# Development and Interpretation of a Deep Learning Algorithm for Multi-State Model Analysis

Najwan Wafeek Majeed
*Diyala Education Directorate, Ministry of Education, 32001 Diyala, Iraq*
*Najwan1980najwan@gmail.com*

Keywords: Deep Learning, Structured Hidden Variable Models, Multi-State Models, Nonlinear System Identification, Generative Models.

Abstract: This research presents the development of a deep learning algorithm grounded in hidden variable models for the analysis of multi-state systems. Accurately estimating the joint probability distribution of interrelated variables remains a significant challenge, particularly in scenarios where the state data is either unavailable or only weakly linked to the observed correlation data. Hidden variable models address this problem by introducing latent or unobserved factors that can effectively capture and explain the variations in the observed data. System identification, which focuses on developing mathematical models of dynamic systems based on observed input-output data, shows a natural synergy with machine learning methodologies. By leveraging this connection, we propose structural simplifications to hidden variable models that aim to improve their capacity to represent critical state variables and enhance their performance in system identification applications. These simplifications make the models more interpretable and computationally tractable, while still preserving the essential dynamics of multi-state systems, thereby improving the reliability and practicality of their deployment.

# 1 INTRODUCTION

Estimating joint probability distributions of interdependent features is a fundamental challenge in machine learning. Hidden variable models address this by introducing latent variables z with prior distribution p(z), where observed data x is generated from p(x|z) [1] The joint distribution is given by:

$$p(x,z) = p(z)p(x|z). \quad (1)$$

The marginal distribution $p(x) = \int p(x/z)p(z)dz$ is typically intractable. We therefore employ variational inference, introducing an approximate posterior $q_{\emptyset}(z\backslash x)$ to derive the Evidence Lower Bound (ELBO) [2]:

$$log\ p_{\theta}(x) = log \int p_{\theta}(z)p_{\theta}(x/z). \quad (2)$$

This bound decomposes into:

1. Reconstruction term
$$= Eq_{\emptyset}(\ log\ p_{\theta}(x\backslash z)\ ). \quad (3)$$
2. KL divergence:
$$- Dkl\ q_{\emptyset}(z\backslash x)\ \backslash p_{\theta}(z)) = F(\theta, \Phi).$$

The parameter estimation process is thus formalized as [3]:

$$\theta^*, \emptyset^* = arg\ max_{\theta,\emptyset}\ F(\theta, \Phi). \quad (4)$$

- E-step:
$$\Phi^{K+1} = arg\ max_{\Phi}\ F(\theta^k, \Phi).$$
- M-step:
$$\theta^{K+1} = arg\ max_{\theta}\ F(\theta, \Phi^{k+1}). \quad (5)$$

This framework forms the basis of Variational Autoencoders (VAEs), where neural networks parameterize both the encoder $q_{\emptyset}$(z | x) and decoder $p_{\theta}$(x | z).

# 2 METHODOLOGY

## 2.1 Variational Autoencoder with Adaptive Structure

A variational encoder is a latent variable model that uses neural networks to approximate conditional distributions. The generative distribution takes the form:

$$p(z)p(\ x \mid z\ ).$$

The prior p(z), is unrestricted, but its choice critically impacts the model's effectiveness. While any p(z|x) is theoretically valid, the optimal distribution enables the decoder p(x|z) to generate accurate samples efficiently. In VAEs, the latent space is modeled as a standard normal distribution because any complex distribution can be approximated by transforming normal samples. This ensures accurate generation, as the decoder (a neural network) can universally approximate p(x|z) The core training [4] challenge is sampling z such that p(x|z) assigns high likelihood to observed data. To achieve this, an encoder network maps input x to a latent distribution z that maximizes the probability of reconstructing x. The general diagram of this generative and recognitive model is shown in Figure 1.

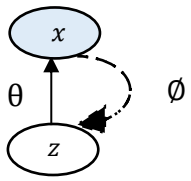


Figure 1: The generative model of the changeable self-encoder (17) - the bold lines of the generative model, the dashed lines show the inference model or the recognition network.

