Neural Hybrid Automata: Learning Dynamics with Multiple Modes and Stochastic Transitions

Michael Poli¹,³,*, Stefano Massaroli²,*, Luca Scimeca³, Seong Joon Oh³, Sanghyuk Chun⁴, Atsushi Yamashita², Hajime Asama², Jinkyoo Park¹, Animesh Garg⁵,⁶

Abstract

Effective control and prediction of dynamical systems often require appropriate handling of continuous–time and discrete, event–triggered processes. Stochastic hybrid systems (SHSs), common across engineering domains, provide a formalism for dynamical systems subject to discrete, possibly stochastic, state jumps and multi–modal continuous–time flows. Despite the versatility and importance of SHSs across applications, a general procedure for the explicit learning of both discrete events and multi–mode continuous dynamics remains an open problem. This work introduces Neural Hybrid Automata (NHAs), a recipe for learning SHS dynamics without a priori knowledge on the number of modes and inter-modal transition dynamics. NHAs provide a systematic inference method based on normalizing flows, neural differential equations and self–supervision. We showcase NHAs on several tasks, including mode recovery and flow learning in systems with stochastic transitions, and end–to–end learning of hierarchical robot controllers.

1 Introduction

Behaviors emerging from the interaction of continuous and discrete–time dynamics in the presence of uncertainty are described through the language of stochastic hybrid systems (SHSs). Such discrete events can bring along abrupt changes in the state, and in complex multi–mode systems, may also cause a switch between system modes, and corresponding underlying continuous dynamics [1]. Communication networks [2], [3], where changes in communication protocol can happen at certain levels of traffic congestion, and biological systems [4]–[6] are example domains where the SHS modeling paradigm has proven fruitful.

Data–driven identification and learning of hybrid systems are known to be challenging due to the entanglement of continuous flows and discrete events [7]; finding a generally applicable technique remains an open problem, particularly in the common scenarios where no a priori knowledge on the number and type of system modes is given. The aim of this work is to apply continuous neural models [8]–[10] to the learning SHSs. We introduce a compact descriptive language for this task, decomposing the system into a set of core primitives. Prior work is integrated into the framework, highlighting in the process limiting assumptions and areas of further improvement.

To address the shortcomings of existing techniques, we introduce Neural Hybrid Automata (NHA) as a general procedure designed to enable learning and simulation of SHSs from data. NHAs are comprised of three components: a dynamics module, taking the form of an neural differential equation (NDE) [8]–[10] capable of approximating a different vector field for each mode, a discrete latent state tracking the internal mode of the target system, and an event module determining the time to next event. In particular, our approach does not require prior knowledge on the number of modes.

*Equal contribution. Author order was decided by flipping a coin. ¹KAIST ²The University of Tokyo ³NAVER AI Lab ⁴Naver Clova ⁵University of Toronto, Vector Institute ⁶Nvidia. Corresponding author: Michael Poli (poli_m@kaist.ac.kr)

Preprint. Under review.
The synergy among NHA components ensures a broader range of applicability compared to previous attempts, which in example do not directly tackle multi–mode hybrid systems [11]–[14]. NHAs are shown to enable mode recovery and learning of systems with stochastic transitions, with additional applications in end–to–end learning of hierarchical robot controllers.

2 Background

We introduce required background on the formalism of stochastic hybrid systems (SHSs) and event handling for their numerical simulation. We then provide further contextualization on previous approaches, introducing in the process a unified language for SHS learning tasks.

2.1 Stochastic Hybrid Systems

A stochastic hybrid system (SHS) [2], [15] is a right-continuous stochastic process \( X_t \) taking values in \( \mathbb{X} \subseteq \mathbb{R}^{n_x} \) with a latent mode process \( Z_t \) conditioning the dynamics of \( X_t \), where \( t \geq 0 \). \( Z_t \) is another right-continuous stochastic process that takes values in a finite set \( \mathbb{M} \) of size \( m \). In this context, the set \( \mathbb{M} \) contains identifiers of internal system modes. An event is defined as either a mode switch or a state discontinuity (a jump in \( X_t \)), which can in some cases occur simultaneously. We refer to times at which events \( z \to z' \) occur as random variables \( t_k \in \mathcal{T} \), with associated intensity functions [16]

\[
\lambda_{z\to z'}(t|H_t) \geq 0.
\]

where \( H_t := \{ t_k \in \mathcal{T} : t_k < t \} \) is the history of event times. Intensity, as defined in the classical temporal point process (TPP) sense, can be interpreted as the expected number of events \( z \to z' \) within the time interval \([t, t + dt]\). The dynamics for \( X_t \) when \( Z_t = z \) is defined by

**Flow dynamics:**
\[
\dot{x}_t = F_{z}(t, x_t), \quad (z, t, x_t) \in \mathbb{M} \times \mathbb{T} \times \mathbb{X}
\]

When a jump event \( z \to z' \) is triggered, \( X_t \) can instantaneously jump according to

**Jump dynamics:**
\[
x_{t}^{+} = \psi_{z\to z'}(t, x_t), \quad (z, z', t, x_t) \in \mathbb{M}^2 \times \mathbb{T} \times \mathbb{X}
\]

Jump maps \( \psi_{z\to z'} \) and intensities \( \lambda_{z\to z'} \) describe the behavior during events \( z \to z' \).

2.2 Event Handling for Hybrid Systems

Following [17], to enable forward simulation of SHSs, a convenient mathematical representation of an event is a function \( g : \mathbb{T} \times \mathbb{X} \to \mathbb{R} \) which nullifies only at any event time \( t^* \), thus providing the differential equation integration algorithm with a termination or restart condition i.e.

\[
t^* \text{ is an event } \iff g(t^*, x_{t^*}) = 0.
\]  

(2.1)

The particular form of \( g \) induces a jump set on \( \mathcal{D} \subseteq \mathbb{T} \times \mathbb{X} \), \( \mathcal{D} := \{ t, x : g(t, x_t) = 0 \} \), and determines transitions from roots of \( g \) to regions of the state–space \( \mathbb{X} \) where \( g \neq 0 \). Notably, this construct enables utilization of root finding methods [18] in a neighborhood of \( t^* \) to accurately zero in on the event time.

The same simulation technique can be extended to the many jump sets case typical of multi–mode systems, by equipping the condition function with identifiers \( z, z' (g_{z\to z'}) \) which induces jump sets \( \mathcal{D}_{z\to z'} \).

Simulating stochastic events While the event function approach appears to be limited to the deterministic setting, it also subsumes stochastic events whose aleatoric uncertainty is encoded by an intensity \( \lambda(t|\mathcal{H}_t) \) [1], [13]. Without loss of generality let us consider a single intensity function which is henceforth denoted as \( \lambda^* := \lambda(t|\mathcal{H}_t) \). Recalling that the cumulative distribution function (CDF) of inter–event times is \( 1 - \exp \{- \int_{t_0}^{t^*} \lambda^*_t dt \} \), standard inverse transform sampling [1], [19] yields

\[
t^* : 0 = s - \log \int_{t_0}^{t^*} \lambda^*_t dt, \quad s \sim \text{Uniform}(0, 1)
\]

(2.2)

as a special case of (2.1). Approaches developed for learning TPPs, including in the context of Neural ODEs [8], [11], [20], introduce a parametric formulation for the intensity \( \lambda^*_\theta \) and optimize via direct TPP likelihood objectives. The integral is in general intractable, thus these methods require either a numerical approximation or the augmentation of additional states to compute it alongside the ODE.

2
We introduce Neural Hybrid Automata (NHA), a model for learning of SHSs. A NHA is comprised of a dynamics module, a discrete latent state and an event predictor. A general overview of a NHA is depicted in Figure 1. We start with a description of each module and their interconnections, followed by a step-by-step procedure for NHA training.

