Mixed Effects Neural ODE:  
A Variational Approximation for Analyzing the Dynamics of Panel Data

Jurijs Nazarovs, Rudrasis Chakraborty, Songwong Tasneeyapant, Sathya N. Ravi, Vikas Singh

1Department of Statistics, University of Wisconsin Madison  
2Department of Biostatistics & Med. Info., University of Wisconsin Madison  
3Amazon Lab 126  
4 Department of Computer Science, University of Illinois at Chicago

Abstract

Panel data involving longitudinal measurements of the same set of participants taken over multiple time points is common in studies to understand childhood development and disease modeling. Deep hybrid models that marry the predictive power of neural networks with physical simulators such as differential equations, are starting to drive advances in such applications. The task of modeling not just the observations but the hidden dynamics that are captured by the measurements poses interesting statistical/computational questions. We propose a probabilistic model called ME-NODE to incorporate (fixed + random) mixed effects for analyzing such panel data. We show that our model can be derived using smooth approximations of SDEs provided by the Wong-Zakai theorem. We then derive Evidence Based Lower Bounds for ME-NODE, and develop (efficient) training algorithms using MC based sampling methods and numerical ODE solvers. We demonstrate ME-NODE’s utility on tasks spanning the spectrum from simulations and toy data to real longitudinal 3D imaging data from an Alzheimer’s disease (AD) study, and study its performance in terms of accuracy of reconstruction for interpolation, uncertainty estimates and personalized prediction.

1 INTRODUCTION

Observational studies in the social and health sciences often involve acquiring repeated measurements over time for participants/subjects. If most participants stay enrolled, we can consider each row in the panel to correspond to longitudinal measurements or records of an individual, at regularly or irregularly sampled time points. Modeling development or growth trends while accounting for variability within and across individuals leads to the need for statistical models for analysis of such “panel data” [Kreindler and Lumsden, 2006; Katsev and L’Heureux, 2003].

The modeling of temporal processes can be set up as a regression task, where a function (with unknown parameters) that is plausible for the domain is estimated using the observed longitudinal data samples. Apart from splines and tools from functional data analysis, a common alternative is to use differential equations [Chen and Wu, 2008; Liang and Yq, 2013; Fang et al., 2011], which provides expressive power and many computational tools developed over decades. However, differential equations do not directly account for variability within and across subjects – hallmark features of panel data. To capture some of these characteristics, the widely used Auto-Regressive Models (ARMA) literature incorporates white noise type functions in differential equation models, leading to various forms of stochastic differential equation (SDE) [Brockwell, 2001; MaCurdy, 1982; Hedeker and Gibbons, 2006] where

$$z_t = f_\mu(z, t) dt + L_{\Sigma}(z, t) \circ d\beta(t)$$  (1)

where $z_t \in \mathbb{R}$, $f$, $L$ denote the drift and noise sensitivity functions with unknown parameters $\mu$ and $\Sigma$ respectively.

Suppose we are given a set of (partial) measurements $\{z_{tk}\}$ at certain $K$ time points $\{t_k\}_{k=1}^K$, and an efficient numerical scheme to simulate the SDE in (1). Then, the unknown parameters $\mu$ and $\Sigma$ can be found by simply maximizing the likelihood function $p(z_t|\mu; \Sigma)$. In principle, it is straightforward to extend the model in (1) to high dimensional $z_t$. However, there are two main technical challenges in this setting with stand-alone likelihood based methods: (i) such an approach requires a large number of longitudinal measurements which is often infeasible, especially in the applications that motivate our work [Marinescu et al., 2018]; (ii) numerical schemes to simulate nonlinear SDEs in high dimensions are quite involved. Indeed, these issues become pronounced when we assume that the observed data is not $z$ but $x$ which are actually measurements governed by a

*Corresponding authors: Ravi and Singh.

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).
process that reflects the dynamics $z$.

The literature provides a principled way, called Bayesian filtering, to tackle the problem described above, see [Särkkä and Solin, 2014, Ch 7. Let us consider that the object or measurement $x$ is evolving as non-linear function $D$ from a latent measure of dynamics/progression $z_t$ [Pierson et al., 2019; Hyun et al., 2016; Whitaker et al., 2017]: $x_t = D(z_t) + \varepsilon_t$. So, it is natural to think of the observable $x$ of an unknown dynamics $z$ – which we can call the “latent” representation. The goal in Bayesian filtering, which aligns nicely with our task, is to compute, $p(z|x_1, ..., x_{t_K})$. Interestingly, under some assumptions, closed form solutions for the posterior distributions $p(z|x_1, ..., x_{t_K})$ are available, see Chapter 10 in [Särkkä and Solin, 2019]. However, these assumptions are hard to verify in general, and direct utility of these approaches for modern applications is not obvious.

Main ideas/contributions. The most important takeaway from the description above is not the mechanics of how Bayesian filtering is carried out in practice, rather, what it seeks to estimate. If we focus on the key object of interest – the conditional distribution – we realize that recent works in machine learning do provide a recipe that exploits the universal approximation properties of neural networks to represent fairly complex conditional distributions. In this case, the parameters are simply trained using off-the-shelf procedures, and DE numerical solvers are required for (1). While both ODE and SDE solvers are available, SDE is typically less efficient [Li et al., 2020; Liu et al., 2019]. Notice that when $L_{x,t} \equiv 0$, that is, the observable $x$ follows an ODE and corresponding solvers can be applied, the approach that would instantiate this idea has already been successfully tried in [Yildiz et al., 2019]. However, given the Bayesian filtering motivation above, is there a way to utilize ODE solvers, while preserving stochastic nature of (1)?

Our development begins by rewriting the noise term in (1) with a series of basis functions with standard normal coefficients. We show that this modification enables incorporating random effects in our predictions – which appropriately models the variability of the data, the requirement for successful analysis of panel data. More importantly, using our approach, we show that for a special class of latent SDE based models, which we will refer as Mixed Effects Neural ODE (ME-NODE), the parameters of the underlying neural network can be trained efficiently without backpropagating through any SDE solvers. To achieve this, we derive the Evidence Lower Bound loss – where widely available libraries for numerical ODE solvers – are sufficient and directly applicable. We show applications to brain imaging where our formulation can provide personalized prediction together with uncertainty, a feature of Bayesian methods.

## 2 BACKGROUND/NOTATION

In this section we present the notations and some basic concepts we use in the paper.