In variational autoencoders, assuming a normal latent distribution is a key limitation. This restricts the model's ability to capture multi modal distributions and prevents the encoder from accurately approximating the true posterior. Despite this limitation, the variational autoencoder (VAE) still performs well in a wide range of applications. The training and sample generation process in this network is governed by the following relations (6) [5]:

$$q_{\emptyset}(z\backslash x) \sim N(\mu_{en},\ \sigma_{en}^2)\begin{cases} \mu_{en} = NN_{\mu_{en}}(x) \\ log\ \sigma_{en}^2 = NN_{\sigma_{en}}(x) \end{cases}$$
$$p_{\theta}(x\backslash z) \sim N(\mu_{de}, \sigma_{de}^2)\begin{cases} \mu_{de} = NN_{\mu_{de}}(z) \\ log\ \sigma_{de}^2 = NN\sigma_{de}\,(z) \end{cases} \quad (6)$$

Each latent sample z is generated using the parameters $\mu_{en}$, $\sigma_{en}^2$, following a standard normal distribution. This is enabled by the reparameterization trick, which addresses the issue that sampling is inherently non-differentiable and would otherwise prevent gradient-based training. By reparameterizing the latent variables, the model remains fully trainable using gradient descent. The model parameters are optimized by minimizing the Evidence Lower Bound (ELBO), where both expectation and maximization steps are performed jointly. This joint update creates a gap between the ELBO and the true log-likelihood, as decoder parameters affect reconstruction loss and encoder parameters shape the latent space. To reduce this gap, a weighting scheme for encoder samples was introduced, leading to the weighted autoencoder. A key limitation of the standard VAE is its assumption of Gaussian distributions for both encoder and decoder, chosen for simplicity sampling is easy, and the KL divergence between Gaussians has a closed form expression. While this facilitates training, it limits the flexibility of the latent space. Modifying the encoder distribution complicates decoder training, and no general framework exists for arbitrary prior and posterior distributions. To address this, the Adversarial Autoencoder (AAE) was proposed, treating the VAE as a special case and using adversarial training to relax the Gaussian assumptions imposed on the latent space [5].

This enables modeling more complex distributions and provides solutions for memoryless or static data types lacking temporal structure.However, in the context of identifying nonlinear dynamic systems, it is essential to generalize these models to handle sequential and time-dependent data. Developing such extensions is critical for adapting VAEs to dynamic system identification tasks [6].

## 2.2 Ordinal Latent Variable Models for Time-Dependent Data

Most of the information in our environment is dynamic and continuously changing. Mathematically, this type of information is represented as a sequence. A sequence is a vector with a physical dimension, which in most systems is modeled as time hence the term time series. Unlike ordinary (static) signals, time series contain two types of information:

1) The numerical values of the data;
2) The temporal structure or timing of the values.

Because of this dual nature, processing time series data is more complex than processing static data. correlation analysis or mutual information may be required to determine an appropriate value for mmm From a machine learning perspective, analyzing time series involves estimating the joint probability distribution $p(y_{1:T})$ across all time steps. In general, directly modeling this joint distribution is infeasible due to the high correlation between time steps. To make the problem tractable, we rely on conditional independence assumptions and temporal causality. For example, even if we assume that each variable xt is binary, modeling the joint distribution over a sequence of length T would require $2^{T} - 1$ free parameters clearly impractical. Using the chain rule

of probability, we can decompose the joint distribution over time as follows (7):

$$p(y_{1:T}) = \Pi_{t=1}^{T} p(y_t \backslash y_{1:t-1}). \quad (7)$$

This factorization is causally consistent, since each time step depends only on past values. However, this formulation still presents challenges, as the condition set $y_{1:t-1}$ grows over time and can vary in size. For instance, if we aim to model this distribution using a neural network, we would need a network whose input size dynamically increases with time—which is impractical. Thus, we must simplify and impose constraints on the temporal structure of the sequence. One common simplification is to limit the dependency window. Rather than allowing each time step to depend on the entire history, we assume it depends only on a fixed number mmm of past steps. This leads to the order-mmm Markov model, represented as (8):