**Event module**  
Intensity-free parametrizations for stochastic mode transitions allows NHAs to sample next event times without solving integrals \( \int \lambda_z \, dt \) for all target modes \( z' \) reachable from current \( z \). From event \( k \) at time \( t_k \), NHAs determine next event times \( t_{k+1} \) through a conditional normalizing flow modeling, for each possible pair of \((z, z')\), the density of corresponding *inter-event* times \( \tau_{z \to z'} = t_{k+1} - t_k \). Let \( \tau_{z \to z'} \) be the parametrized conditional density obtained by the normalizing flow and let \( \mathcal{T}(z, z', t_k) \) be a collection of conditional samples from \( p_{z \to z'} \) (one for each pair \( z, z' \)), i.e.

\[
\mathcal{T}(z, z', t_k) = \left\{ \tau_{z \to z'} \sim p_{z \to z'}(\tau|\mathcal{H}_{t_k}) \right\}_{z,z' \in \mathbb{Z}}.
\]
Using (2.2), we can thus sample an event given the current mode $z$ and the previous event time $t_k$ as:

$$t_{k+1} = t_k + \min_{z' \in \mathcal{Z}} T(z, z', t_k).$$  

(3.1)

Note that the next mode $z'$ after the event is simultaneously obtained as $z' = \arg \min T(z, z', t_k)$. Sampling strategy (3.1), differently from (2.2), relies on the normalizing flow to explicitly model the density rather than defining it implicitly through $\int \lambda_t^z dt$. When event time $t_k$ is reached, a parametrized jump map conditioned on $(z, z')$ is applied to the state $x^+ = \psi_{z \rightarrow z'}(t_k, x)$. Normalizing flow $p_{z \rightarrow z'}$ and jump map $\psi_{z \rightarrow z'}$ together define the full event module of an NHA.

It should be noted that (3.1) always samples the quickest-to-occur event from the normalizing flow, which implies that no other event occurs between $t_k$ and $t_{k+1}$. While the history can be compressed into a fixed-length vector following [23] through application of sequence models e.g. RNNS, we note that for hybrid systems equipped with deterministic events, providing $(x_{t_k}, t_k)$ as conditioning inputs for $p_{z \rightarrow z'}(\tau)$ is sufficient since ODE solutions with deterministic transitions are uniquely determined by the initial condition. Finally, deterministic events are a special case of stochastic events [2] that can be well-represented with a Dirac $\delta$ function, of which the normalizing flow learns a smooth approximation with continuous support.

Dynamics module To enable approximation of different mode-dependent vector fields, we parametrize the flow map $F_z(t, x_t)$ of a SHS as a data-controlled neural ordinary differential equation (Neural ODE) [9] with parameters $\omega$, driven between each pair of event times $t_k, t_{k+1}$ by discrete latent mode $z$

$$\dot{x}_t = F_z(t, x_t, \omega) \quad t \in [t_k, t_{k+1}]$$  

(3.2)

Finiteness of admissible values in the latent mode state i.e. $m$ ensures $F$ is capable of approximating a finite number of different vector fields, one for each mode. In particular, we consider one-hot representations for latents $z \in \mathbb{R}^m$. In batched data settings, (3.2) can be integrated in parallel across $n_b$ batches of initial conditions $x_{t_k} \in \mathbb{R}^{n_b \times n_x}$ with different modes, provided the latent is also batched $z \in \mathbb{R}^{n_b \times m}$. The combination of a given dynamics and event module, applied in turn as depicted in Figure 1, enables simulation of trajectories of a SHS. We now describe their training procedure.

# 4 Neural Hybrid Automata Module Training

Here, we detail the training procedure for each NHA component. Our only assumption is availability to a trajectory segmentation routine tasked with separating the trajectories, or flows, into a collection of subtrajectories $X_t$ of potentially of different length, each produced by the system in a different mode. The routine can be as simple as detection of discontinuities in the solution by inspecting finite-differences of observations across timestamps [24], or involve additional steps such as change-point detection [25]. Providing exact event times to NHAs is not required; the segmentation routine need only partition the full dataset in $n$ disjoint sets $X_t$ s.t. $\bigcup X_t = X$ and $\bigcap X_t = \emptyset$. In addition, no knowledge of the number of modes, or topology of transitions between modes is made available to NHAs, as these are rarely available in practice.
Self-supervised mode recovery  The first stage of learning an NHA is designed to approximate the continuous dynamics under each SHS mode while simultaneously identifying modes $z$. We achieve this by framing subtrajectory reconstruction as a pretext task for mode recovery, via a reconstruction objective $\mathcal{L}_r = \frac{1}{n} \sum_{i=0}^{n} f_r(X_i, \hat{X}_i)$, being $\hat{X}_i$ subtrajectories reconstructed by the flow decoder $F_z$ via the model

$$\dot{x} = F_z(t, x, \omega) \quad t \in [t_k, t_{k+1}).$$

(4.1)

Here, a latent encoder $\mathcal{E}$ with parameters $\theta$ is tasked with extracting a latent mode state $z \in \mathbb{M}$ to steer the decoder $F_z$ towards a more accurate reconstruction. Representation limitations of Neural ODEs [9], [26] ensure that to fit the above objective the the encoder $\mathcal{E}$ has to cluster the trajectories to enable the data–control decoder to represent different vector fields for each system mode. Finiteness of admissible values in the latent state is enforced by defining $z$ as one–hot encoded sample from a parametrized categorical distribution. Backpropagating through the sampling procedure is performed via straight–through gradients [27]. System (4.1) can be regarded as an ODE trajectory autoencoder with a categorical bottleneck.

Finally, we note that trajectory segmentation serves multiple purposes during NHA training. Forward integration is significantly sped up since the ODE solves can now be parallelized across subtrajectories $X_i$ as independent samples of a batch of data, avoiding a sequential solve on full SHS trajectories. The speedups can be dramatic for multi–mode SHSs, the focus of this work, where data trajectories may need to be longer to sufficiently explore different modes.

Event and jump supervision  In addition to the learning of mode dynamics, self–supervised mode recovery objectives provides direct supervision for normalizing flows $p_{z \rightarrow z'}$ and jump maps $\psi_{z \rightarrow z'}$. More specifically, we collect times $\tau_{k}^{z \rightarrow z'}$, and jump state pairs $(x, x^\tau)$ for each pair of modes $(z, z')$ corresponding to a transition between pairs of subtrajectories clustered as $z$ (first) and $z'$ (second) by the encoder $\mathcal{E}$. We then train the jump maps $\psi_{z \rightarrow z'}$ to approximate $x \mapsto x^\tau$, and the mode conditional normalizing flow to approximate the density $p_{z \rightarrow z'}(\tau)$.

Gradient pathologies in joint learning of flows and events  When attempting simultaneous learning on the full trajectory, the parameters of flow $F_z$ can be subjected to wrong gradients from reconstruction objectives, arising from overreliance on or underreliance of the flow model on certain modes. This phenomenon can bias training, and provides strong motivation behind our segmentation–first approach, since each subtrajectory $X_i$ is associated only to a single mode.\footnote{While speedups are dependent on full trajectory and average subtrajectory lengths, in our experiment we observe at least an order of magnitude (more than 20x) in wall–clock speedups for a single training iteration.}

A visualization is provided in Figure 3, through an idealized learning task of a two–mode system. Overreliance of the flow model on mode $z = 0$, due to overestimation of event time $t_k$, leads to a decomposition of gradients $\nabla_{\omega_0} \mathcal{L}_r$; in green, gradients pushing the trajectory closer to the solution, in red, incorrect gradients pushing the trajectory further away from the ground–truth and closer to a solution belonging to a different mode. Appendix A further develops theoretical considerations on the nature of these gradients.