**Notation.** For a time-varying vector $z = (z_0, \ldots, z_n)$, (with $n$ time points) we denote a vector without the $j$-th component as $z_{-j} = (z_0, \ldots, z_{j-1}, z_{j+1}, \ldots, z_n)$. Often each time point $z_t$ is vector valued, and in the rest of the paper we denote it by $z_t \in \mathbb{R}^p$, where $p$ is the number of variables. Thus, to denote a time-varying vector where each time point represents a $p$ dimensional vector we use $z = (z_0, \ldots, z_n)$, where each $\{z_t\}_{t=1}^n \subseteq \mathbb{R}^p$. We denote the indicator function as $\mathbb{1}_x(y)$: it is 1 if $y = x$, and 0 otherwise. We refer to a general form of ODE as $\dot{z}_t = h(z_t)$, where $h(z_t)$ defines a “trajectory” and depends on the current value of the process at time $t$. Without loss of generality, in this section we assume that the DE is defined on $\mathbb{R}$.

**Smooth Approximations of SDE.** Consider the standard form of the Stratonovich SDE given by (1). By the Wong and Zakai theorem [Hairer and Pardoux, 2015], the solution of the following equation:

$$\dot{z}_t = f(z, t) + L(z, t) \sum_{n=1}^{N} b_n \varphi_n(t),$$

where $b_n \sim \mathcal{N}(0, 1)$ and $\{\varphi_n\}$ are a suitable set of basis functions. Based on mild simplifying assumptions, we get

$$\dot{z}_t = f(z, t) + g(z, t)b,$$

where $b \sim \mathcal{N}(0, 1)$.

**Proof.** Assume that $\exists g(z, t) < \infty$ and $\forall n, \exists \sigma_n(z, t) < \infty$ such that

$$\varphi_n(t)L(z, t) = \frac{\sigma_n(z, t)}{\sqrt{\sum_{n=1}^{N} \sigma_n^2(z, t)}} g(z, t).$$

Then,

$$\sum_{n=1}^{N} \varphi_n(t)L(x, t) = g(z, t) \sqrt{\sum_{n=1}^{N} \sigma_n^2(z, t)} \sum_{n=1}^{N} \xi_n(z, t),$$

where $\xi_n(z, t) = b_n \sigma_n(z, t) \sim \mathcal{N}(0, \sigma_n(z, t))$. If for some $\delta > 0$,

$$\left(\sum_{n=1}^{N} \sigma_n^2 \right)^{2+\delta} \sum_{n=1}^{N} \mathbb{E} \left(\xi_n - \mu_n\right)^{2+\delta} \rightarrow_{N \rightarrow \infty} 0,$$

then according to Lyapunov’s Central Limit Theorem

$$\frac{1}{\sqrt{\sum_{n=1}^{N} \sigma_n^2}} \sum_{n=1}^{N} \xi_n(z, t) \rightarrow\mathcal{N}(0, 1).$$

Then, $\dot{z}_t = f(z, t) + g(z, t)b$ is an approximation of Stratonovich’s SDE (1), where $b \sim \mathcal{N}(0, 1)$. \qed
Our goal is to learn a latent representation of a task, $x$ where $p$ time-varying physical process/dynamics $\epsilon$ for random effects, and $\nu$ here, $m$ are the number of observations and variables respectively. Here, $\nu$ is a non-linear (vector-valued) function, $\beta \in \mathbb{R}^m$ is a vector of fixed effects, $b^i \sim \mathcal{N}(0, \Sigma_b)$ is a vector of random effects, $U^i$ is a design matrix (modeling choice) for random effects, $\phi^i \in \mathbb{R}^n$ is the response variable and $\epsilon^i \sim \mathcal{N}(0, \Sigma_{\epsilon})$ represents a noise term.

**Task.** Our goal is to learn a latent representation of a time-varying physical process/dynamics $z$, and distribution $p(z|x_1, \ldots, x_{t_f})$. Based on the Bayesian filtering discussion above, we focus on variational approximation techniques.

**Learning latent representations with a VAE.** Variational auto-encoders (VAE) [Kingma and Welling 2013] enable learning a probability distribution on a latent space. Then, we can draw samples in the latent space – and the decoder can generate samples in the space of observations. In practice, the parameters of the latent distribution are learned by maximizing the evidence lower bound (ELBO) of the intractable likelihood:

$$\log p(x) \geq -KL(q(z) \| p(z)) + \mathbb{E}_{q(z)}[\log p(x \mid z)] \tag{7}$$

where $z$ is a sample in the latent space from the approximate posterior distribution $q(z)$, with a prior $p(z)$, and $x$ is a reconstruction of a sample (e.g., an image) with the likelihood $p(x \mid z)$. A common choice for $q$ is $\mathcal{N}(\mu, \Sigma)$, where $\mu$ and $\Sigma$ are trainable parameters [Kingma and Welling 2013].

**3 MIXED EFFECTS NEURAL ODE**

Given a latent representation $z$ of a time-varying process with $z = (z_1, \ldots, z_n)$ and $z_t \in \mathbb{R}^d$, we now model the latent representation $z$ as a mixed effects neural ODE. We will assume each of the $p$ variables to be independent, so we will seek to learn $p$ mixed effects models. Without any loss of generality, below we assume $z \in \mathbb{R}$ to denote the latent representation of the time-varying process at a time point $t$.

**Modeling random effect $b^i$ in a network $\Gamma$.** Mixed effects in the context of ODE is a well studied topic in longitudinal data analysis literature [Wang et al. 2014, Liang and Yu 2013]. Formally, for subject $i$, given $z^i$, $b^i$, and (population level) fixed effects $\beta$, we assume that there exist a smooth function $h$ such that,

$$\dot{z}^i = h(z^i, \beta, b^i). \tag{8}$$

Due to the universal approximation properties of neural networks [Zhou 2020], such models are a sensible choice to express the nonlinear function $h$. Recall the non-linear mixed effects model from (6), and let us model

$$h(z^i, \beta, b^i) = \nu(\eta(z^i)\beta + U^i b^i).$$
Here, \( \eta : \mathbb{R}^n \to \mathbb{R}^{n \times m} \) is a non-linear function, with \( n \) and \( m \) being the number of observations and variables respectively and \( \beta \in \mathbb{R}^m \). With a choice of \( U^i = \eta(z^i) \), we can model \( h(z^i, \beta, b^i) = \Gamma(z^i)(\beta + b^i) \), where \( \Gamma \) is a neural network with \( \Gamma(z^i) \in \mathbb{R}^{n \times m} \). Now, we can derive the expressions for representing mixed effects in ODE, parameterized by a neural network as

\[
    h(z^i, \beta, b^i) = \Gamma(z^i)w^i, \tag{9}
\]

where \( w^i \sim \mathcal{N}(\beta, \Sigma_b) \) is a mixed effect for subject \( i \). This can be thought of as a projection from \( \mathbb{R}^m \) to \( \mathbb{R}^n \). We use \( q(z_0) \) to sample initial points of ODE \( z_0 \).