$$p(y_{1:T}) = \Pi_{t=1}^{T} p(y_t \backslash y_{t-m:t-1}). \quad (8)$$

This model assumes that only a fixed-length history is relevant for predicting future values. Examples of such models are illustrated in Figure 2. While this approach simplifies modeling, it introduces the challenge of determining the memory length mmm. For practical applications, this model is often used when the underlying process has short-term dependencies. For systems with long-term memory, additional techniques such as Auto-regressive (AR) models, commonly used in time series analysis, are a special case of the Markov model. However, there is a more general and comprehensive framework that can encompass all of these modeling strategies: the state space model (SSM). State space models can be viewed as a time-unfolded version of hidden variable models. According to the hidden variable perspective, state space models assume that at each time step, there is a latent (hidden) mechanism that drives the observed data. This hidden state must evolve over time, maintaining temporal dependencies so that the model can learn and capture the dynamics of the system. In the following sections, we will explore state space models in greater detail [8]. Many real-world phenomena are dynamic and are best represented as time series sequences where temporal order carries critical meaning. These sequences can be modeled as Figure 2.

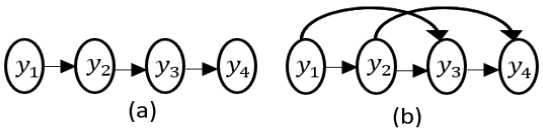


Figure 2: The m-order Markov model.

This simplification enables tractable training, but introduces the challenge of selecting an appropriate memory length mmm. Common solutions include: Auto-Regressive (AR) Models: A specific case of the Markov assumption, where past values linearly predict the current step. Recurrent Neural Networks (RNNs): Address variable-length dependencies through recurrent structures, although they struggle with long-range dependencies. Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU): Mitigate vanishing gradients, enabling learning over longer sequences. Transformers: Recently, attention-based models like Transformers have outperformed RNN-based methods on sequence tasks by allowing direct access to all previous time steps.

## 2.3 Research Sample

Assume we are given a sequence of observations $X_{1:T}$ that depends on a sequence of inputs $u_{1:T}$. In this case, the objective is to model the conditional probability distribution:

$$p_\theta(X_{1:T} \backslash u_{1:T}).$$

In the previous section, we assumed an auto-regressive setting, where the only available information was the sequence $X_{1:T}$ itself. The formulation above is a more general version of the time series modeling problem. If we replace the input ut with past values of $X_t$ we recover an auto-regressive model. In other words, auto-regression is a special case of this formulation when there is no external input or stimulus. To model this distribution, following the principles of hidden variable models, we introduce a latent variable $z_t$ which depends on either the input ut (in general models) or the past output (in auto-regressive models), and also on the previous latent state $z_{t-1}$ to capture temporal dynamics. In the context of state space models (SSMs), this latent variable is referred to as the state variable. Given this setup, we express the conditional distribution as follows:

$$p_\theta(X_{1:T} \backslash u_{1:T}) = \int p_\theta(X_{1:T}\, z_{1:T} \mid u_{1:T}) dz_{1:T}. \quad (9)$$

Assuming that the state variables are continuous (in the discrete case, the integral becomes a summation), the joint distribution can be factorized as:

$$p_\theta(X_{1:T}\, z_{1:T} \mid u_{1:T}) = p_\theta(z_1) \Pi_{t=1}^{T} p_\theta(X_T / z_T) \Pi_{t=1}^{T} p_\theta(z_T \backslash z_{T-1}, u_T). \quad (10)$$

- $p_\theta(x_t \mid z_t)$ is the observation model;
- $p_\theta(z_t \backslash z_{t-1}, u_t)$ is the state transition model.

In some formulations, the observation model may also condition on the input: $p_\theta(\mathrm{x_t} \mid \mathrm{z_t}, u_t)$. A graphical model of such a state space system is shown in Figure 3.

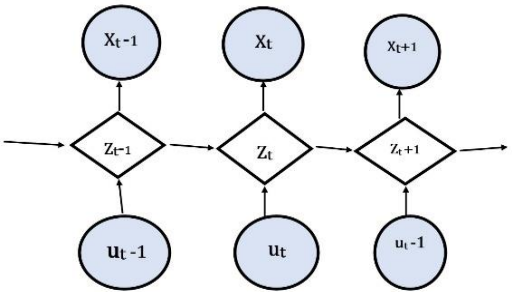


Figure 3: The graphical model of a state space system.

Depending on how the state transition and observation models are defined, various state space model variants can be constructed. The primary goal in all cases is to infer the conditional distribution of the output sequence given the inputs, which is achieved by marginalizing out the latent states. This requires estimating:

$$p_\theta(z_{1:T} \mid x_{1:T}, u_{1:T}).$$

In most real-world cases, exact inference of this posterior is intractable, and only under certain conditions does an exact solution exist. For example:

- If both the observation and transition models are linear,
- If both distributions are Gaussian.

