Stochastic Differential Equation Networks for Time Series at Edge

Yimin Dai*
Interdisciplinary Graduate School
Nanyang Technological University
Singapore

Li-Lian Wang
School of Physical and Mathematical Sciences
Nanyang Technological University
Singapore

Rui Tan
College of Computing and Data Science
Nanyang Technological University
Singapore

ABSTRACT

Stochastic differential equation networks (SDENets), a subset of continuous-time neural networks, offer the natural ability to model continuous time-series data with greater expressivity than discrete-time neural networks. However, SDENets face challenges related to stability and high computational overhead. In this paper, we introduce SE-SDENet, a stable and efficient variant of SDENet. Leveraging the inherent capability of SDENets to model randomness in time-series data, we establish a theoretical framework that ensures SE-SDENet's stability during training by regulating the dynamics of each neuron. Additionally, we propose an efficient training and inference framework that enables SE-SDENet to achieve low forward-pass complexity and to dynamically adjust its complexity at run-time. Evaluation with four datasets and four edge devices demonstrates that SE-SDENet achieves a 6.6x higher throughput than the solver-based SDENet and exhibits improved stability in handling noisy data and long-term predictions. A validation on a robot vehicle shows that SE-SDENet can dynamically adjust its complexity at run-time to meet varying resource constraints.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; • **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

Continuous-time neural network, time series processing, edge computing

1 INTRODUCTION

Time-series data processing plays a pivotal role in enabling the functionality of Internet-of-Things (IoT) devices. IoT systems continuously collect data over time from various sensors and embedded devices, generating streams of sequential data that reflect dynamic changes in the physical environment. The ability to accurately process time-series data is essential for IoT devices to make fast, context-aware decisions in diverse domains, such as processing inertial data for activity recognition [1], energy consumption monitoring [2], and healthcare from photoplethysmogram [19].

However, existing feedforward networks, such as multilayer perceptrons (MLPs), do not explicitly model the time dimension, hindering their ability to effectively capture the relationships between data points at different time steps. Recurrent neural networks (RNNs) use feedback to develop time-series data processing capabilities. Transformers [29] use attention mechanisms to model the interactions between data points collected at different times. However, as both RNNs and transformers assume discrete time and often uniform steps with a fixed time interval, they require heuristic data preprocessing (e.g., up/down-sampling and interpolation) to address the deviations from the assumption, such as varied sampling rate and irregular sampling. Overall, they both belong to the class of discrete-time neural networks (DT-NNs).

Continuous-time neural networks (CT-NNs) [32], which incorporate neuron-level dynamics, have been introduced to model time-series data. In CT-NNs, the behavior of each neuron is governed by a function with continuous time as an independent and explicit variable. A prominent subset of CT-NNs uses continuous-time differential equations (DEs) to model nonlinear mappings. As the differential equation contains learnable parameters, the neuron can adjust its internal dynamics during the training. Among these, the neural ordinary differential equation networks (ODENets) [6] have attracted significant attention due to their flexibility and smoothness in learning dynamics. ODENets, as part of the broader class of CT-NNs, offer two distinct advantages.

First, due to their continuous-time formulation, ODENets are capable of handling irregularly sampled time-series data without the assumption of fixed sampling intervals, making them highly adaptable to IoT scenarios where data collection intervals can be inconsistent. Second, ODENets exhibit enhanced expressivity compared with DT-NNs like RNNs [12], as ODENets can model more complex temporal relationships through the interaction of neuron-level dynamics and feedback mechanisms at the network level.

To meet the latency requirements, data collected by IoT sensors is processed directly on nearby edge devices. However, in practical deployments on edge devices, CT-NNs face two significant limitations: *computation complexity* and *computation stability*. CT-NNs' expressivity is bottlenecked by the performance of numerical differential equation solvers. Specifically, since most differential equations have no closed-form solutions, the use of a numerical solver incurs computational overhead proportional to the product of the prediction steps and the solver's step size, resulting in a quadratic complexity term. On the other hand, edge devices need to process data from multiple sources, often requiring several models to run in parallel. Fluctuations in the number of concurrently running models lead to variations in the computational resources available to each model. This requires the model to have the ability to dynamically adjust its complexity. Although a solver can modify complexity by adjusting the step size, this approach compromises accuracy. In addition, CT-NNs suffer from the instability problem. Specifically, a minor perturbation in the input will be amplified over time. These perturbations, small but widely present due to environmental noises, sensor drifts, etc, can significantly affect the

*Also with College of Computing and Data Science, Nanyang Technological University, Singapore.

long-term behavior of the system due to the accumulative effect of errors.

In this paper, based on two characteristics of time-series data collected from sensors, we address the above two challenges to facilitate the deployment of CT-NNs on resource-constrained edge devices to fully utilize CT-NNs' expressivity. To enhance stability, we replace the ODE with a stochastic differential equation (SDE), creating a model referred to as *SDENet* [18]. SDENet assumes that time-series data follow a stochastic process, allowing it to incorporate random fluctuations through stochastic terms in the differential equation. This property makes SDENets particularly suited to IoT scenarios, where sensor data is inherently noisy or influenced by external factors. Leveraging this inherent capability in modeling randomness, we establish a theoretical framework that ensures SDENet's stability during training by controlling the regularity of each neuron's dynamics. Our design ensures stability during the training phase, rather than simply verifying it after training, thereby preventing unstable dynamics from arising in the model design phase.

Furthermore, we design an efficient, solver-free SDENet training and inference framework based on the stochastic collocation method (SCM). The core idea is to approximate the stochastic process by interpolating over process values on a set of preselected points instead of discretizing the time horizon as in numerical solvers. Therefore, SCM can predict the stochastic process at arbitrary timestamps with constant time complexity. Moreover, we design a complexity-aware training framework, enabling SDENet to adjust its model complexity according to run-time demands. The complexity-aware training framework treats the complexity requirement as an input and aims at minimizing the accuracy loss by choosing the optimal parameters of SDENet for the required complexity level.

We name the proposed model as *stable and efficient SDENet*, in short, *SE-SDENet.* The main contributions of this paper are summarized as follows:

- We establish a theoretical framework that guarantees the stability of SE-SDENet. Evaluation demonstrates that, compared with an ODENet, LTC [12], SE-SDENet is less sensitive to noise and exhibits better stability, particularly with a large number of prediction steps.
- We design a complexity-aware and computationally lightweight training and inference framework for SE-SDENet. Evaluation demonstrates that SE-SDENet can be deployed on low-end devices with down to 256 KB memory. Tests on four edge devices across four datasets indicate that SE-SDENet achieves up to 6.6x higher throughput than SDENet which uses a numerical solver.
- We implement SE-SDENet on a robot vehicle to process real sensing data traces. The evaluation shows that SE-SDENet can dynamically adjust its complexity to meet varying resource constraints while maintaining high accuracy.

Paper organization: §2 presents background, related work, and the assumptions in the design. §3 presents the theoretical framework that guarantees the stability of SE-SDENet. §4 introduces the design of the lightweight and adaptive training and inference framework. §5 and §6 present evaluation and case-study experiments. §7 discusses the limitations. §8 concludes this paper.

2 BACKGROUND

2.1 Related Work

2.1.1 Resource-efficient CT-NNs. The design of resource-efficient CT-NNs is essential to deploying the CT-NNs on edge devices. The study [6] designs an adjoint-based training framework for back-propagation through time and achieves $O(1)$ memory complexity by compromising backward-pass accuracy. The work [17] proposes a differentiable surrogate function for the solver's time cost and optimizes based on the function to reduce the computation time. The work [24] designs a lightweight neural network to approximate the computation-heavy higher-order term in the solver to reduce the computation complexity. The work [11] derives a closed-form solution of a bio-inspired ODE and achieves time-complexity comparable to DT-NNs. However, all the existing methods for reducing time complexity cannot dynamically adjust computational complexity at run-time, resulting in under- or over-utilization of available computational resources in dynamic edge environments. In this work, we will design SE-SDENet to fill this gap.

2.1.2 Stability of CT-NNs. A subset of CT-NNs [12] is proved to be stable with the dynamic modeled by specific stable ODEs. The work [15] improves the stability of the CT-NNs by introducing Lyapunov-stable equilibrium points in its solution, ensuring that small perturbations in the input will not lead to drastically different outcomes. In our theoretical model, we quantify how small perturbations affect the output of the SDENet over time and derive the sufficient condition to ensure the stability of the SDENet.

2.2 Preliminaries on SDENet

The notation convention in this paper is as follows. Take the letter x as an example. x denotes a scalar; $\mathbf{x}$ denotes a column vector; $\mathcal{X}$ denotes a set; $\mathbf{X}$ denotes a matrix. $\mathbf{x}_{[i]}$ denotes the i^{th} element of $\mathbf{x}$; $\mathcal{X}_i$ denotes the i^{th} item of $\mathcal{X}$; $\mathbf{X}_i$ denotes the i^{th} row of $\mathbf{X}$.

An $\mathbb{R}^d$ process from time $[0,T]$ is denoted as $\mathbf{X}^{d\times T} = \{\mathbf{x}_t\}_{t\in[0,T]}$. The $\mathbf{x}_t$ denotes the $\mathbf{X}$'s value at timestamp t, where $\mathbf{x}_t$ is a vector in $\mathbb{R}^d$. Therefore, $\mathbf{x}_{t_{[i]}}$ is the i^{th} value of $\mathbf{X}$ at timestamp t. Assume $\mathbf{X}^{d\times T}$ is a stochastic process and its initial value $\mathbf{x}_0$ follows the distribution α, we can describe $\mathbf{X}$'s dynamic by its rate of change over time with a stochastic differential equation:

$$\mathrm{d}\mathbf{x}_t = f(\mathbf{x}_t, t)\mathrm{d}t + g(\mathbf{x}_t, t)\mathrm{d}\mathbf{b}_t, \tag{1}$$

$$\mathbf{x_0} \sim \alpha, \tag{2}$$

where $\mathbf{B} = \{\mathbf{b_t}\}_{t\in[0,T]}$ is the Brownian process in $\mathbb{R}^w$. The Brownian process provides a mathematically convenient way to introduce randomness into differential equations. The $f : [0,T]\times\mathbb{R}^d \to \mathbb{R}^d$, $g : [0,T]\times\mathbb{R}^d \to \mathbb{R}^{d\times w}$ are called the drift and diffusion functions. Here, we illustrate the above formulation with an example of gait sensing. Suppose $\mathbf{X}^{3\times T}$ represents the three-dimensional angular velocity of lower limb motion measured by a gyroscope. The drift function f represents the deterministic dynamics of motion, such as the periodic gait cycle observed during the subject's walking.

The diffusion function g accounts for the uncertainty in the inertial measurement unit (IMU) data, including the instability of the gyroscope caused by temperature fluctuations. A key advantage of modeling dynamics with the SDE is that it offers a principled framework for capturing inherent uncertainty of sensing systems. For instance, the effect of thermal noise on a gyroscope can be modeled as additive Gaussian noise, allowing the diffusion function to be defined as a constant, such as $g(\mathbf{x}_t, t) = \gamma$, where $\gamma \in \mathbb{R}^3$ is a constant representing the gyroscope's temperature-induced drift.