### Remark
Observe the difference between standard SDE in (1) – where the noise is added at each step \( t \) – and our formulation, where \( w^i \) (or \( b^i \)) is sampled once for subject \( i \) and completely defines the trajectory through \( h(z^i, \beta, b^i) \) for all steps of time \( t \). This is crucial from the computational perspective: with this strategy, we can simply apply existing ODE solvers whereas backpropagating through a blackbox SDE requires specialized solutions, which are typically slower [Li et al. 2020, Liu et al. 2019].

### Initializing ODE \( h(z^i, \beta, b^i) \) with an encoder \( E \)
Often in real-world analysis tasks involving panel data, the initial point of the process \( z_0 \) is not observed. While it can be learned as a parameter [Huang et al. 2008], it is desirable to also provide uncertainty pertaining to the learned \( z_0 \). For this reason, we learn the distribution \( q(z_0) = \mathcal{N}(\mu, \sigma) \), by training an encoder \( E \) to map observed data \( x = (x_1, \ldots, x_n) \) (at all \( n \) time points) to parameters of \( q(z_0) \), \( \mu \) and \( \sigma \):

\[
    (\mu, \sigma) = E(x) \tag{10}
\]

We use \( q(z_0) \) to sample initial points of ODE \( z_0 \).

### Mapping \( z \) to \( x \) via decoder \( D \)
Given the latent representation \( z \) and the non-linear function \( D \), which can recover the output \( x^i \) for a subject \( i \) at time point \( t \), we can model the output of the dynamic process (e.g., in our application, a brain image) \( x^i \) as a non-linear transformation of the latent measure of progression \( z^i \):

\[
    x^i_t = D(z^i_t) + \epsilon_t, \tag{11}
\]

where \( \epsilon_t \) is measurement error at each time point. This idea has been variously used in the literature [Pierson et al. 2019, Hyun et al., 2016, Whitaker et al. 2017].

### The final model
Combining Equations (10) and (11) we obtain our Mixed Effects Neural ODE model in (12) and illustrated in Figure 1

\[
    \begin{bmatrix}
    z_0^i \sim \mathcal{N}(\mu, \sigma), \text{ where } \mu, \sigma = E(x^i) \\
    w^i = \beta + b^i \sim \mathcal{N}(\beta, \Sigma_b) \\
    \dot{z}_t^i = \Gamma(z^i_t)w^i \\
    x^i_t = D(z^i_t) + \epsilon_t
    \end{bmatrix} \tag{12}
\]

**Synopsis.** Here, for subject \( i \), we use the encoder \( E \) to map the observed data \( x^i \) to parameters of the distribution of ODE initialization \( z_0^i \). Then, we parameterize the derivative of ODE \( \dot{z}_i \) with a neural network \( \Gamma(z^i) \) and mixed effects \( w^i \), and use \( D \) to map solution of ODE to the original space.

### Structure of the latent space
When the latent space \( z \) can be embedded in a low dimensional space, it is natural to ask whether simulating the ODE \( \dot{z} \) in (12) can be accomplished efficiently. The following result shows a link between our model in (12) and random projection ideas [Vempala, 2005]. Moreover, in contrast to Neural ODE [Chen et al., 2018], random projections allow using high dimensional representations, \( \Gamma(z^i_t) \), and mapping it back to \( \mathbb{R} \) using random projections. This provides expressive power but also approximately preserves the distance (using JL lemma).

**Lemma 1** (Random projection). With a certain choice of \( h(z^i) \) in the ODE formulation and given a mixed effect \( w \) with a choice of approximate posterior \( q(w) \) as a Normal distribution, the solution to ME Neural ODE (Equation (12)) is a random projection of a solution to Neural ODE [Chen et al. 2018]. Figure 2(right part).

**Proof.** Let us use the following notations,

- (a) \( d\tilde{z}_t = f(z_t) \) \( dt \) defines trajectory of Neural ODE setup;
- (b) \( d\tilde{w} = \Gamma(\tilde{z}_t) \cdot wd\tilde{t} \) defines trajectory of our ME setup;
- (c) \( \tilde{z}_t = z_t \cdot w \) : random projection of \( z_t \). Then we have,

\[
    d\tilde{z}_t = w \cdot dz_t = w \cdot f(z_t) dt = f \left( \frac{\tilde{z}_t}{w} \right) \cdot wd\tilde{t}
\]

If \( \Gamma(\tilde{z}_t) = f \left( \frac{\tilde{z}_t}{w} \right) \), then \( \tilde{z}_t \) is random projection of \( z_t \) and \( \tilde{z}_t = z_t \cdot w \).

While having a model, efficient training is still unresolved. Next, we show how for (12), we can derive ELBO-like bounds using Approximate Bayesian Computation (ABC) [Wilkinson 2013, Fearnhead and Prangle 2010].

### Connection with approximation of Stratonovich’s SDE
Observe that if we select \( f \) and \( g \) in (3) as \( f(z, t) = \Gamma(z, t)\beta \) and \( g(z, t) = \Gamma(z, t)\Sigma_b^{1/2} \), the approximation of Stratonovich’s SDE becomes ME-ODE defined in (9), i.e., we set \( \tilde{z}_t = \Gamma(\cdot)w \), where \( w = \beta + b\Sigma_b^{1/2} \sim \mathcal{N}(\beta, \Sigma_b) \).

Note that our ODE/SDE based derivation of the expression in (9) coincides with the mixed effects form proposed in [Xiong et al. 2019] for single panel (time) data.

**Remark.** Recall that in (8), we assumed that \( h \) is smooth. In theory, the smoothness assumption is justified due to Wong-Zakai approximation, see [3]. In practice, this can be achieved by choosing a sufficiently fine discretization.
3.1 MODEL TRAINING: ME-NODE ELBO

The objective of the training scheme we describe now is to learn (a) distribution of \( z_0 \), (b) fixed effect \( \beta \), (c) variance of random effect \( \Sigma_b \). To reduce clutter, in this section, we drop index \( i \) (which specifies a subject).

