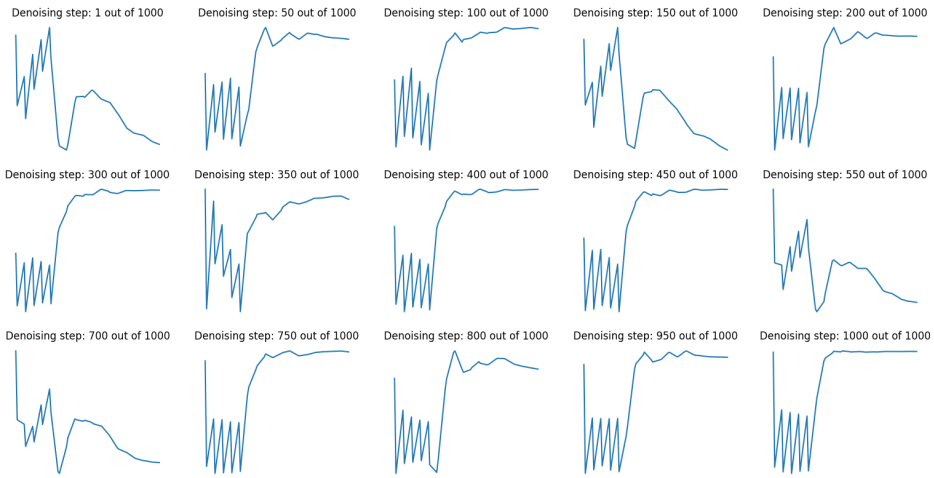



Figure 3.7: Visualization of the denoising process for the neural ODE-based model.

simple AE model instead, see Figure 3.8.

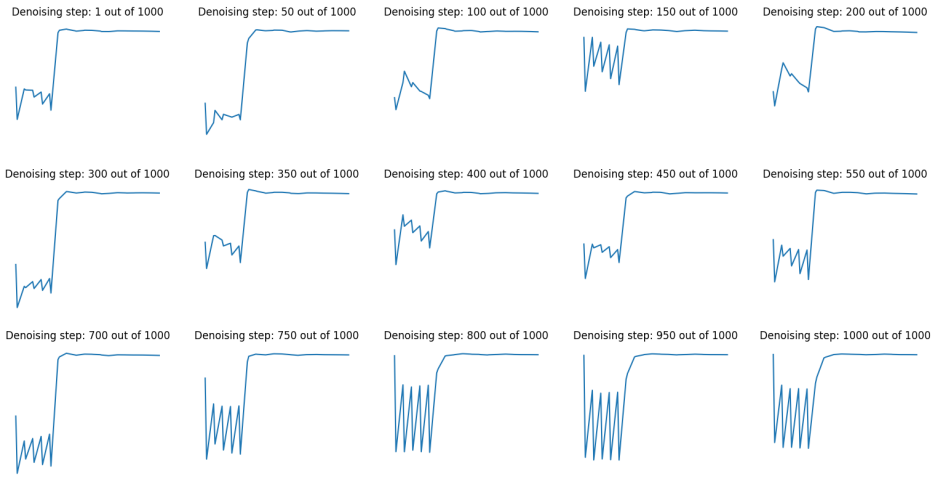



Figure 3.8: Visualization of the denoising process for the simple Autoencoder model.

### 3.2.4 Analysis

Visually, the samples generated by the models appear to be of quite high quality for both models. Firstly, they have the characteristic look of PD time series, with peaks and troughs at dosing times. Furthermore, when compared to the true population mean, the values appear to be very reasonable. This is especially true in the model using neural ODEs, where the sampled mean often lies directly on top of the true population mean. Another thing of interest is that the variability of the samples varies over the trajectory. We can observe that the biggest differences between the samples are observed at the peaks and the troughs, while the spread is much thinner after the last dose has been given, and the response slowly returns to the baseline value. While you do not want too large a spread, you would expect to see noticeable differences between the samples. In these plots, the neural ODE model may appear to have very little spread, but this is partly due to the line width of the mean plots. When you plot only the samples, the amount of spread looks reasonable. However, it is not possible to say which model has the better spread from the visualization alone. One needs to take a look at the metrics as well.

Firstly, the metrics verify that the neural ODE model is, in fact, better at producing samples with expected values around the population average. Not only is the mean of the samples closer to the population mean on average than for the model without a neural ODE, but the variation in the difference is also much lower. Interestingly, we can also see that the neural ODE model captures the variability in the population much better than the model without a neural ODE.

Earlier in the thesis, we have shown that neural ODEs have significant advantages when used in Discriminative models for PKPD modeling. These results now indicate that neural ODEs are also beneficial for Generative PKPD models. While the

Generative models were not used for any other task except this comparison, their development was nonetheless informative.

Moving on to the visualization of the de-noising process, we see that many of the early states are not as noisy as one might first expect. One can still see the structure of a PKPD time series. We have the following explanation for this: The neural ODE and the decoder take the noisy encoding and give structure. The decoder has learnt to map encodings to viable time series. This means that even if an encoding is a bit noisy, it can still be translated into a semi-realistic PKPD time series. More importantly, the information about what dosing regimen we want is given to the decoder. Therefore, even if the encodings themselves are mostly random, the resulting time series will still resemble a time series of the given dosing regimen, because the decoder holds so much information. If we think more closely about what information an encoding contains, it is mostly information about covariates. But since different values of covariates do not completely change the structure of a PD time series, the same should hold for the random encoding.

Since we want to steer our samples into a specific dosing regimen, we will necessarily have to handle that information in the decoder. Therefore, the decoded time series will always have some structure related to our dosing regimen, despite the fact that the encoding consisted mostly of noise. This is a property that is a consequence of what the model is supposed to do.

While there is, of course, quite a bit of randomness in these types of plots, there is one difference that is consistent between the two models. The neural ODE model struggles much more with getting the end of the trajectory to a correct flat line, while that is something the simple AE model gets correct immediately in all cases. This suggests some differences in the internal structures of the two models, although it is difficult to articulate exactly what these might be. It could be because for the output of a neural ODE to be constant, it would need a derivative equal to exactly zero, which is not feasible in the early stages of the denoising of the encodings.

In conclusion, there seems to be some slight differences in the inner workings of the two models, as one might expect. More importantly, we have seen that the use of a neural ODE for a generative PKPD model can increase the quality of the generated samples, which answers our second research question.

## 3.3 Research Question 3

### 3.3.1 Performance of the neural ODE trained with flow matching on PK data

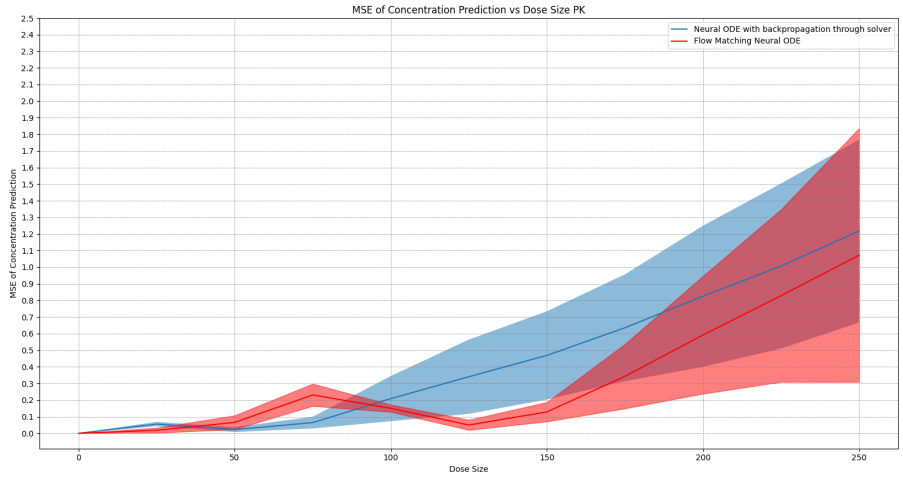



Figure 3.9: Comparison of how the neural ODE trained with flow matching extrapolates to new dosing sizes vs. the neural ODE trained with backpropagation through the solver.

### 3.3.2 Analysis

Firstly, we note that we omitted figures for the training and validation loss of the flow matching model. This is because the training loss during flow matching is very random, due to the fact that we need to randomly sample values to train our neural ODE on. We also omitted comparisons for the number of doses, as both models were once again stable with respect to the number of doses, and these comparisons were not informative.

The flow matching algorithm performed roughly the same with respect to extrapolation across dose size. However, when comparing the training times of the two different training methods, flow matching is substantially better. This was tested by timing the training of the two models. In order to make a direct comparison, we used the same step size for each model, 0.1 hours, and trained each model until convergence. Averaging over five runs, the training time for the neural ODE was 305.4 minutes, while the average training time for the flow matching model was 18.2 minutes. This demonstrates the utility of using flow matching as a training algorithm for PKPD time series. The flow matching model actually took more epochs to converge, but the time required for each epoch was significantly less. The reason flow matching takes more epochs is that for each epoch, only a small portion of the total time series is sampled, while when backpropagating through the solver, all of the time steps are trained at once. However, we note that the time series we are

training with are particularly well suited for the flow matching method, as they are nearly cyclical. For example, the behavior of the trajectories between the 3rd and 6th doses is nearly identical, making it very easy for the model to generalize what it has learned, regardless of which point the random time is sampled from. In the case of more complex time series that show more diverse behavior, it is possible that flow matching would take more epochs to train, as it would not be able to generalize as easily.

### 3.3.3 Performance of the neural ODE trained with flow matching on PD data

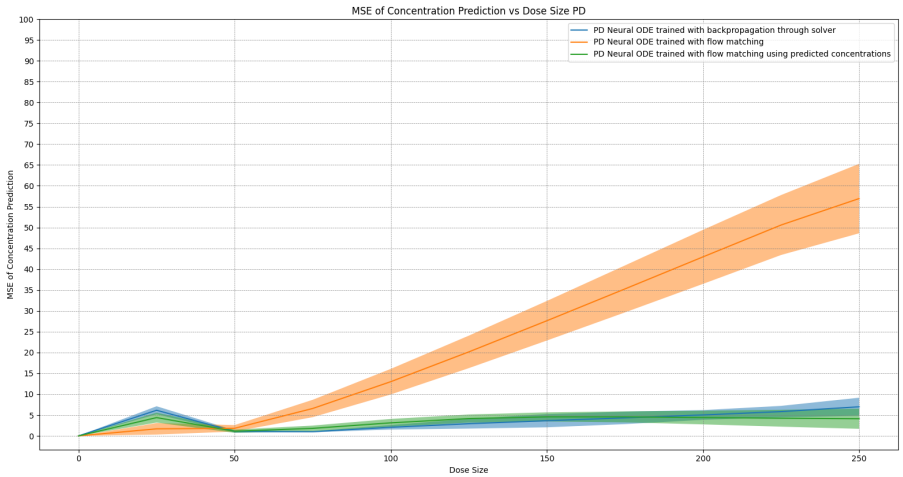



Figure 3.10: Comparison of the extrapolation capabilities of a neural ODE trained with backpropagation through the solver vs. flow matching

### 3.3.4 Analysis

We did not perform a comparison of the training times between the PD model trained with backpropagation through the solver and the flow matching PD model, as their training was almost identical to that of the PK models, and therefore, no new insights would be gained from them.

The failure of flow matching to model the PD trajectories using the dosing regime highlights one of the drawbacks of the flow matching method. When using flow matching, one gives the model a very strong prior, in the form of the vector field to be approximated, and this prior can lead to poor learning capabilities if it is not appropriate for the task at hand. In the case of the PD trajectories, the effect of the dose size on the response is indirect, as it has a direct effect first on the concentration and then on the response. In this case, making our model conform to a relatively simple TVF is most likely a poorly chosen prior. However, when we use our predicted concentrations as inputs instead, this assumption no longer holds the model back, as the current concentration has a direct effect on the change in the

response. We believe this is why flow matching fails when attempting to model the response when using the dosing regime, but is successful when using the predicted concentrations.

On the other hand, the original neural ODE model is successfully able to extrapolate for PD data using the dosing regime, and this is because there are no priors forced onto the vector field that it learns. It is thus able to learn a more complex relationship between the dose size and the response.

Overall, we consider our method of training neural ODEs with flow matching to be a success. The models were able to extrapolate to new dosing sizes with the same accuracy as our original neural ODE model, at a fraction of the training time. However, they also showed limitations that could only be partially alleviated by modifying their TVF.