SDENet utilizes multiple interconnected neurons, each satisfying an SDE, to model the above stochastic process. SDENet consists of M layers of neurons. The $\Omega = \{\omega_1, \ldots, \omega_{\mathrm{M}}\}$ represents the set of all neurons, where $\omega_{\mathrm{i}[j]}$ denotes the j^{th} neuron in the i^{th} layer. The input to the neuron $\omega_{1[j]}$ in the first layer is $\mathbf{x}_{0_{[j]}}$ and the timestamp t. For the neuron $\mathbf{x}_{i_{[j]}}$ in the subsequent i^{th} layer, its input is the weighted sum of the outputs from the $(i-1)^{th}$ layer and the timestamp t. Each neuron's state $\mu \in \mathbb{R}$ follows the SDE in the form $\mathrm{d}\mu_t = f(\mu_t, t)\mathrm{d}t + g(\mu_t, t)\mathrm{d}\mathbf{b}_t$. The μ_0 is equal to the input of the neuron. The neuron's output is the state at timestamp t, i.e., μ_t. The function f and g are modeled by two neural networks. Given the input μ_0 and t, the output μ_t is calculated by the Itô calculus as $\mu_t = \mu_0 + \int_0^t f(\mu_s, s)\mathrm{d}s + \int_0^t g(\mu_s, s)\mathrm{d}\mathbf{b}_s$ [14]. Since the integrals of most functions do not have closed-form solutions, numerical methods are developed to approximate the calculation above. The basic idea behind numerical solvers is to discretize the interval $[0, t]$ into $\frac{t}{\Delta t}$ small segments, each having a time duration of Δt, where $\Delta t \ll 1$. Different numerical methods are then used to approximate the integral within each segment. Therefore, for each neuron, such a method requires at least $O\left(\frac{t}{\Delta t}\right)$ time complexity. Thus, to obtain the output of SDENet $\mathbf{x}_t$, a complexity of $O\left(\frac{t}{\Delta t} \cdot d \cdot M\right)$ is needed. The time complexity described above increases with t. On the other hand, when the initial value $\mathbf{x}_0$ is subjected to a small perturbation, this perturbation accumulates over time t and is propagated across each layer of the SDENet. This may cause the final output to deviate significantly from the true value, potentially even leading to arithmetic overflow in the system.

2.3 Assumptions

In our design, based on the following two assumptions, we derive the sufficient condition to ensure the stability of each neuron in SDENet and embed SCM in SE-SDENet to approximate the value $\mathbf{x}_t$ with a constant time complexity.

Assumption 1. *$\mathbf{X}$ is a smooth process.*

Assumption 2. *The parameters of the diffusion function are independent of those of the drift function.*

A *smooth process* refers to a time series where changes occur continuously over time without abrupt jumps or discontinuities. Most sensor data follow Assumption 1, such as acceleration readings from an accelerometer, temperature measurements, or pressure sensor data, where changes occur gradually due to physical constraints. Differently, network latency trace is a counterexample, because of the potential abrupt changes. Assumption 1 allows us to assume that the diffusion function g is Lipschitz-continuous, which further enables the derivation of Theorem 1 in this paper. The validity of Assumption 2 depends on whether the noise in the dynamic process is correlated with the process itself. Most sensing data are affected by environmental noises and inherent sensor imperfections or heterogeneity, such as the influence of ambient noises on acoustic data, thermal noises on gyroscopes, and structural differences in microphones on the sound collected. Since these factors are in general independent of the dynamic process itself, Assumption 2 is widely valid. Assumption 2 ensures the correctness of SCM. In §5.2, we will quantitatively analyze the impact of data smoothness on the accuracy of SE-SDENet, as well as the effect of violating Assumption 2 on SE-SDENet's performance.

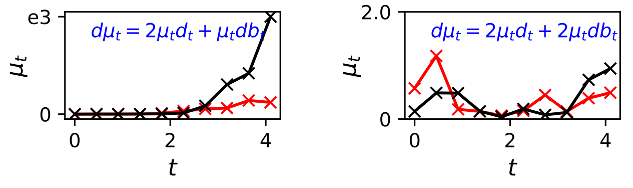


Figure 1: Neuron output differences over time when the inputs are with a difference of 0.2. *Left*: setting $2c_{f_i} - c_{g_i}^2 > 0$ results in diverged outputs. *Right*: $2c_{f_i} - c_{g_i}^2 \leq 0$ results in converged outputs.

3 STABILITY OF SDENET

As discussed in §2.2, the inherent ability of SDENet in modeling the stochastic nature of data does not guarantee stability when the input $\mathbf{x}_0$ is subject to perturbations. This section proposes the sufficient condition for a single neuron to satisfy bounded-input bounded-output (BIBO) stability. We begin with analyzing how variations in a neuron's output evolve over time t when there are differences in its input values. Based on this analysis, we derive a sufficient condition for the stability of a neuron. Note that when every neuron is stable, the SDENet is also stable. Subsequently, we present a practical algorithm for designing and training a stable SDENet that satisfies this sufficient condition.

3.1 The Sufficient Condition for BIBO Stability of SDENet

We begin with defining the BIBO stability of a neuron.

Definition 1 (BIBO Stability). *A neuron is BIBO stable if, for every bounded input $s \in \mathbb{R}$ with $\mathbb{E}(|s|) < \infty$, the state μ_t remains bounded for all $t \geq 0$.*

Based on the Assumption 1, we can assume that the dynamic of each neuron in SDENet is smooth. Specifically, we can assume the drift function and diffusion function in the SDE $\mathrm{d}\mu_t = f(\mu_t, t)\mathrm{d}t + g(\mu_t, t)\mathrm{d}\mathbf{b}_t$ are c_f-Lipschitz and c_g-Lipschitz continuous with respect to μ_t respectively. Suppose z_0, y_0 are two different inputs of the neuron, and z_t ,y_t are the corresponding outputs at timestamp t, then we have the following theorem.

Theorem 1. *For any positive t, the 2-Wasserstein distance $\mathbb{W}_2$ between z_t and y_t has the following upper bound: $\mathbb{W}_2(z_t, y_t) \leq e^{(2c_f - c_g^2)t}\mathbb{W}_2(z_0, y_0) + \Upsilon$. The Υ is a constant that is related to c_f, c_g, and the linear growth bounds of the drift and diffusion function.*

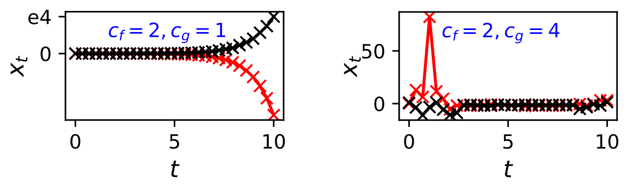


Figure 2: Verify the algorithm for learning a stable SDENet. *Left*: diverged output with $c_{f_i} = 2, c_{g_i} = 1$. *Right*: converged output with $c_{f_i} = 2, c_{g_i} = 4$.

The proof of the theorem in the general case, where $\mathbf{z_t}, \mathbf{y_t} \in \mathbb{R}^d$ is provided in the appendix. The theorem indicates that if $2c_f \le c_g^2$, the difference of the two outputs will not grow unboundly over time. Fig. 1 plots the evolution of z_t and y_t when the initial values are set as $z_0 - y_0 = 0.2$, under the conditions $2c_f - c_g^2 > 0$ and $2c_f - c_g^2 < 0$, respectively. As shown, when $2c_f - c_g^2 < 0$, z_t and y_t converge to approximately the same value over time. If the diffusion function g is set to a constant 0, the SDENet reduces to an ODENet. Informally, we have $\mathbb{E}[(z_t - y_t)^2] \le e^{2c_f t}\mathbb{E}[(z_0 - y_0)^2] + C$, where C is a constant. This inequality suggests that small differences in initial conditions could potentially lead to exponentially growing differences in solutions of an ODE over time. However, when the dynamics of each neuron satisfy an SDE, we can achieve BIBO stability for the neuron's dynamic by adjusting the smoothness of the drift and diffusion functions. Theorem 2 can be directly derived from Theorem 1, as a bounded difference between z_t and y_t implies both values are bounded.

Theorem 2. *Given a neuron, whose dynamic is defined by* $\mathrm{d}\mu_t = f(\mu_t, t)\mathrm{d}t + g(\mu_t, t)\mathrm{d}\mathbf{b}_t$ *and the function* $f(\cdot)$ *and* $g(\cdot)$ *are* c_f*-Lipschitz and* c_g*-Lipschitz continuous with respect to* μ_t *respectively, the neuron is BIBO stable if* $2c_f - c_g^2 \le 0$.

3.2 Learn a Stable SDENet

This section describes the details of designing and training a stable SDENet based on Theorem 2. Because the drift and diffusion functions in each neuron are modeled by two MLPs, we describe how to estimate the Lipschitz constants of the MLPs and regularize these constants during the training so that the sufficient condition for BIBO stability is met.

Estimator of the MLP's Lipschitz constant. Given a fully connected L-layer MLP and $\Theta_\mathbf{i}$ as the i^{th} layer's weight matrix, an upper bound of the MLP's Lipschitz constant c is $c = \prod_{i=1}^{L} \|\Theta_\mathbf{i}\|_\infty$ [30].

Regularize the MLP's Lipschitz constant. Let $\{\Theta_{\mathbf{f_i}}\}_{i=1}^{L_f}$ and $\{\Theta_{\mathbf{g_i}}\}_{i=1}^{L_g}$ be the weight matrices of the MLPs associated with the drift and diffusion functions of a neuron, respectively. L_f and L_g are the number of layers in the two MLPs. We define two sets of trainable parameters $\{c_{f_i}\}_{i=1}^{L_f}$ and $\{c_{g_i}\}_{i=1}^{L_g}$ as the Lipschitz constant bounds for each layer of the two MLPs. Based on the estimator, we normalize the weight matrix $\Theta_{\mathbf{f_i}}$ and $\Theta_{\mathbf{g_i}}$ at each layer to satisfy $\|\Theta_{\mathbf{f_i}}\|_\infty \le c_{f_i}$ and $\|\Theta_{\mathbf{g_i}}\|_\infty \le c_{g_i}$. The normalization is achieved by $\Theta_{\mathbf{f_i}} = \frac{\Theta_{\mathbf{f_i}}}{\|\Theta_{\mathbf{f_i}}\|_\infty} \cdot \ln(1 + e^{c_{f_i}})$ and the same operation is applied on

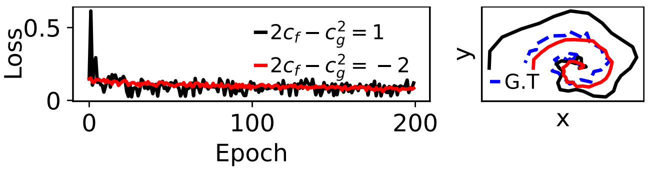


Figure 3: *Left*: training losses for two SDENets with $2c_f - c_g^2 = 1$ and $2c_f - c_g^2 = -2$. *Right*: predicted trajectories compared to the ground truth (G.T.).