At a high level, our approach is to infer the random effects \( b \) by learning it as a parameter. For our purposes, learning \( b \) corresponds to ensuring that \( b \) satisfies the following key requirement (accounting for a small reconstruction error): \( b \) needs to be random by design for statistical reasons such as uncertainty quantification. Using our model in (12), it is easy to see that this requirement is satisfied because \( b \) is sampled from \( \mathcal{N}(0, \Sigma_b) \). A common strategy to satisfy the requirement is to use a VAE [Chen et al., 2020]. It is known that in such probabilistic models computing the marginal likelihood \( p(x) \) is usually intractable. Let \( p(z, w) \) be the prior joint distribution, \( q(z, w) \) as approximate joint posterior, and \( p(x | z, w) \) as likelihood of reconstruction. Using concepts from §2 we can derive a lower bound for the \( p(x) \) of our ME-NODE model as:

\[
\log p(x) = \log \int p(x | z, w) p(z, w) \frac{q(z, w)}{q(z, w)} d(z, w) \\
= \log \mathbb{E}_{q(z, w)} \left( p(x | z, w) \cdot \frac{p(z, w)}{q(z, w)} \right) \\
\geq \mathbb{E}_{q(z, w)} \log p(x | z, w) - KL(q(z, w)||p(z, w)),
\]

where (13) follows from the marginalization property and then we use Jensen’s inequality. Next, we define \( q(z, w) \) which is needed to compute ELBO. Note that in the following description, \( \xi | \psi \) refers to the random variable \( \xi \) conditioned on a value of \( \psi \), regardless of what the value is, i.e., it can be \( \xi | \psi = 0 \) or \( \xi | \psi = 1 \).

**Defining \( q(z, w) \).** Assuming that \( z_0 \) and \( w \) are independent random variables, we get

\[
q(z, w) = q(z_0, z_0 \cdot z, w) = q(z_0)q(z_0 | z_0)q(w).
\]

Recall that \( z_0 = (z_1, \ldots, z_n) \) is a vector of ODE solutions at time step \( t \), except \( t = 0 \). At every step \( t, z_t \) is a random variable, which is a function of \( z_0 \) and \( w \). However, with a fixed initial point \( z_0 \) and mixed effect \( w \), the progression follows a defined trajectory (i.e., there is no randomness). It means that \( z_t | z_0 \) is deterministic and hence the distribution \( q(z_t | z_0, w) \) is degenerate [Danielsson, 1994], which results in \( q(z_0 | z_0, w) = \mathbb{I}_{z_0 \mid z_0, w} \). Thus,

\[
q(z, w) = \mathbb{I}_{z_0 \mid z_0, w} q(z_0) q(w).
\]

**Note.** While the derivation from (13) to (14) is well defined for point mass distributions stated in (15), the use of 'KL', although consistent with [Bai et al., 2020] (pp 3, (4)–(6)) is not ideal (because \( \log(0) \) and thus 'KL' is undefined). We will avoid using 'KL' notation in the loss in (16).

**MC approximation of \( E_q(z, w) g(z, w) \).** The key in computing (14) is to estimate \( E_q(z, w) g(z, w) \) for a given function \( g(z, w) \). Based on parameterization of \( q(z, w) \) in (15),

\[
E_q g(z, w) = \int_{z_0, w} g(z, w) \cdot \mathbb{I}_{z_0 \mid z_0, w} q(z_0) q(w) dz_0 dw.
\]

While the integration may be intractable, it can be estimated by Monte Carlo (MC) techniques. Sampling \( (z^m, w^m) \) from \( q \), we compute \( \frac{1}{M} \sum_{m=1}^{M} g^*(z^m, w^m) \), where

\[
g^*(z^m, w^m) = g(z^m, w^m) \cdot \mathbb{I}_{z_0 \mid z_0, w^m}.
\]

This type of sampling is called likelihood-free rejection sampling [Del Moral et al., 2012]: we reject all samples \( (z^m, w^m) \), which do not generate observed \( z_{obs} \).

**The final loss.** Given this MC approximation (with \( M \) samples), with the approximate posterior \( q(z, w) \) defined in (15) and with a similarly defined prior \( p(z, w) = \mathbb{I}_{z_0 \mid z_0, w} p(z_0) p(w) \), the final loss is

\[
\frac{1}{|S|} \sum_{s \in S} \left( \log p(x | z^s, w^s) - \log \frac{q(z_0) q(w^s)}{p(z_0) p(w^s)} \right),
\]

where \( S \) is a set: \( \forall s \in S : \mathbb{I}_{z_0 \mid z_0, w^s} = 1 \).

**Remark.** While a MC approximation in (16) is an unbiased estimator of the Lower Bound in (14), its variance is \( O \left( \frac{1}{|S|} \right) \). This leads to efficiency issues in that it may require a large \( M \) until we get \( z_0 \) and \( w \) to generate \( z_{-0} \) exactly along the observed trajectory to populate the set \( S \). But we can address this problem using ABC methods [Wilkinson, 2013, Fearnhead and Prangle, 2010].

**Efficient sampling: approximating \( \mathbb{I}_{z_0 \mid z_0, w} \).** ABC recommends finding samples of \( z_0 \) and \( w \) to generate trajectories \( z_{-0} \) which are approximately equal to the observed one, rather than exactly equal. The idea in [Marin et al., 2012] proposes using \( \mathbb{I}_{y \{z \}} \) as \( \mathbb{I}_{A_{xy} \{z \}} \), where \( A_{xy} = \{z \mid d(z, y) \leq \epsilon \} \) is an \( \epsilon \)-neighborhood of \( y \), and \( d \) is a distance function. For a direct application of these methods on \( \mathbb{I}_{z_0 \mid z_0, w} \), we must have access to \( z_{obs} \) in the latent space, which is unavailable unless the encoder \( E \) and the decoder \( D \) are identity functions. Nonetheless, we can approximate \( \mathbb{I}_{z_0 \mid z_0, w} \) by comparing if the decoded \( z_0 \mid z_0, w \) indeed corresponds to \( z_{obs} \), i.e., we need to compute \( \mathbb{I}_{z_0 \mid z_0, w} \{D(z_0 | z_0, w) \} \). We simply use the mean squared error (MSE) as the distance for this comparison.

By decreasing \( \epsilon \), we can improve the quality of the samples for MC estimation, but at higher compute cost. However, because we learn the distribution of \( z_0 \), during the first steps of training, our model provides poor reconstructions. For this reason, setting \( \epsilon \) to a small value at the beginning of
the training is inefficient. Therefore, we make $\varepsilon$ adaptive through the training, by choosing the sample, closest to our observed trajectory, i.e., sample with smallest distance $d$, and $\varepsilon$ is a function of the initial point $z_0$.

**Choice of $q(w)$**. To optimize the ELBO, it is necessary to define the approximate posterior $q(w)$ and prior $p(w)$. While we assumed in [9] and Lemma [10] that the true distribution of $w$ is Normal, $q(w)$ and $p(w)$ remain design choices for the user. For example, if we believe that the correlation structure of the data is sparse, then we have the following choices: Horseshoe [Carvalho et al., 2009], spike-and-slab with Laplacian spike [Deng et al., 2019] or Dirac spike [Bai and De Gaetano, 2005, Bouriaud et al., 2019]. However, for our experiments, we found that modeling $q(w)$ and $p(w)$ as Normal is sufficient.