It could be said that having to define a valid TVF is a disadvantage, as a key advantage of using a neural network is that you can create an accurate model without many prior assumptions. However, there are two key points to consider, one being that the assumptions made on the TVF in this thesis were very general and could be applied to practically any PKPD model. The other point is that this problem could be removed in a future work. For example, we could lift our one-dimensional concentration or response values into a higher-dimensional latent space, and then perform linear flow matching in this space, and then decode back into our original one-dimensional space. Perhaps in this way our models could learn more complicated vector fields that do not have to be specified beforehand.

## 3.4 Research Question 4

### 3.4.1 Uncertainty Quantification

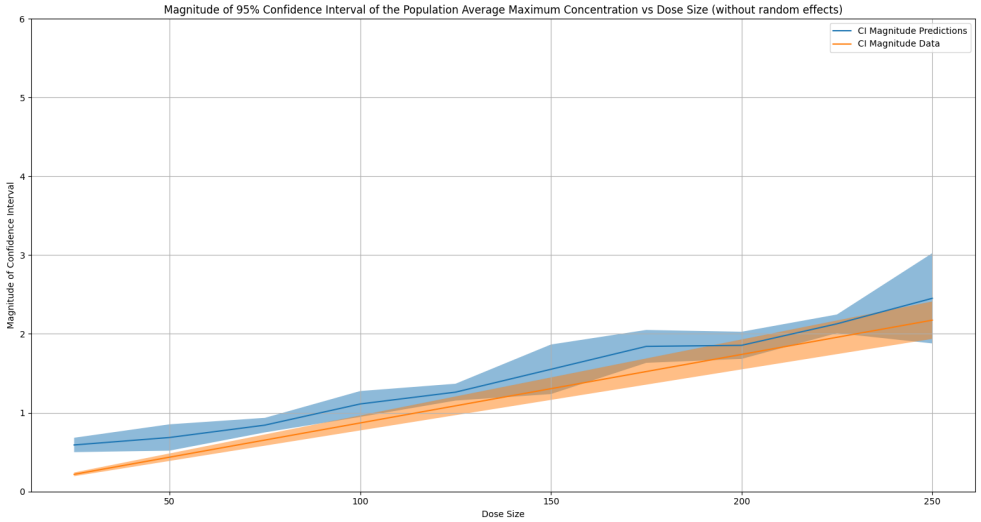



Figure 3.11: SWAG uncertainty as a function of dose size when trained on data without random effects and a SWAG learning rate of 1e-4

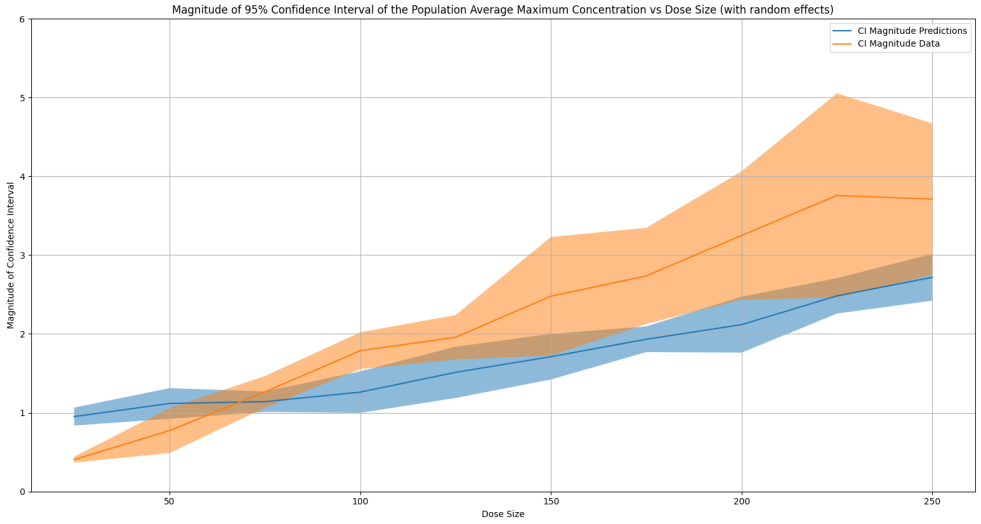



Figure 3.12: SWAG uncertainty as a function of dose size when trained on data with random effects and a SWAG learning rate of 1e-4.

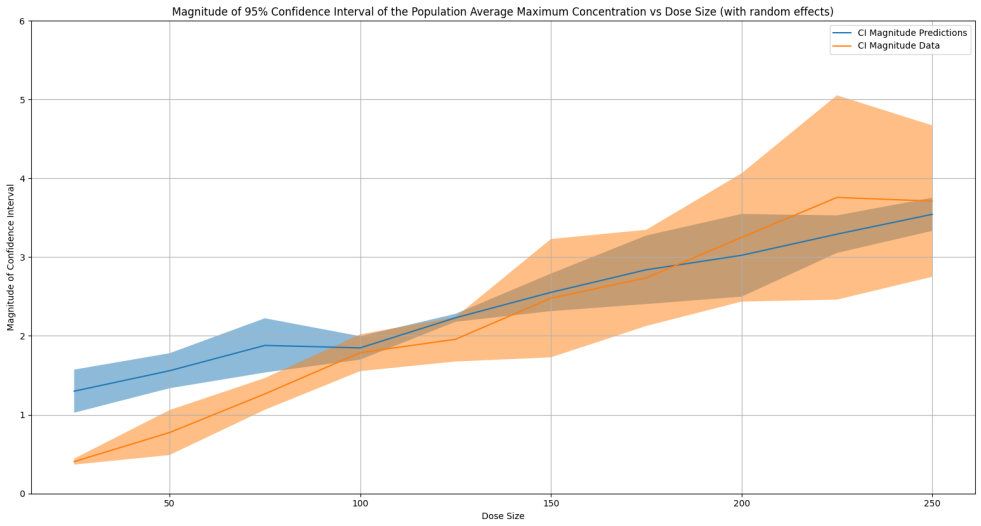



Figure 3.13: SWAG uncertainty as a function of dose size when trained on data with random effects and a SWAG learning rate of 5e-4.

### 3.4.2 Analysis

SWAG shows promise in capturing the uncertainty of the data-generating distribution it was trained on. Unfortunately, when controlling for learning rate, the difference in uncertainty between data with random effects and data without random effects is not as large as we had hoped. Instead, the main difference comes from the learning rate used to sample weights during SWAG. Clearly, the step sizes should have a significant effect on the posterior distribution that SWAG produces, but unfortunately, the effect from the data being trained on is not as significant as expected.

Thus, to effectively model the uncertainty in the new dosing regimes using SWAG, one would need to tune the learning rate based on the uncertainty in the training data. In the plots above, we can see that estimated model uncertainty from SWAG can effectively predict the uncertainty in the data for larger dose sizes when the estimated model uncertainty is at least as large or larger than the data uncertainty within the training dosing sizes, which in this case is the range from 0 to 100 mg.

Another drawback of the current implementation of uncertainty quantification is that there is no way to identify the different sources of uncertainty. This means that SWAG cannot be used to determine the magnitude of the random effects to include in your model. With SWAG, all of the uncertainty of the model is combined, so that the uncertainty in the impact of the covariates is combined with the uncertainty in the effects of the dosing sizes. One possible future direction to find the sources of uncertainty could be to try to freeze different parts of the model, for example, the encoding of the patient covariates, and test if this could be used to pinpoint the various sources of uncertainty.

### 3.4.3 SHAP Analysis

To demonstrate the results of SHAP analysis, we present beeswarm summary plots generated using the Python library shap. The beeswarm plot shows the effect of the various inputs to the model on the final output. The beeswarm plot has a color bar that tells us the relative value of the input, and along the x-axis, we have the SHAP values. To simplify the analysis as much as possible, we kept the current concentration, number of doses taken, and time since the last dose constant at 0, 1, and 0, respectively. This is how the beginning of all PK trajectories start. The beeswarm plot provides a clear visualization of the trends in the data, as shown below.

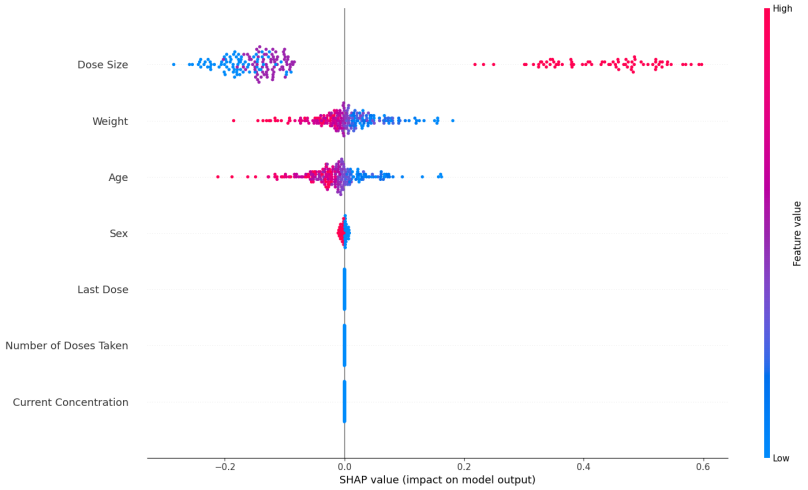



Figure 3.14: SHAP Analysis when model is trained on data without random effects.

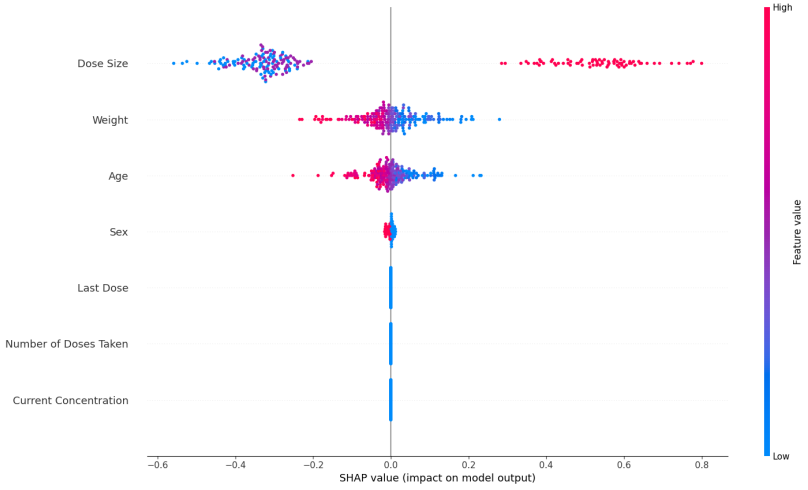



Figure 3.15: SHAP Analysis when model is trained on data with random effects.

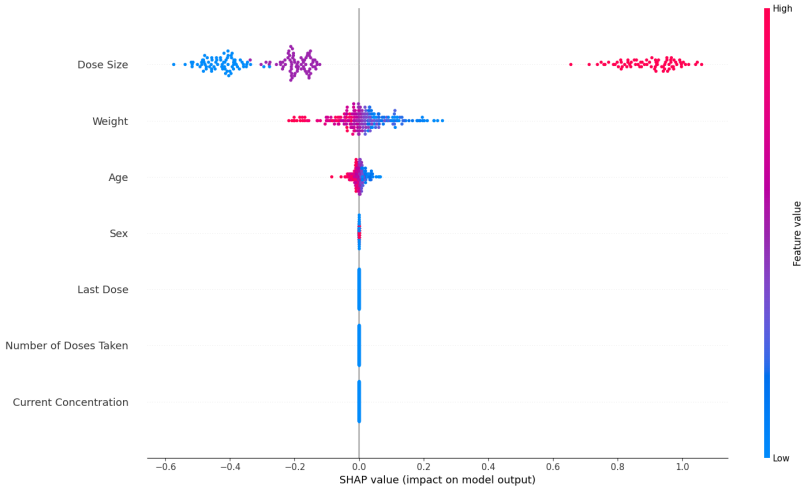



Figure 3.16: SHAP Analysis when model is trained on data with random effects, but only weight has an effect on the underlying PKPD model.

### 3.4.4 Analysis

SHAP analysis was very effective in identifying relevant covariates, even when random effects were included in the training. This result demonstrates that a trained neural ODE model can be used to determine which covariates to include in a mechanistic PKPD model. However, these results were only qualitatively successful. Our models were unable to capture the full quantitative range of the effects of the covariates on the resulting PK trajectories. This could be because the model is trained to keep its predictions of the trajectories close to the mean, in order to minimize the loss as much as possible. This problem is particularly apparent when random effects are included, as these have a larger impact on the resulting trajectories than the covariates themselves.