$\Theta_{\mathbf{g_i}}$. The logarithmic term is included to prevent the unattainable negative Lipschitz constant [20]. To reduce the number of parameters, the two sets $\{c_{f_i}\}_{i=1}^{L_f}$ and $\{c_{g_i}\}_{i=1}^{L_g}$ are shared across all neurons. During training, we add the term $2\prod_{i=1}^{L_f} c_{f_i} - \prod_{i=1}^{L_g} c_{g_i}^2$ to the loss function of each neuron to satisfy the sufficient condition in Theorem 2. Fig. 2 demonstrates the output differences of a neuron for different inputs when setting c_{f_i} and c_{g_i} to different values and $L_f = L_g = 1$. It can be observed that when $2c_{f_i} - c_{g_i}^2 > 0$, the two outputs diverge over time; conversely, they converge. This verifies the effectiveness of the neuron stability and the above method for learning a stable neuron.

3.3 Case Study on Robot Motion Modeling

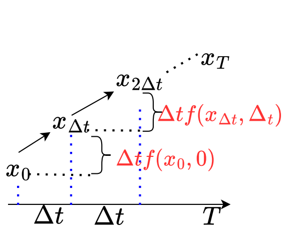
In this section, we conduct a case study to compare the training and prediction performance of the SDENet when the sufficient condition for stability is met or not. We collect real-world motion data from a robotic arm moving in a two-dimensional plane and train the SDENet to predict the arm's trajectory $\{(x_t, y_t)\}_{t\in[0,T]}$. The randomness of the motion comes from the oscillations of the arm and the inaccurate motion control. We train two SDENets, with $2c_f - c_g^2 = 1$ and $2c_f - c_g^2 = -1$. The left part of Fig. 3 shows when the sufficient condition is met, the oscillation of the training loss is smaller, and convergence is achieved more quickly. In contrast, when the condition is not satisfied, the training loss may oscillate due to instability in the model, which causes fluctuations in the gradient. The right side of the Fig. 3 plots the ground truth trajectory along with the trajectories predicted by the two SDENets. When the sufficient condition is met, prediction errors remain small and do not accumulate, keeping the trajectory near the ground truth.

4 COMPUTATION-EFFICIENT SDENET

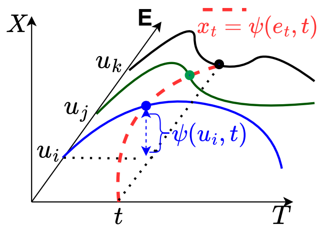
This section begins by introducing how the stochastic collocation method (SCM) approximates a stochastic process $\mathbf{X}$ with low time complexity, followed by the design of SE-SDENet, which incorporates SCM in its structure. Finally, we present the complexity-aware training and inference framework for dynamically adjusting SE-SDENet's time complexity at run-time.

4.1 Design Overview

Given a process $\mathbf{X}^{1\times T} = \{x_t\}_{t\in[0,T]}$ in $\mathbb{R}$ over the interval $[0, T]$, the SCM assumes there is another stochastic process $\mathbf{E}^{1\times T} = \{e_t\}_{t\in[0,T]}$ such that $x_t = \psi(e_t, t)$. Here, $\mathbf{E}$ can be interpreted as a measure of randomness in the system. For example, if $\mathbf{X}$ represents a series of IMU readings, then $\mathbf{E}$ could represent the IMU drift. We define the



(a) Solver-based method.

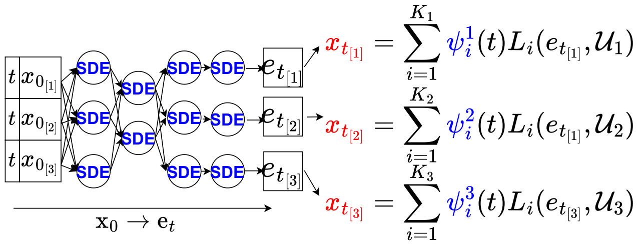


(b) SCM

Figure 4: Comparison of the solver-based method and the SCM for approximating the x_t. In the solver-based method (*left*), the time horizon is discretized. In contrast, SCM (*right*) discretizes the random space spanned by e_t using predefined collocation points $\{u_i\}_{i=1}^K$. The assumption is that $x_t = \psi(e_t, t)$, as shown in the red curve. Each curve in blue, green or black is under a different realization of e_t. Each point on the curve takes the value $\psi(u_i, t)$. SCM then approximates x_t by interpolating among these trajectories using Lagrange polynomials.

collocation point set $\mathcal{U} = \{u_i\}_{i=1}^K$ as a set of predefined nodes in the random space spanned by e_t, where $K \geq 1$ is the number of points. Then we define the function $\psi_i(t) = \psi(u_i, t)$ as the value of function ψ at these collocation points over t. The SCM seeks an approximate solution of x_t by interpolating among the values $\{\psi_i(t)\}_{i=1}^K$ in the form $\hat{x_t} = \sum_{i=1}^K \psi_i(t) L_i(e_t, \mathcal{U})$. $L_i(e_t, \{u_s\}_{s=1}^K) = \prod_{j=1, j\neq i}^K \frac{e_t - u_j}{u_i - u_j}$ defines a set of the Lagrange polynomial basis. Each polynomial $L_i(e_t, \mathcal{U})$ is constructed to be equal to 1 at its corresponding collocation point u_i and 0 at all other collocation points u_j for $j \neq i$, which allows the weighted sum of Lagrange polynomials bases to match the process values $\psi(u_i, t)$ exactly at each collocation point u_i. In $\hat{x_t} = \sum_{i=0}^K \psi_i(t) L_i(e_t, \mathcal{U})$, the term $\psi_i(t)$ captures the deterministic dynamic of the process x_t, while $L_i(e_t, \mathcal{U})$ captures the randomness inherent to the process as e_t is a measure of randomness at timestamp t. In the SCM, the deterministic dynamic and the randomness are modeled by two independent terms. The formulation is aligned with Assumption 2. Fig. 4 compares the solver-based method with the SCM. Instead of discretizing the time horizon, SCM discretizes the random space spanned by e_t using predefined collocation points $\mathcal{U}$. Consequently, the time complexity of SCM depends only on the number of collocation points, scaling as $O(K)$. Additionally, the existence and uniqueness of the functions $\{\psi_i(\cdot)\}_{i=1}^K$ have been established [26], ensuring that when an MLP is used to approximate ψ_i, the optimal parameters of the MLP are uniquely defined, thereby accelerating the training process.

Despite these advantages, applying SCM to design SE-SDENet still presents three challenges. First, when SCM is applied to a d-dimensional multivariate process $\mathbf{x}_t \in \mathbb{R}^d$, the number of collocation points grows exponentially with d. Specifically, if we select K_i collocation points for each dimension, the time complexity becomes $O(\prod_{i=1}^d K_i)$ because we need sufficient collocation points to cover the random space spanned by $\mathbf{e}_t$. Second, we need to determine



t
$x_{0_{[1]}}$
t
$x_{0_{[2]}}$
t
$x_{0_{[3]}}$
SDE
SDE
SDE
SDE
SDE
SDE
SDE
SDE
SDE
SDE
SDE
SDE
SDE
$e_{t_{[1]}}$
$e_{t_{[2]}}$
$e_{t_{[3]}}$
$\mathbf{x}_0 \to \mathbf{e}_t$
$x_{t_{[1]}} = \sum_{i=1}^{K_1} \psi_i^1(t) L_i(e_{t_{[1]}}, \mathcal{U}_1)$
$x_{t_{[2]}} = \sum_{i=1}^{K_2} \psi_i^2(t) L_i(e_{t_{[1]}}, \mathcal{U}_2)$
$x_{t_{[3]}} = \sum_{i=1}^{K_3} \psi_i^3(t) L_i(e_{t_{[3]}}, \mathcal{U}_3)$

Figure 5: The structure of the SE-SDENet for $\mathbf{x}_t \in \mathcal{R}^3$. It first learns the randomness in X modeled by $\mathbf{e}_t$ from $\mathbf{x}_0$. Then, SE-SDENet approximates $\mathbf{x}_t$ dimension by dimension with $\hat{\mathbf{x}}_{t_{[j]}} = \sum_{i=1}^{K_j} \psi_i^j(t) L_i(\mathbf{e}_{t_{[j]}}, \mathcal{U}_j)$. Blue parts contain learnable parameters and red parts are the outputs.

$\mathbf{e}_t$ based on $\mathbf{x}_0$ with low time complexity. Lastly, selecting and adjusting collocation points to adjust the time complexity remains a significant challenge.

Let $\mathbf{x_{t[j]}}$ denote the value of the j^{th} dimension of $\mathbf{x_t}$. To address the first challenge, if we can ensure that $\mathbf{e}_{t_{[j]}}$ and $\mathbf{e}_{t_{[i]}}$ are independent for all $0 < i, j \leq d$, then SCM can be applied independently to each dimension for estimation. For each dimension, we use the SCM method for the univariate case with $\hat{\mathbf{x}}_{t_{[j]}} = \sum_{i=0}^{K_j} \psi_i^j(t) L_i(\mathbf{e}_{t_{[j]}}, \mathcal{U}_j)$. The $\mathcal{U}_j = \{u_i^j\}_{i=1}^{k_j}$ is one of the d sets of collocation points $\mathcal{U} = \{\mathcal{U}_j\}_{j=1}^d$. The structure of SE-SDENet for $\mathbf{x_t} \in \mathbb{R}^3$ is shown in Fig. 5. SE-SDENet is separated into two parts: the first part utilizes an SDENet structure to generate $\mathbf{e_t}$ from $\mathbf{x_0}$ and t, and the second part applies SCM on each dimension of $\mathbf{e_t}_{[j]}$ to approximate the corresponding value of $\mathbf{x_t}_{[j]}$. Assume the time complexity for the first part is $O(M)$, where M is the number of neurons. The time complexity of SE-SDENet for an $\mathbb{R}^d$ process is reduced to $O(d \cdot \bar{K} + M)$, where $\bar{K}$ is the average number of collocation points. The following subsection introduces how to learn the randomness modeled by $\mathbf{e}_t$ with constant time complexity.

4.2 Learn the Randomness Modeled by $\mathbf{e}_t$

As described in §4.1, we assume that $\mathbf{x}_t = \psi(\mathbf{e}_t, t)$. Therefore, we can represent $\mathbf{e}_t = \psi^{-1}(\mathbf{x}_t, t)$. Consequently, we can learn the dynamics defined by ψ^{-1} using an SDENet, named as $\psi^{-1}Net$. In the final layer of $\psi^{-1}Net$ shown in Fig. 5, the i^{th} neuron only receives input from the i^{th} neuron in the previous layer to ensure that each dimension of $\mathbf{e}_t$ are independent. Next, we describe how to select the structure of the SDE for each neuron of $\psi^{-1}Net$ based on the properties of $\mathbf{e}_t$, allowing us to efficiently obtain $\mathbf{e}_t$. Note that $\psi^{-1}Net$ is not a general SDENet, as it assumes the independence of each dimension in $\mathbf{e}_t$.