Then the problem admits a closed-form solution known as the Kalman Filter. If the relationships are nonlinear, an approximate version of the Kalman Filter (such as the Extended Kalman Filter or Unscented Kalman Filter) can be used. For most practical applications, especially when models are nonlinear or complex, approximate inference techniques such as variational inference or sampling-based methods (e.g., Monte Carlo methods) are required [9]. The architecture, structurally reminiscent of the state space model, is a recurrent neural network (RNN), shown in Figure 4.

$$d_t = f_\theta\,(d_{T-1}, u_{T:\theta}).$$

A model that structurally resembles a state space model. The left panel shows an RNN with an input sequence, while the right panel shows the case with no external input. Both configurations are fundamentally similar in that they encode hidden states over time using a deterministic update function:

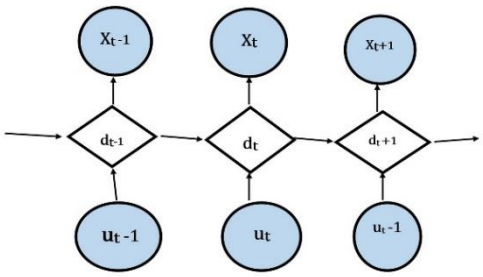


Figure 4: Presents a Recurrent Neural Network (RNN).

$$p_\theta(d_T \backslash d_{t-1}, u_t) = \delta\big(d_T - f_\theta\,(d_{t-1}, u_t)\big).$$

There are many variants of RNNs, each developed to enhance the network's ability to retain memory and capture long-term dependencies. Despite differences in architecture (e.g., LSTM, GRU), they share the common goal of modeling state transitions via deterministic mappings. It's important to note that the assumption of knowing the exact model (i.e., the form of $f_\theta$ is often unrealistic in real applications, where the relationship between variables is unknown. However, the flexibility of neural networks allows both transition and observation models to be approximated, making them highly effective. This has led to the emergence of Deep State Space Models, which use deep neural networks to represent both the hidden dynamics and observation mappings. While these models lack closed-form solutions, approximate training and inference methods (e.g., variational inference, ELBO maximization) are employed.

In this research, Recurrent Neural Networks are explored and utilized as a flexible and powerful instance of state space models due to their effectiveness in modeling sequential data and hidden state dynamics [10]. Recurrent Neural Networks (RNNs) are a special class of neural network architectures capable of modeling variable-length sequences through deterministic mappings. Figure 5 illustrates the structure of an RNN in two configurations: one without input and one with input. In the second configuration (the right panel), the external stimulus input is included. If we replace the input with the previous output (i.e., auto-regressive feedback), the structure becomes self-contained.

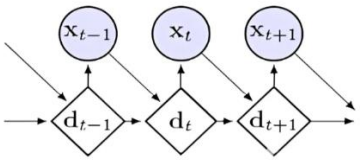


Figure 5: The structure of an RNN in two configurations.

Architectures such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) belong to this category of RNNs. An RNN can be viewed as a special case of a state space model, where a Dirac delta function represents the probability distribution of the system. This means the observation model becomes:

$$p_\theta(d_T \backslash d_{t-1}, u_t) = \delta\big(d_T - f_\theta\,(d_{t-1}, u_t)\big).$$

From the above structure, the deterministic nature of typical RNNs. However, in specific applications such as text generation, the output distribution can be modeled probabilistically, rather than deterministically.

- Stochastic Recurrent Neural Networks. To introduce stochasticity into the modeling of sequences especially when representing uncertainty and capturing latent variabilitya Stochastic Recurrent Neural Network (SRNN) is used. In such architectures, a connection is established between the random variables across time to ensure consistency in variance propagation. There are two types of relationships.
- Indirect connection. The sampled random variable influences the deterministic variable in the next time step.
- Direct connection. The sampled random variable directly influences the next random variable.

$$p_\theta(X_{1:T}\, z_{1:T}\,, d_{1:T}|u_{1:T}) = \Pi_{t=1}^{T}\, p_\theta(x_t|z_T, d_T) p_\theta(z_t|z_{t-1}, d_t)$$
$$p_\theta(d_T|d_{t-1}, u_t). \qquad (11)$$