5 Results and Discussion

We evaluate Neural Hybrid Automata (NHA) through extensive experiments, with a focus on investigating the performance and robustness of each NHA module. A summary of experiments, objectives and ablations is provided here for clarity:

\footnote{Due to inaccuracies or noise in the segmentation algorithm, these partitions might not be perfectly separated into different modes. We experimentally investigate these effects on NHA training in Appendix B.}
\( \dot{x}_1 = (\log 2)x_1 \quad \eta \)
\( \dot{x}_2 = 0 \quad \eta_{\text{nack}} \)
\( \dot{x}_1 = 0 \quad \eta \)
\( \dot{x}_2 = 0 \quad \eta \)

**Figure 4:** [Left] Reconstruction of system trajectories through NHA vector field decoders \( F_z \) and corresponding modes \( z \) encoded by \( \mathcal{E} \) for Reno TCP. Although the encoder is initialized with more modes (10) than there are in the underlying system (3), mode clustering is sparse and accurate. [Right] Flow reconstruction test MSE for different classes of decoders. NHA decoders (10 modes) can reconstruct the flows as well as other NODE baselines, with the added benefit of being able to recover mode labels during training.

- **Reno TCP:** we carry out a quantitative evaluation on quality of learned flows (mean squared error) and quality of mode clusters recovered during self-supervision (\( v \)-measure). We also verify the robustness of NHAs to overclustering and amount of data required for event module training.
- **Mode mixing in switching systems:** we highlight and varify robustness against mode mixing, a phenomenon occurring during learning of multi-mode systems through alternative softmax parameterization of latent \( z \), such as through softmax instead of categorical samples.
- **Behavioral control of wheeled robots:** NHAs enable task-based behavioral control. We investigate a point-to-point navigation task where a higher level reinforcement learning (RL) planner determines mode switching for a lower-level optimal controller.

### 5.1 System with Stochastic Transitions

We apply NHAs to a dataset of internal state trajectories of a network transmission controller (TCP), the Reno TCP scheme [2]. The system has two states, five stochastic transitions and three modes as shown through an automata representation in Figure 5. Here, we qualitatively validate the performance of dynamics and event modules of NHAs. We simulate 40 trajectories of the system, each 200 seconds long, and segment them. No a priori knowledge on the mode of each subtrajectory is provided to the model. We perform self-supervised mode recovery to train \( F_z \) and \( \mathcal{E} \), in the process labeling the subtrajectories, then train event module normalizing flows and jump functions with the mode labels obtained. Training cross-validation strategy, with a final test fold of 15. More details on the system, architectures and data generation are reported in the Appendix B.

**Mode recovery results** First, we perform self-supervised mode recovery and verify (i) whether the mode conditioned NHA decoder \( F_z \) offers test-time TCP trajectory reconstruction of equal or better quality than other Neural ODE variants, and (ii) quality of the mode label clusters assigned by the NHA encoder and robustness to different number of latent modes \( m \). We measure (i) via test mean squared error (MSE) on reconstructed trajectories, and (ii) via \( v \)-measure [28], a metric taking values in \([0, 1]\), computed as the harmonic mean between cluster completeness and homogeneity. A \( v \)-measure of 1 indicates perfect clustering. As baselines, we collect for (i) the performance of 3 Neural ODE (NODE) variants, a zero-augmented NODE, a data-controlled NODE (DC-NODE) [9] where the latent \( z \) is the output of a multi-layer encoder, and a Latent NODE where \( z \) is sampled via reparametrization of a Normal [8]. We also provide baseline performance of a series of popular clustering algorithms tasked to cluster the subtrajectories: k-means++ [29], hierarchical [30] and DBSCAN [31]. Figure 4 provides qualitative and quantitative results for stage (i). As established by 1, NHA mode recovery outperforms all baselines by a wide margin.
We investigate softmax whereas with v–measure values close to 1. Surprisingly, we observe providing the NHA encoder with a larger number of latent modes than the 3 of the system improves clustering results. Additional details on data pre–processing, metrics and baseline design and tuning are provided in Appendix B.

**Event module results** Next, we leverage the mode labels recovered as supervision for the event module of an NHA. In all cases, we train three–layer MLPs as jump maps and two–layer spline flows [32] as normalizing flows. Figure 6 visualizes the learned densities for each stochastic transition for the standard training regime of \( n = 5 \) trajectories. We also perform an ablative study on the quantity of data required to extract sufficient supervision signal for both components of the event module. The results are included in Table 1. We find that a single trajectory is sufficient, with relative performance gains quickly dropping off after \( n = 3 \).

| Model          | \( m = 3 \)       | \( m = 5 \)       | \( m = 10 \)      |
|----------------|-------------------|-------------------|-------------------|
| k–means++     | 0.20 ± 0.02       | 0.24 ± 0.02       | 0.30 ± 0.06       |
| hierarchical  | 0.23 ± 0.01       | 0.24 ± 0.01       | 0.31 ± 0.06       |
| DBSCAN        | 0.66 ± 0.02       | 0.68 ± 0.02       | 0.69 ± 0.01       |
| NHA           | **0.86 ± 0.02**   | **0.91 ± 0.02**   | **0.96 ± 0.03**   |

**Table 1:** Quality of recovered mode clusters from NHA self–supervised training and baseline clustering algorithms in the TCP task. Hyperparameter \( m \) is the number of clusters provided to each algorithm. For DBSCAN, values of \( m \in [3, 5, 10] \) map instead to its primary parameter \( \epsilon \in [0.1, 0.5, 1] \) [31].

| Model          | Metric           | \( n = 1 \)          | \( n = 3 \)          | \( n = 5 \)          | \( n = 10 \)         |
|----------------|------------------|----------------------|----------------------|----------------------|----------------------|
|                | \( p_{z \to z'} \) NLL ↓ | 2.761                | 2.375                | 2.362                | 2.313                |
|                | \( \psi_{z \to z'} \) MSE (10^{-3}) ↓ | 1.435                | 0.018                | 0.009                | 0.003                |

**Table 2:** Quality of fit for event module components, normalizing flows \( p \) and jump maps \( \psi \). Training performed with supervising mode labels from \( n \) trajectories of TCP. We report test MSE and negative log–likelihood (NLL) estimated from a base normalizing flow model trained on ground–truth data from \( n = 500 \).

**5.2 Deterministic Switching System**

We investigate mode mixing in a three–mode switching linear system (SLS) [13]. The deterministic nature of mode transitions, as well as the absence of state jumps enables direct training of NHAs on data–trajectories without prior segmentation. This allows us to perform an ablative study on mode mixing, a phenomenon arising from latent modes \( z \) produced by the encoder \( \mathcal{E} \) via soft alternatives to categorical samples, such as by using soft tmax activations.

**Mode mixing and overclustering** We train NHAs on reconstruction of SLS trajectories. Each encoder is provided, at initialization, one additional latent mode over the three of the system. The conditioned flow \( F_z \) is constructed with three–layer MLPs. Rather than segmenting the data, we repeat sampling for \( z \) at each integration step. Figure 7 shows the state space switching boundaries and mode vector fields learned by an NHA and a baseline producing \( z \) via soft tmax rather than as categorical samples. The additional freedom provided by softmax latents \( z \in \mathbb{R}_+^3 \), \( \sum z_i = 1 \) allows fitting the trajectories by nonlinearly mixing different vector fields to approximate a single one. Instead, categorical samples cannot mix the vector fields; this ensures that the learned clustering is either sparse as shown in the Fig. 7, or latent values dedicated to the approximation of the same underlying mode dynamics are forced to learn the same vector field. Appendix B contains a visualization and analysis for this second case.