Choice of the embedded SDE. Let each neuron in ψ^{-1}-Net satisfies the autonomous dynamic $\mathrm{d}m_t = f(m_t)\mathrm{d}t + g(m_t)\mathrm{d}b_t$. §4.1 explains that $\mathbf{e}_t$ can be interpreted as a dynamic process representing variables that influence the system's randomness. This implies that the drift function $f(\cdot)$ is dependent on the diffusion function $g(\cdot)$. Therefore, we can have $f(m_t) = \hat{f}(g(m_t))$. Expressing the drift term as a function of the diffusion term allows us to control the influence of randomness on the system's behavior, ensuring that the drift remains consistent with the stochastic properties of $\mathbf{e}_t$. In our design, we choose $f(m_t) = g(m_t) + \frac{1}{2}g(m_t)g'(m_t)$, resulting in the SDE $\mathrm{d}m_t = [g(m_t) + \frac{1}{2}g(m_t)g'(m_t)]\mathrm{d}t + g(m_t)\mathrm{d}b_t$. In the next part, we will show that the SDE has an approximate closed-form solution so we can get the output of each neuron with $O(1)$ time complexity and derive $\mathbf{e}_t$ with $O(M)$ time complexity.

Approximate the closed-form solution of m_t. We perform integration by substitution with $v_t = h(m_t) = \int_{m_0}^{m_t} \frac{1}{g(s)}\mathrm{d}s$ to transform the SDE $\mathrm{d}m_t = [g(m_t) + \frac{1}{2}g(m_t)g'(m_t)]\mathrm{d}t + g(m_t)\mathrm{d}b_t$ into a linear SDE $\mathrm{d}v_t = \mathrm{d}t + \mathrm{d}b_t$. With Itô's Lemma [14], we have $dv_t = \mathrm{d}h(m_t) = \{[g(m_t) + \frac{1}{2}g'(m_t)q(m_t)]\frac{\mathrm{d}h(m_t)}{\mathrm{d}t} + \frac{1}{2}g^2(m_t)\frac{\mathrm{d}^2h(m_t)}{\mathrm{d}^2t}\}\mathrm{d}t + g(m_t)\frac{\mathrm{d}h(m_t)}{\mathrm{d}t}\mathrm{d}b_t$. Since $\frac{\mathrm{d}h(m_t)}{\mathrm{d}t} = g(m_t)$, it follows that $\mathrm{d}v_t = \mathrm{d}t + \mathrm{d}b_t$. The linear SDE has a closed-form solution $v_t = t + b_t + h(m_0)$, so we have $m_t = h^{-1}(t + b_t + h(m_0))$.

Implementation details. To facilitate the calculation of $h(\cdot)$ and $h^{-1}(\cdot)$, we choose the MLP associated with $g(\cdot)$ to be a two-layer MLP with the Sigmoid activation function. The closed-form expressions for $h(\cdot)$ and $h^{-1}(\cdot)$ are derived in [21] and we model $g'(\cdot)$ with another MLP. We regularize $g(\cdot)$ and $g'(\cdot)$'s Lipschitz constants to c_q and $c_{q'}$ with the method described in §3.2. It can be derived that the drift function's Lipschitz constant is $c_q + 2c_qc_{q'}$. Therefore, to satisfy the sufficient condition for neuron stability, we should have $c_q^2 - 2c_qc_{q'} - 2c_q \geq 0$. Solving this quadratic inequality results in $c_q \geq 2(c_{q'} + 1)$.

4.3 Optimal Selection of Collocation Points

Based on the design in §4.2, the time complexity of the SCM for the process in $\mathbb{R}^d$ is $O(d \cdot \bar{K} + m)$. The value of $\bar{K}$ directly influences the complexity of the SCM. Therefore, adjusting the value of $\bar{K}$ provides SE-SDENet with the flexibility to balance the time complexity and accuracy. In what follows, we present how to dynamically adjust the value of $\bar{K}$ and select the corresponding collocation points at run-time.

Initialize the collocation points. We initialize d collocation point sets $\{\mathcal{U}_i\}_{i=1}^d$, with K_i points in each set. In implementation, we set $K_i = K$ for all dimensions so $\bar{K} = K$. Each point u_j^i in the i^{th} set is $u_j^i = \cos\left(\frac{(2j-1)\pi}{2K}\right)$. The initialization reduces interpolation error near the boundaries [26].

Adjust the number of collocation points. At run-time, we aim to reduce K to adjust complexity while maintaining accuracy. Here, we describe the algorithm for the j^{th} collocation point set, and the algorithms for other sets are identical. Let F_e^j denote the probability density function (PDF) of $\mathbf{e}_{t_{[j]}}$. During training, we collect numerical samples of $\mathbf{e}_{t_{[j]}}$ and approximate F_e^j using Gaussian kernel density estimation, where the kernel bandwidth is determined by Silverman's rule [27]. To enhance the accuracy of the SCM, we allocate more collocation points in the region where the density of $\mathbf{e}_{t_{[j]}}$ is higher and remove the points in low-density regions. Based on this idea, we construct a binary vector $\mathbf{s_j}$ to determine whether a collocation point should be removed. Suppose we want to select $\hat{K}$ collocation points out of K of them, the vector $\mathbf{s_j}$ is defined as follows: $\mathbf{s_j}[i] = 1$ if the probability $F_e^j(u_i^j)$ is among the largest $\hat{K}$ values of $\{F_e^j(u_1^j), \ldots, F_e^j(u_K^j)\}$; otherwise, $\mathbf{s_j}[i] = 0$,

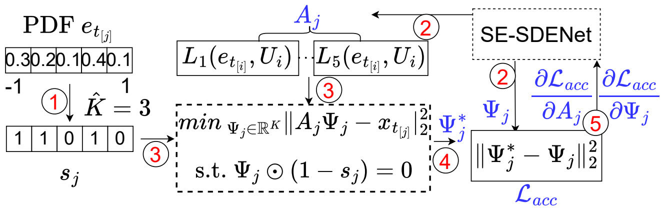

PDF $e_{t_{[j]}}$
0.3
0.2
0.1
0.4
0.1
-1
1
1
$\hat{K} = 3$
1
1
0
1
0
s_j
A_j
$L_1(e_{t_{[i]}}, U_i)$
$L_5(e_{t_{[i]}}, U_i)$
2
SE-SDENet
3
3
$min_{\Psi_j \in \mathbb{R}^K} \|A_j\Psi_j - x_{t_{[j]}}|_2^2$
s.t. $\Psi_j \odot (1 - s_j) = 0$
Ψ_j^*
4
2
Ψ_j
$\frac{\partial \mathcal{L}_{acc}}{\partial A_j}$
$\frac{\partial \mathcal{L}_{acc}}{\partial \Psi_j}$
5
$\|\Psi_j^* - \Psi_j\|_2^2$
$\mathcal{L}_{acc}$

Figure 6: Complexity-aware training: *Step 1*: suppose there are 5 collocation points originally, the value range of $\mathrm{e}_{\mathrm{t}[j]}$ is (-1,1). We want to retain only 3 points, so we choose the 3 points with the highest probability. *Step 2*: the SE-SDENet generates the Ψ_j and $\mathrm{A_j}$. *Step 3*: solve the convex optimization problem to obtain optimal value for Ψ_j,Ψ_j^*. *Step 4 & 5*: calculate the loss for accuracy and update the model parameters with gradients $\frac{\partial \mathcal{L}_{acc}}{\partial \mathrm{A_j}}$ and $\frac{\partial \mathcal{L}_{acc}}{\partial \Psi_\mathrm{j}}$, where both of the gradients depend on $\mathrm{s_j}$.

which means that we remove the i^{th} collocation point. An example of reducing the number of collocation points from 5 to 3 is shown in step 1 of Fig. 6. Note that K is initialized as a large value and we always have $\hat{K} \leq K$. The next section describes how to train SE-SDENet to minimize the impact on accuracy when adjusting the number of collocation points. We name the training procedure *complexity-aware training*.

4.4 Complexity-aware Training

The complexity-aware training framework regards $\mathbf{s_j}$ as an input of the SE-SDENet and updates SE-SDENet's parameters based on $\mathbf{s_j}$. This requires designing the loss function so that the gradient with respect to parameters of SE-SDENet depends on $\mathbf{s_j}$. The parameters of SE-SDENet can be divided into two parts. The first part generates $\mathbf{e}_{t_{[j]}}$, then generates the K Lagrange polynomial bases $\mathbf{A_j} = [L_1(\mathbf{e}_{t_{[j]}}, \mathcal{U}_j), \ldots, L_K(\mathbf{e}_{t_{[j]}}, \mathcal{U}_j)]$. The second part generates the weights for bases, denoted as $\mathbf{\Psi_j} = \left[\psi_1^j(t, \mathbf{s_j}_{[1]}) \ldots \psi_K^j(t, \mathbf{s_j}_{[K]})\right]^T$. Here, we slightly modify the original SCM and regard $\mathbf{s_j}$ as an input for generating the weights. If $\psi_i^j(t) \to 0$ when $\mathbf{s_j}_{[i]} \to 0$, then the impact of removing the collocation point u_i^j on the interpolation accuracy is small because the collocation point contributes little to the weighted sum in the SCM. Based on this principle, the optimal value for $\mathbf{\Psi_j}$, which is denoted by $\mathbf{\Psi_j^*}$, can be obtained by solving the following optimization problem with sparsity constraint:

$$\min_{\mathbf{\Psi_j} \in \mathbb{R}^K} \|\mathbf{A_j}\mathbf{\Psi_j} - x_{t_{[j]}}\|_2^2 \quad \text{s.t.} \quad \mathbf{\Psi_j} \odot (1 - \mathbf{s_j}) = 0,$$

where the $\odot$ represents the Hadamard product. The objective function is designed to minimize the modeling error. The constraint ensures that $\psi_i^j(u_i^j, \mathbf{s_j}_{[i]}) \to 0$ as $\mathbf{s_j}_{[i]} \to 0$. We observe that this optimization problem is convex, allowing us to directly obtain the optimal solution $\mathbf{\Psi_j^*}$. The loss function for accuracy is set as $\mathcal{L}_{acc} = \|\mathbf{\Psi_j^*} - \mathbf{\Psi_j}\|_2^2$. Naturally, the gradient $\frac{\partial \mathcal{L}_{acc}}{\partial \mathbf{\Psi_j}}$ is related to $\mathbf{s_j}$ and we now introduce how the gradient $\frac{\partial \mathcal{L}_{acc}}{\partial \mathbf{A_j}}$ depends on

