KalmanHD: Robust On-Device Time Series Forecasting with Hyperdimensional Computing

Ivannia Gomez Moreno
CETYS University, Campus Tijuana
ivannia.gomez@cetys.edu.mx

Xiaofan Yu
University of California, San Diego
x1yu@ucsd.edu

Tajana Rosing
University of California, San Diego
tajana@ucsd.edu

***Abstract*—Time series forecasting is shifting towards Edge AI, where models are trained and executed on edge devices instead of in the cloud. However, training forecasting models at the edge faces two challenges concurrently: (1) dealing with streaming data containing abundant noise, which can lead to degradation in model predictions, and (2) coping with limited on-device resources. Traditional approaches focus on simple statistical methods like ARIMA or neural networks, which are either not robust to sensor noise or not efficient for edge deployment, or both. In this paper, we propose a novel, robust, and lightweight method named KalmanHD for on-device time series forecasting using Hyperdimensional Computing (HDC). KalmanHD integrates Kalman Filter (KF) with HDC, resulting in a new regression method that combines the robustness of KF towards sensor noise and the efficiency of HDC. KalmanHD first encodes the past values into a high-dimensional vector representation, then applies the Expectation-Maximization (EM) approach as in KF to iteratively update the model based on the incoming samples. KalmanHD inherently considers the variability of each sample and thereby enhances robustness. We further accelerate KalmanHD by substituting the expensive matrix multiplication with efficient binary operations between the covariance and the encoded values. Our results show that KalmanHD achieves MAE comparable to the state-of-the-art noise-optimized NN-based methods while running 3.6-8.6x faster on typical edge platforms. The source code is available at https://github.com/DarthIV02/KalmanHD**

***Index Terms*—Time-Series Forecasting, Hyperdimensional Computing, Kalman Filter**

I. Introduction

The widespread applications of the Internet of Things (IoT) in healthcare [26], transportation [36], and smart cities [2] highlight its importance in modern society. Sensor time series data, where observations are recorded sequentially over time, play a significant role in IoT [34], [6]. Time series forecasting has gained increased attention in IoT research [32]. It involves predicting future values based on historical data [34], offering valuable insights for various domains [34] and enabling resource conservation [12]. Consequently, the need for timely decision-making next to the source of IoT data becomes imperative [8], [22]. In contrast to cloud computing, edge computing brings data processing and analysis closer to the data source. It allows real-time training and inference to be performed on edge devices [9]. Forecasting models have increasingly adopted edge computing to enable timely decision-making, save communication costs, and support operations in remote areas, thereby facilitating scalability [9], [30].

However, edge devices possess limited computational, storage, and energy resources, making accurate forecasting a challenge. Additionally, the time series data streams are often plagued by noise, such as Gaussian noise [28] simulating sensor disruptions, missing samples [19] due to power issues or Poisson noise [28] due to electrical disturbances. These noise factors pose another significant challenge, potentially reducing forecasting accuracy by up to 30% [35] and impacting performance. Forecasting models must effectively leverage time series information from multiple edge sensors, as well as efficiently use on-board resources.

Early research has suggested that statistical models (e.g., BHT-ARIMA [27]) and linear approaches (e.g., support vector regression [3] and random forest [10]) perform well with limited data. Recently, Neural Networks (NN) and deep learning models, such as recurring neural networks (RNN) and long-short-term memory (LSTM) [32], have emerged as popular approaches for forecasting problems due to their strong adaptability and robustness to noise. However, all the above approaches often require multiple iterations (epochs) of the complete data stream to capture the relationships between previous values, which is not adequate for edge settings with streaming data input. Addtionally, NN-based methods require a large volume of data samples and exhibit a resource-intensive, slow training process [12], [37].

Hyperdimensional Computing (HDC) [18] is an emerging computing paradigm that mimics the functioning of human brain. It involves mapping input values to high-dimensional sparse vectors, named *hypervectors*. Training is performed on hypervectors with well-defined and highly parallelizable operations. This makes HDC lightweight to train, robust against hardware errors [11] and conducive to single-pass training under limited observations. All of these benefits align with the challenges that come with edge forecasting applications. While the vast majority of HDC research has focused on classification problems [11], RegHD [14] is the only regression algorithm in HDC domain. RegHD encodes time series data into a binary hypervector which is used to forecast the next value. The encoding and forecasting operations are simple thus save computational costs. However, RegHD requires multiple epochs to converge and is susceptible to *sensor noise*, making it inappropriate for edge scenarios. A new method that is both lightweight and robust to *sensor noise* is needed.