In general, categorical sampling is effective when recovery of system modes is a model objective, whereas soft tmax or other soft relaxations can be viable if only black–box fitting of data is desired.
5.3 End–To–End Learning of Hierarchical Switching Controllers for Dynamical Systems

Beyond SHS identification, the NHA framework enables learning of task–based hierarchical controllers comprising a low–level controller \( u_t := u(t, x_t, z) \) dependent on the discrete mode \( z \) provided by a higher–level policy \( \pi \). Each NHA module is adapted as:

\[
\begin{align*}
\text{dynamics module:} & \quad \dot{x}_t = F(t, x_t, u_t)dt \\
\text{event module:} & \quad z' \leftarrow \pi(t, x_t, z) \quad \text{high–level planner}
\end{align*}
\]  

(5.1)

Within this context, latent state \( z \) can be regarded as a system set–point (e.g. a desired value of state \( x \)) determined by the planning policy \( \pi \) to achieve a certain task, which the low–level controller has then to carry out. Both \( \pi \) and \( u \) are parametrized by neural networks, and the training is done end–to–end. The obtained hierarchical control scheme is sample efficient, since system dynamics available a priori are included in \( F \). From the perspective of \( \pi \), however, the dynamics are disentangled from the planning objective. Indeed, the higher level policy need only learn how to set and switch between objectives \((z \to z')\), and not how to control the system to reach them. For this reason we train the model using a loss function partitioned in two terms as \( \mathcal{L} = \mathcal{L}_u + \mathcal{L}_\pi \).

**Results** We consider learning controllers for navigation of two–wheeled differential drive robots [33]. The higher–level model–free policy \( \pi \) is here trained via REINFORCE gradients [34], [35] to
select the nearest resource target every 5 seconds. We set 5 different resources within a map, and train all robots to: one, select the nearest resource target; two, drive the wheels to reach the target. The training of both controllers is carried out concurrently, where target (or mode) selection is performed by $\pi$, whereas the behavioural controller $u_z$ has to reach the target chosen by $\pi$ via low–level steering control inputs. As shown in Figure 8, convergence of both control policies occurs after around 4000 episodes of training. Figure 8 also visualizes the resulting navigation behavior at test–time on two new resource layouts, where we alternate between two different sets of targets.

5.4 Generalizable Insights and Empirical Observations

The task of learning SHSs involves several moving components. Ablative experiments have been performed to address specific questions on the robustness of NHAs. We detail some empirical heuristics observed to improve performance, and report areas of further improvement.

- Overclustering stabilizes training We empirically observe providing NHA with more latent modes than the system stabilizes training. We conjecture the additional choice allows the model to use different modes during exploration without always conflicting with other already "assigned" modes, phenomenon which is more frequent, in example, when the number of modes exactly matches that of the system. In these cases, dropout in the encoder $E$ appears to improve performance.

- Noisy segmentation of trajectories We investigate, for the TCP experiment, robustness of mode recovery to incorrect segmentation (Appendix B) and number of NHA latent modes $m$ (Table 1). Extending NHAs to include a finetuning step for trajectory segmentation, in example leveraging ideas from [13] might improve robustness of the segmentation routine and thus the overall approach.

6 Related Work

Hybrid system identification and Markov models Hybrid system identification is a relatively recent development in dynamical system theory [7]. A majority of existing literature focuses on (linear) piecewise affine systems (PWA) [36], [37]. [38] proposes a clustered symbolic regression algorithm for learning input–output maps rather than dynamics. Existing approaches involving continuous optimization [21] do not consider event stochasticity and mode recovery. Identification of SHSs is an even smaller field, with limited success outside specific cases [15].

Continuous–depth and contact models Neural differential equations and continuous–depth models, initially concerned with unimodal systems [8], [9], have seen preliminary application to the learning of temporal point processes [11], [39]. Although some of these works tackle stochastic events and marked point processes, multimodality and explicit learning of the flows is not considered. [12] examine interventions as events, and develop a continuous architecture for modeling the lasting effect of a given intervention on the dynamics. Differentiable contact approaches [14], [40] introduce physics–compatible models designed to recover deterministic hybrid dynamics of mechanical systems from data. [13] develops, through implicit differentiation, a method for direct optimization of event times. Although Neural Event ODEs do not directly address multimodality, a potential synergy between the approach of [13] and NHAs could preserve the advantage of our flow–parallel mode recovery, namely integration speed and sidestepping of gradient pathologies outlined in Section IV. Section 2.3 provides a summary of limitations for these existing methods.

7 Conclusion

Hybrid systems represent a versatile and general class of systems, with applications across engineering disciplines [15]. In this work, we investigate challenges related to the learning of SHSs from data and introduce Neural Hybrid Automata (NHA), a step–by–step method leveraging neural differential equations, density estimation and self–supervised system mode recovery. NHAs are shown to be effective in various settings, including flow and event learning in systems with stochastic transitions.
References

