A Further Model Specifications

For better readability, Sec. 3 omitted element-wise activation functions for query/key/value vectors. For example, for Eq. 21, the key and value vectors, $\boldsymbol{k}$ and $\boldsymbol{v}$, are generated as follows

$$\beta(s), \boldsymbol{x}_{\boldsymbol{k}}(s), \boldsymbol{x}_{\boldsymbol{v}}(s) = \boldsymbol{W}_{\text{slow}}\boldsymbol{x}(s) \tag{28}$$

$$\boldsymbol{k}(s) = \text{softmax}(\boldsymbol{x}_{\boldsymbol{k}}(s)) \tag{29}$$

$$\boldsymbol{v}(s) = \tanh(\boldsymbol{x}_{\boldsymbol{v}}(s)) \tag{30}$$

Eq. 19 for the query generation has to be replaced by

$$\boldsymbol{q}(T) = \text{softmax}(\boldsymbol{W}_q\boldsymbol{x}(T)) \tag{31}$$

These specifications are analogous in the CDE cases, i.e., in Eqs. 24-25.

While softmax has already been identified as a crucial component for stability in prior works on discrete-time FWPs [9], we apply an additional $\tanh$ to the value vectors (Eq. 30). Such usage of $\tanh$ in the output of vector fields for CDEs has been advocated by Kidger et al. [4] to improve the stability of the ODE solver. Indeed, we also generally found this beneficial, and sometimes crucial, for stable training of our models. For the Delta rule (Eq. 24), since we modify the value vector after projection (to take into account the "value" which is currently associated with the key vector), we consider two different ways of applying $\tanh$ as follows:

$$\boldsymbol{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s)) = \sigma(\beta(s)) \begin{cases} \big(\tanh(\boldsymbol{x}_{\boldsymbol{v}}(s)) - \boldsymbol{W}(s)\boldsymbol{k}(s)\big) \otimes \boldsymbol{k}(s) & \text{pre-delta} \\ \tanh(\boldsymbol{x}_{\boldsymbol{v}}(s) - \boldsymbol{W}(s)\boldsymbol{k}(s)) \otimes \boldsymbol{k}(s) & \text{post-delta} \end{cases} \tag{32}$$

For the task involving very long sequences (> 4000 frames), we found the "post-delta" version to be crucial for successful training.

B Further Experimental Details

Our basic settings for data preparation and training are based on those used by prior works [4, 37] (we use their public implementations) for fair comparisons with the corresponding baselines.

Dataset/Task details. The essential information about the datasets has already been presented in Sec. 4. The number of sequences in the training set is about 24 K for the Speech Commands dataset and about 28 K for the PhysioNet Sepsis dataset.

Model/Training details. As mentioned in Sec. 3, all our FWP models make use of multiple computational heads, and their NODE/NCDE layer is followed by the standard Transformer feedforward block [6]. Layer normalisation [62] is also applied inside their vector field and feedforward block. The number of heads n_{head} and the feedforward inner dimension d_{ff} are hyper-parameters of the model, in addition to the size d_{model} of the NODE/NCDE layer. For all models considered in the main text, the number of layers is one (see Sec. C.1 below for a discussion of deeper models). We conducted hyper-parameter search in the following ranges, and selected the best configuration for each setting based on its validation performance.

For Speech Commands: $d_{\text{model}} \in \{32, 80, 128, 160\}$, $n_{\text{head}} \in \{4, 8, 16, 32\}$ and $d_{\text{ff}} = \{80, 4 * d_{\text{model}}\}$, with a learning rate $\eta \in \{1e-5, 5e-5\}$ and a batch size of 1024. The best Delta CDE model is obtained for ($d_{\text{model}} = 80$, $n_{\text{head}} = 4$, $d_{\text{ff}} = 320$, $\eta = 5e-5$).

For PhysioNet Sepsis: $d_{\text{model}} \in \{32, 80, 128, 160\}$, $n_{\text{head}} \in \{8, 16, 32\}$ and $d_{\text{ff}} \in \{64, 4 * d_{\text{model}}\}$, with a learning rate $\eta \in \{1e-5, 2e-5, 3e-5, 4e-5, 5e-5, 6e-5, 8e-5, 1e-4\}$ and a batch size of 1024. The best Delta CDE model is obtained for ($d_{\text{model}} = 80$, $n_{\text{head}} = 16$, $d_{\text{ff}} = 64$, $\eta = 3e-5$) in the "OI" case, and for ($d_{\text{model}} = 160$, $n_{\text{head}} = 32$, $d_{\text{ff}} = 64$, $\eta = 4e-5$) in the "no-OI" setting.