**Calibration.** One feature of our model is that learned distribution of mixed effects $w$ can be used for personalized prediction during extrapolation [Wang et al., 2014] [Ditlevsen and De Gaetano, 2005] [Bouriaud et al., 2019]. First, we train our model to learn the parameters of distribution of $z_0$, fixed effects $\beta$, and variance of random effect $\Sigma_b$. Then at test time, we make use of the observed temporal data for a previously unseen test subject $x^{obs}$. Given the learned distribution of mixed effects $w$, we want to find a sample $w \sim w^*$, which minimizes error w.r.t. $x^{obs}$. This selection provides the most appropriate mixed effect $w$ corresponding to $x^{obs}$. We call this process calibration: a solution to the optimization of ELBO in (16) requires using the appropriate distribution of mixed effects $w^i$.

The output from this phase is the prediction for subject $i$, $\hat{x}^i$ for $t \in [0, T]$ is assumed to be provided. Then, we

1. Select a personalized mixed effect $w^i \sim w^i$, according to the calibration scheme.
2. Then, we use the selected mixed effect sample $w^i$, to generate personalized prediction for the subject $i$ for either interpolation or extrapolation.

The output from this phase is the prediction for subject $i$, $\hat{x}^i$ for $t \in [0, \ldots, T^*]$, where $T^*$ can be extrapolated time, i.e., $T^* \geq T_i$, and/or a denser interpolation in $[0, T]$.

| Estimated parameters | $n_{z_0}$, $n_w$ | True values |
|----------------------|-----------------|-------------|
| $\hat{\mu}$         | 1.252, 1.258, 1.316, 1.313 | 1.30 |
| $\hat{\sigma}$      | 0.003, 0.002, 0.005, 0.016 | 0.01 |
| $\hat{\beta}$       | 0.311, 0.315, 0.311, 0.319 | 0.3  |
| $\hat{\sigma}_b$    | 0.036, 0.051, 0.054, 0.060 | 0.01 |
| MSE (all)            | 0.0017, 0.0011, 0.0006, 0.0005 | 0.0005 |

Table 1: The first four rows show the estimated parameters for specific choices of $n_{z_0}$ and $n_w$. Here we use the following convention: $(n_{z_0}, n_w) = (i, j)$ denotes we draw $i$ and $j$ number of samples from $z_0$ and $w$ respectively. The last row presents the MSE values for the estimated parameters.

### 4 EXPERIMENTS

We evaluate our model on five temporal datasets: (1) simulations, (2) MuJoCo hopper, (3) rotating MNIST, and (4) two different Neuroimaging datasets, representing disease progression in the brain.

**Goals.** We will evaluate: (a) ability to learn mixed effects, given different types of correlations in the data (b) the effect of mixed effects dimension $m$ on extrapolation power and confidence of the model, and (c) the ability to preserve statistical group differences in the data in latent representations.

The baselines are given separately for each experiment. We provide description of hardware and neural networks architectures, including encoder/decoder in appendix.

#### 4.1 SYNTHETIC DATASET

We start with a synthetic setup where all parameters are known. Using a ODE solver and conditioning on $z_0$ and $w$, we generate a solution of the mixed effect ODE

$$\begin{align*}
z_0^i & \sim \mathcal{N}(\mu = 1.3, \sigma = 0.01) \\
w^i & \sim \mathcal{N}(\beta = 0.3, \sigma_b = 0.01) \\
\dot{z}_t^i & = \dot{z}_t^i w^i \tag{17}
\end{align*}$$

We set the encoder $E$ and decoder $D$ in (12) to the identity transformation. Given 1000 (80 : 20 split for train/test) numerical solutions of the ODE in (17), we uniformly sampled 20 time points from $[0, 3]$, and use the first 10 time steps for interpolation and the last 10 for extrapolation.

**Parameters.** As the optimization of ELBO in (16) requires samples, we evaluate the performance of our model by varying the number of samples for $z_0$ and $w$ (denoted by $n_{z_0}$ and $n_w$ respectively). The results in Table 1 suggest that MSE goes down with an increase of $n_{z_0}$ or $n_w$.

In Figure 2 (top panel), we show samples (trajectories) drawn from the learned model (blue lines) with the real trajectories (`x` marker). Notice that the sampled trajectories from the learned model almost cover the “range” of real trajectories and the results appear meaningful.

**Mixed effects.** To evaluate generation of a personalized pre-
As our model implicitly learns correlation structure of the data and check: (i) relation between mixed effect dimension \( m \) (which we can think of as a dimension of random projection) and performance of the model, (ii) the performance of personalized prediction for extrapolation in comparison with standard BNN.

**Data description.** Similar to the setup in ODE2VAE [Yildiz et al., 2019], we construct a dataset by rotating the images of different handwritten digits, in order to learn a digit specific mixed effect model. In ODE2VAE, digits were rotated by 22.5°. We used a slightly different scheme: for a sampled digit we randomly choose an angle from the set of 1, 4, or 8 angles from the range \( [-\pi/4, \pi/4] \) and apply it at all time steps. For example, if we choose the set containing 4 angles, then a sampled digit is rotated using one of the 4 angles, selected randomly. In order to simulate a practical scenario, we spread out the initial points, by randomly rotating a digit by angles from \( -\pi/2 \) to \( \pi/2 \). We generate 10K samples of different rotating digits for 20 time steps and split it in two equal sets: interpolation and extrapolation.

**Effect of mixed effect (random projection) dimension \( m \).** Recall from Lemma 1, we showed that mixed effects \( \mathbf{w} \) in (12) can be considered as a random projection. So, we can expect that with an increase in \( m \), MSE should decrease as it leads to a richer latent representation of a trajectory. We see that this is indeed true as shown in Figure 4 (left panel). Here, we demonstrate the MSE of reconstruction for three different handwritten digits, in order to learn a digit specific mixed effect model. In ODE2VAE, digits were rotated by 22.5°. We used a slightly different scheme: for a sampled digit we randomly choose an angle from the set of 1, 4, or 8 angles from the range \( [-\pi/4, \pi/4] \) and apply it at all time steps. For example, if we choose the set containing 4 angles, then a sampled digit is rotated using one of the 4 angles, selected randomly. In order to simulate a practical scenario, we spread out the initial points, by randomly rotating a digit by angles from \( -\pi/2 \) to \( \pi/2 \). We generate 10K samples of different rotating digits for 20 time steps and split it in two equal sets: interpolation and extrapolation.