The deterministic update step is:

$$p_\theta(d_T \backslash d_{t-1}, u_t) = \delta(d_T - d_t\,)$$
$$p_\theta(d_T \backslash d_{t-1}, u_t) = \delta\big(d_T - f_\theta(d_{t-1}, u_t)\big).$$

Using the sifting property of the Dirac delta function, the joint distribution simplifies to:

$$p_\theta(X_{1:T}, z_{1:T}|\tilde{d}_{1:T}) = \Pi_{t=1}^{T}\, p_\theta(z_t|z_{T-1}, \tilde{d}_t) p_\theta(x_t|z_t, \tilde{d}_t). \quad (12)$$

To train this model, as with other latent variable models, we use the variational lower bound. The evidence lower bound (ELBO) for the log-likelihood of the observations is given by:

$$log\ p_\theta(X_{1:T}\,|u_{1:T}\,, d_0) \geq E_q(log\, p_\theta(X_{1:T}, z_{1:T}\,|u_{1:T}, d_0) - log\, q_\phi(z_{1:T}|X_{1:T}, u_{1:T}, d_0) = F(\theta, \emptyset) \qquad (13)$$

To compute this bound and train the model, we must choose a suitable variational distribution (i.e., inference model). As in previous sections, we adopt a delayed probability distribution for the inference network. An important note about this architecture is that, due to the temporal dependencies between random variables, generating valid random samples may require access to future time steps—introducing computational and design challenges.

# 3 DISCUSSION

In a sequence generator model, it is essential to statically incorporate uncertainty into the modeling process. More precisely, the random mapping layer must also depend on previous time steps to ensure that a certain level of stochasticity is included in the model. This concept aligns with the state space model, with the difference that the structure of the recurrent autoencoder neural network has been modified. In other words, the hidden layer now follows a structured form, obeying a first-order Markov process, as illustrated in Figure 7. Compared to the model in Figure 6, the recurrent neural network architecture is enhanced by introducing links between hidden variable, enabling stochastic information flow across time. This brings the model closer to state - space representations influenced by Gaussian noise. As a result, state transitions become nonlinear, increasing training complexity in stochastic RNNs due to the need for inference over future states.

## 3.1 Simulation

In this section, we compare the simulation results of the original model structures and the newly modified architecture. To do so, we use three benchmark systems, described below.

## 3.2 Linear Gaussian System

The first system used for simulation is a Linear Gaussian system, described by (14):

$$x_{k+1} = [0.7\ \ 0.8\ \ 0\ \ \ 1\ ]\, X_k + [-1\ 01]\, u_\kappa + W_k$$
$$y_k = [1\ 0] x_k + v_k. \qquad (14)$$

Where:

- $W_k$: process noise;
- $v_k$: measurement noise.

Both $W_k$ and $v_k$ follow zero-mean Gaussian distributions. The variance of the measurement noise is 1.0, while that of the process noise is 0.5.For training and validation, the input is uniformly sampled from the range . Both datasets are generated using 50 independent runs, and the results are averaged.mFor testing, the input is a composite sinusoidal signal:

$$u_k = sin(2k\pi 10) + sin(2k\pi 5),$$
$$u_k = sin(102k\pi) + sin(52k\pi),$$

The simulation includes:

- 6,000 samples for training;
- 2,000 samples for validation and testing.

## 3.3 Narendra-Li Benchmark System

This system, originally proposed by Narendra, is a hypothetical nonlinear system with no real-world physical implementation. It has been widely used as a benchmark for nonlinear system identification.
The system's dynamics and outputs are defined by (13) and (14). In this context, represents the measurement noise. Unlike the original benchmark, which does not include noise, the Giri-modified version introduces Gaussian noise to increase the task complexity: ϵx~N(0,0.5)

## 3.4 Third Benchmark System

The third benchmark system is the Wiener–Hammerstein process, selected to demonstrate the effectiveness of the introduced structures. This system includes process noise, and the data was collected by researchers at the University of Eindhoven. it is available on the nonlinear benchmark systems website.

# 4 RESULTS

This section presents the results related to the identification (modeling) of the benchmark systems introduced earlier. All simulations were performed using the Monte Carlo method. Specifically, for each input, 100 samples were generated from the encoder's output. The decoder output is calculated as the average of these 100 samples, representing the mean mapping across simulations. Table 1 also shows the proposed NN comparison table (the proposed NN comparison results), which presents the RMS and NLL metrics for the studied structures.