### 3.4.5 Neural ODE interpreter

To evaluate how well the neural ODE interpreter works, it is a good idea to see whether it can correctly interpret a simple vector field. That is, we define a vector field as any function we would like, and then we see if the interpreter returns the correct ODE. This is a good first step, because if the interpreter is to be able to interpret neural ODEs, then it must be able to interpret explicit ODEs.

By running the interpreter on six different ODEs, we obtain the results shown in Figure 3.17.

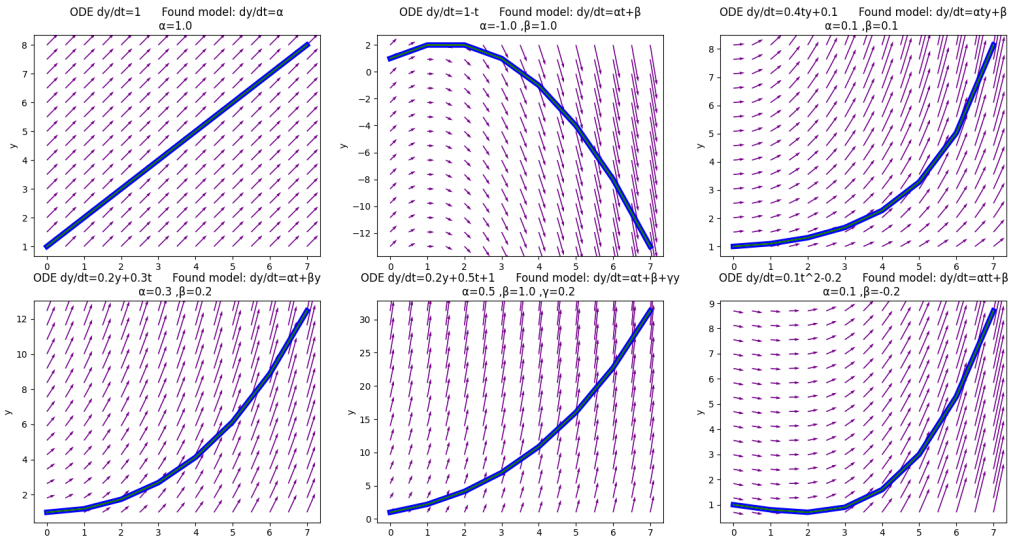



Figure 3.17: Interpreter used on some basic explicit ODEs. The green line represents the true trajectory given to the interpreter, and the blue line represents the trajectory of the interpreted ODE. There are two vector fields in the plot, a blue one representing the interpreted ODE and an orange one representing the true ODE. When these two overlap, they become purple.

It seems that there is no issue for the interpreter to pick up on the underlying ODE in these cases. This allows us to trust it more when it interprets neural ODEs, which is the next step. The setup is as follows: We generate a dataset by simulating ODE trajectories according to some ground truth ODE. We create one main trajectory that contains most data points and some auxiliary trajectories in its vicinity. The neural ODE is then trained on this dataset until it converges, at which point we run the interpreter on the neural ODE. This is done for the same ODEs that were in the previous Figure. See the Figure 3.18 below for the results.

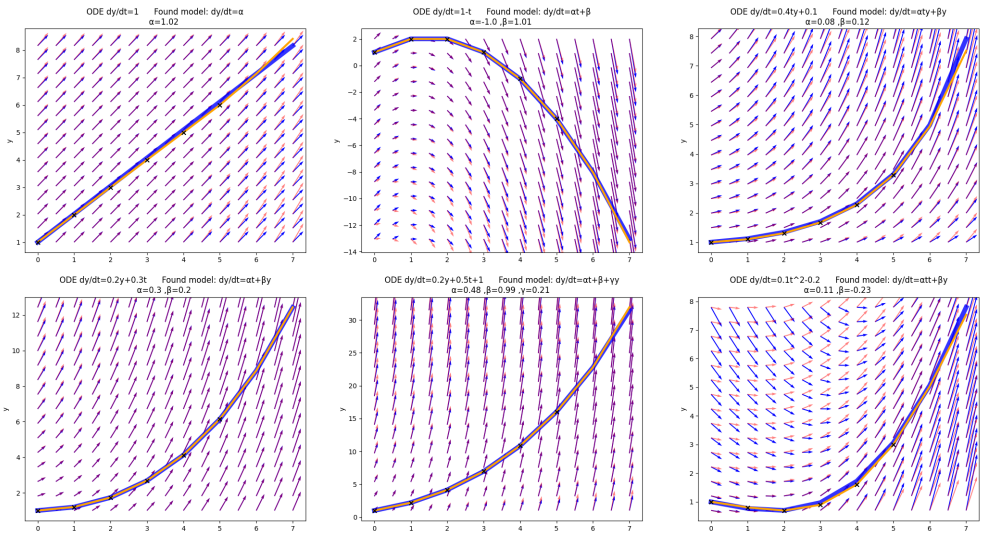



Figure 3.18: Interpreter used on a neural ODE which was trained on some basic explicit ODEs. The green line represents the true trajectory given to the interpreter, and the blue line represents the trajectory of the interpreted ODE. There are two vector fields in the plot, a blue one representing the interpreted ODE and an orange one representing the neural ODE. When these two overlap, they become purple.

It appears that the interpreter is able to interpret the neural ODE, particularly locally around the main trajectory. Given how neural networks work, it makes intuitive sense that the vector field of the neural ODE diverges from a simple ODE expression for points far away from the training data. The issue is more with the neural ODE rather than with the interpreter, as it appears that neural ODEs only learn a neat closed-form ODE locally. That the two diverge is not that big of an issue, since one would make most predictions quite close to the training data, so this is where a good interpretation is the most important. In fact, in some settings, the interpreter could be used as an improvement of neural ODEs, by first interpreting it locally and then using the interpretation as the global prediction!

Due to time constraints, the interpreter did not get tested for the PKPD model. Nevertheless, this serves as proof of concept that a closed-form expression for neural ODEs is possible, at least in the region close to the underlying dataset.

## 3.5 Research Question 5

### 3.5.1 Results of Adverse Effects Model

Due to the noisy nature of the adverse effects data, we cannot expect the model to be able to predict every single fluctuation in the dataset correctly. The goal lies more in capturing the trends of the dataset. We must therefore make sure that the model does not overfit to the training data, which we can see by investigating the model's prediction on the training data, see Figure 3.19.

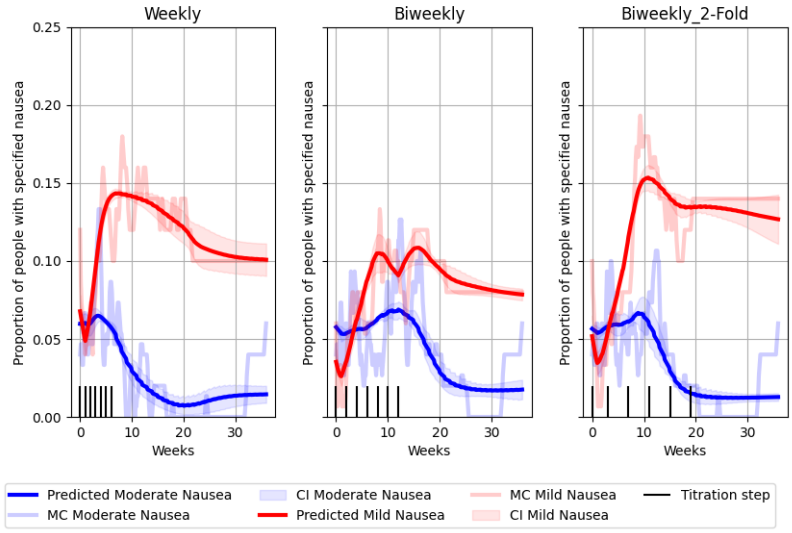



Figure 3.19: The model's prediction on the titration schemes in the training data

It appears that the model follows the data nicely, without being overly expressive. The prediction of the model on the eight unseen titration schemes can be seen in Figure 3.20.

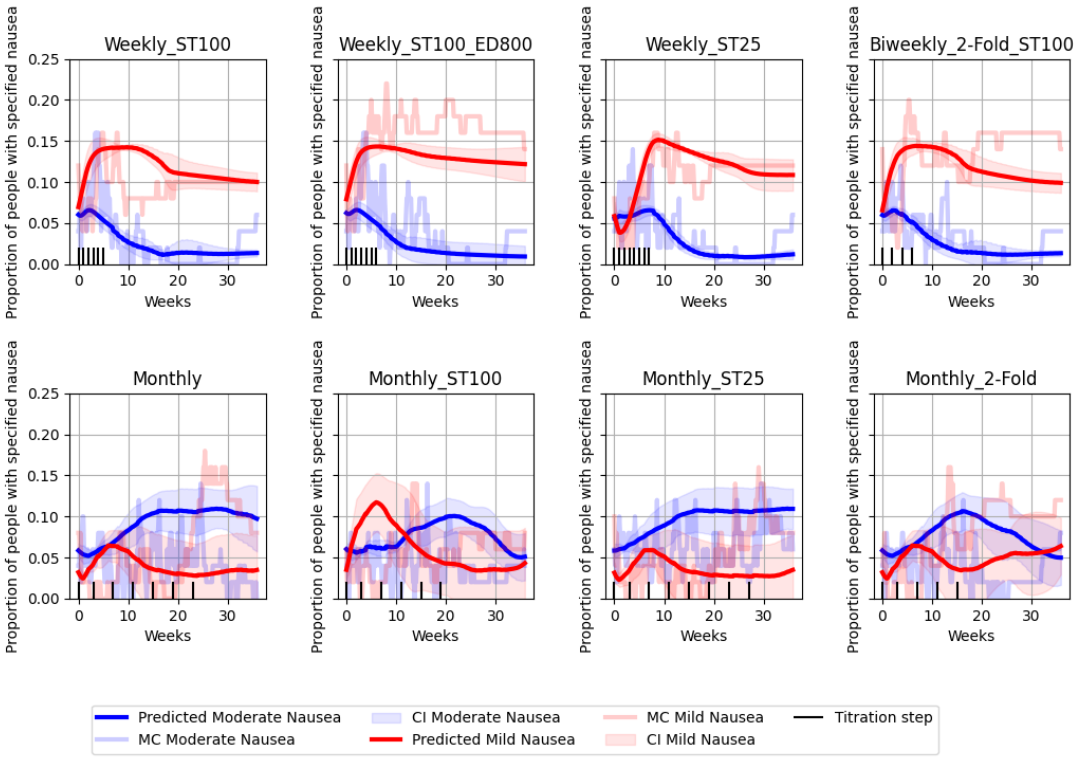



Figure 3.20: The model's prediction on the titration schemes in the test data

We then reran the same tests for different combinations of titration schemes in the training data as a way to find the best combination. Instead of trying every single combination of three titration schemes as the training data, we limit ourselves to only the combinations where there is exactly one weekly, one bi-weekly, and one monthly titration scheme in the training data. These are the most diverse training sets possible, which we therefore expect to give the best performance. By ignoring combinations that are expected to perform worse, a lot of computational power is saved. See figure 3.21 for the results.

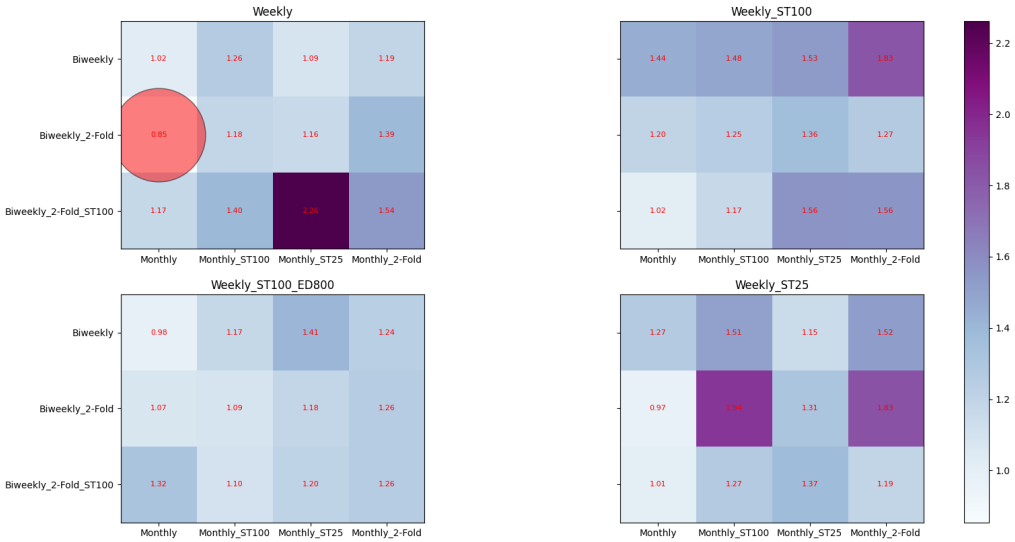



Figure 3.21: MSE-loss on the test-set for different variations of training data. The best-performing model is marked by a red circle.