For EigenWorms: $d_{\text{model}} \in \{64, 128, 160\}$, $n_{\text{head}} \in \{8, 16, 32\}$ and $d_{\text{ff}} \in \{32, 64, 80, 128\}$, with a learning rate $\eta \in \{5e-5, 1e-4, 3e-4, 5e-4\}$ and a batch size set to 181 to include all training sequences (70% of the 259 total sequences in this dataset, again following the prior work [37]). The best overall Delta CDE model is obtained for ($d_{\text{model}} = 128$, $n_{\text{head}} = 16$, $d_{\text{ff}} = 64$, $\eta = 1e-4$) in the log-signature depth-1 case.

The best hyper-parameters for each model are listed in Table 3.

Table 3: Hyper-parameters. Their definitions can be found in Appendix B. H/O/D denote Hebb/Oja/Delta variants respectively. The top part shows hyper-parameters of the direct NODE models for Speech Commands and PhysioNet Sepsis, and those of the NRDE models for EigenWorms. The bottom part is for the NCDE models.

		Speech Commands			PhysioNet Sepsis OI			PhysioNet Sepsis no-OI			EigenWorms		
Type	Param.	H	O	D	H	O	D	H	O	D	H	O	D
O/RDE	d_{model}	128	128	160	160	160	128	32	160	80	160	160	128
	n_{head}	32	32	16	32	32	16	8	32	16	16	16	16
	d_{ff}	512	512	80	64	64	64	64	64	64	128	128	64
	η (1e-5)	5	5	1	6	6	2	10	4	4	30	30	10
CDE	d_{model}	128	128	80	32	80	80	160	160	160	160	128	128
	n_{head}	32	32	4	8	16	16	32	32	32	16	16	16
	d_{ff}	512	512	320	256	64	64	64	64	64	128	64	64
	η (1e-5)	5	5	5	6	2	3	4	6	4	30	30	10

Numerical solver details. For all tasks, we use the numerical solver settings of the baseline models. For all supervised learning experiments in the main text, we use the '`rk4`' Runge-Kutta variant with default tolerance parameters in `https://github.com/rtqichen/torchdiffeq` [1]: relative/absolute tolerance values are 1e-7/1e-9. For the RL experiment (Sec. C.3), '`dopri5`' is used instead, with relative/absolute tolerance values of 1e-5/1e-6. As mentioned in Sec. 5, we do not tune these configurations specifically for our models. Further enhancements in terms of performance or stability may be obtained by tuning them.

For any further details, we refer to our code.

C Extra Experiments

C.1 Deeper Models

As the datasets used here were rather small, one-layer models yield satisfactory performance. In general, however, deeper architectures are common for Transformers [63]. Here we show methods to increase the depth of FWP-based NODE/NCDE models. While the possibility of stacking multiple NCDE layers has been mentioned previously [38], no practical result has been reported.

The following equations are for the direct NODE case of our FWP models (the CDE case is analogous). Let L and l denote positive integers. The forward computation of layer l in an L-layer model at step t is as follows

$$\boldsymbol{W}(t,l) = \boldsymbol{W}(t_0,l) + \int_{s=t_0}^{t} \boldsymbol{F}_{\theta_l}^{(l)}(\boldsymbol{W}(s,l), \boldsymbol{x}(s,l-1))ds \tag{33}$$

$$\boldsymbol{x}(t,l) = \text{FFN}\left(\boldsymbol{W}(t,l)\text{softmax}(\boldsymbol{W}_q\boldsymbol{x}(t,l-1))\right) \tag{34}$$

where FFN denotes the standard Transformer feedforward block, $\boldsymbol{W}(t,l)$ denotes the fast weight matrix, $\boldsymbol{F}_{\theta_l}^{(l)}$ is the vector field with parameters θ_l, and $\boldsymbol{x}(t,l)$ denotes the output of layer $l \geq 1$ while $\boldsymbol{x}(t,0)$ is the external control signal. The output of layer l, $\boldsymbol{x}(t,l)$, thus becomes the control signal for the next layer $l+1$, and so on. This yields a system of L *coupled ODEs*. The fast weight matrix for each layer at time T can be obtained by calling the ODE solver for the corresponding coupled ODEs

$$\boldsymbol{W}(T,1), ..., \boldsymbol{W}(T,L) = \text{ODESolve}([\boldsymbol{F}_{\theta_1}^1, ..., \boldsymbol{F}_{\theta_L}^L], [\boldsymbol{W}(t_0,1), ..., \boldsymbol{W}(t_0,L)], t_0, T) \tag{35}$$

The final output $\boldsymbol{x}(T,L)$ can be computed recursively via Eq. 34 for $t = T$ starting from $l = 1$.

We conduct experiments in the EigenWorms dataset with the goal of improving our best single-layer model of Table 2. We tune the 2-layer model using the same hyper-parameter search space used for the 1-layer model. However, the best obtained 2-layer performance is 86.7 % (4.1) which is worse

than the best 1-layer performance of 91.8 % (3.4). A natural next step towards obtaining potential performance gains from deeper/larger models is to apply regularisation techniques to our models (we leave that to future work).