**Calibration for personalized prediction.** Here we choose \( m = 50 \) and compare our model for interpolation with ODE2VAE using 3 different rotation settings: 1, 4, and 8 available angles. The calibration results in Table 3 show that our model significantly outperforms the baseline ODE2VAE; however, making the correlation structure

| velocities | 1   | 4   | 8   |
|------------|-----|-----|-----|
| Interpolation | NODE | 7.4 | 5.4 | 5.5 |
| This work   | 5.7 | 4.6 | 4.0 |

| Extrapolation | NODE | 166.1 | 82.1 | 80.3 |
| This work     | 164.4 | 81.2  | 80.0  |

Table 2: MSE (in scale of \( 10^{-3} \)) on MuJoCo Hopper data set, generated for three settings: 1, 4 and 8 initial velocities. We compare these two models using identical neural networks with the same number of levels and hyperparameters, however, in our model the dimension of mixed effect is \( m = 50 \).
of the data more complicated (increasing number of possible rotation angles), does lead to a larger MSE. This is expected: with an increase in complexity of correlation in data, the learning task (and thereby, prediction) becomes harder.

Recall that in order to generate a personalized prediction for subject $i$ we have to sample mixed effect $w^i$ resulting in trajectory closed to the observed. Thus, if the model fails to learn the distribution $q(w)$ accurately, sampling such $w^i$ is less likely, and will result in a larger interpolation error.

For extrapolation, we compare with a standard BNN approach in Fig. [4]. We observe that for each extrapolation step, we obtain, on average, much smaller MSE and smaller variance during the initial steps of extrapolation.

**Extrapolation steps.** Earlier, the number of steps for extrapolation were smaller than the number of observed steps, used for calibration. In Fig. [5] we show results of interpolation and extrapolation, varying the number of observed time steps used for calibration. Expectedly, decreasing the number of steps to be small for calibration yields a smaller number of steps where the extrapolation is meaningful.

| angles | 1    | 4    | 8    |
|--------|------|------|------|
| Interpolation | ODE2VAE | 0.0648 | 0.0644 | 0.0640 |
| Ours-50 | 0.0000 | 0.0014 | 0.0027 |

Table 3: MSE of two models, given different complexity of the data. Low interpolation error of our model indicates the properly learned mixed effect distribution $q(w)$.

In this section, we conduct experiments on two longitudinal brain imaging datasets obtained from Alzheimer’s Disease Neuroimaging Initiative (ADNI) (adni.loni.usc.edu), both of which describe AD progression through time, but are derived from two different imaging modalities.

**Effect of number of dimensions $m$.** We conducted experiments for different values of the mixed effect (random projection) dimension $m$, see appendix. We find that while for any $m$ interpolation looks similar to real data, the further we move in extrapolation, the more noticeable the differences are. For example, for $m = 1$ some frames look blurry and in the last steps of extrapolation, the rotation is wrong. Increasing the dimension of random projection to $m = 20$ improves image quality, but does not fix rotation. Increasing dimension further to $m = 50$, not only improves quality of digits, but also leads to a better prediction of rotation.

(A) TADPOLE. TADPOLE dataset includes data for 276 participants with 3 time points. It represents Florbetapir (AV45) Positron Emission Tomography (PET) scans, which measure the level of amyloid-beta pathology in the brain [Marinescu et al., 2018]. Scans were registered to a template (MNI152) to derive the 82 gray matter regions. Thus, each sample, at time $t$ is a 82 dimensional vector, i.e., $x^t \in \mathbb{R}^{82}$.

Given a few time points (3 time points), we evaluate generation of a personalized prediction for a subject in an interpolation setting. We compare our personalized prediction with ground truth and prediction using a standard BNN approach. The predictions of both our model and BNN
Figure 7: Left-Right Ground truth (1, 3) and prediction (2, 4) of two slices (side and top) of 3D brain at time point 3.

Figure 8: Distribution of distances resulted from permutation test with original distances (dashed lines), and corresponding p-values. The difference between trajectories is significant up to the 7th extrapolated step and degradation from 8.

are based on samples from the same distribution \( q(z, w) \). However, Fig. 6 shows that the calibration of our model (Fig. 6, second row) provides better prediction than BNN (Fig. 6, third row). Even though the learned distribution of mixed effects \( q(w) \) is capable of providing the correct trajectory (calibrated prediction), the direct application of the model without personalized calibration (BNN) leads to high subject-wise uncertainty.

(B) ADNI. Our second dataset from ADNI contains processed MRIs (3D brain scans) of size \( 105 \times 127 \times 105 \) per subject at 3 time steps. The subjects are divided into two groups: diagnosed with Alzheimer’s disease (abnormal: 377 subjects) and healthy controls (normal: 152 subjects).

Given high resolution 3D images, we would like to evaluate whether our model is able to learn the distribution of mixed effects and perform calibration for personalized prediction. Similar to TADPOLE, we conduct an interpolation experiment and provide representative samples of brain images in Figure 7. We find by inspecting the axial/sagittal/coronal views that our model yields meaningful brain images. To evaluate the extrapolation capability, given limited number of time steps (only 3), we perform a statistical test. Recall that our method explicitly models the mixed effect term inside the trajectory to learn the data hierarchy. If our method works as intended, there should be a statistical difference between latent space of trajectories for normal and diseased/abnormal groups. Ideally, this difference should be preserved for several more extrapolation steps. To check this, we train our model on 3 time points. During testing, we use 3 observed time points for calibration and extrapolate for 5 more time steps: we get latent trajectories defined for 8 (3 interpolation and 5 extrapolation) time points. Finally, the resultant trajectories are used to evaluate differences between normal and abnormal groups via a permutation test. The resultant distribution of distances and p-values for interpolation and extrapolation is in Fig. 8. As expected, for interpolation and some steps of extrapolation (up to step 7) differences between trajectories is significant (with p-value \( \leq 0.1 \)), and becomes less significant with more extrapolation steps.

5 DISCUSSIONS AND CONCLUSIONS

We proposed a novel ME-NODE model that enables us to incorporate both fixed and random effects for analyzing the dynamics of panel data. Our evaluations on several different tasks show that the ME-NODE loss function can be trained using existing ODE solvers in a stable and efficient manner. We see various benefits from incorporating mixed effects, (1) model explicitly learns the correlation structure of the data during the training, which improves prediction accuracy in setups where samples can be grouped by some criteria; (2) in contrast to generative models where only initial point is sampled from the distributions, by fixing initial point \( z_0 \) we can still provide uncertainty of the predictions, because from one initial point we can sample different trajectories; (3) since our model learns random effects for individual \( i \), it allows personalized prediction, given a short history of data, which is useful in biomedical or scientific applications with a limited number of time points per individual along trajectory. One limitation of our approach is that there is not an explicit noise handling mechanism for test time calibration and prediction. This is problematic in large scale high dimensional settings. For example, say that the encoding distribution \( \mathcal{N}(\mu, \Sigma) \) produces a small fraction of noisy trajectories. Even in small noise settings, filtering them for robust personalized prediction requires solving complex optimization problem [Bakshi and Kothari, 2021] and so handling noise is especially an open problem in real time, edge deployments. The code is available at [https://github.com/vsingh-group/panel_me_ode](https://github.com/vsingh-group/panel_me_ode).