[1] C. G. Cassandras and J. Lygeros, *Stochastic hybrid systems*. CRC Press, 2018.
[2] J. P. Hespanha, “Stochastic hybrid systems: Application to communication networks,” in *International Workshop on Hybrid Systems: Computation and Control*, Springer, 2004, pp. 387–401.
[3] G. Sun, C. G. Cassandras, Y. Wardi, C. G. Panayiotou, and G. F. Riley, “Perturbation analysis and optimization of stochastic flow networks,” *IEEE Transactions on Automatic Control*, vol. 49, no. 12, pp. 2143–2159, 2004.
[4] R Alur, C Belta, F Ivancic, V Kumar, M Mintz, G Pappas, H Rubin, J Schug, and G. Pappas, “Hybrid modeling and simulation of biological systems,” *Hybrid Systems: Computation and Control*, vol. 2034, pp. 19–32, 2001.
[5] J. P. Hespanha and A. Singh, “Stochastic models for chemically reacting systems using polynomial stochastic hybrid systems,” *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 15, no. 15, pp. 669–689, 2005.
[6] X. Li, O. Omotere, L. Qian, and E. R. Dougherty, “Review of stochastic hybrid systems with applications in biological systems modeling and analysis,” *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 2017, no. 1, pp. 1–12, 2017.
[7] F. Lauer and G. Bloch, “Hybrid system identification,” in *Hybrid System Identification*, Springer, 2019, pp. 77–101.
[8] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, “Neural ordinary differential equations,” in *Advances in neural information processing systems*, 2018, pp. 6571–6583.
[9] S. Massaroli, M. Poli, J. Park, A. Yamashita, and H. Asama, “Dissecting neural odes,” *arXiv preprint arXiv:2002.08071*, 2020.
[10] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. Ramadhan, “Universal differential equations for scientific machine learning,” *arXiv preprint arXiv:2001.04385*, 2020.
[11] J. Jia and A. R. Benson, “Neural jump stochastic differential equations,” in *Advances in Neural Information Processing Systems*, 2019, pp. 9843–9854.
[12] D. Gwak, G. Sim, M. Poli, S. Massaroli, J. Choo, and E. Choi, “Neural ordinary differential equations for intervention modeling,” *arXiv preprint arXiv:2010.08304*, 2020.
[13] R. T. Chen, B. Amos, and M. Nickel, “Learning neural event functions for ordinary differential equations,” *arXiv preprint arXiv:2011.03902*, 2020.
[14] Y. D. Zhong, B. Dey, and A. Chakraborty, “A differentiable contact model to extend lagrangian and hamiltonian neural networks for modeling hybrid dynamics,” *arXiv preprint arXiv:2102.06794*, 2021.
[15] C. G. Cassandras and J. Lygeros, “Stochastic hybrid systems,” *Automation and Control Engineering*, vol. 24, 2007.
[16] D. J. Daley and D. Vere-Jones, *An introduction to the theory of point processes: volume II: general theory and structure*. Springer Science & Business Media, 2007.
[17] L. Shampine and S Thompson, “Event location for ordinary differential equations,” *Computers & Mathematics with Applications*, vol. 39, no. 5-6, pp. 43–54, 2000.
[18] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.
[19] J. G. Rasmussen, “Temporal point processes: The conditional intensity function,” *Lecture Notes, Jan., 2011.*
[20] R. T. Chen, B. Amos, and M. Nickel, “Neural spatio-temporal point processes,” *arXiv preprint arXiv:2011.04583*, 2020.
[21] F. Lauer, G. Bloch, and R. Vidal, “A continuous optimization framework for hybrid system identification,” *Automatica*, vol. 47, no. 3, pp. 608–613, 2011.
[22] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2.
[23] O. Shchur, M. Biloš, and S. Günnemann, “Intensity-free learning of temporal point processes,” *arXiv preprint arXiv:1909.12127*, 2019.
[24] S. Massaroli, F. Califano, A. Faragasso, M. Risiglione, A. Yamashita, and H. Asama, “Identification of a class of hybrid dynamical systems,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 875–882, 2020.
[25] S. Aminikhanghahi and D. J. Cook, “A survey of methods for time series change point detection,” *Knowledge and information systems*, vol. 51, no. 2, pp. 339–367, 2017.
[26] E. Dupont, A. Doucet, and Y. W. Teh, “Augmented neural odes,” in *Advances in Neural Information Processing Systems*, 2019, pp. 3134–3144.
[27] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *arXiv preprint arXiv:1308.3432*, 2013.
[28] A. Rosenberg and J. Hirschberg, “V-measure: A conditional entropy-based external cluster evaluation measure,” in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 410–420.
[29] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” Stanford, Tech. Rep., 2006.
[30] F. Murtagh and P. Contreras, “Algorithms for hierarchical clustering: An overview,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.
[31] D. Birant and A. Kut, “St-dbscan: An algorithm for clustering spatial–temporal data,” *Data & knowledge engineering*, vol. 60, no. 1, pp. 208–221, 2007.
[32] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, “Neural spline flows,” *arXiv preprint arXiv:1906.04032*, 2019.
[33] S. K. Malu and J. Majumdar, “Kinematics, localization and control of differential drive mobile robot,” *Global Journal of Research In Engineering*, 2014.
[34] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[35] J. Peters and S. Schaal, “Policy gradient methods for robotics,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006, pp. 2219–2225.
[36] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, and R. Vidal, “Identification of hybrid systems a tutorial,” *European journal of control*, vol. 13, no. 2-3, pp. 242–260, 2007.
[37] R. L. Westra, M. P. Ralf, and L. Peeters, “Identification of piecewise linear models of complex dynamical systems,” *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 14 863–14 868, 2011.
[38] D. L. Ly and H. Lipson, “Learning symbolic representations of hybrid dynamical systems,” *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 3585–3618, 2012.
[39] Y. Rubanova, T. Q. Chen, and D. K. Duvenaud, “Latent ordinary differential equations for irregularly-sampled time series,” in *Advances in Neural Information Processing Systems*, 2019, pp. 5321–5331.
[40] A. Hochlehnert, A. Terenin, S. Sæmundsson, and M. Deisenroth, “Learning contact dynamics using physically structured neural networks,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2021, pp. 2152–2160.
[41] A. L. Mitchell, M. Engelcke, O. P. Jones, D. Surovik, S. Gangapurwala, O. Melon, I. Havoutis, and I. Posner, “First steps: Latent-space control with semantic constraints for quadruped locomotion,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5343–5350.
[42] Y. D. Zhong and N. Leonard, “Unsupervised learning of lagrangian dynamics from images for prediction and control,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.
[43] A. M. Johnson, S. A. Burden, and D. E. Koditschek, “A hybrid systems model for simple manipulation and self-manipulation systems,” *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1354–1392, 2016.
[44] P. Holmes, R. J. Full, D. Koditschek, and J. Guckenheimer, “The dynamics of legged locomotion: Models, analyses, and challenges,” *SIAM review*, vol. 48, no. 2, pp. 207–304, 2006.
[45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.
[46] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, “Pyro: Deep universal probabilistic programming,” *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 973–978, 2019.
[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., “Scikit-learn: Machine learning in python,” the Journal of machine Learning research, vol. 12, pp. 2825–2830, 2011.

[48] J. R. Dormand and P. J. Prince, “A family of embedded runge-kutta formulae,” Journal of computational and applied mathematics, vol. 6, no. 1, pp. 19–26, 1980.
A Additional Discussion and Theory

A.1 Neural Hybrid Automata: Modules and Hyperparameters

We provide a notation and summary table for Neural Hybrid Automata (NHA). The table serves as a quick reference for the core concepts introduced in the main text.

1. **Dynamics**: tasked with approximating continuous dynamics of each mode by conditioning a Neural ODE on mode $z$.

2. **Mode Encoder**: only used during self-supervised mode recovery. Labels every subtrajectory $X_i$ with a mode $z$ to ensure mode-conditioned decoder $F_z$ can reconstruct it despite Neural ODE representation limitations (uniqueness of solutions given an initial condition).

3. **Event Module**: determines during simulation (i) when events happen, and what types of events i.e. mode transitions ($z \rightarrow z'$) through $p_{z\rightarrow z'}$, (ii) what happens during such events i.e. jumps on the state via $\psi_{z\rightarrow z'}$. Normalizing flow $p_{z\rightarrow z'}$ is trained to approximate densities $p(\tau_{z\rightarrow z'}|\mathcal{H})$.

The only NHA hyperparameter beyond module architectural choices is $m$, or number of latent modes provided to the model at initialization. Performance effects of changing $m$ have been explored in Section 5.2 and Appendix B.2. Appendix B.2 further provides analyzes potential techniques to prune additional modes.
A.2 Gradient Pathologies

We provide some theoretical insights on the phenomenon of gradient pathologies with the simple example of a one-dimensional linear hybrid system with two modes and a timered jump,

\[ \dot{x}_t = \begin{cases} ax_t & t < \tau \\ bx_t & t \geq \tau \end{cases}, \quad x_t^+ = cx_t \quad t = \tau \]

We let the system to evolve in a compact time domain \( T = [0, 1] \) such that \( \tau \in T \). Given an initial condition \( x_0 \in \mathbb{R} \), the solution \( x_1 \) can be obtained as follows

\[ x_\tau = e^{a\tau}x_0 \]
\[ x_\tau^+ = cx_\tau = ce^{a\tau}x_0 \]
\[ x_1 = e^{b(1-\tau)}x_\tau^+ = ce^{b(1-\tau)}e^{a\tau}x_0 = ce^{a\tau+b(1-\tau)}x_0 \]

1. integrate 1st flow until \( t = \tau \)
2. at \( t = \tau \) apply jump
3. integrate 2nd flow from \( t = \tau \) to \( t = 1 \)

Alternatively, we can compactly write the solution at any time \( t \in T \) as

\[ x_t = \begin{cases} e^{at}x_0 & t < \tau \\ ce^{a\tau+b(t-\tau)}x_0 & t \geq \tau \end{cases} \]

Using the previous equation we can compute the gradient of solutions w.r.t. the parameters \( a, b, c, \tau \).

In particular, we have

\[ \frac{dx_t}{da} = \begin{cases} te^{at}x_0 & t < \tau \\ \tau e^{at+b(t-\tau)}x_0 & t \geq \tau \end{cases} \]
\[ \frac{dx_t}{db} = \begin{cases} 0 & t < \tau \\ (t-\tau)ke^{a\tau+b(t-\tau)}x_0 & t \geq \tau \end{cases} \]
\[ \frac{dx_t}{dc} = \begin{cases} 0 & t < \tau \\ e^{a\tau+b(t-\tau)}x_0 & t \geq \tau \end{cases} \]
\[ \frac{dx_t}{d\tau} = \begin{cases} 0 & t < \tau \\ (a-b)ke^{a\tau+b(t-\tau)}x_0 & t \geq \tau \end{cases} \]

Now let us consider a loss function computed on the mesh solution points of the trajectory

\[ L = \sum_{k=1}^{K} \gamma(x_{t_k}), \quad 0 < t_1 < \cdots < t_S < 1, \ t_s \neq \tau \]

of which we wish to obtain the minimizers \( a^*, b^*, c^*, \tau^* \) via e.g. application of gradient descent methods. The gradient of the cost function w.r.t. any of the parameters \( \theta \in \{a, b, c, \tau\} \) is given by

\[ \frac{dL}{d\theta} = \sum_{k=1}^{S} \frac{\gamma(x_{t_k})}{dx} \frac{dx_{t_k}}{d\theta}. \]

Simultaneous estimation of both the optimal dynamic parameters \( a^*, b^*, c^* \) and a randomly initialized event time \( \tau^* \), will result in gradients of certain parameters to vanish or be completely incorrect.