## 4.1 Performance of the Model for the Gaussian Linear System

For this simulation, the hidden layer of the recurrent neural network has a dimension of 80, and the random layer has a dimension of 10. A regression layer is also used for modeling. The input sequence length is set to 60. The effective error value and the negative log-likelihood were calculated as 104 and 0.99, respectively, for the structured neural network variable encoder. For the structured recurrent random network, these values are 0.93 and 0.94.

## 4.2 Narendra-Li Benchmark System Results

Figure 8 presents the output of the proposed models for the Narendra–Li benchmark system, comparing the Structured VAE-RNN system and the Structured STORN model.In this case, the model architecture is identical to the previous one, except that the input sequence length is 50. The effective error value and the maximum similarity function for the structured variable autoencoder–structured recurrent neural network were 0.92 and 0.76, respectively. For the structured stochastic recurrent neural network, the values were 1.01 and 0.96. In other words, for this benchmark, the modified structured recurrent neural network with the variable autoencoder performed better.

## 4.3 Wiener-Hammerstein Benchmark System Results

Figure 9 illustrates the output of the proposed models for the Wiener–Hammerstein benchmark system. The structure used here is the same as in the previous simulations, with the difference that the input sequence length is 150.

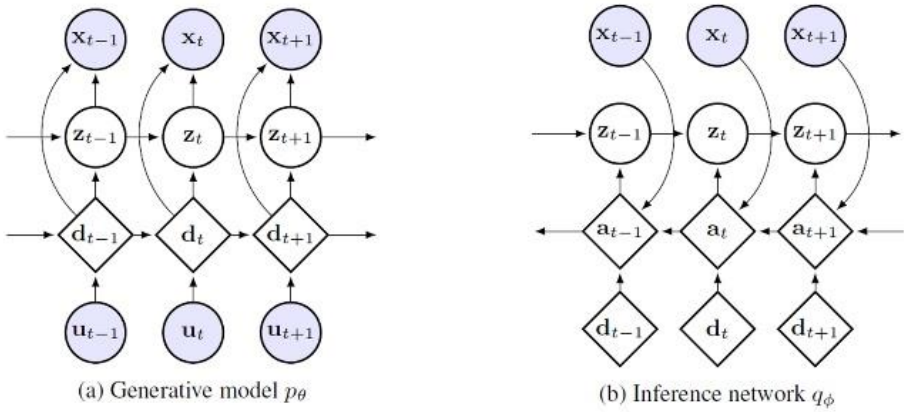


Figure 6: The architecture of a generative SRNN.

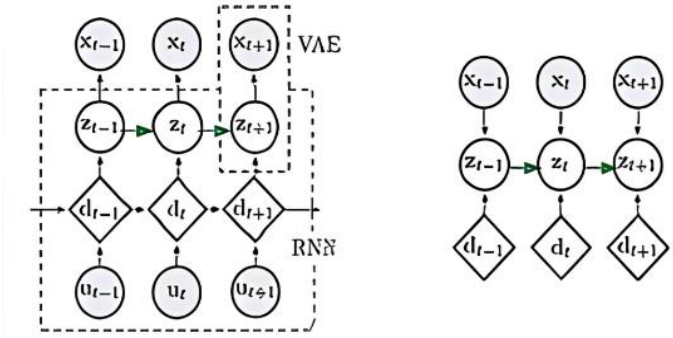


Figure 7: Modified recurrent neural network model (Autoencoder structure).