The results from the grid search indicate that the three best titration schemes to have in the training data are Weekly, Biweekly-2-Fold, and Monthly. It is then interesting to see what the extrapolation capabilities look like in this best-case scenario. This is illustrated in Figure 3.22.

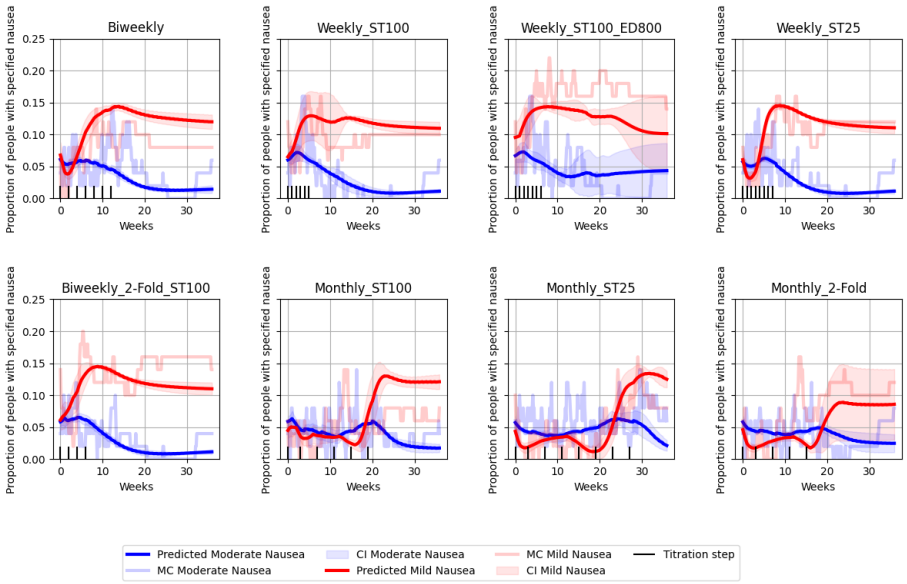



Figure 3.22: Model performance on the titration schemes in the test data, when the model is trained on the optimal combination of titration schemes.

### 3.5.2 Analysis

In the first case with the original set-up, the model appears to make reasonable predictions for the top four titration schemes, while struggling with the four at the bottom. The reason for this is most likely explained by the fact that the bottom four titration schemes have monthly titration. This was not present in the training data, and so the model struggles. However, the predictions are not entirely off, and one should keep in mind that the underlying data itself is from a model that contains its own assumptions and flaws. Furthermore, the data is, as previously mentioned, highly variable.

The fact that the model struggles with the titration schemes that are vastly different from those in the training dataset is not surprising, but it did, however, raise an interesting question. How well would the model perform if we changed which titration schemes are present in the training data? Perhaps a more diverse training set will allow the model to generalize even better. This led us to run the grid search for the optimal combination of titration schemes to have in the training data. Investigating this is not only interesting from a model evaluation view, but also from a practical application point of view. If changing the titration schemes in the training data indeed increases performance, then this suggests that those titration schemes may be a good starting point in real clinical trials, as they appear to be highly informative.

Examining the results of the best-case scenario, the model now handles monthly titrations significantly better than previously, as expected. Note that the model is still able to predict well on the other types of titration schemes. The worst prediction is now on the titration scheme that ends at 800mg, which is expected, as there is no example of 800mg doses in the training data.

In conclusion, the model performs well in predicting new titration schemes, provided that they are not significantly different from those in the training data. This is to be expected of any model, the further you move away from the training data the worse the performance is. Furthermore, if one were to put this model to use on real data, it could be beneficial to test the Weekly, Biweekly-2-Fold, and the Monthly titration schemes, so that the data from these trials can be used for training the model.

If we had gotten a fully working neural ODE interpreter in Section 3.4.5, it would have been interesting to see what it would make of this model and if it could have given insight into the underlying mechanics. Nevertheless, these results demonstrate the usefulness of neural ODEs even in a case where ODEs do not generate the data. While this model did not achieve the same extrapolation capabilities as the PKPD model, the lack of mechanistic understanding of the problem was not a significant issue. This answers our fifth and final research question.

# 4

# Conclusions and Future works

In conclusion, the project has successfully demonstrated the benefits of using neural ODEs for discriminative PKPD modeling, as evidenced by a comparison with other models. The advantages of the neural ODE model are especially evident in its ability to extrapolate to new dose sizes, where it vastly outperforms the other contenders. While the different models can extrapolate quite well for dosing sizes close to the training data, the neural ODE achieves remarkable results for dose sizes that are more than twice the size of those in the training data. Furthermore, the most significant problem with neural ODEs, their training time, is addressed through the flow-matching technique. This significantly decreases the training time of the neural ODE, although it still takes longer than the other models.

The model also showed the ability to accurately capture information about the underlying PKPD model, which could then be used to build a mechanistic PKPD model. This is done partly through the uncertainty quantification of the model using the SWAG technique, which demonstrably follows the underlying uncertainty of the data well, although without giving an account for the source of the uncertainty. Furthermore, SHAP values are used to study the ability of the model to pick up on which covariates have an effect on the PKPD response. The model performs quite well in this regard, indicating that it could be used to investigate the importance of covariates. To allow for even more interpretability of the model, a technique for going from neural ODE to a closed-form ODE expression was developed. Although this technique was not tested for the PKPD model, it shows promising results for simple toy examples.

The project also finds that neural ODEs allowed for a better Generative model for PKPD time series. By adding a neural ODE component, the quality of the samples improved significantly, reflecting both the mean and the spread of the population more accurately.

Lastly, a model for the mechanistically complex area of Adverse Effects was developed. This model demonstrates that neural ODEs can be effectively utilized when the data is generated from a Markov Chain model. The model also gives some insight into which titration schemes carry the most information for model training.

While the project led to many results and conclusions, there are still many areas of

the project that could be investigated even further.

Firstly, an interesting next step would be to use real-world clinical data, which would be a great way to demonstrate real-world applicability of the model. One way this could be done is by comparing the conclusions reached by using this model with the ones reached by the current PKPD modeling process. If they reach similar conclusions, that would indicate great utility in this type of model. Even if the conclusions differ, it may still be of interest, as one could then investigate why this is the case and which of the two approaches is more trustworthy.

When one uses the model on real-world data, many new challenges can be faced. For example, the time points of the samples would most likely not always be the same. As mentioned previously in the thesis, we have investigated a setup where the samples are taken within a window of 1 hour. We saw that this had little to no effect on the performance, which we attributed to the fact that the PKPD curves are relatively flat at these time points, and therefore, the exact time point where the sample is taken does not matter. If the time window for the samples were larger, than this could start to degrade the performance of our model. However, we would expect that the neural ODE model would be the least effected compared to the RNN or MLP, due to its continuously defined latent state. Because this latent space is defined very finely, the model in effect learns what the concentration should be at any time in the time series, not just at the sampled times.

Another way in which the real-world case might be different is that there might be missing data. One must expect that for some patients, it was not possible to get a sample at every point. Due to the structure of the data we expect this to be remedied by a simple data insertion technique. For example, if the peak value after the third dose is missing, one could insert the mean of the peak values after the second and fourth doses. The cyclical nature of the PKPD time series affords us the ability to perform this type of insertion. We expect that this would work well in most cases, and the results should only be slightly affected.

Typically, we would expect real-world PKPD data to contain a high amount of noise, which could create significant challenges for our model. However, the underlying model already contains high amounts of random effects, while still achieving great performance. The model is therefore able to handle noisy data well, which indicates that this should not be an issue for use in the real world. It is important to note that this most likely requires datasets with a population size similar to those used in our project. We had 100 patients in each cohort, which would be realistic for real-world clinical trials. Noise from other sources, such as measurement error, which are not included in any sort of underlying model, could also be present in real world data. However, assuming the noise is at a typical level for medical test results, this would probably also not significantly affect the results.

Regarding how the model can be applied in practice, an effort could be made to automate PKPD modeling. However, due to the lack of interpretability, this is

not a realistic goal. Instead, it is more reasonable to use the model in conjunction with traditional methods. One way this could be done is by using SHAP-values on neural ODE model to identify relevant covariates, which gives information about how the traditional PKPD model should be set up. Another approach is to use both methods and compare their results, which could lend more credibility to the process.

Going back to future works, one could also investigate how the model performs when a more complicated underlying PKPD model generates the data. It would be interesting to see whether the results are similar for a two-compartment or a three-compartment PK model. Along the same lines, it would be interesting to investigate what happens when one changes the dosing regimens on which the model is trained. This could indicate which dosing regimens are the most informative, helping the decision of which dosing regimens to test.

It would also be interesting to see further investigation into the interpretability of neural ODEs. While our project shows some hope in understanding what goes on in the black-box, there is still a long way to go before we can completely understand our models. As mentioned earlier, this would be of particular interest for our application of PKPD modeling. This is because a good understanding of the model plays a crucial part in determining its trustworthiness and, hence, its practical usefulness.

# Bibliography


[1] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural ordinary differential equations*, 2019. arXiv: 1806.07366 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1806.07366.

[2] J. Lu, K. Deng, X. Zhang, G. Liu, and Y. Guan, "Neural-ode for pharmacokinetics modeling and its advantage to alternative machine learning models in predicting new dosing regimens," *iScience*, vol. 24, p. 102 804, Jun. 2021. DOI: 10.1016/j.isci.2021.102804.

[3] D. Duvenaud, "Automatic model construction with gaussian processes," Ph.D. dissertation, Apollo - University of Cambridge Repository, 2014. DOI: 10.17863/CAM.14087. [Online]. Available: https://www.repository.cam.ac.uk/handle/1810/247281.

[4] B. Meibohm and H. Derendorf, "Basic concepts of pharmacokinetic/pharmacodynamic (pk/pd) modeling," *International journal of clinical pharmacology and therapeutics*, vol. 35, pp. 401–13, Nov. 1997.

[5] CUSABIO-Team, *What is compartment modelling in pharmacokinetics?* 2025. [Online]. Available: https://www.cusabio.com/c-21197.html.

[6] D. Valderrama, A. V. Ponce-Bobadilla, S. Mensing, H. Fröhlich, and S. Stodtmann, "Integrating machine learning with pharmacokinetic models: Benefits of scientific machine learning in adding neural networks components to existing pk models," *CPT: Pharmacometrics & Systems Pharmacology*, vol. 13, no. 1, pp. 41–53, 2024. DOI: https://doi.org/10.1002/psp4.13054. eprint: https://ascpt.onlinelibrary.wiley.com/doi/pdf/10.1002/psp4.13054. [Online]. Available: https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1002/psp4.13054.

[7] L. Keutzer, H. You, A. Farnoud, *et al.*, "Machine learning and pharmacometrics for prediction of pharmacokinetic data: Differences, similarities and challenges illustrated with rifampicin," *Pharmaceutics*, vol. 14, no. 8, 2022, ISSN: 1999-4923. DOI: 10.3390/pharmaceutics14081530. [Online]. Available: https://www.mdpi.com/1999-4923/14/8/1530.

[8] A. Tang, "Machine learning for pharmacokinetic/pharmacodynamic modeling," *Journal of Pharmaceutical Sciences*, vol. 112, no. 5, pp. 1460–1475, 2023, ISSN: 0022-3549. DOI: https://doi.org/10.1016/j.xphs.2023.01.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022354923000126.

[9] M. Uno, Y. Nakamaru, and F. Yamashita, "Application of machine learning techniques in population pharmacokinetics/pharmacodynamics modeling,"

*Drug Metabolism and Pharmacokinetics*, vol. 56, p. 101 004, 2024, ISSN: 1347-4367. DOI: https://doi.org/10.1016/j.dmpk.2024.101004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1347436724000107.