Specifically, we note that parameter \( \tau \) determines, beyond the specific time when the jump event occurs, also which parameters are responsible for computation of solution points \( x_{t_k} \). Consider
the following two scenarios, where mode 1 is the first vector field of (A.1) and 2 is the second (post–event):

1. Initialization of $\tau$ is an over–estimation of $\tau^*$ at the beginning of training. If this is the case, for $t_s$ such that $\tau > t_s > \tau^*$ the mode is missclassified i.e. should be 2, but is still 1 due to the delayed event time $\tau$. The gradient w.r.t $b$ of loss computed on solution points $x_{tk}, \tau > t_s > \tau^*$ is then wrongfully set to zero.

2. $\tau$ is an under–estimation of $\tau^*$. The same reasoning applies, except that for $\tau^* > t_s > \tau$ the mode is misclassified to 2, although it should be 1. This, in turn, affects the gradients for $b$, which results different than 0 despite the fact that $b$, from (A.1) should not be affecting the solution at points $t_s < \tau^*$. Any value taken by this gradient is thus incorrect, and can greatly affect the optimization procedure.

We have shown how gradient pathologies exist even in the idealized linear case. In nonlinear systems with multiple events (including stochasticity) these effects can have a great empirical effect on a training procedure. The trajectory segmentation first approach of NHAs is designed to minimize their impact.

### A.3 Extensions and Limitations

**Automata reconstruction via symbolic regression** NHAs with categorical encoders recover either a representation using the minimum number of modes necessary, corresponding to those of the system, or can be pruned due to immunity to mode mixing (discussed in Appendix B.3).

This property allows application of symbolic regression (SR) to reconstruct an analytic expression for each differential equation driving a system mode. This step grants domain experts a method to validate and certify the results, and enables construction of a human–readable automata representation for the SHS.

Clustered SR results can be improved by leveraging the universal differential equation approach employed in example by [10] for unimodal differential equations, by utilizing the decoder $F_z$ as an interpolating source of additional trajectory data for each mode.

**Latent hybrid automata from observations** Learning methods for dynamical systems often introduce structure in latent space to enable control and identification from raw observations [41], [42]. Practical application for hybrid systems, such as robotic manipulation [43], locomotion [44], and traffic networks [2], might benefit from learning models structurally equipped with latent NHAs.

Optimal design of NHA modules for latent applications remains a difficult open question, as the analysis of deep models with latent spaces designed to evolve in continuous–time is also in its infancy.

**Unified benchmark for model development** Despite the importance of hybrid systems in engineering, wide differences in techniques across domains have historically made it difficult to develop and preserve a unified set of benchmarks.

Evidence from other deep learning disciplines e.g. computer vision highlights the importance of consistent and curated benchmark datasets to track and measure the impact architecture and method optimizations. We argue further benchmark design, along with larger datasets, to be a necessary step required to trigger an ImageNet–like [45] moment for general neural differential equations and thus also NHAs. As an additional challenge, we note that performance of continuous neural models is in general highly impacted by the numerical method used for forward and backward inference, with optimal methods usually system or application dependent. This makes decoupling architecture improvements from the numerical underpinnings harder than for traditional models.

### A.4 Detailed Feature Comparisons with Related Work

Table 3 compares the proposed method with recent learning based approaches in terms of features. We use $\checkmark$ for features that are either absent or incompatible with a given method, or features that have not been tested or verified, although the method itself may be adapted to include it.

We consider the following:
• **Recovery of flows and events:** can state–space vector fields be learned along with the events. In [11], learned latent dynamics aid in the intensity parametrization of the point process. State–space dynamics are not learned simultaneously with point process maximum likelihood training. [13] trains a neural network vector field along with a parametrized event function.

• **Stochastic events:** has the method been shown to be compatible with stochastic events. The formulation of [13] can parametrize stochastic events via inverse sampling, but no experiments have been performed, likely due to difficulties in learning stochastic events from a full trajectory.

• **Mode identification:** does the method recover modes of a multi–mode hybrid system, and can the vector field approximate a different dynamics for each. NHAs are the first method to tackle this setting.

• **Adaptive end–time:** can the method adjust event times by calculating gradients with respect to integration end–times. Core contribution of [13] is an implicit differentiation formulation to adapt end-times. While adaptive segmentation has been discussed as being compatible with NHAs, no targeted experiments on this technique have been carried out. The extension is left as future work.

• **Intensity–free parametrization:** does the method use intensity–free parametrizations to avoid numerically solving integrals to sample from next–event densities. [11], [13] parametrize the intensity as only a single mode is considered. NHA use normalizing flows to approximate these densities directly, since intensity parametrization scale poorly as the number of system modes increases.

### A.5 Broader Impact

This work represents a first attempt in developing a data–driven, learning based technique for stochastic hybrid system (SHS) identification and control. As discussed in the main text, existing methods currently rely on strict assumptions that severely limit their utilization in practice. Applications domain the SHS formalism provides an accurate language to describe a target system are most likely to be affected by the availability of NHAs as a method to improve partial mathematical models using data or construct from scratch a model of the system and its automata representation. The impact of NHAs here is thus likely to be similar in scope to the impact of neural differential equations in science and engineering.

We do not anticipate significant negative environmental impact from the adoption of NHAs as these models are still orders of magnitude smaller than other large deep learning architectures for domains such as natural language.

### B Experimental Details

**Hardware and software resources** Experiments have been performed on a workstation equipped with a 48 threads AMD Ryzen Threadripper 3960X, a NVIDIA GeForce RTX 3090 GPUs and two NVIDIA RTX A6000. All models and datasets fit in a single GPU. The software implementation of NHA leverages the PyTorch framework. ODE solvers and numerical methods for hybrid systems have been developed from scratch and are included in the submission.

**Common experimental settings** In all experiments, unless specified, the NHA mode encoder $E_\theta$ is capped with a softmax activation computing the probabilities of a categorical distribution from which then the one–hot mode $z$ is sampled.

| Method                  | Recovery of flow + events | Stochastic events | Mode identification | Adaptive end–time | Intensity–free parametrization |
|-------------------------|---------------------------|-------------------|---------------------|------------------|-------------------------------|
| NJSDE [11]              | ✓                         | ✓                 | ✓                   | ❌               | ❌               |
| Neural Event ODE [13]   | ✓                         | ❌                 | ✓                   | ✓                | ❌               |
| **NHA (this work)**     | ✓                         | ✓                 | ❌                   | ✓                | ✓                |

*Table 3: Feature comparison between neural models for hybrid systems. ❌ is used to indicate features that are either not compatible, or have not been verified in the original work.*
where simultaneously ensuring that the gradients backpropagate directly through the probabilities.