$\mathbf{s}_\mathrm{j}$. Based on the chain rule $\frac{\partial \mathcal{L}_{acc}}{\partial \mathrm{A}_\mathrm{j}} = \frac{\partial \mathcal{L}_{acc}}{\partial \mathbf{\Psi}^*_\mathrm{j}} \cdot \frac{\partial \mathbf{\Psi}^*_\mathrm{j}}{\partial \mathrm{A}_\mathrm{j}}$, the only unknown part is $\frac{\partial \mathbf{\Psi}^*_\mathrm{j}}{\partial \mathrm{A}_\mathrm{j}}$. The Karush–Kuhn–Tucker (KKT) conditions [16] establish the relationship between $\mathbf{\Psi}^*_\mathrm{j}$ and $\mathbf{A}_\mathrm{j}$. Let λ_j represent the Lagrange multipliers associated with the constraint in the dual problem. The stationary condition in KKT conditions is $2\mathbf{A}_\mathrm{j}{}^\top(\mathbf{A}_\mathrm{j}\mathbf{\Psi}^*_\mathrm{j} - \mathbf{x}_{\mathrm{t}_{[\mathrm{j}]}}) + \lambda_\mathrm{j} \odot (1 - \mathbf{s}_\mathrm{j}) = 0$. Therefore, the gradient $\frac{\partial \mathbf{\Psi}^*_\mathrm{j}}{\partial \mathrm{A}_\mathrm{j}}$ can be derived as $\frac{\partial \mathbf{\Psi}^*_\mathrm{j}}{\partial \mathrm{A}_\mathrm{j}} = \frac{(\mathrm{A}_\mathrm{j}{}^T \mathrm{A}_\mathrm{j})\mathrm{x}_{\mathrm{t}_{[\mathrm{j}]}} - \left(\mathrm{A}_\mathrm{j}{}^T \mathrm{x}_{\mathrm{t}_{[\mathrm{j}]}} - \frac{\lambda_\mathrm{j}}{2} \odot (1 - \mathrm{s}_\mathrm{j})\right) 2\mathrm{A}_\mathrm{j}}{(\mathrm{A}_\mathrm{j}{}^T \mathrm{A}_\mathrm{j})^2}$, which depends on $\mathbf{s}_\mathrm{j}$. The full process of the *complexity-aware training* is shown in Fig. 6. Our approach ensures that the gradients of SE-SDENet are related to the required model complexity. This design enables SE-SDENet's parameters to adapt effectively to different complexity requirements while maintaining high accuracy.

5 PERFORMANCE EVALUATION

We evaluate SE-SDENet in three perspectives. First, we benchmark its accuracy, efficiency and stability in handling time series on a workstation computer across six datasets, including four tasks for time series forecasting and two tasks for time series classification. In addition, we evaluate the impact of violating Assumptions 1 and 2. Next, we assess SE-SDENet's accuracy and efficiency on edge devices. Finally, we evaluate its ability to adjust the time complexity at run-time.

5.1 Benchmark of SE-SDENet

Implementation. We conduct experiments on a workstation computer with an Intel Xeon Gold 6246 CPU and an NVIDIA RTX 8000 GPU. We set the number of collocation points K to 500. Implementation details for learning $\mathbf{e}_t$ from $\mathbf{x}_0$ are covered in §4.2. A two-layer MLP with the *tanh* activation functions learns each weight corresponding to the Lagrange polynomial base.

Baselines and datasets. Our experiments consider the following time series processing models:

(1) **SDENet** and **SE-SDENet** have the same number of neurons, with each neuron satisfying a general SDE, $\mathrm{d}u_t = f(u_t, t)\mathrm{d}t + g(u_t, t)\mathrm{d}\mathbf{b}_t$. It solves the SDE with the 4th-order stochastic Runge–Kutta solver.
(2) **LTC** [12] is a bio-inspired ODENet with each neuron satisfying the ODE $\frac{\mathrm{d}\mu_t}{\mathrm{d}t} = -\omega\mu_t + m_t$, where $m_t = A(\mu_t) \cdot (a - \mu_t)$. The LTC has been proven to be BIBO stable [12].
(3) **CFC** [11] approximates the ODE in LTC with the closed-form solution $\mu_t \approx A(-\mu_t)(\mu_0 - a)\mathrm{e}^{-[\omega + A(\mu_t)]t} + a$. It is 6x faster in inference when compared with LTC and can fit into a microcontroller (MCU) with only 256 KB static random-access memory (SRAM) [7].
(4) **TCN** [3] is a convolutional neural network model that uses 1D convolutional layers with causal padding to capture sequential dependencies in time-series data.
(5) **TimesFM** [8] is a pre-trained time-series foundation model with 500 million parameters. In the following experiments, the model is not fine-tuned for each task. Note that fine-tuning it takes 51x more time than training the SE-SDENet. Thus, the fine-tuning is impractical at edge devices. The training dataset of TimesFM includes data similar to the LibCity and ETT datasets used in our experiments.

The accuracy metric of the time series forecasting task is the Mean Absolute Percentage Error (MAPE), defined as MAPE $= \frac{1}{nT}\sum_{i=1}^{n}\sum_{t=1}^{T}\frac{\|\mathrm{y}_t^i - \hat{\mathrm{y}_t^i}\|_2^2}{\|\mathrm{y}_t^i\|_2^2}$, where y_t^i is the i^{th} ground truth at time t, $\hat{\mathrm{y}_t^i}$ is the predicted value, n is the number of test samples and T is the number of prediction steps. The accuracy metric of the time series classification task is the Top-1 Accuracy (T-ACC). We select datasets with different sampling rates to evaluate the SE-SDENet across different time series tasks.

(1) **OU Process** is a simulated 1D Ornstein-Uhlenbeck process with 50,000 samples and a sampling interval of 0.01 seconds. The process is defined as $dx_t = \theta(\alpha - x_t)\mathrm{d}t + \sigma\mathrm{d}b_t$.
(2) **PPG-DaLiA** [25] includes 30,000 photoplethysmography (PPG) samples with a sampling interval of 0.3 seconds for heart rate estimation.
(3) **LibCity** [31] consists of 40,000 samples with a sampling interval of 15 minutes for point-based traffic flow prediction.
(4) **ETT** [33] consists of 70,000 hourly-sampled energy consumption samples for energy demand forecasting.
(5) **HASC-PAC2016** [13] includes 19,172 samples collected by IMU on wrist with a sampling interval of 0.02 seconds for human activity recognition.
(6) **AMIGOS** [22] includes electroencephalogram traces from 40 subjects for emotion recognition, sampled at 128 Hz. Five bands of electroencephalogram data are used as input features. A sliding window with a 1-second window size and a 0.2-second overlap is applied to segment the trace into input samples.

Accuracy of SE-SDENet. We begin by evaluating the accuracy of SE-SDENet across the six datasets. The prediction step T on the four forecasting datasets is set to 100, 60, 12, and 4, respectively. As shown in Table 1, SE-SDENet performs the best on the OU dataset, ETT dataset and AMIGOS dataset. For the PPG-DaliA dataset, LTC achieves the lowest MAPE (12.0%) and SE-SDENet achieves the second lowest result (13.3%). In the LibCity and HASC-PAC2016 datasets, LTC marginally outperforms SE-SDENet by less than 1%. Overall, SE-SDENet consistently demonstrates strong performance across data with diverse sampling rates, particularly in scenarios with strong stochastic dynamics like hourly electric consumption. We also observe that, without fine tuning, the TimesFM cannot outperform other models.

Furthermore, we analyze the effect of the number of prediction steps, T, on the accuracy of the four forecasting tasks. Fig. 7 shows the impact of T on accuracy. The MAPE of solver-free models, i.e., SD-SDENet and CFC, increases by 4.1% and 5.92% on average when T increases, respectively. In contrast, LTC and SDENet show higher increases of 12.4% and 10.9%. Solver-free models avoid cumulative errors and the SE-SDENet's BIBO stability ensures bounded prediction errors over time.

Training and inference overhead of SE-SDENet. We benchmark SE-SDENet regarding training resource usage and inference latency on the OU dataset. We exclude TimeFM as we do not fine-tune it. The training is conducted on the NVIDIA RTX8000 GPU until the loss converges (i.e., $\mathcal{L}_{acc}$ drops below 10^{-2} for SE-SDENet,

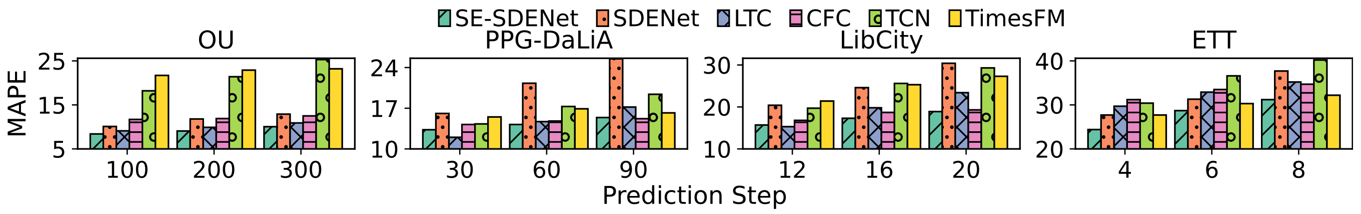
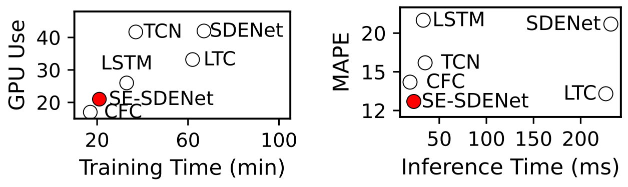


Figure 7: Influence of the number of prediction steps T on the accuracy.

Table 1: Comparison of models across datasets in terms of MAPE (forecasting) and T-ACC (classification).

	SE-SDENet	SDENet	LTC	CFC	TCN	T-FM
OU	**8.4%**	10.1%	9.1%	11.7%	18.2%	21.7%
PPG	13.3%	16.1%	**12.0%**	14.2%	14.3%	15.5%
LibCity	15.7%	20.4%	**15.3%**	16.8%	19.7%	21.4%
ETT	**24.4%**	27.7%	29.7%	31.2%	30.4%	27.7%
HASC	0.795	0.782	**0.811**	0.791	0.754	0.767
AMIGOS	**0.765**	0.742	0.712	0.723	0.687	0.494



(a) GPU usage and training time **(b) MAPE VS Inference Time**

Figure 8: Benchmark of SE-SDENet on training and inference efficiency.

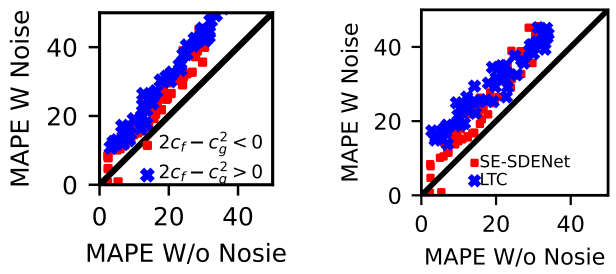


Figure 9: *Left*: comparison of stability when the stability condition is met or not. *Right*: comparison of stability between SE-SDENet and LTC.

MAPE in training drops below 10^{-2} for other models). Fig. 8a shows SE-SDENet uses 24.4% GPU resources and the second shortest training time. Fig. 8b indicates SE-SDENet achieves the lowest MAPE and the second shortest inference time, 9x faster than SDENet and comparable in efficiency to CFC, which uses an ODE with a closed-form solution in each neuron.