[10] E. Sibieude, A. Khandelwal, P. Girard, J. S. Hesthaven, and N. Terranova, "Population pharmacokinetic model selection assisted by machine learning," *Journal of Pharmacokinetics and Pharmacodynamics*, vol. 49, no. 2, pp. 257–270, Apr. 2022, ISSN: 1573-8744. DOI: 10.1007/s10928-021-09793-6. [Online]. Available: https://doi.org/10.1007/s10928-021-09793-6.

[11] O. Qutishat, *Development of artificial neural networks for the prediction of outlying and influential individuals from pharmacokinetic and pharmacodynamic models*, 2022.

[12] C. Ogami, Y. Tsuji, H. Seki, *et al.*, "An artificial neural networkpharmacokinetic model and its interpretation using shapley additive explanations," *CPT: Pharmacometrics & Systems Pharmacology*, vol. 10, no. 7, pp. 760–768, 2021. DOI: https://doi.org/10.1002/psp4.12643. eprint: https://ascpt.onlinelibrary.wiley.com/doi/pdf/10.1002/psp4.12643. [Online]. Available: https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1002/psp4.12643.

[13] N. A. Daryakenari, S. Wang, and G. Karniadakis, *Cminns: Compartment model informed neural networks – unlocking drug dynamics*, 2024. arXiv: 2409.12998 [q-bio.QM]. [Online]. Available: https://arxiv.org/abs/2409.12998.

[14] H. Yu, S. Ueckert, L. Zhou, *et al.*, "Exposureresponse modeling for nausea incidence for cotadutide using a markov modeling approach," *CPT: Pharmacometrics  Systems Pharmacology*, vol. 13, pp. 1582–1594, Jul. 2024. DOI: 10.1002/psp4.13194.

[15] Alexander Amini and Ava Amini, *Mit introduction to deep learning*, 2025. [Online]. Available: https://introtodeeplearning.com/.

[16] M. M. Hammad, *Artificial neural network and deep learning: Fundamentals and theory*, 2024. arXiv: 2408.16002 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2408.16002.

[17] R. M. Schmidt, *Recurrent neural networks (rnns): A gentle introduction and overview*, 2019. arXiv: 1912.05911 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.05911.

[18] Y. Rubanova, R. T. Q. Chen, and D. Duvenaud, *Latent odes for irregularly-sampled time series*, 2019. arXiv: 1907.03907 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1907.03907.

[19] D. S. Bräm, U. Nahum, J. Schropp, M. Pfister, and G. Koch, "Low-dimensional neural odes and their application in pharmacokinetics," *Journal of Pharmacokinetics and Pharmacodynamics*, vol. 51, no. 2, pp. 123–140, Apr. 2024, ISSN: 1573-8744. DOI: 10.1007/s10928-023-09886-4. [Online]. Available: https://doi.org/10.1007/s10928-023-09886-4.

[20] I. B. Losada and N. Terranova, "Bridging pharmacology and neural networks: A deep dive into neural ordinary differential equations," *CPT: Pharmacometrics & Systems Pharmacology*, vol. 13, no. 8, pp. 1289–1296, 2024. DOI: https:

//doi.org/10.1002/psp4.13149. eprint: https://ascpt.onlinelibrary.wiley.com/doi/pdf/10.1002/psp4.13149. [Online]. Available: https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1002/psp4.13149.

[21] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, *Flow matching for generative modeling*, 2023. arXiv: 2210.02747 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2210.02747.

[22] R. Irwin, A. Tibo, J. P. Janet, and S. Olsson, *Semlaflow – efficient 3d molecular generation with latent attention and equivariant flow matching*, 2025. arXiv: 2406.07266 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2406.07266.

[23] J. Ho, A. Jain, and P. Abbeel, *Denoising diffusion probabilistic models*, 2020. arXiv: 2006.11239 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2006.11239.

[24] Y. Li, *Ts-diffusion: Generating highly complex time series with diffusion models*, 2023. arXiv: 2311.03303 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2311.03303.

[25] X. Yuan and Y. Qiao, *Diffusion-ts: Interpretable diffusion for general time series generation*, 2024. arXiv: 2403.01742 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2403.01742.

[26] S. Kim, J. Yoo, J. Kim, Y. Cha, S. Kim, and S. Hong, *Simulation-free training of neural odes on paired data*, 2024. arXiv: 2410.22918 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2410.22918.

[27] X. Zhang, Y. Pu, Y. Kawamura, *et al.*, *Trajectory flow matching with applications to clinical time series modeling*, 2024. arXiv: 2410.21154 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2410.21154.

[28] K. Baron and M. R. Gastonguay, "Simulation from ode-based population pk/pd and systems pharmacology models in r with mrgsolve," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:63947586.

[29] W. He, Z. Jiang, T. Xiao, Z. Xu, and Y. Li, *A survey on uncertainty quantification methods for deep learning*, 2025. arXiv: 2302.13425 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2302.13425.

[30] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, *Averaging weights leads to wider optima and better generalization*, 2019. arXiv: 1803.05407 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1803.05407.

[31] W. Maddox, T. Garipov, P. Izmailov, D. Vetrov, and A. G. Wilson, *A simple baseline for bayesian uncertainty in deep learning*, 2019. arXiv: 1902.02476 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1902.02476.

[32] S. M. Lundberg and S. Lee, "A unified approach to interpreting model predictions," *CoRR*, vol. abs/1705.07874, 2017. arXiv: 1705.07874. [Online]. Available: http://arxiv.org/abs/1705.07874.

[33] L. S. Shapley, *A Value for N-Person Games*. Santa Monica, CA: RAND Corporation, 1952. DOI: 10.7249/P0295.

[34] S. Lipovetsky and M. Conklin, "Analysis of regression in game theory approach," *Applied Stochastic Models in Business and Industry*, vol. 17, no. 4, pp. 319–330, 2001. DOI: https://doi.org/10.1002/asmb.446. eprint: https:

//onlinelibrary.wiley.com/doi/pdf/10.1002/asmb.446. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/asmb.446.

[35] C. Herrera, F. Krach, and J. Teichmann, *Neural jump ordinary differential equations: Consistent continuous-time prediction and filtering*, 2021. arXiv: 2006.04727 [stat.ML]. [Online]. Available: https://arxiv.org/abs/2006.04727.

[36] N. Mallik, E. Bergman, C. Hvarfner, *et al.*, *Priorband: Practical hyperparameter optimization in the age of deep learning*, 2023. arXiv: 2306.12370 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2306.12370.

# 5

# Appendix A

See Figure 5.1 for the comparison of generative models for the second dosing regimen.

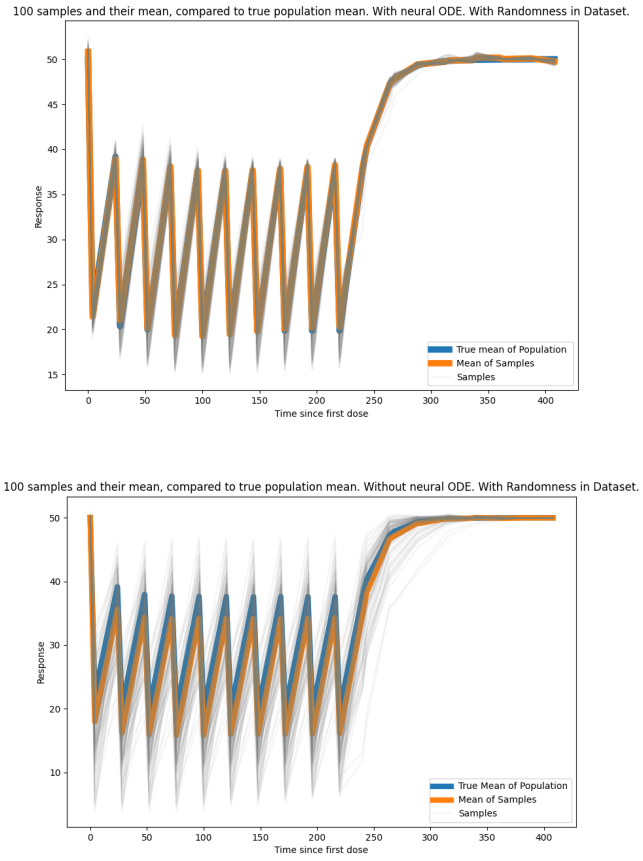



Figure 5.1: Comparison of the model with neural ODE (top) and the model without (bottom). 100 samples are generated from each, the mean of these samples are calculated and compared to the true population mean. The input into the diffusion model is that the dosing regimen is 5 doses of 100mg.

# 6

# Appendix B

## Uncertainty Quantification

While the Uncertainty Quantification mentioned in the main body of the thesis provides substantial insight into the uncertainty of our model, we had hoped to achieve a more precise metric. The goal was to have the standard deviation create a confidence interval for the true underlying population mean. This way, one can interpret the resulting standard deviation probabilistically, similar to the standard deviations in purely statistical models. Otherwise, the standard deviation merely measures the spread of predictions, allowing for less strong statements about how sure you are that the population lies in some interval. The idea behind this topic was to develop a type of uncertainty quantification that aligns with the approach used by pharmacometricians in their PKPD modeling. We sought to establish a connection between the two.

The focus here is on trying to empirically prove that the estimated uncertainty has some real meaning, and not on finding a new method for uncertainty quantification. The methods used for the uncertainty quantification are Monte Carlo Dropout and Bagging.

In the real world, developing a test for the quantified uncertainty becomes a very tough task. However, due to our access to unlimited simulated data, we can actually investigate this in a practical manner. To understand how, we start by clearly stating what we want from our uncertainty metric. We want: "Given some dataset that we train our model on, we want a metric that allows us to determine the probability that the true value lies within some range of the predicted value." In particular, we want there to be a 68% probability of the true value lying within 1 standard deviation of the prediction, and a 95% probability of the true value lying within 2 standard deviations. For any specific uncertainty quantification method, we can test if this is the case.

In our particular case, the procedure is the following: We first simulate 21 datasets, each with its own level of noise and populations with varying covariates. Models are then trained on the datasets, using either MCDropout or Bagging. These models then extrapolate to two new dosing regimens: 5 doses of 25mg and 5 doses of 150mg. At this point, we check the proportion of times the true mean lies within either one or two standard deviations of the predicted mean. If the standard deviation is

correctly tuned according to wishes specified above, then we would expect the mean to lie within one standard deviation approximately 68% of the time, and within two standard deviations approximately 95% of the time.

It is important to note that these proportions are calculated individually for each of the different time points. This is because the uncertainty at each time point varies, and we want to know how well calibrated the uncertainty is at each time point. If we were to bundle all time points, the ratios would be much better than in practice. For example, we always know that the concentration at time zero is 0, and so the model will always get this correct, and therefore the true value will always lie within one standard deviation. We do not want this to affect our evaluation of the uncertainty quantification at the peaks and the troughs, which are the points we are actually interested in.

We visualize the results of the approach by plotting time series of the proportion of times the true mean was within either 1 or 2 standard deviations, respectively. As we are interested in the peaks and troughs, we mostly care about the proportions at those sample points. As mentioned above, we tested two different methods of uncertainty quantification, Monte Carlo Dropout and Bagging. Their results are presented in Figures 6.1 and 6.2, respectively.

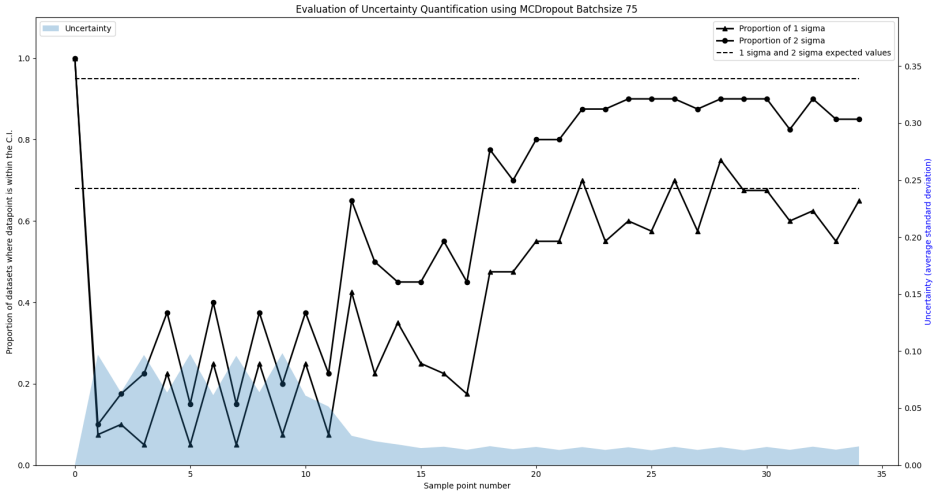



Figure 6.1: Evaluation of the Uncertainty Quantification through MCDropout

In both cases, the results clearly indicate that the standard deviations do not have the interpretations we wished for.