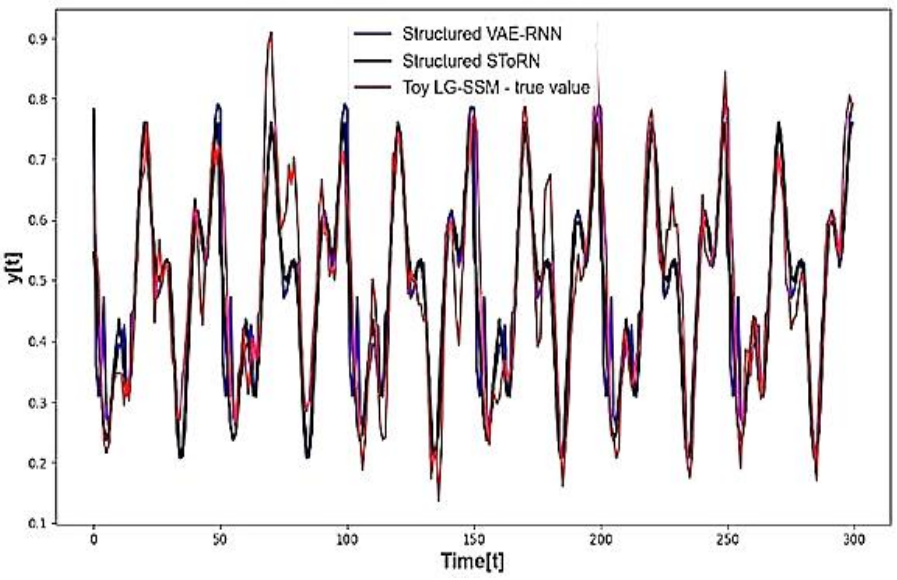


Figure 8: Presents the output of the proposed models for the Narendra–Li benchmark system.

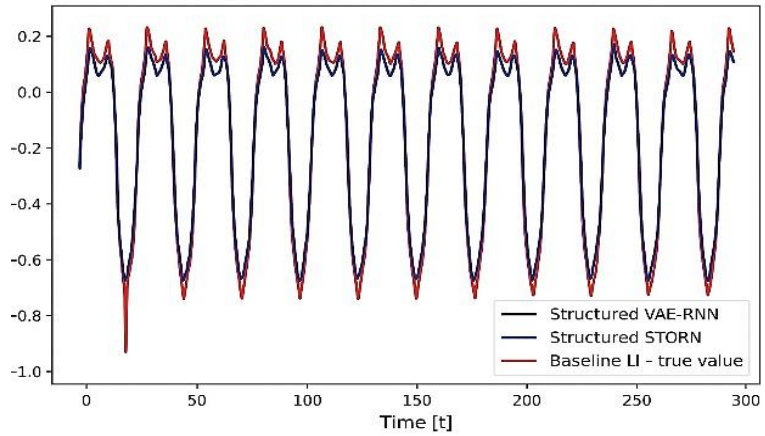


Figure 9: The output of the proposed models for the Wiener Hammerstein benchmark system.

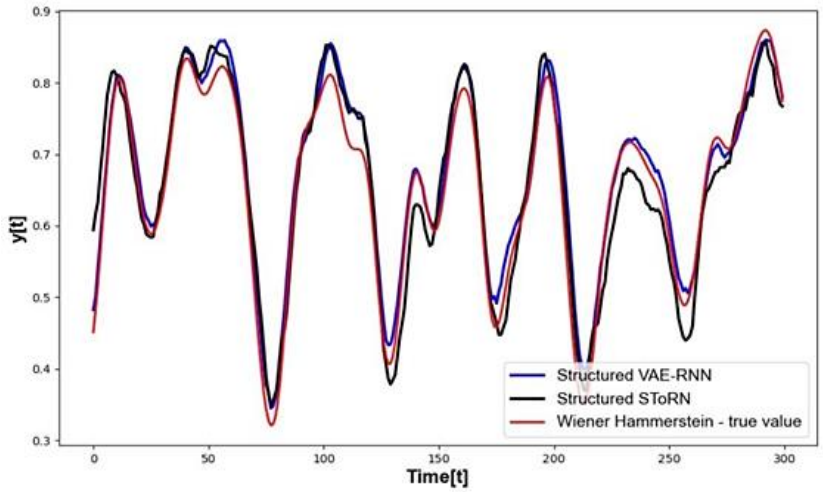


Figure 10: The performance of the Structured VAE-RNN and Structured STORN systems.