which can be implemented with a \texttt{stop\_gradient} (the Neural ODEs is zero–augmented \cite{9} i.e. \( \mathcal{E}_\theta \)). The complete experiment on learning the Reno TCP system involves multiple stages: (i) mode clustering (ii) include standard clustering algorithms \texttt{k–means++} \cite{29}, hierarchical \cite{30} and \texttt{DBSCAN} \cite{31}. We use \texttt{scikit–learn} \cite{47} implementation of all baseline algorithms. We perform light hyperparameter tuning on ground–truth labels for \texttt{k–means++} and hierarchical to optimize their

| Simulation Hyperparameter | Value                  |
|----------------------------|------------------------|
| Number trajectories       | 40                     |
| ODE solver                | Dormand–Prince          |
| Tolerances (abs, rel, event) | \(10^{-6}, 10^{-6}, 10^{-4}\) |
| \(\tau_{\text{eff}}\)    | 3                      |
| \(\eta\)                 | 1                      |
| \(n_{\text{ack}}\)       | 2                      |
| \(p_{\text{drop}}\)      | 0.05                   |
| \(\kappa\)               | 4                      |

| Training Hyperparameter   | Value                  |
|----------------------------|------------------------|
| Training iterations (mode recovery) | 4000               |
| Encoder \(\mathcal{E}_\theta\) learning rate | \(5 \cdot 10^{-4}\)   |
| Decoder \(F_z\) learning rate | \(10^{-2}\)          |
| Optimizer                 | \texttt{Adam}         |
| \(\mathcal{E}_\theta\) layer dimensions | \([64, 65, 64, m]\) |
| \(\mathcal{E}_\theta\) activation | ReLU                  |
| \(\mathcal{E}_\theta\) dropout | \([0.3, 0.3, 0.3]\)  |
| \(F_z\) layer dimensions | \([2 + m, 2]\)        |
| Training iterations (event training) | 4000               |
| Learning rate             | \(2 \cdot 10^{-3}\)   |
| Optimizer                 | \texttt{Adam}         |
| \(\psi_{z \rightarrow z'}\) layer dimensions | \([2, 32, 2]\) |

Table 4: Hyperparameters of [Left] TCP data simulation [Right] NHA in the TCP experiment, mode recovery and event module training.

Gradients through the sampling operation are computed via \textit{straight–through–estimation} (STE) \cite{27}, which can be implemented with a \texttt{stop\_gradient} (e.g. \texttt{detach} in PyTorch) operation present in modern deep learning frameworks. Let \(z\) be the one–hot representation of system mode, and \(p\) a vector of probabilities for the corresponding categorical distribution, computed as output of a neural network parametrized NHA encoder \(\mathcal{E}_\theta\). STE can be realized in a single line as \(z – p.\texttt{stop\_gradient}() + p\). STE ensures the output of the encoder \(\mathcal{E}_\theta\) is strictly one–hot encoded, while simultaneously ensuring that the gradients backpropagate directly through the probabilities.

The data–controlled Neural ODE decoder \(F_z\) incorporates mode information to select

\[
\hat{x} = \sum_{i=1}^{m} z_i \cdot f_i(t, x, \omega_i)
\]

where \(z\) is one–hot encoded, and thus only a single neural network vector field \(f_i\) with parameters \(\omega_i\) determines the solution.

**B.1 Identification of Reno TCP**

**Experimental setup**  Tables 4 provide hyperparameters for data simulation and training of NHA. Unless otherwise specified simulation, training and testing is repeated for 10 different random seeds.

The complete experiment on learning the Reno TCP system involves multiple stages: (i) mode recovery and (ii) training of the NHA event module. As baselines for (i), we consider several Neural ODE variants similar to NHA decoder \(F_z\), with latent \(z\) obtained in different ways. Latent Neural ODEs obtain \(z\) as sampled for Normal distribution \(\mathcal{N}_\theta := \mathcal{N}(\mu_\theta, \sigma_\theta)\) parametrized by a neural network matching the architecture of \(\mathcal{E}_\theta\). Reparametrization is used to backpropagate through the sampling procedure. Data–controlled Neural ODEs (DC–NODEs) are comparable to Latent Neural ODEs model with the major difference in the computation of latents as \(z = g_\theta(X)\) with \(g_\theta\) once again matching \(\mathcal{E}_\theta\). All decoders \(F_z\) are equivalent, except in the case of Augmented Neural ODEs (the Neural ODEs is zero–augmented \cite{9} i.e. \(z := 0\)) where the absence of the encoder is balanced by a more expressive decoder with three–layers. The normalizing flows \(\psi_{z \rightarrow z'}\) in NHA event modules are designed as spline flows \cite{32} with two layers. We use the standard implementation of spline flows in Pyro \cite{46}.

During mode recovery, all models are trained on 5–folds of 5. Training is performed by parallel integration across subtrajectories \(X_i\) using the \texttt{Runge–Kutta4} explicit solver. All gradients are computed via reverse–mode automatic differentiation. We test models with lowest cross–validation MSE loss, since cross–validation v–score would not be available without ground–truth labels. We find that lowest reconstruction loss often correlated with best v–measure. Baselines for mode clustering (ii) include standard clustering algorithms \texttt{k–means++} \cite{29}, hierarchical \cite{30} and \texttt{DBSCAN} \cite{31}. We use \texttt{scikit–learn} \cite{47} implementation of all baseline algorithms. We perform light hyperparameter tuning on ground–truth labels for \texttt{k–means++} and hierarchical to optimize their
performance in the range $m \in [3, 5, 10]$. DBSCAN is similarly tuned to optimize its performance with parameter $\epsilon$ indicating the maximum size of neighbourhoods around a data point. Subtrajectories classified as noise by DBSCAN are counted as incorrectly clustered.

We observe DBSCAN performance to be correlated to NHA self-supervised mode recovery. Both methods excel when density of data points under some metric is indicative of cluster separation. However, NHA self-supervision relies on the additional inductive bias of data points in a subtrajectory representing observations of a solution of an ODE, whereas DBSCAN does not. The denser the trajectories, the more restricting the Neural ODE representation limitations, and the easier each cluster is to find. Due to the similarity in their working principle, DBSCAN performance can be used as a quick sanity check to determine whether the dataset is suitable to NHA mode recovery.

### B.2 Robustness to Segmentation Noise

We investigate robustness of NHA mode recovery and related baselines to a noisy segmentation in subtrajectories $X_i$. To simulate incorrect segmentations, we collect segmentation indices and perturb them by adding or removing an uniformly sampled from $[1, 10]$. Each index has a $p$ probability of being corrupted by noise, and we repeat mode recovery with $p \in [0.1, 0.3, 0.5]$ (3 times per $p$). Shifting left or right by values sampled from $[1, 10]$ results in significant data corruption; certain subtrajectories, being shorter than 10 points, can be completely absorbed into a different subtrajectory. Table 5 reports the differences in v–measure compared to the results of Section 5.1. For each baseline, we report best results in terms of hyperparameter $m \in [3, 5, 10]$, typically $m = 10$, corresponding to $\epsilon = 1$ for DBSCAN. Although specific perturbation patterns may affect the model more than others, the trend uncovered by the results is clear.

### B.3 Switching Linear System and Mode Mixing

#### Experimental setup
We considered the two–dimensional switching linear system reported in [20], described by the dynamics

$$\begin{align*}
    (\dot{x}_t, \dot{y}_t) &= f(x_t, y_t) := \begin{cases}
        (-y_t, x_t + 2) & \text{if } x_t \geq 2 \\
        (-1, -1) & \text{if } x_t < 2 \land y_t \geq 0 \\
        (1, -1) & \text{if } x_t < 2 \land y_t < 0
    \end{cases}
\end{align*} \quad (B.1)
$$

We performed an ablation study on the effect of the categorical sampling for the mode selection in NHAs in presence of redundant “free” modes. In particular, we considered the following learning model

$$\begin{align*}
    (\dot{x}_t, \dot{y}_t) &= \sum_{i=1}^{4} w_i^f f_i(x_t, y_t) \quad (B.2)
\end{align*}$$

with one redundant mode. $F_i (i = 1, 2, 3, 4)$ was two-layers neural networks with 32 neurons each, softplus activation on the first hidden layer and hyperbolic tangent activation on the second one. We then defined two variants of the model: a first variant with $w_i = (w_i^1, w_i^2, w_i^3, w_i^4)$ directly obtained via softmax normalization of the output of a neural network $g$,

$$w_i = \text{softmax} \; g(x_t, y_t);$$
and a second one where \( w_t \) is obtained by a categorical sample conditioned by \( g(x_t, y_t) \), i.e.

\[
\forall t \quad w_t \sim \text{categorical}(\text{softmax} \ g(x_t, y_t))
\]

\( g \) was fixed as a neural network made up by two layers with 64 units and SiLU (swish) activation. The two models were trained on a \( L_1 \) reconstruction loss of nominal trajectories of the system (B.1). We introduced a regularization term penalizing the squared error on un–normalized finite differences of nominal/reconstructed trajectories as a proxy for the vector field information.