While the approach was an interesting way to utilize our access to infinite data, it was not something where you could expect good results. Because there is no clear connection between the two uncertainties we tried to connect, the test was a long shot. Another issue with the approach lies in the fact that in order for the uncertainty estimation to be deemed good, the extrapolation also has to perform well.

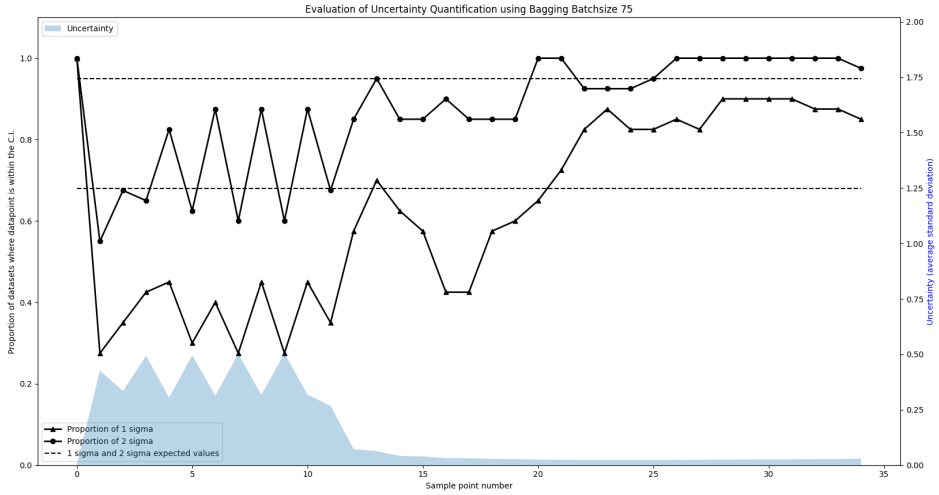



Figure 6.2: Evaluation of the Uncertainty Quantification through Bagging

Because of this, the fact that the proportions are too low might not just be the result of bad methods of uncertainty estimation, but rather because of poor extrapolation. While our main model does extrapolate well overall, the setup in this particular test makes the extrapolation a bit tougher. For example, it was not possible to allow for dropout while also keeping the same extrapolation capabilities. Furthermore, the number of models being trained allows for much less detailed hyperparameter tuning.

While this approach was naive in retrospect, with some modification, it could probably have given better results. One could either try more diverse methods for the uncertainty quantification, or move away from the necessity of a normal distribution towards something more realistic. While we had ideas on how we might make this approach better, we ultimately opted to explore other parts of the project, which we deemed to be more promising.

# 7

# Appendix C

## Jump Models

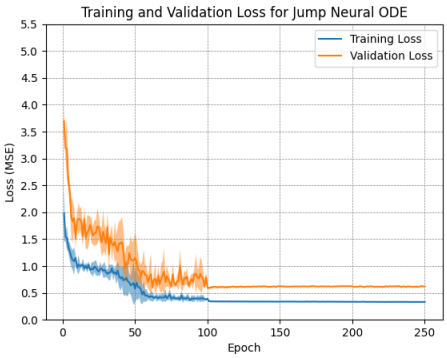
The initial results of the comparison of the different models (see Results for Research Question 1) showed that the neural ODE strongly outperformed the other two models in the ability to extrapolate across dose size, which was given the highest priority in terms of comparisons. However, we suspected that the performance of the models could be significantly improved by adjusting the architectures of our models to better reflect the underlying process we were trying to capture. By integrating the structure of this process into our models, we hoped that the performance of the RNN and MLP could reach parity with the neural ODE model.

These new models, referred to as jump models, consisted of two components: a "jump network" that could capture the increase in concentration resulting from taking a dose, and a second network, identical to the original models, that would capture the decay of the concentration. This "jump network" approach is similar to that found in [35], except that instead of updating the state with an observation, it is updated with a predicted peak concentration based on dose size and patient covariates. We hypothesized that such a model would prove better at extrapolation, as it would allow the model to separate the effects of the clearance of the drug and the irregular perturbations to the system due to dosing intake.

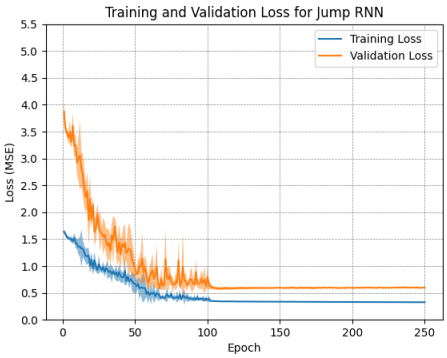
While these models did lead to some small performance improvement for the RNN, the difference in performance was not enough to justify including them in the main body of the text. It is interesting to note that the performance of the neural ODE model was made significantly worse, further supporting the conclusion that it is the smoothness of the neural ODEs that contributes to their performance.

The results for the Jump models on PK data are shown on the next page.

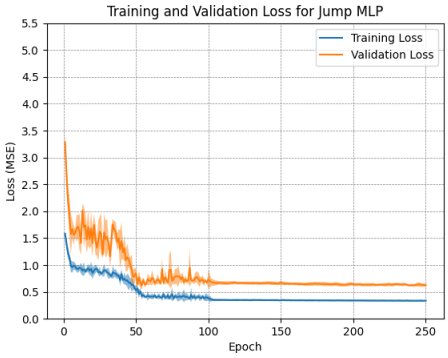
## Results for the Jump Models on PK Data




(a) Jump ODE Loss

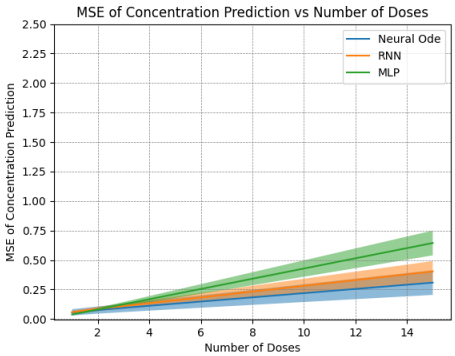



(b) Jump RNN Loss

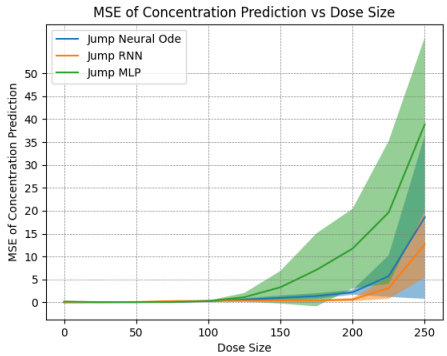



(c) Jump MLP Loss

Figure 7.1: Training/validation Losses for jump models



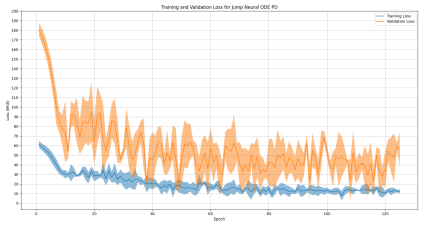

(a) Number of Doses



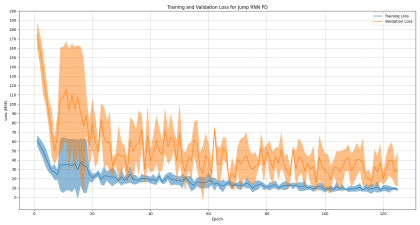

(b) Dose Size

Figure 7.2: Extrapolation across dose size and number of doses for jump models

## Results for the Jump Models on PD Data

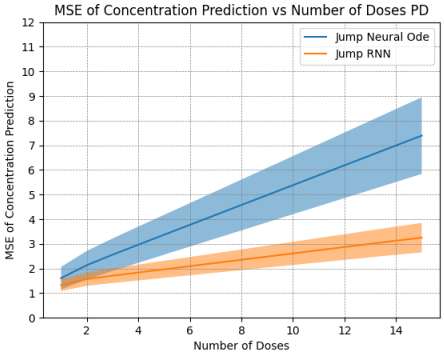


(a) Jump ODE PD Training/Validation Loss

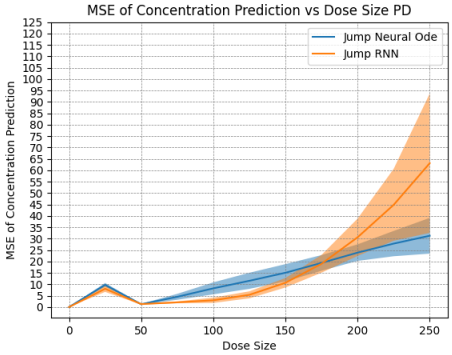


(b) Jump RNN PD Training/Validation Loss

Figure 7.3: Training/validation Losses for base models on PD data




(a) Number of Doses




(b) Dose Size

Figure 7.4: Extrapolation across dose size and number of doses for models trained on PD data

# 8

# Appendix D

In this Appendix we go into detail on the architectures of the different models implemented over the course of the thesis.

## Architecture of the PK and PD Models used for Research Question 1

Unlike what was commonly found in other papers involving using neural ODEs for time series, in this model, the latent space is the same as our output space. This means that the trajectory of the neural ODE is the concentration, rather than a higher-dimensional representation. Initially, models utilizing a high-dimensional latent space were tested. However, it turned out that the encoding and decoding of this hidden space ultimately proved to be superfluous, leading to increased overfitting and poor generalization. This discrepancy may be because this model does not incorporate any time series data to produce its predictions, unlike the other models. Time series modeling using neural ODEs is typically performed by providing the model with an initial trajectory and then having it predict its subsequent evolution, whereas our model predicts an entire time series from scratch. Thus, there is not much information to store in a latent space, which may lead to the observed overfitting.

The neural ODE model comprises three different neural networks. Two of the networks act as encoders, one for the patient covariates and the other for dosing information. Included in this is the time since last dose, which is calculated using the time since the first dose at each step of the ODE solver. The separation into two decoders is done because the patient information remains constant throughout the trajectory and therefore only needs to be performed once. Once the encodings are created, they are passed as input into the final network. This is the neural ODE network that calculates the derivative at a point. Together with an ODESolver, this yields the model's output.

This architecture defines a model that is similar to how mathematical models describe the effects of a drug on our body. The change in concentration over time is determined by the dose size, the time since the last dose, and personal characteristics (such as weight, age, and sex). Schematics of the architectures can be seen on the next page.