Table 1: RMS and NLL Metrics for the Studied Neural Network Structures.

| Model | RMS Error - Toy-LGSSM | RMS Error - Narendra-Li | RMS Error - Wiener-Hammerstein | NLL - Toy-LGSSM | NLL - Narendra-Li | NLL - Wiener-Hammerstein |
|---|---|---|---|---|---|---|
| VRNN | 1.48 | 0.89 | 0.86 | 1.82 | 1.31 | 1.33 |
| VAERNN | 1.56 | 0.84 | 0.74 | 1.95 | 1.34 | 1.29 |
| STORN | 1.43 | 0.64 | 0.74 | 1.79 | 1.20 | 1.01 |
| S-VAE-RNN | 1.08 (↓26% vs VAE-RNN) | 0.76 (↓9.5% vs VAE-RNN) | 0.63 (↓15% vs VAE-RNN) | 0.93 (↓52% vs VAE-RNN) | 0.91 (↓32% vs VAE-RNN) | 0.91 (↓29% vs VAE-RNN) |
| S-STORN | 0.94 (↓34% vs STORN) | 1.01 (↑58% vs STORN) | 0.74 (No Δ vs STORN) | 0.92 (↓49% vs STORN) | 0.92 (↓23% vs STORN) | 0.92 (↓9% vs STORN) |

For the structured variable autoencoder–structured recurrent neural network, the effective error value and maximum similarity function were 63 and 0.91, respectively. For the structured stochastic recurrent neural network, these values were 74 and 0.92. Figure 10 shows the performance of the Structured VAE-RNN and Structured STORN systems on the Wiener–Hammerstein benchmark.

# 5 CONCLUSIONS

Latent variable models, adapted from machine learning, provide a powerful and flexible framework for nonlinear system identification by learning probabilistic state representations that capture complex system dynamics. With only minimal structural adjustments, these models can match - and in many cases surpass - traditional identification methods in both accuracy and generalization capability. Their ability to represent uncertainty and model nonlinearities makes them particularly well-suited for scenarios where conventional approaches struggle. The results of this study highlight the strong potential of integrating generative modeling techniques into control-oriented applications, enabling more robust, adaptive, and data-efficient system designs. As these models continue to evolve, their incorporation into real-time control and decision-making pipelines could offer significant advantages in fields such as robotics, autonomous systems, and industrial process optimization.

# REFERENCES

[1] K. Hua, D. Wojdyla, A. Carnicelli, C. Granger, X. Wang, and H. Hong, "Network meta-analysis with individual participant-level data of time-to-event outcomes using Cox regression", Statistical Methods in Medical Research, vol. 44, no. 5, p. e70027, Feb. 2025, doi: 10.1002/sim.70027.

[2] J. A. Sparano, M. R. Crager, G. Tang, R. J. Gray, S. M. Stemmer, and S. Shak, “Development and validation of a tool integrating the 21-gene recurrence score and clinical-pathological features to individualize prognosis and prediction of chemotherapy benefit in early breast cancer,” J. Clin. Oncol., vol. 39, no. 6, pp. 557–564, 2021.

[3] M. J. Ramezankhani, M. Blaha, M. Mirbolouk, F. Azizi, and F. Hadaegh, “Multi-state analysis of hypertension and mortality: Application of semi-Markov model in a longitudinal cohort study,” BMC Cardiovasc. Disord., vol. 20, no. 1, pp. 1–13, 2020.

[4] J. D. Kalbfleisch and R. L. Prentice, The Statistical Analysis of Failure Time Data, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, 2020.

[5] P. K. Andersen, Ø. Borgan, R. D. Gill, and N. Keiding, Statistical Models Based on Counting Processes. Berlin, Germany: Springer Science & Business Media, 2023.

[6] L. Meira-Machado and M. Sestelo, “Estimation in the progressive illness-death model: A nonexhaustive review,” Biometrical J., vol. 61, no. 2, pp. 245–263, 2019.

[7] Z. Yan, J. Liu, L. T. Yang, and N. Chawla, “Big data fusion in Internet of Things,” Information Fusion, vol. 100, pp. 32–33, 2022.

[8] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, “Efficient GAN-based anomaly detection,” Inst. for Infocomm Res., Singapore, School of Computer Science, Nanyang Technol. Univ., 2018.

[9] M. Ahmed, A. N. Mahmood, and J. Hu, “A survey of network anomaly detection techniques,” J. Netw. Comput. Appl., vol. 60, pp. 19–31, 2020.

[10] R. Daneshfaraz, E. Aminvash, A. Ghaderi, J. Abraham, and M. Bagherzadeh, “SVM performance for predicting the effect of horizontal screen diameters on the hydraulic parameters of a vertical drop,” Appl. Sci., vol. 11, p. 4238, 2021.