C.2 Ablation Studies on CDE-based Models

In Sec. 3.2 on NCDE based FWPs, we mention two optional model variations skipped in the main text: the Hebb variant which uses $\boldsymbol{x}'$ to generate keys and $\boldsymbol{x}$ to generate values, and NCDE based models which only use $\boldsymbol{x}'$ (instead of $\boldsymbol{x}$ and $\boldsymbol{x}'$) in the vector field (see footnote 4). Here we provide the corresponding ablation studies.

Using $\boldsymbol{x}'$ for key and $\boldsymbol{x}$ for value generation in the Hebb variant. In Sec. 3.2 on FWP based CDEs, we note that there are two possible formulations for the Hebb variant. The following equations highlight the difference between the two formulations (depending on the variable, $\boldsymbol{x}$ or $\boldsymbol{x}'$, used to generate keys/values):

$$\mathsf{F}_\theta\big(\boldsymbol{W}(s), \boldsymbol{x}(s), \boldsymbol{x}'(s)\big)\boldsymbol{x}'(s) = \sigma(\beta(s)) \begin{cases} \boldsymbol{W}_k\boldsymbol{x}(s) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \boldsymbol{x}\text{-keys, } \boldsymbol{x}'\text{-values as in Eq. 24} \\ \boldsymbol{W}_v\boldsymbol{x}(s) \otimes \boldsymbol{W}_k\boldsymbol{x}'(s) & \boldsymbol{x}\text{-values, } \boldsymbol{x}'\text{-keys} \end{cases} \tag{36}$$

We also use different equations for the fast net forward computation where, for consistency, we generate the query from the same input ($\boldsymbol{x}$ or $\boldsymbol{x}'$) used to generate keys (even if this is not a strict requirement):

$$\boldsymbol{y}(T) = \begin{cases} \boldsymbol{W}(T)^\top \boldsymbol{W}_q\boldsymbol{x}(T) & \boldsymbol{x}\text{-keys, } \boldsymbol{x}'\text{-values} \\ \boldsymbol{W}(T)\boldsymbol{W}_q\boldsymbol{x}'(T) & \boldsymbol{x}\text{-values, } \boldsymbol{x}'\text{-keys} \end{cases} \tag{37}$$

Experimentally, we found this second variant ("$\boldsymbol{x}$-values, $\boldsymbol{x}'$-keys") to perform worse on the Speech Commands task. It obtained a test accuracy of 83.9 % (0.4) compared to 89.5 % (0.3) achieved by the first variant we reported in the main text (Table 1).

FWP based NCDEs using only $\boldsymbol{x}'$. In Sec. 3.2, we note that tractable FWP based NCDEs can also be obtained by using only $\boldsymbol{x}'$ in the vector field (instead of $\boldsymbol{x}$ and $\boldsymbol{x}'$). The equations for the corresponding models can be obtained by replacing $\boldsymbol{x}$ by $\boldsymbol{x}'$ in Eq. 24:

$$\mathsf{F}_\theta\big(\boldsymbol{W}(s), \boldsymbol{x}'(s)\big)\boldsymbol{x}'(s) = \sigma(\beta(s)) \begin{cases} \boldsymbol{W}_k\boldsymbol{x}'(s) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \text{Hebb} \\ \big(\boldsymbol{W}_k\boldsymbol{x}'(s) - \boldsymbol{W}(s)^\top \boldsymbol{W}_v\boldsymbol{x}'(s)\big) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \text{Oja} \\ \big(\boldsymbol{W}_v\boldsymbol{x}'(s) - \boldsymbol{W}(s)\boldsymbol{W}_k\boldsymbol{x}'(s)\big) \otimes \boldsymbol{W}_k\boldsymbol{x}'(s) & \text{Delta} \end{cases} \tag{38}$$

On the Speech Commands task, we found it crucial to use both $\boldsymbol{x}$ and $\boldsymbol{x}'$. The second variants using only $\boldsymbol{x}'$ perform much worse (< 65 %) than the variants using both $\boldsymbol{x}$ and $\boldsymbol{x}'$, across all three learning rule schemes.