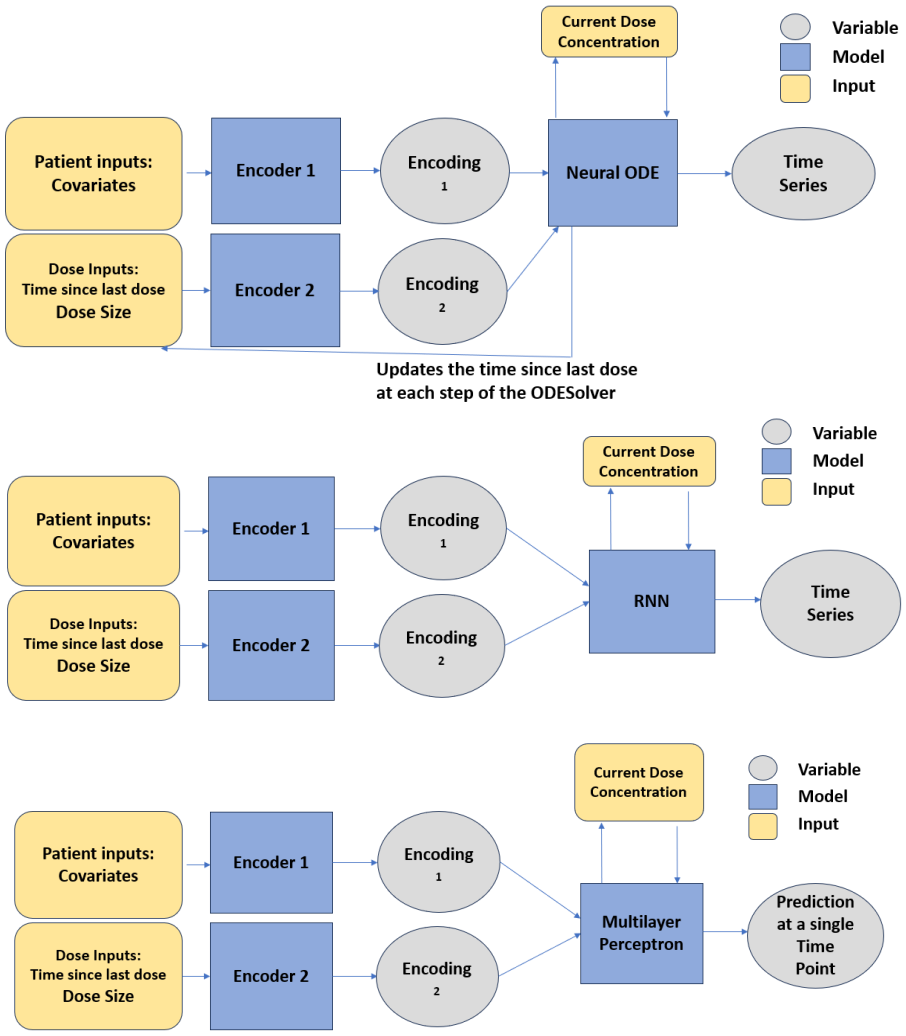



Figure 8.1: The models of the neural ODE model (top), the RNN (middle), and the multilayer perceptron (bottom)

## Architecture of the Generative Models used for Research Question 2

An important detail of the generative models is that the dosing regimen is given as input when generating a sample. The reason for this is that, for any practical purposes, we would want to be able to specify which dosing regimen to generate the time series for. Otherwise, control over the specific dosing regimen is lost, as the sampled encoding could be more representative of another dosing regimen than the one specified in the input. We currently compare two models, both of which use a DDPM as the generative component. The primary difference is that one utilizes neural ODEs, while the other does not. At first, we tried to use a VAE as the generative component, but these models did not achieve satisfactory performance.

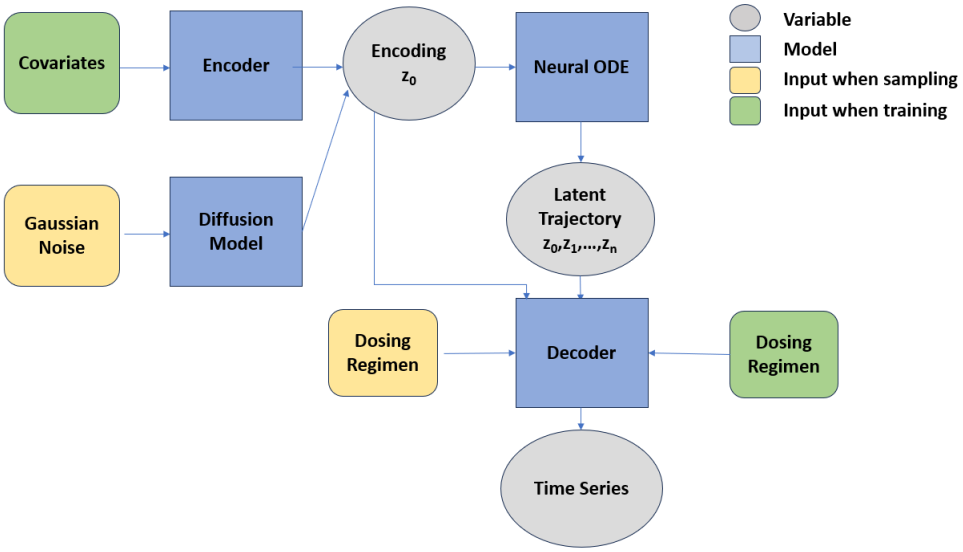



Figure 8.2: Overview of a Generative Model for PKPD time series, using neural ODEs.

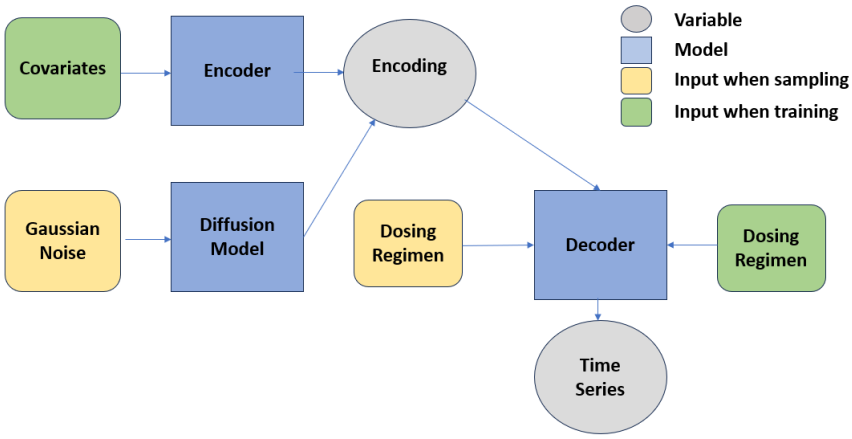



Figure 8.3: Overview of a Generative Model for PKPD time series, without using neural ODEs.

Going into a bit more detail about the architecture, the encoder is a simple MLP that encodes the covariates of a patient into a four-dimensional latent space. This encoding is then used as the initial value of the latent neural ODE, which gets solved in time for the time points where data is sampled. The neural network that parameterizes the derivative is a simple MLP. For each time point in the trajectory, the decoder uses this value, along with encoded information about the dosing regimen, encoded covariates, and the time since the last dose, to make a prediction at the specific time point. Ultimately, this yields a time series. When we want to sample a

time series, we use a diffusion Model to get the encoding, which then goes into the rest of the model.

## Architecture of the Flow Matching PK and PD Models used for Research Questions 3 and 4

The most significant difference in the architecture of the neural ODE model trained with flow matching is that it now contains two separate neural ODEs, one for predicting the increase in concentration and one for predicting the decrease in concentration. This allowed for simpler inputs when modeling the decay, as well as slightly improved performance.

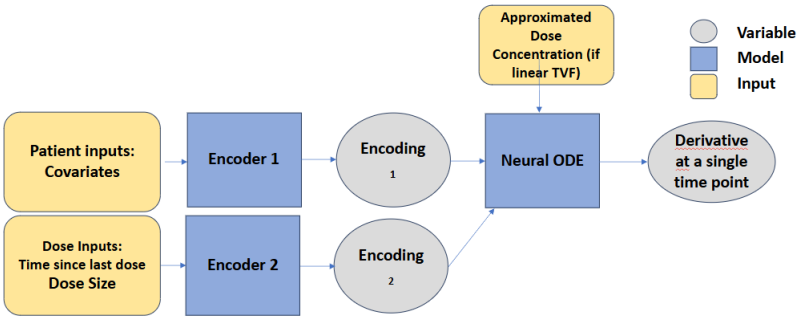



Figure 8.4: Architecture for the neural ODE trained with flow matching for the increase in concentration.

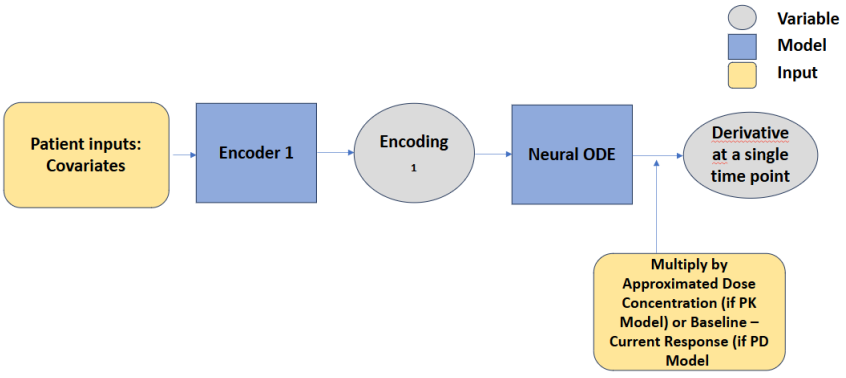



Figure 8.5: Architecture for the exponentially decaying neural ODE trained with flow matching for the decrease in concentration.

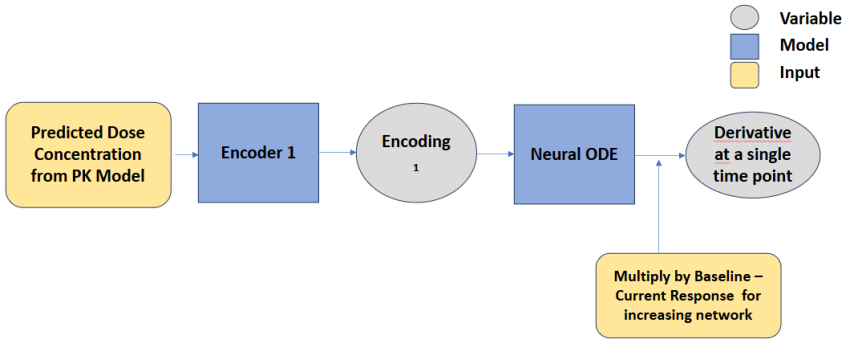



Figure 8.6: Architecture for the PD flow matching model trained with predicted concentration as input.

## Architecture of the Adverse Events Neural ODE Model used for Research Questions 5

We first note that this model was trained with backpropagation through the solver, as the flow matching training techniques were implemented in parallel with work on this model and thus were not ready to be used for this model.

The model encodes the vector of cumulative doses into a two-dimensional latent space, where the encoding is used as the initial value of a neural ODE. This neural ODE then creates a trajectory with 36 time points in the two-dimensional space. The value at each of these time points is then concatenated with the time since the last dose change at that particular time point. These three values are decoded into two values: representing the predicted Mild Nausea and predicted Moderate Nausea at that time point. The decoder is the same for all 36 time points. See Figure 8.7 for an overview of the model.

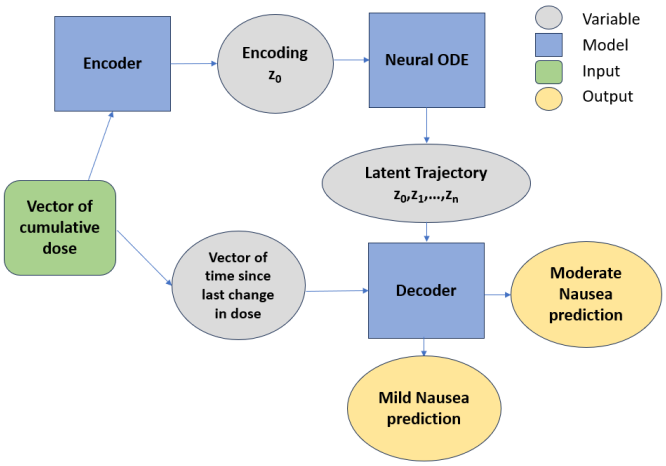



Figure 8.7: Overview of the Adverse Effects model

The rationale behind having a latent dimension of size 2 is that this represents the two different trajectories of the Mild and Moderate predictions. In the neural ODE, each dimension can be seen as its own trajectory, with some dependency on the other trajectory. Intuitively, it makes sense that the trajectories can affect each other, as Mild Nausea and Moderate Nausea are highly correlated.

## Hyperparameter Tuning

To improve the performance of the discriminative neural ODE model, we implemented the PriorBand technique [36] to find the optimal configuration of hyperparameters. Using PriorBand, we tried extensively to find a configuration that could outperform our manual choice for the discriminative neural ODE model. This did not help at all, and through this process, we did not end up with a single configuration that could converge, despite allowing plenty of resources for the hyperparameter search. There are many possible reasons for this. Firstly, the models take quite a lot of resources to converge, which means that getting a good estimate of a configuration's performance requires more resources than we can reasonably allocate to it. Secondly, it is possible that the priors for the hyperparameter space were not well chosen. While it has been shown that even poor priors will yield a good result in the long run, it might be the case that we did not allocate enough computational resources to reach this stage. This could be a possible explanation, however, we believe that the priors chosen were reasonable. The priors were chosen to align with the knowledge gathered from our initial hyperparameter tuning, while also favoring simple models.