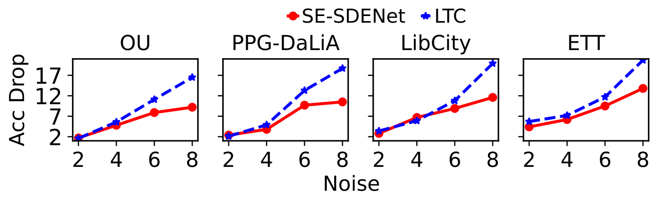


Figure 10: Accuracy drop of SE-SDENet and LTC under increasing noise level.

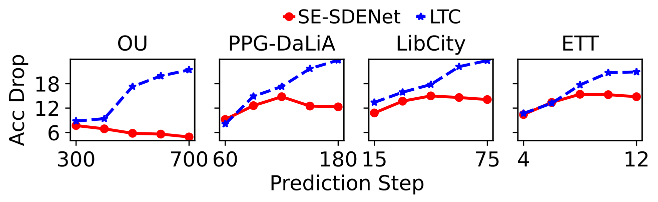


Figure 11: Accuracy drop of SE-SDENet and LTC under increasing prediction step with 4% relative noise intensity.

Stability of SE-SDENet. Left part of Fig. 9 compares the performance of SE-SDENet when the sufficient condition (i.e., $2c_f - c_g^2 \leq 0$) for BIBO stability is met or not on four datasets. We add Gaussian noise with a magnitude of 5% of the original data's magnitude to the input. When the stability condition is satisfied, the MAPE of SE-SDENet on noisy data increases by 5.4% on average, which is just 0.4% higher than the noise magnitude. In contrast, when the condition is unsatisfied, the MAPE increases by 9.8% on average.

We further compare the stability of SE-SDENet with that of LTC, which has been proven to satisfy BIBO stability [12]. The right part of Fig. 9 shows the MAPE of LTC on noisy data increases by 6.9% on average, which is higher than the 5.4% increase for SE-SDENet. In Fig. 10, we compare the stability of SE-SDENet and LTC under different noise levels. As noise intensity increases from 2% to 8%, SE-SDENet's accuracy drops by an average of 8.45% across the four datasets, less than LTC's 15.7% drop. Additionally, SE-SDENet's accuracy decline rate decreases as noise intensity increases. Finally, we set the noise intensity to 4% and increase the number of prediction steps to investigate the effect of noise accumulation over many prediction steps. According to Theorem 1, in both the presence and absence of noise, the divergence between the two learned processes converges to a fixed distance. Consequently, as the number of prediction steps T increases, the accuracy drop of SE-SDENet may decrease, as shown in Fig. 11. In contrast, LTC's accuracy drop

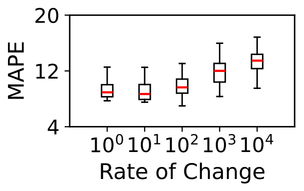


Figure 12: Influence of data smoothness on accuracy.

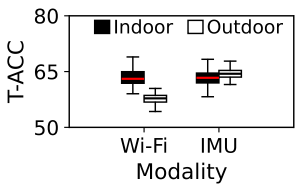


Figure 13: Effect of violating Assumption 2.

does not exhibit any trend of decline, but instead, progressively increases as T increases.

5.2 Impact of Assumption Violations

This section evaluates the influence of smoothness and the effect of violating Assumption 2 on the accuracy of SE-SDENet.

Influence of data smoothness on the accuracy. The OU process dataset is generated from the SDE $\mathrm{d}x_t = \theta(\alpha - x_t)\mathrm{d}t + \sigma \mathrm{d}B_t$, where θ controls smoothness. We set $\alpha = 1$, $\sigma = \frac{\theta}{2}$, and vary θ from 1 to 10,000 to assess the effect of data smoothness on accuracy. Fig. 12 shows that SE-SDENet's accuracy remains stable for θ between 1 and 100 and only decreases by 7.4% as θ increases to 10,000. This suggests that as long as the data remains continuous, its smoothness has a relatively minor impact on SE-SDENet's accuracy.

Effect of violating Assumption 2 on the accuracy. In IMU-based human recognition systems, noise is independent of human motion. However, in Wi-Fi-based human recognition systems, noise caused by the multi-path effect is influenced by movement and is more pronounced indoors than outdoors. As a result, Assumption 2 is violated. Using the Vi-Fi dataset [4], we train SE-SDENet for human recognition based on IMU and Wi-Fi RSSI data, comparing indoor and outdoor accuracy to assess the impact of violating Assumption 2. Fig. 13 shows that the accuracy difference between indoor and outdoor environments for IMU-based SE-SDENet is 2.3%, whereas for Wi-Fi-based SE-SDENet, indoor accuracy is 7.6% higher than outdoor accuracy. This indicates that violating Assumption 2 affects the accuracy of SCM, thereby affecting the accuracy of SE-SDENet.

5.3 Evaluation in Edge Environments

In this section, we deploy SE-SDENet on various edge devices to assess its performance in handling edge computing challenges. First, we demonstrate that SE-SDENet can run on highly resource-constrained devices, such as a microcontroller with only 256 KB SRAM. Next, we evaluate SE-SDENet's throughput across different devices to verify its efficiency.

Deployability on resource-constrained devices. In this section, we demonstrate that SE-SDENet can be deployed on resource-constrained devices by adjusting the number of collocation points K during the design phase. We test SE-SDENet on three microcontrollers with SRAM ranging from 256 KB to 891 KB. We train SE-SDENet with different K on a workstation computer and test it on the PPG-DaLiA dataset with $T = 15$. We choose different K to keep SRAM usage at approximately 50% of each device's available memory. Table 2 shows the selected microcontrollers, corresponding K, average run-time RAM usage, and average MAPE with or without noise. The results indicate that average SRAM usage remains close to 50% on each device, with accuracy decreased by 5.9%, 5.5%, and 4.7% only, compared with SE-SDENet trained with $K = 500$. In §5.4, we will further analyze the impact of K on SE-SDENet's performance. Meanwhile, when the input contains Gaussian noise with a relative intensity of 3%, the accuracy drop of SE-SDENet on MCUs remains below 3%, demonstrating that SE-SDENet can directly handle noisy inputs on MCUs.

Table 2: Deploy on resource-constraint MCUs.

Device	RAM (KB)	K	Size (KB)	RAM Use	MAPE	Noise MAPE
Arduino Nano 33	256	30	29.1	49.3%	19.1%	20.9%
STM32F7	331	50	39.3	56.7%	18.7%	20.1%
STM32H7	891	80	71.0	50.5%	17.9%	18.8%

Throughput on different devices. In this experiment, we run a single model per thread and execute multiple threads in parallel. The number of concurrently running models is gradually increased until the device's computational resources are fully utilized. This setup simulates realistic edge computing environments, ensuring maximum resource utilization. For the setup, we use the STM32H7 microcontroller and include three more powerful edge devices: an Android smartphone with a Snapdragon 865 CPU, a personal computer (PC) with an Intel Xeon W-2133 CPU, and the NVIDIA Jetson AGX Orin. We choose the LibCity dataset for this experiment and set K to 50, 200, 300, and 500 across the four devices. Fig. 14 shows that when running a single model, SE-SDENet achieves approximately 5.1×, 3.4×, 2.7×, and 3.1× higher throughput than SDENet on the four devices, respectively. When the available computational resources are not fully utilized by the parallel models, SE-SDENet achieves 6.6× higher throughput than SDENet on average. In the comparison, we include a vanilla RNN with only three layers, denoted by *vRNN*, as a new baseline to achieve the upper bound of the throughput. Note that the vRNN achieves a low accuracy (42.9% MAPE) on the LibCity dataset. On four different devices, the throughput of SE-SDENet reaches 59%, 79%, 81%, and 89% of the throughput of vRNN, respectively. Compared with CFC, which utilizes a closed-form solution, the throughput of SE-SDENet differs from that of CFC by no more than ±5.1%. The result highlights SE-SDENet's small inference latency and small memory footprint, which are critical for edge environments.

5.4 Influence of K

This section analyzes the impact of the number of collocation points K on the efficiency and accuracy of SE-SDENet.

Influence of K on the efficiency. We first evaluate the influence of K on the time for solving the optimization problem for selecting the optimal collocation points. When K increases from 50 to 1100, the time for solving the problem increases by 6.4×, 5.1×, and 3.1× on the Android smartphone, the CPU-based PC, and the GPU-powered Jetson Orin, respectively. When $K = 1100$, the maximum solving latency on the phone is 106 ms. We then evaluate

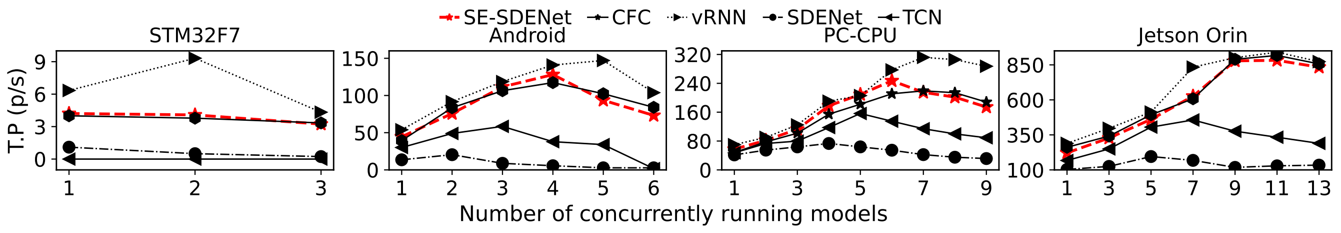


Figure 14: Throughput comparison of SE-SDENet, SDENet, TCN, CFC, RNN across different edge devices as the number of concurrently running models increases.

Table 3: Inference time and CPU/GPU usage for different numbers of collocation points.

K	Inference Time (ms)			Avg CPU/GPU Usage		
	Phone	PC	Jetson	Phone	PC	Jetson
50	2.1	1.28	0.71	7.4%	1.9%	2.1%
100	3.5	3.66	1.6	10.1%	3.2%	2.7%
300	11.8	8.23	4.7	13.2%	6.3%	4.1%
500	43.2	20.25	8.1	17.2%	9.5%	7.2%
700	65.3	34.3	14.4	22.1%	11.6%	9.1%
900	97.2	70.5	26.4	29.8%	12.5%	10.3%
1100	124.3	87.8	38.7	39.4%	15.3%	12.4%

the inference latency and corresponding GPU/CPU usage for varying K values on the PPG-DaLiA dataset. The CPU/GPU usages on the three devices are measured by the *dumpsys* service, the *psutil* library, and the *tegrastats* utility, respectively. The number of prediction steps is 90. Table 3 presents the results obtained on three different devices. For inference time, when K increases from 100 to 500, the inference latency increases by 9.6×, 5.5×, and 5.1× on the three devices, respectively. Similarly, the CPU/GPU usage increases by 1.7×, 3.9×, and 3.8× on the three devices, respectively. These results demonstrate that changes in the number of collocation points significantly impact inference time and computational resource usage, providing opportunities for adjusting the inference delay and resource usage by dynamically adjusting the number of collocation points.