ACKNOWLEDGMENTS

This work was supported by NIH grants RF1 AG059312 and RF1 AG062336. SNR was supported by UIC start-up funds. We thank Seong Jae Hwang for sharing code and describing the experiments in [Hwang et al., 2019].
References

Jincheng Bai, Qifan Song, and Guang Cheng. Efficient variational inference for sparse deep learning with theoretical guarantee. *arXiv preprint arXiv:2011.07439*, 2020.

Ainesh Bakshi and Pravesh K Kothari. List-decodable subspace recovery: Dimension independent error in polynomial time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1279–1297. SIAM, 2021.

Olivier Bouriaud, G Stefan, and Laurent Saint-André. Comparing local calibration using random effects estimation and bayesian calibrations: a case study with a mixed effect stem profile model. *Annals of Forest Science*, 76(3):1–12, 2019.

Peter J Brockwell. Continuous-time arma processes. *Handbook of statistics*, 19:249–276, 2001.

Carlos M Carvalho, Nicholas G Polson, and James G Scott. Handling sparsity via the horseshoe. In *Artificial Intelligence and Statistics*, pages 73–80. PMLR, 2009.

Jianwei Chen and Hulin Wu. Efficient local estimation for time-varying coefficients in deterministic dynamic models with applications to hiv-1 dynamics. *Journal of the American Statistical Association*, 103(481):369–384, 2008.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31:6571–6583, 2018.

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Jon Danielsson. Stochastic volatility in asset prices estimation with simulated maximum likelihood. *Journal of Econometrics*, 64(1-2):375–400, 1994.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. An adaptive sequential monte carlo method for approximate bayesian computation. *Statistics and Computing*, 22(5):1009–1020, 2012.

Eugene Demidenko. *Mixed models: theory and applications with R*. John Wiley & Sons, 2013.

Wei Deng, Xiao Zhang, Faming Liang, and Guang Lin. An adaptive empirical bayesian method for sparse deep learning. *Advances in neural information processing systems*, 2019:5563, 2019.

Susanne Ditlevsen and Andrea De Gaetano. Mixed effects in stochastic differential equation models. *REVSTAT-Statistical Journal*, 3(2):137–153, 2005.

Yun Fang, Hulin Wu, and Li-Xing Zhu. A two-stage estimation method for random coefficient differential equation models with application to longitudinal hiv dynamic data. *Statistica Sinica*, 21(3):1145, 2011.

Paul Fearnhead and Dennis Prangle. Semi-automatic approximate bayesian computation. *Arxiv preprint arXiv*, 1004:70, 2010.

Martin Hairer and Étienne Pardoux. A wong-zakai theorem for stochastic pdes. *Journal of the Mathematical Society of Japan*, 67(4):1551–1604, 2015.

Donald Hedeker and Robert D Gibbons. *Longitudinal data analysis*, volume 451. John Wiley & Sons, 2006.

Liam Hodgkinson, Chris van der Heide, Fred Roosta, and Michael W Mahoney. Stochastic normalizing flows. *arXiv preprint arXiv:2002.09547*, 2020.

Seong Jae Hwang, Zirui Tao, Won Hwa Kim, and Vikas Singh. Conditional recurrent flow: Conditional generation of longitudinal samples with applications to neuroimaging. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

Jung Won Hyun, Yimei Li, Chao Huang, Martin Styner, WeiLi Lin, Hongtu Zhu, Alzheimer’s Disease Neuroimaging Initiative, et al. Stgp: Spatio-temporal gaussian process models for longitudinal neuroimaging data. *Neuroimage*, 134:550–562, 2016.

Sergei Katsev and Ivan L’Heureux. Are hurst exponents estimated from short or irregular time series meaningful? *Computers & Geosciences*, 29(9):1085–1089, 2003.

David M Kreindler and Charles J Lumsden. The effects of the irregular sample and missing data in time series analysis. *Nonlinear dynamics, psychology, and life sciences*, 2006.

Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pages 3870–3882. PMLR, 2020.

Hua Liang and Yao Yu. Parameter estimation for hiv ode models incorporating longitudinal structure. *Statistics and Its Interface*, 6(1):9–18, 2013.
Xuanqing Liu, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

Thomas E MaCurdy. The use of time series processes to model the error structure of earnings in a longitudinal data analysis. *Journal of econometrics*, 18(1):83–114, 1982.

Jean-Michel Marin, Pierre Pudlo, Christian P Robert, and Robin J Ryder. Approximate bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012.

Razvan V Marinescu, Neil P Oxtoby, Alexandra L Young, Esther E Bron, Arthur W Toga, Michael W Weiner, Frederik Barkhof, Nick C Fox, Stefan Klein, Daniel C Alexander, et al. Tadpole challenge: Prediction of longitudinal evolution in alzheimer’s disease. *arXiv preprint arXiv:1805.03909*, 2018.

Emma Pierson, Pang Wei Koh, Tatsunori Hashimoto, Daphne Koller, Jure Leskovec, Nicholas Eriksson, and Percy Liang. Inferring multidimensional rates of aging from cross-sectional data. *Proceedings of machine learning research*, 89:97, 2019.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5320–5330, 2019.

Simo Särkkä and Arno Solin. Lecture notes on applied stochastic differential equations, 2014. *Version as of December*, 4, 2014.

Simo Särkkä and Arno Solin. *Applied stochastic differential equations*, volume 10. Cambridge University Press, 2019.

Santosh S Vempala. *The random projection method*, volume 65. American Mathematical Soc., 2005.

L Wang, Jiguo Cao, JO Ramsay, DM Burger, CJL Laporte, and JK Rockstroh. Estimating mixed-effects differential equation models. *Statistics and Computing*, 24(1):111–121, 2014.

Gavin A Whitaker, Andrew Golightly, Richard J Boys, Chris Sherlock, et al. Bayesian inference for diffusion-driven mixed-effects models. *Bayesian Analysis*, 12(2):435–463, 2017.

Richard David Wilkinson. Approximate bayesian computation (abc) gives exact results under the assumption of model error. *Statistical applications in genetics and molecular biology*, 12(2):129–141, 2013.

Yunyang Xiong, Hyunwoo J Kim, Bhargav Tangirala, Ronak Mehta, Sterling C Johnson, and Vikas Singh. On training deep 3d cnn models with dependent samples in neuroimaging. In *International Conference on Information Processing in Medical Imaging*, pages 99–111. Springer, 2019.

Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. Ode2vae: Deep generative second order odes with bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 13412–13421, 2019.

Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794, 2020.
APPENDIX

A PROOFS

A.1 DERIVATION FOR FINAL LOSS

\[
\log p(x) \geq \mathbb{E}_{q(z, w)} \log p (x | z, w) - KL(q(z, w) || p(z, w)) \\
\approx \frac{1}{M} \sum_{m=1}^{M} \log p (x | z, w) \cdot 1_{z \sim p(z)} \{ z_0 | z_0^m, w^m \} + \\
\frac{1}{M} \sum_{m=1}^{M} \log \frac{q(z, w)}{p(z, w)} \cdot 1_{z \sim q(z)} \{ z_0 | z_0^m, w^m \} \\
= \frac{1}{M} \sum_{m=1}^{M} \left( \log p (x | z, w) + \log \frac{q(z, w)}{p(z, w)} \right) \cdot \\
1_{z \sim p(z)} \{ z_0 | z_0^m, w^m \} \\
= \frac{1}{|S|} \sum_{s \in S} \left( \log p (x | z^s, w^s) + \log \frac{q(z_0^s, w^s)}{p(z^s, w^s)} \right),
\]

where \( S \) is a set, such that \( \{ \forall s \in S : 1_{z \sim p(z)} \{ z_0 | z_0^s, w^s \} = 1 \} \) and \( |S| \) is its size.

B EXPERIMENTS

B.1 ROTATING MNIST

B.1.1 Ability to capture different angles

In Figure 9 we provide visualization of 2 samples with the same digit style, but 2 different angles of rotation through interpolation and extrapolation.

B.2 ADNI

Following ADNI setup from the main paper, in Figures 10 and 11 we provide another examples of our model, comparing with BNN. To evaluate the result visually, we provide a difference between real and prediction, for both our method and BNN. We see that our model gives much better results.

B.3 HARDWARE SPECIFICATIONS AND ARCHITECTURE OF NETWORKS

All experiments were executed on NVIDIA - 2080ti, and detailed code will be provided in github repository later.

For Rotating MNIST (2d data) encoder/decoder is described in Figure 12 and for ADNI (3d data) encoder is described in Figure 13 and decoder in Figure 14.
Figure 9: Visualization for two samples from data set with 8 possible angles (slow rotation–top, fast rotation – bottom). Because of space limitation we show only 3 time steps of interpolation (1, 5, 10) and all steps of extrapolation (11-20) – blue frame. We show the calibration effect of our model on extrapolation, compare to BNN. We see that with slow rotation interpolation on all 10 steps for bnn is a little worse than our, but still sensible, while extrapolation is not good anymore. Same time, for fast rotated data even for interpolation BNN provides worse results, and very bad for extrapolation. While our method is good for both.

Figure 10: Top: Our method, Bottom: BNN. From left: Truth, prediction, difference between truth and prediction. According to difference (3 column), our method performs better than BNN.
Figure 11: **Top:** Our method, **Bottom:** BNN. **From left:** Truth, prediction, difference between truth and prediction. According to difference (3 column), our method performs better than BNN.
Encoder structure used in ROTATING MNIST

```python
encoder = nn.Sequential(
    nn.Conv2d(input_dim, 12, ks,
              stride=1, padding=1),
    nn.ReLU(),
    nn.Conv2d(12, 24, ks,
              stride=2, padding=1),
    nn.ReLU(),
    nn.Conv2d(24, output_dim, ks,
              stride=2, padding=1),
    nn.Flatten(2),
    nn.Linear(49, 1),
    nn.Flatten(1)
)
```

Decoder structure used in ROTATING MNIST

```python
extend_to_2d = nn.Linear(input_dim, 49 * input_dim)
decoder = nn.Sequential(
    nn.ConvTranspose2d(input_dim, 24,
                       ks, stride=2, padding=1, output_padding=1),
    nn.ConvTranspose2d(24, 12, ks, stride=2, padding=1, output_padding=1),
    nn.ConvTranspose2d(12, output_dim, ks, stride=1, padding=1),
    nn.Sigmoid(),
)
```

Figure 12: Description of Encoder and Decoder used in experiment with 2d data structure: Rotating MNIST.
Encoder structure used in ADNI

encoder = nn.Sequential(
    nn.Conv3d(input_dim, 8, kernel_size=3, stride=1, padding=1, bias=False),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=2, return_indices=True),
    nn.Conv3d(8, 16, kernel_size=3, stride=1, padding=2, bias=False),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=2, return_indices=True),
    nn.Conv3d(16, 32, kernel_size=3, stride=1, padding=1, bias=False),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=2, return_indices=True),
    nn.Conv3d(32, 64, kernel_size=3, stride=1, padding=2, bias=False),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=2, return_indices=True),
    nn.Conv3d(64, 128, kernel_size=3, stride=1, padding=1, bias=False),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=2, return_indices=True),
    nn.Conv3d(128, 256, kernel_size=3, stride=1, padding=1, bias=False),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=2, return_indices=True),
    nn.Conv3d(256, output_dim, kernel_size=3, stride=1, padding=1, bias=False),
    nn.ReLU(),
    nn.Flatten(2),
    nn.Linear(8, 1),
    nn.Flatten(1)
)
Decoder structure used in ADNI

```python
extend_to_3d = nn.Linear(input_dim, 2 * 2 * 2 * input_dim)

decoder = nn.Sequential(
    nn.ConvTranspose3d(in_channels=input_dim,
                      out_channels=256,
                      kernel_size=3,
                      padding=1),
    nn.ReLU(),
    nn.MaxUnpool3d(kernel_size=2),
    nn.ConvTranspose3d(in_channels=256,
                      out_channels=128,
                      kernel_size=3,
                      padding=1),
    nn.ReLU(),
    nn.MaxUnpool3d(kernel_size=2),
    nn.ConvTranspose3d(in_channels=128,
                      out_channels=64,
                      kernel_size=3,
                      padding=1),
    nn.ReLU(),
    nn.MaxUnpool3d(kernel_size=2),
    nn.ConvTranspose3d(in_channels=64,
                      out_channels=32,
                      kernel_size=3,
                      padding=2),
    nn.ReLU(),
    nn.MaxUnpool3d(kernel_size=2),
    nn.ConvTranspose3d(in_channels=32,
                      out_channels=16,
                      kernel_size=3,
                      padding=1),
    nn.ReLU(),
    nn.MaxUnpool3d(kernel_size=2),
    nn.ConvTranspose3d(in_channels=16,
                      out_channels=8,
                      kernel_size=3,
                      padding=2),
    nn.ReLU(),
    nn.MaxUnpool3d(kernel_size=2),
    nn.ConvTranspose3d(in_channels=8,
                      out_channels=output_dim,
                      kernel_size=3,
                      padding=1),
    nn.ReLU()
)
```

Figure 14: Decoder for ADNI