Because of the poor performance, PriorBand was not investigated further. It was not used in combination with any of the other models except for the discriminative neural ODE trained with backpropagation through the solver. However, it is very possible that it would have been more effective when used for the flow matching neural ODE model, as the training is much faster, allowing for a quicker exploration of the hyperparameter space.

# 9

# Appendix E

## Implementation of the TVFs

### The Linear TVF

The first implementation of flow matching used the slope between the interpolated data point and the next closest training point to define the flow, its implementation is explained below:

$$
\begin{gathered}
f(t) = mt + b \\
f^{'}(t) = m \\
f(t_0) = x_0 \\
f(t_f) = x_{after} \\
f(t_0) = x_0 \implies m \cdot t_0 + b = x_0 \\
f(t_{after}) = x_{after} \implies x_{after} = m \cdot t_{after} + b \implies \\
x_{after} - x_0 = m \cdot (t_{after} - t_0) \implies m = \frac{x_{after} - x_0}{t_{after} - t_0} = \frac{dx}{dt}_{training}
\end{gathered}
\tag{9.1}
$$

We denote this a linear TVF because any trajectory determined by this vector field will be a straight line. A visualization of the training vector field defined in this manner is shown in Figures 9.1 and 9.2 below.

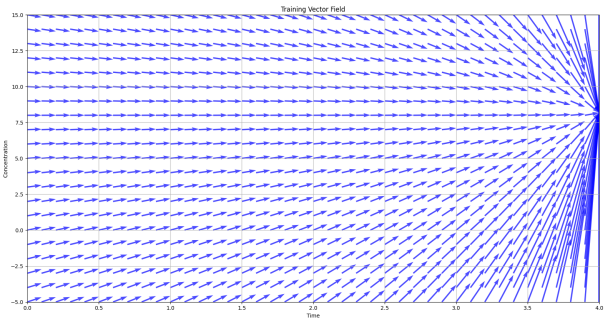



Figure 9.1: Visualization of the vector field with a linear flow, for the increase in concentration after a dose is taken.

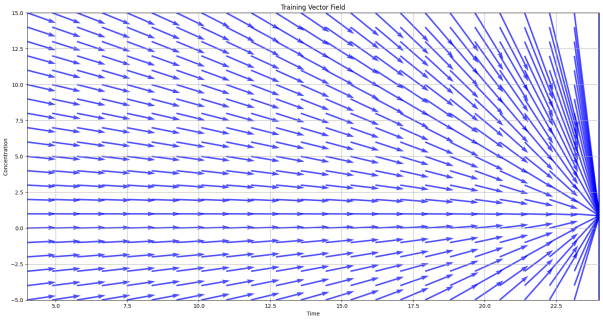



Figure 9.2: Visualization of the vector field with a linear flow, for the decrease in the concentration after concentration has reached its peak.

From the visualization of the vector field, we can see that our conditional field defines a flow that moves towards a fixed point, the location of which is determined by the peak concentration found in our time series data. It is necessary to sample an approximate concentration value, rather than simply using linear interpolation, in order for the neural ODE to learn a more global vector field. Without this sampling procedure, the global behavior of the trajectories would be completely unknown to our model, which would inhibit it from effectively extrapolating to new patient covariates and new dosing regimes. This was confirmed during the initial training of our model with flow matching, where it was found that performance was significantly degraded if the standard deviation was too small.

Training with a vector field of this type worked relatively well, and we could accurately extrapolate to new dosing sizes, however, there were also some significant issues. One conceptual issue is that we know that the shape of the curve during the increase and decrease in concentration is different, however we train them with the same type of vector field. This was a particularly large problem for the decay, as the decrease in concentration should be proportional to the current concentration, not a linear function. This led our model to overshoot at the end of trajectories and end up with negative concentration. Although this is relatively easy to solve by passing our final trajectory through a ReLU function, this is not a very satisfactory result.

## The Exponentially Decaying TVF

In order to define our TVF so that it follows exponential decay, we need to calculate the exponential decay constant from our data points. This can be done using two consecutive (decreasing) data points, $(t_{before}, x_{before})$, $(t_{after}, x_{after})$ as follows. First we simply redefine our time points as $(0, x_{before})$, $(t_{after} - t_{before}, x_{after})$, then using the ansatz $f(t) = Ne^{-\lambda t}$ we have

$$
\begin{aligned}
f(0) = Ne^{-\lambda \cdot 0} = N = x_{before} \\
f(t_{after} - t_{before}) = x_{before} e^{-\lambda(t_{after} - t_{before})} = x_{before} \implies \\
\frac{x_{before}}{x_{after}} = e^{\lambda(t_{before} - t_{after})} \implies \\
\lambda(t_{after} - t_{before}) = ln(\frac{x_{before}}{x_{after}}) \implies \lambda = \frac{ln(\frac{x_{before}}{x_{after}})}{t_{after} - t_{before}}
\end{aligned}
\tag{9.2}
$$

Then we have that $-\lambda \cdot x_0 = \frac{dx}{dt}_{training}$. We visualize our TVF defined in this way below.

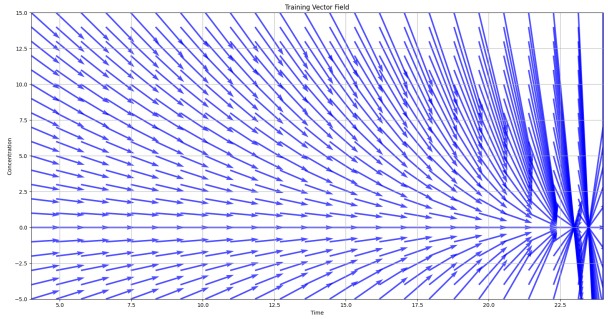



Figure 9.3: Visualization of the TVF defined using exponential decay, for the decrease in the concentration after concentration has reached its peak.

## The Polynomial TVF

There is also a subtle problem with using a linear TVF to model the increase in concentration after a dose is taken. We can see that the vector field remains flat immediately after a dose is applied if the patient's concentration is already at the peak expected concentration. This is not realistic behavior, we would still expect the concentration to rise after a dose is applied even if the patient already has a positive concentration of the drug. This problem did not become readily apparent until attempting to predict PK time series for dosing regimes with different times between each dose, for example, 12 hrs or 48 hours between each dose instead of 24 hours between each dose. Our model failed to extrapolate in this instance because it had to learned to associate higher concentrations in the blood with smaller increases in concentration given a certain dosing size. Thus, in order to train a model that could generalize to new dosing times, we had to make the flow depend on time since the last dose rather than on the current concentration.

Our first attempt to do this was to remove the current concentration as an input to our model and make the vector field independent of the current concentration. However, this led to problems with generalizing to new dosing regimes. This is because the current concentration of the drug in the body has a negative impact

on the increase in concentration, so ignoring this entirely leads the model to vastly overestimate the concentrations as the dose size increases. This led us to instead use a polynomial to define the flow.

$$
\begin{gathered}
f(t) = at^2 + bt + c \\
f^{'}(t) = 2at + b \\
f(t_{before}) = x_{before} \\
f(t_{after}) = x_{after} \\
f^{'}(t_{after}) = 0 \\
f(t_{before}) = x_{before} \implies at_{before}^2 + bt_{before} + c = x_{before} \implies \\
c = x_{before} - at_{before}^2 - bt_{before} \\
f^{'}(t_{after}) = 0 \implies 2t_{after}a + b = 0 \implies b = -2t_{after}a \\
f(t_{after}) = x_{after} \implies x_{after} = \\
at_{after}^2 + t_{after}b + x_{before} - at_{before}^2 - bt_{before} \implies \\
x_{after} - x_{before} = a(t_{after}^2 - t_{before}^2) + b(t_{after} - t_{before}) = \\
a(t_{after}^2 - t_{before}^2) - 2t_{after}a(t_{after} - t_{before}) = \\
a(2t_{after}t_{before} - t_{before}^2 - t_{after}^2) \implies a = \frac{x_{after} - x_{before}}{2t_{after}t_{before} - t_{before}^2 - t_{after}^2} \implies \\
b = \frac{8(x_{after} - x_{before})}{t_{after}^2 + t_{before}^2 - 2t_{after}t_{before}}
\end{gathered}
\tag{9.3}
$$

We can now define our flow independently of the concentration by setting it to $f^{'}(t) = 2at + b$, with $a$ and $b$ determined from the data, while still ensuring that the derivative approaches zero as we reach the expected maximum concentration time.

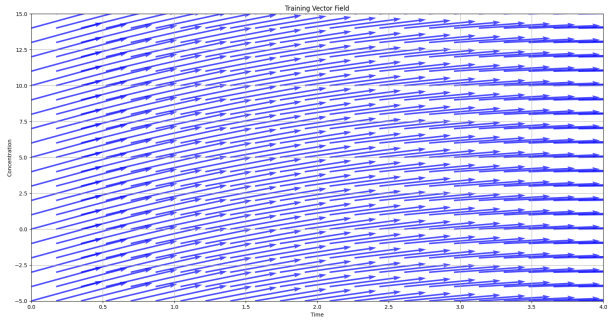
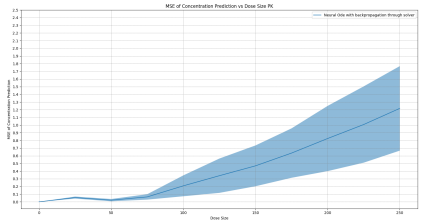


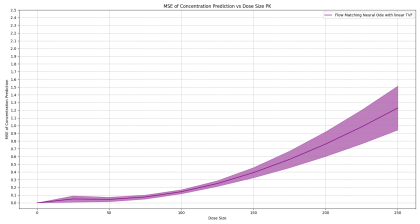

Figure 9.4: Visualization of the TVF defined using the polynomial, for the increase in the concentration after a dose has been taken.

# Results

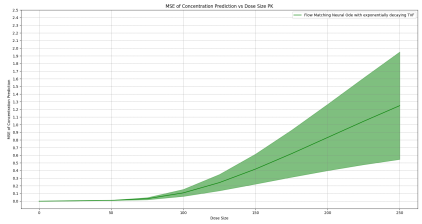
Here we show all of the results related to the different training vector fields used when training with the flow matching algorithm and their comparison to the neural ODE trained with backpropagation through the solver. Linear TVF denotes using the slope for both the increase and decrease in concentration, exponentially decaying TVF denotes using the TVF defined with exponential decay for the decrease in concentration and the slope for the increase, while nonlinear TVF denotes using the TVF defined with exponential decay for the decrease in concentration and the TVF defined with the polynomial for the increase in concentration, which was the model used in the main body of the report.



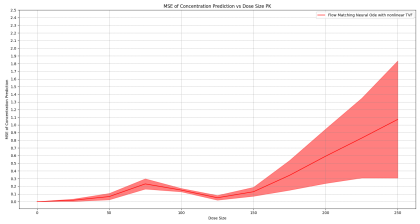
(a) Neural ODE trained with backpropagation through the solver



(b) Flow Matching Neural ODE trained with linear TVF



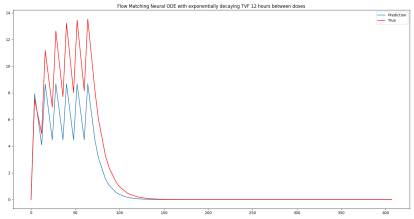
(c) Flow Matching Neural ODE trained with exponentially decaying TVF



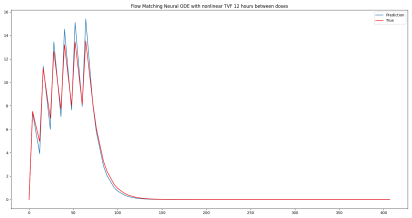
(d) Flow Matching Neural ODE trained with nonlinear TVF

Figure 9.5: Comparison of the neural ODEs extrapolation across dose size with different training methods

The different TVFs all perform roughly the same at extrapolating to new dosing sizes. The most significant difference in performance between the different types of TVFs is observed when testing the model on a shorter time interval between each dose. As noted in the method section, when using a linear TVF to model the increase in concentration after a dose, the model learns to associate specific dose sizes with a particular maximum concentration. Thus, it is not able to model the accumulation of the drug in the body. The nonlinear TVF bypasses this by not having the TVF depend on the current concentration, but as seen in the time series, this does not lead to perfect extrapolation.



(a) Flow Matching Neural ODE trained with exponentially decaying TVF



(b) Flow Matching Neural ODE trained with nonlinear TVF

Figure 9.6: Visual comparison of the neural ODEs extrapolation to a smaller time between doses