Influence of K on the accuracy. We examine the impact of adjusting K on accuracy during testing and compare our method of adjusting K with uniformly sampling K collocation points. During training, the maximum K is set to 600. Fig. 15 shows that as K decreases during testing, our method consistently achieves lower average MAPE than uniform sampling. The blue dashed line represents the MAPE at $K = 600$. When K decreases to 500, 400, 300, and 200, MAPE only increases by 1.07%, 1.89%, 2.12%, and 2.93% on average, respectively. However, as shown in Table 3, the computational resource usage of SE-SDENet can be reduced by approximately 50%, and the latency decreases by around 70% when K is reduced from 600 to 200. This demonstrates that reducing K can significantly lower computational cost with minimal impact on average accuracy. This is achieved by the complexity-aware training framework, which optimizes the SE-SDENet's weights based on different time complexity requirements. In the meantime, we also observe greater accuracy fluctuations as K decreases. At $K = 200$, the average MAPE variance is 2.4%, compared with 0.94% at $K = 600$. Due to the reduced K, the chosen collocation points cannot cover the entire range of $\mathbf{e_t}$, leading to accuracy fluctuation for some samples.

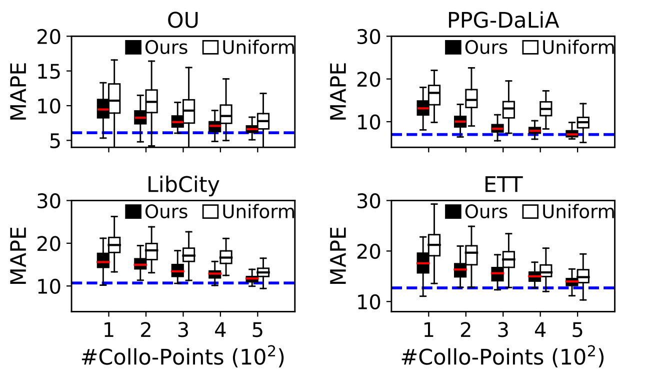


Figure 15: Comparison between the complexity-aware framework and uniform sampling methods. The blue line indicates MAPE at $K = 600$.

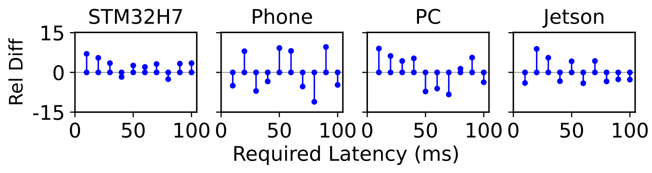
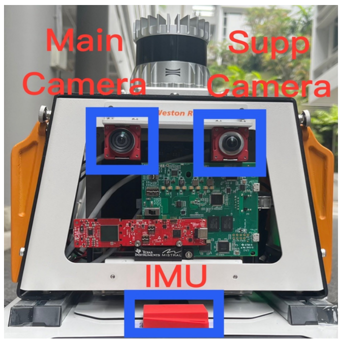


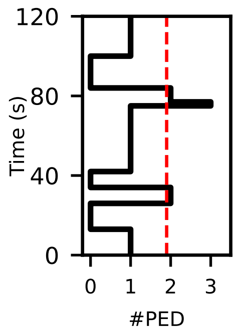
Figure 16: The relative differences between the actual latency and the required latency.

5.5 Evaluation of SE-SDENet's Complexity Adjustment Ability

This section evaluates whether SE-SDENet can adjust its complexity at run-time to meet system latency requirements. First, we profile the deployment device to establish the relationship between K and the corresponding latency on that device. The works [10, 28], which design faster modeling of execution time on specific devices, can serve as a component of our system. We test the OU process dataset on four devices, with specified latency requirements ranging from 10 ms to 100 ms. On the four devices, the maximum values of K are 140, 1000, 1500, and 3000, respectively. In Fig. 16, we show the relative differences between the actual latency and the required latency. The maximum relative differences observed across the four devices are 7.1%, 11.1%, 8.3%, and 8.8%, respectively. Even on the Jeston Orin, with a wide adjustable range of K, the average relative difference maintains below 4.71%.



(a) Robot car and sensors.



(b) #detected pedestrians.

Figure 17: Experiment settings.

6 CASE STUDY: RUN-TIME MODEL COMPLEXITY ADJUSTMENT

This section considers an autonomous robot scenario where model complexity should be dynamically adjusted at run-time to meet computational and throughput constraints. Fig. 17a shows the robotic car we use to collect data from two cameras, an IMU, and motor power data from Robot Operating System (ROS) during operation. We use the YOLOv8-tiny model to process camera data and train an SE-SDENet to predict IMU-measured acceleration in two directions based on motor power in the front and back of the car. In an autonomous driving scenario, the system adapts to varying environments by utilizing different sensor data, leading to fluctuations in available resources. To simulate this setup, we process only the main camera data when fewer than 2 pedestrians are detected; otherwise, we also process the side camera for additional environmental sensing until fewer than 2 pedestrians are detected. Fig. 17b shows the pedestrian count in the test period. All data is processed on a Jetson Orin, where we require SE-SDENet's throughput to remain above 55 samples per second (sps) while ensuring GPU usage stays within available resources.

Fig. 18 shows SE-SDENet's latency and GPU usage on the Jetson for different K. At run-time, we select the maximum K that meets both resource and throughput constraints based on the profiling result. The first row of Fig. 19 plots the available GPU resource and the GPU resource used by the SE-SDENet in 120 seconds. When available GPU resource decreases at around the 19-second and 77-second marks due to the activation of the second YOLO model, SE-SDENet adjusts its model complexity, maintaining at least a 2.4% gap between consumed resource and available resource. The second row of Fig. 18 shows the corresponding throughput. SE-SDENet maintains a minimum throughput of 49.7 sps and achieves at least 55 sps throughput in 90.09% of measurements. The third row shows that the MAE fluctuations remain within 3.7, and the model maintains an average MAE of 6.1.

7 DISCUSSION

Building a foundation model for time series processing based on SE-SDENet is an interesting research direction, as it can reduce the resource usage of foundation models. Our experiments show that increasing the number of collocation points quickly reaches a saturation point in accuracy improvement. However, we find that stacking SE-SDENet enhances long-term sequence prediction accuracy at the cost of reduced stability. The decline in stability may be due to the introduction of excessive randomness. Enhancing the stability of SE-SDENet at the scales of billions of parameters will be beneficial for constructing a foundation model based on SE-SDENet. On the other hand, our current implementation adjusts SE-SDENet's complexity at fixed intervals to meet computational constraints. We believe that leveraging prior knowledge of resource consumption on edge devices to adaptively adjust the frequency of model complexity updates will facilitate the deployment of SE-SDENet on resource-rich devices.

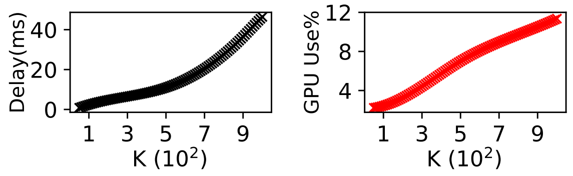


Figure 18: Latency and GPU usage profiling with different K.

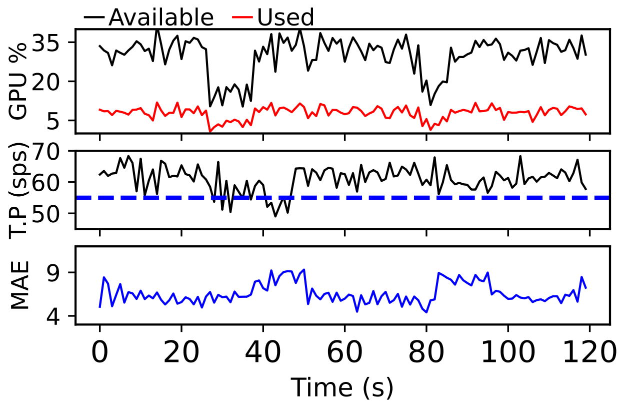


Figure 19: Run-Time GPU usage, throughput, and MAE in 120 seconds.

8 CONCLUSION

This paper presents SE-SDENet, a novel framework that achieves stability, efficiency, and adaptability for continuous-time neural networks deployed on edge devices. Our theoretical framework guarantees SE-SDENet's stability during training. In addition, the complexity-aware training and inference framework allows SE-SDENet to achieve high throughput and gives SE-SDENet the ability to dynamically adjust model complexity in response to varying resource constraints.

ACKNOWLEDGMENTS

This research is supported by the Singapore Ministry of Education under its AcRF Tier-1 grants RT14/22 and RG88/22.

REFERENCES

[1] Mohammad Arif Ul Alam, Nirmalya Roy, and Archan Misra. 2019. Tracking and behavior augmented activity recognition for multiple inhabitants. *IEEE Transactions on Mobile Computing* 20, 1 (2019).

[2] Pandarasamy Arjunan, Harshad D. Khadilkar, Tanuja Ganu, Zainul M. Charbiwala, Amarjeet Singh, and Pushpendra Singh. 2015. Multi-User Energy Consumption Monitoring and Anomaly Detection with Partial Context Information. *BuildSys* (2015).

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv* (2018).

[4] Bryan Bo Cao, Abrar Alali, Hansi Liu, Nicholas Meegan, Marco Gruteser, Kristin Dana, Ashwin Ashok, and Shubham Jain. 2022. Vitag: Online wifi fine time measurements aided vision-motion identity association in multi-person environments. *SECON* (2022).

[5] Ngoc Huy Chau, Éric Moulines, Miklos Rásonyi, Sotirios Sabanis, and Ying Zhang. 2021. On stochastic gradient langevin dynamics with dependent data streams: The fully nonconvex case. *SIAM Journal on Mathematics of Data Science* 3, 3 (2021).

[6] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. *NeurIPS* (2018).

[7] Yimin Dai and Rui Tan. 2024. FedCFC: On-Device Personalized Federated Learning with Closed-Form Continuous-Time Neural Networks. *IPSN* (2024).

[8] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. 2024. A decoder-only foundation model for time-series forecasting. *ICML* (2024).

[9] Thomas Hakon Gronwall. 1919. Note on the derivatives with respect to a parameter of the solutions of a system of differential equations. *Annals of Mathematics* 20, 4 (1919).

[10] Jussi Hanhirova, Teemu Kämäräinen, Sipi Seppälä, Matti Siekkinen, Vesa Hirvisalo, and Antti Ylä-Jääski. 2018. Latency and throughput characterization of convolutional neural networks for mobile computer vision. *MMSys* (2018).

[11] Ramin Hasani, Mathias Lechner, Alexander Amini, Lucas Liebenwein, Aaron Ray, Max Tschaikowski, Gerald Teschl, and Daniela Rus. 2022. Closed-form continuous-time neural networks. *Nature Machine Intelligence* 4, 11 (2022).

[12] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. 2021. Liquid time-constant networks. *AAAI* (2021).

[13] Haruyuki Ichino, Katsuhiko Kaji, Ken Sakurada, Kei Hiroi, and Nobuo Kawaguchi. 2016. HASC-PAC2016: large scale human pedestrian activity corpus and its baseline recognition. *UbiComp* (2016).