**Mode pruning**  Uniqueness theorems for ODE solutions guarantee that, given an initial condition and a mode latent code \( z \), the decoder \( F_z \) will always produce the same trajectory. Immunity to mixing for categorical latents enables mode pruning and recovery of a minimal representation. If \( \mathcal{L}_r \) saturates, the encoder has not been initialized with a sufficient number of modes \( m \). Redundant modes may be pruned, in example, by merging them if a similarity measure between the corresponding vector fields \( F_{z_i}, F_{z_j} \), e.g. difference in a given norm calculated on data trajectories, is small enough.

Figure 10 provides an example result of the second scenario discussed in Section 5.2, where categorical NHA encoders use more than a single latent mode for a target underlying mode. However, due to their immunity to mode mixing, the vector fields are equivalent, and can be merged. We show this in Figure 9, where the L1–norm between \( F_2 \) and \( F_3 \) is shown to be small in the region where the corresponding modes 2 and 3 are active.

**B.4 End–To–End Learning of Hierarchical Controllers for Dynamical Systems**

**Experimental setup**  Our objective is to control a swarm of differential drive robots moving on a planar space. The system dynamics are

\[
\begin{align*}
\dot{x}_1 &= u_v \cos \vartheta \\
\dot{x}_2 &= u_v \sin \vartheta \\
\dot{\vartheta} &= u_r
\end{align*}
\]

where \( \vartheta \) is the orientation of a single robot. The control \( [u_v, u_r] \in \mathbb{R}^2 \) is obtained via the low–level, time–invariant feedback controller \( u_z := u(x_1, x_2, \vartheta, z) \), with \( z \) produced and switched every 5 seconds by the planner. Both the controller and the policy \( \pi \), are parametrized by a neural network. We use Adam for both networks, with learning rates \( 10^{-3}, 10^{-3} \).

The training of low–level controller \( u_z \) and high–level planner \( \pi \) is carried out concurrently. We perform batch training on robot swarms of \( N = 80000 \). At the beginning of each episode, we sample
the initial conditions uniformly in a square region, with each robot rotated according to a random orientation also uniformly sampled in $[0, 2\pi]$. During each episode, we simulate the switching and control behaviour of each robot with respect to two pre-defined map layouts, both shown in the alternating resource pattern of the upper-left plot of Figure 8a. Each map layout has $M = 5$ resource locations, where two auxiliary scalar variables $r_1$ and $r_2$ specify, for each resource, its planar location.

We train policy networks $\pi(x_1, x_2, r_{11}, r_{21}, \ldots, r_{1m}, r_{2m})$ where the inputs correspond to the concatenated robot states, together with the flattened resource locations. The $M$-dimensional softmax output determines a categorical probability distribution over the resources is then used to sample a resource target.

A reward can be assigned to each robot based on the ability to select the correct target. For each robot, we generate a reward of $1$ if the selected target is correct, and $0$ otherwise. Let $s$ be the categorical sample from the distribution over the targets, and $p$ the reference targets computed by $\arg\min_j ||[r_{1j}, r_{2j}] - [x_1, x_2]||$ (note $R \in [0, 80]$ for any one episode), with $x_1, x_2$ being the robot location at the time of switching. The target selector is trained by minimizing a $\mathcal{L}_\pi = -\frac{1}{T} \sum T \ln(\pi(s_p^t) * R)$, where $T$ is the number of alternating maps, and $s_p^t$ denotes the concatenated robot location and resource map with respect to the $t$th map layout used for training.

The target selected by the planned informs low–level controller $u_2$ via the corresponding resource location. In particular, we provide as input to $u_2$ the coordinates $z := [r_1, r_2]$ of the target chosen by $\pi$. This augmented state is used by $u_2$ to resolve the robot’s dynamics and drive the swarm closest to their selected targets. We train $u_2$ by solving a continuous–time optimal control problem with a terminal RMSE loss between the state reached by the robot and the objective set by the policy planner. Here, we integrate the system using the adaptive–step DormandPrince [48] solver with tolerances $10^{-3}, 10^{-3}$.

**Discussion of results** Figure 8 shows the average reward and control loss of the robot swarm during training, with both trends converging after 4000 episodes. Figures 8a and 8b show the generated control of a randomly generated swarm of 100 robots on two new maps. In the first map, the targets consist on alternating patterns of the learned map layouts, generating a straight line pattern which correctly captures the greedy robot policy imposed. The second map consists instead of a new, unseen, map layout within the alternation. The trained model is capable of generalizing the planning and control strategy to account for the new map layout, by redirecting the robots onto their closest resource in a wave-like pattern. On the first tested map, the model achieves 99.8% ± 0.8% average target accuracy for the 100 robot tested batch. On the second tested map, the model achieves 98.5% ± 1.95% average target accuracy for the 100 robot tested batch.

### C Realization of NHAs

#### C.1 Software Implementation of Hybrid Integration

We provide documented Python pseudo–code for the hybrid system adaptive integration algorithm used for dataset generation. This function can handle hybrid systems with multiple modes and transitions. Each possible event requires its own callback function with check_event and jump_map methods. We provide an example of one such callback under odeint_hybrid.

```python
def odeint_hybrid(vf, x, t_span, solver, callbacks, atol, rtol, event_tol):
    # initialize event state tracker, one boolean for each possible event
    # (or edge in the automata representation of the SHS).
    event_states = [False for _ in range(len(callbacks))]
    dt = initial_step_size(f, k1, x, t, solver.order, atol, rtol)

    while t < t_span[-1]:
        # tentative step
        x_step, x_err = solver.step(vf, x, t, dt)

        # check whether any event
```

---

20
# has been triggered in the interval \([t, t + dt]\)

```
new_event_states = [cb.check_event(t + dt, x_step)
                    for cb in callbacks]
```

# has any event state moved from \('False' to \('True'\) in \([t, t + dt]\)?

```
triggered_events = sum([(zp != z) & (z == False)
                        for zp, z in zip(new_event_states, event_states)])
```

# if an event / mode transition has been triggered,
# find exact event time and state

```
if triggered_events > 0:
    x, t = root_find_event(max_iters, event_tol)
```

# if there is a conflict and multiple events are triggered,
# takes always the one with smaller ID

```
zp = min([i for i, ev in enumerate(new_event_states)
          if ev == True])
```

```
t = t + dt
# save state and time BEFORE and AFTER jump
sol.append(x)
eval_times.append(t)
```

```
# apply jump func.
x = callbacks[zp].jump_map(t, x)
```

```
sol.append(x)
eval_times.append(t)
```

# when there are no events,
# proceed as usual with adaptive integration
else:

```
error_ratio = compute_error(x_step, x_err, atol, rtol)
accept_step = error_ratio <= 1
```

```
if accept_step:
    t = t + dt
    sol.append(x)
eval_times.append(t)
```

```
else:
    dt = adapt_step(dt, error_ratio, safety,
                    min_factor, max_factor, order)
```

```
return eval_times, sol
```
# sample one `s` for each batch in z0, to identify events
# as described in Section 2. Exponential, instead of Uniform
# is used to avoid `log` computations.
# Should be sampled again after every event is triggered.
self.s = self.exponential.sample(x0.shape[1])

def check_event(self, t, x):
    raise NotImplementedError

def jump_map(self, t, x):
    raise NotImplementedError