C.3 Model-based Reinforcement Learning (RL)

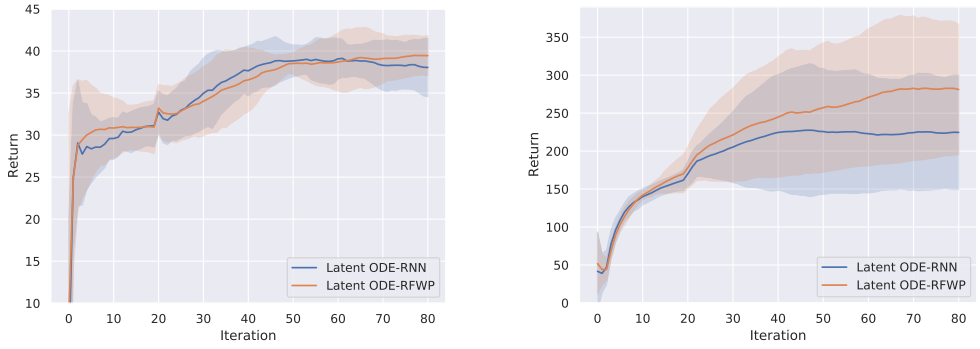
The main focus of this paper is the one of Kidger et al. [4] where it is assumed that at least some bounded (piece-wise) continuous control signal $\boldsymbol{x}(t)$ defined on $[t_0, T]$ is available. However, for the sake of completeness, in Sec. 3.3, we also present the ODE-RFWP and its latent variant in the case where we can only obtain discrete inputs. Here we provide some experimental results with working examples of such models in a model-based RL setting previously proposed by Du et al. [61].

Settings. We use the MuJoCo environments [64]. Our setting follows the version with action repetitions [65] proposed by Du et al. [61], formalised as semi-Markov decision processes [66]. The core difference to the standard setting is that there is a (state-dependent) time gap $\tau > 0$ between the time when an agent takes an action in state $\boldsymbol{s}$ and when it observes a new state $\boldsymbol{s}'$ (and can take a new action), resulting in irregularly timed observations. The time gap τ is a deterministic function of the state $\boldsymbol{s}$. Du et al. [61] proposed a model-based planning approach where an environmental

model is parameterised as an ODE-RNN or Latent ODE-RNN such that an ODE is used to model the latent state transitions between observations. The model is trained to predict state transitions (using mean squared error for ODE-RNNs and negative evidence lower bounds for Latent ODE-RNNs) and time gaps between observations, and it is used for model predictive control (MPC) [67, 68, 69]. The agent is trained by Deep Deterministic Policy Gradient (DDPG) [70, 71] with actor-critic [72, 73, 74] where the predictions of policy and value networks are conditioned on the recurrent (latent) state of the environment model. In Du et al. [61]'s work, the Latent ODE-RNN was shown to outperform the ODE-RNN and the model-free baseline in terms of sample efficiency in *Hopper*, *Swimmer* and *Half Cheetah* environments. Therefore, we compare our Latent ODE-RFWP to the baseline Latent ODE-RNN. We conducted our experiments in *Hopper* and *Swimmer* environments (we excluded *Half Cheetah* which was reported [61] to require larger models and thus more compute). On a side note, there are other works [75] with a focus on RL in continuous-time environments. However, the available environments are still limited in terms of complexity (e.g., CartPole).

Training/Model details. Each training iteration consists of interactions with the environment to collect trajectories, training of the environmental model, and policy optimisation. Following Du et al. [61], each iteration consists of 5 K environmental steps, and we train for up to 400 K steps (corresponding to a total of 80 iterations) in addition to the initial random interactions of 15 K steps. The planning horizon for MPC is set to 10 steps. We set the hidden state size of our RFWP models to 128 like in the baseline. We use 16 computational heads, and a feedforward block size of 512. We found that we can use the same RNN-based encoder used in the baseline also for our RFWP models without notable difference in terms of performance compared to an FWP-based encoder. The same training/model configurations are used for both *Swimmer* and *Hopper*. They are identical to those used by Du et al. [61] (we use their base implementation), apart from the fact that we use the continuous adjoint method to train all models. For further details, we refer to our code.

Results. Figures 1a and 1b show learning curves of baseline Latent ODE-RNN and our RFWP variant in *Swimmer* and *Hopper* environments, respectively. Given a high variability of results across seeds, we present an average over ten training runs. We observe that final performance and sample efficiency of both models are comparable on *Swimmer*, while the RFWP model yields better performance on *Hopper*. These results indicate that the Latent ODE-RFWP model presented in Sec. 3.3 can effectively in practice be used as a replacement of the standard Latent ODE-RNN. We note, however, that the final performance of baseline Latent ODE-RNN is below the one reported by Du et al. [61] (which exceeds an expected return of 350). The only change we introduced to the original setting is to consistently use the continuous adjoint method to train all models (further ablation studies may be needed to evaluate the impact of this change), and to use ten seeds instead of four.



(a) Swimmer (b) Hopper

Figure 1: Return over training iterations in the **Swimmer** and **Hopper** environments, averaged over five test episodes. Smoothing is applied on overlapping windows of size 20. One iteration corresponds to 5 K interactions with the environment. Mean/std are computed for ten runs.