[14] Kiyosi Itô. 1951. On a formula concerning stochastic differentials. *Nagoya Mathematical Journal* 3, 4 (1951).

[15] Qiyu Kang, Yang Song, Qinxu Ding, and Wee Peng Tay. 2021. Stable neural ode with lyapunov-stable equilibrium points for defending against adversarial attacks. *NeurIPS* (2021).

[16] William Karush. 1939. Minima of functions of several variables with inequalities as side constraints. *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago* (1939).

[17] Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. 2020. Learning differential equations that are easy to solve. *NeurIPS* (2020).

[18] Patrick Kidger, James Foster, Xuechen Li, and Terry J Lyons. 2021. Neural sdes as infinite-dimensional gans. *ICLR* (2021).

[19] Eugene Lee and Chen-Yi Lee. 2021. PPG-based smart wearable device with energy-efficient computing for mobile health-care applications. *IEEE Sensors Journal* 21, 12 (2021).

[20] Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. 2022. Learning smooth neural functions via lipschitz regularization. *SIGGRAPH* (2022).

[21] Steffan Lloyd, Rishad A. Irani, and Mojtaba Ahmadi. 2020. Using Neural Networks for Fast Numerical Integration and Optimization. *IEEE Access* 8, 4 (2020).

[22] Juan Abdon Miranda-Correa, Mojtaba Khomami Abadi, Nicu Sebe, and Ioannis Patras. 2018. AMIGOS: A Dataset for Affect, Personality and Mood Research on Individuals and Groups. *IEEE Transactions on Affective Computing* 12, 3 (2018).

[23] YongKyung Oh, Dong-Young Lim, and Sungil Kim. 2024. Stable neural stochastic differential equations in analyzing irregular time series data. *arXiv* (2024).

[24] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. 2020. Hypersolvers: Toward fast continuous-depth models. *NeurIPS* (2020).

[25] Indlekofer Ina Reiss, Attila and Philip Schmidt. 2019. PPG-DaLiA. UCI Machine Learning Repository.

[26] Jie Shen, Tao Tang, and Li-Lian Wang. 2011. *Spectral methods: algorithms, analysis and applications.* Springer Science & Business Media.

[27] Bernard W Silverman. 2018. *Density estimation for statistics and data analysis.* Routledge.

[28] Tianxiang Tan and Guohong Cao. 2021. Efficient execution of deep neural networks on mobile devices with npu. *IPSN* (2021).

[29] A Vaswani. 2017. Attention is all you need. *NeurIPS* (2017).

[30] Aladin Virmaux and Kevin Scaman. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *NeurIPS* (2018).

[31] Jingyuan Wang, Jiawei Jiang, Wenjun Jiang, Chao Li, and Wayne Xin Zhao. 2021. LibCity: An Open Library for Traffic Prediction. *SIGSPATIAL* (2021).

[32] Xin Wang and Edward K Blum. 1992. Discrete-time versus continuous-time models of neural networks. *J. Computer and System sciences* 45, 1 (1992).

[33] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. *AAAI* (2021).

A PROOF OF THEOREM 1

Proof. Suppose $\mathbf{z}_0 \in \mathcal{R}^d$ and $\mathbf{y}_0 \in \mathcal{R}^d$ are initial values of the dynamic $\mathrm{d}\mathbf{x}_t = f(\mathbf{x}_t, t)\mathrm{d}t + g(\mathbf{x}_t, t)\mathrm{d}\mathbf{b}_t$ and $\mathbf{z}_t \in \mathcal{R}^d$ and $\mathbf{y}_t \in \mathcal{R}^d$ are the corresponding values at timestamp t. Considering the process $\mathbf{r}_t = \mathbf{z}_t - \mathbf{y}_t$, we first compute the value of $\|\mathbf{r}_t\|^2$. Based on Itô's lemma [14] for the function $f(\mathbf{r}_t) = \|\mathbf{r}_t\|^2 = \mathbf{r}_t^\top \mathbf{r}_t$, we have:

$$\mathrm{d}\|\mathbf{r}_t\|^2 = 2\mathbf{r}_t^\top (d\mathbf{z}_t - d\mathbf{y}_t) + \mathrm{Tr}((g(\mathbf{z}_t) - g(\mathbf{y}_t))^\top (g(\mathbf{z}_t) - g(\mathbf{y}_t)))\mathrm{d}t.$$

Substituting the expressions for $d\mathbf{z}_t$ and $d\mathbf{y}_t$ into the above expession gives $\mathrm{d}\|\mathbf{r}_t\|^2 = 2\mathbf{r}_t^\top (f(\mathbf{z}_t) - f(\mathbf{y}_t))\, \mathrm{d}t +$ $2\mathbf{r}_t^\top (g(\mathbf{z}_t)\mathrm{d}\mathbf{b}_t - g(\mathbf{y}_t)d\mathbf{b}_t) + \mathrm{Tr}((g(\mathbf{z}_t) - g(\mathbf{y}_t))^\top (g(\mathbf{z}_t) - g(\mathbf{y}_t)))\mathrm{d}t$. Take expectation on both sides and note that expectation of the terms involving $\mathrm{d}\mathbf{b}_t$ vanishes to zero due to zero mean of the $\mathbf{b}_t$, we obtain $\frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}\|\mathbf{r}_t\|^2 = \mathbb{E}\left[\mathrm{Tr}((g(\mathbf{z}_t) - g(\mathbf{y}_t))^\top (g(\mathbf{z}_t) - g(\mathbf{y}_t)))\right] + 2\mathbb{E}\left[\mathbf{r}_t^\top (f(\mathbf{z}_t) - f(\mathbf{y}_t))\right]$. Based on Assumption 1, there are constants c_f and c_g such that $\|f(\mathbf{z}_t) - f(\mathbf{y}_t)\| \le c_f\|\mathbf{r}_t\|$, and $\|g(\mathbf{z}_t) - g(\mathbf{y}_t)\| \le c_g\|\mathbf{r}_t\|$. Therefore, we can bound the left side of the expectation by $\frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}\|\mathbf{r}_t\|^2 \le 2c_f\mathbb{E}\|\mathbf{r}_t\|^2 + c_g^2\mathbb{E}\|\mathbf{r}_t\|^2$. By applying Gronwall's inequality [9], we obtain $\mathbb{E}\|\mathbf{r}_t\|^2 \le \mathbb{E}\|\mathbf{r}_0\|^2 e^{(2c_f + c_g^2)t}$. Based on the relationship between the $\mathbb{W}_2$ distance and the L_2 distance, we have $\mathbb{W}_2(\mathbf{z}_t, \mathbf{y}_t) \le \mathbb{W}_2(\mathbf{z}_0, \mathbf{y}_0)e^{\left(\frac{1}{2}(2c_f - c_g^2)t\right)} + e^{c_g^2 t}\mathbb{W}_2(\mathbf{z}_0, \mathbf{y}_0)$.

In what follows, we show that there exists an upper bound for $e^{c_g^2 t}\mathbb{W}_2(\mathbf{z}_0, \mathbf{y}_0)$, denoted by Υ. Assume that $\mathbf{z}_0$ and $\mathbf{y}_0$ are the solutions of $\mathrm{d}\hat{\mathbf{x}}_t = g(\hat{\mathbf{x}}_t, t)\mathrm{d}\mathbf{b}_t$ at time $\hat{t}$, with initial values $\hat{\mathbf{z}}$ and $\hat{\mathbf{y}}$, respectively. The diffusion function g is identical to that of the original process. To make this proof self-contained, we restate two existing results as the following two lemmas to establish an upper bound for $\mathbb{W}_2(\mathbf{z}_0, \mathbf{y}_0)$.

Lemma 1 (Theorem 2.3 in [5]). *If $\exists a > 0$ and a Lyapunov function $F(\cdot)$, such that $\forall \hat{\mathbf{x}}_\mathbf{t}$, $\frac{1}{2}\,\mathrm{Tr}\left(g(\hat{\mathbf{x}}_\mathbf{t}, t)\, g(\hat{\mathbf{x}}_\mathbf{t}, t)^T\, \nabla^2_{\hat{\mathbf{x}}_\mathbf{t}} F(\hat{\mathbf{x}}_\mathbf{t})\right)$ $+ \frac{\partial F}{\partial t} \le aF(\hat{\mathbf{x}}_\mathbf{t})$, then $\exists b > 0$, such that $\mathbb{W}_{1,2}(F(\mathbf{z}_t), F(\mathbf{y}_t)) \le be^{-at}\mathbb{W}_{1,2}(F(\hat{\mathbf{z}}_0), F(\hat{\mathbf{y}}_0)))$. The $\mathbb{W}_{1,2}$ is the geometric ergodicity of the SDE.*

Lemma 2 (Theorem 3.5 in [23]). *Given a Lyapunov function $F(\cdot)$, $\mathbb{W}_2(\mathbf{z}_t, \mathbf{y}_t) \le \sqrt{2\mathbb{W}_{1,2}(F(\mathbf{z}_t), F(\mathbf{y}_t))}$.*

Based on the above two lemmas, if $\exists a > 0$ and a Lyapunov function $F(\cdot)$ satisfies the condition in Lemma 2 , then $\exists b$, such that $\sqrt{2\mathbb{W}_{1,2}(F(\mathbf{z}_0), F(\mathbf{y}_0))} \le e^{-\frac{1}{2}at}\sqrt{2b\mathbb{W}_{1,2}(F(\hat{\mathbf{z}}), F(\hat{\mathbf{y}}))}$. Therefore, we have $\mathbb{W}_2(\mathbf{z}_0, \mathbf{y}_0) \le e^{-\frac{1}{2}at}\sqrt{2b\mathbb{W}_{1,2}(F(\hat{\mathbf{z}}), F(\hat{\mathbf{y}}))}$. Next, we verify that setting $a = 2c_g^2$ and $F(\hat{\mathbf{x}}) = \|\hat{\mathbf{x}}\|^2$ satisfies the condition in Lemma 2. Simplifying the left side of the condition gives $\frac{1}{2}\,\mathrm{Tr}\left(g(\mathbf{x}_\mathbf{t})\, g(\mathbf{x}_\mathbf{t})^T \cdot 2\mathbf{I}_\mathbf{d}\right) = d \cdot g(\mathbf{x}_\mathbf{t})\, g(\mathbf{x}_\mathbf{t})^T \le c_g^2\|\mathbf{x}_t\|^2 = aF(\mathbf{x}_t)$. Therefore, we have $e^{c_g^2 t}\mathbb{W}_2(\mathbf{z}_0, \mathbf{y}_0) \le e^{c_g^2 t}e^{-\frac{1}{2}at}\sqrt{2b\mathbb{W}_{1,2}(F(\hat{\mathbf{z}}), F(\hat{\mathbf{y}}))}$ $= \sqrt{2b\mathbb{W}_{1,2}(F(\hat{\mathbf{z}}), F(\hat{\mathbf{y}}))}$. We define $\Upsilon = \sqrt{2b\,\mathbb{W}_{1,2}(F(\acute{\mathbf{z}}), F(\acute{\mathbf{y}}))}$, which completes the proof. □