In this paper, we propose a novel, lightweight and robust forecasting method, named KalmanHD, for time series forecasting at the edge. We consider multi-sensor time series data and three common types of *sensor noise*, i.e., Gaussian noise, missing values and Poisson noise. KalmanHD integrates Kalman Filter (KF) with HDC to increase resilience to *sensor noise*, while preserving the lightweight and single-pass properties of HDC training. KF is a commonly used technique to estimate unknown state variables based on observations [17] under the assumption that the incoming samples are subject to Gaussian noise.

The design of KalmanHD faces two primary challenges. First, we need to decide which parts of the process should operate in the hyperdimensional space and which should remain in the original dimensions. Second, we have to adapt operations to the hyperdimensional space while maintaining the original intent of KF. To tackle these challenges, we begin with KF as a base and incrementally incorporate HDC elements, ensuring the learning process remains intact. Specifically, KalmanHD uses the Expectation-Maximization algorithm to train a forecasting weight hypervector, which multiplied by the hypervectors encoded from raw time series data, produces a single value forecast for the next time step.

The main contributions of this paper are as follow:

- We propose KalmanHD as a novel on-device forecasting method specifically designed for time series data at the edge. KalmanHD integrates the power of HDC with Kalman Filter to enable *sensor noise*-resilient prediction while achieving efficient and single-pass training.
- We further enhance the computational efficiency of KalmanHD by approximating the matrix multiplications in Kalman Filter with binary operations. This refinement leads to a notable decrease in computational complexity.
- We perform comprehensive evaluations using five multi-sensor datasets representing practical IoT applications like traffic and electrical monitoring. Our model outperforms the HDC baselines by up to 72% in the presence of sensor noise, highlighting its robustness. Furthermore, our model exhibits 3.6-8.6x faster execution compared to neural-network approaches.

II. Related Works

A. Time Series Forecasting

Time series forecasting is essential for monitoring edge scenarios and making local decisions, such as energy usage and conservation [32]. Initial approaches, like Autoregressive Integrated Moving Average (ARIMA) [4], were widely used for their statistical properties and versatility. ARIMA combines autoregression (AR) and moving average (MA) to model past and future values' relationships. However, ARIMA incurs high computational costs and may not capture relationships among correlated time series. Adaptations like SARIMA [4] and ARIMAX [33] have risen to capture the non-linearities in time series, but long-term predictions remain challenging [29].

An alternative forecasting approach for large panels of related time series involves neural networks (NN) [3], [7], [29]. These networks can simultaneously forecast multiple time series, capturing longer-term relationships within the data. Different architectures like MLPs, CNNs, LSTMs, and other NN approaches have been used in this scope [29]. However, neural networks still face challenges in terms of training complexity and practical applicability [20].

Despite their potential, most of these works focus on cloud-based computing without memory or energy constraints. Only a few studies have specifically addressed forecasting on the edge [9], [30]. However, noise in edge devices is often disregarded or handled using deep networks [21].

B. Robust Forecasting

Recent works have focused on robust NN models to handle Gaussian noise and missing values in time series data. E-Sense [28] uses a Mixture of Experts technique, combining multiple layers of a Convolutional Neural Network (CNN) and a parallel LSTM model to address various types of noise, including Gaussian noise. Another relevant work is PFVAE [16], which employs LSTM as an auto-encoder and variational auto-encoder (VAE) for time series prediction to mitigate the effects of Gaussian noise. However, achieving a high level of robustness necessitates a substantial quantity of training data and utilization of computational resources.

C. Hyperdimensional Computing

Compared to NNs, HDC exhibits superior energy efficiency and faster learning rate, as proven by multiple classification algorithms such as TempHD [24], drive style classification [25] and SemiHD [15], to name a few. We refer the readers to Chang *et al.* [5] for a comprehensive review. It is noteworthy that HDC is not inherently robust against noise in the original data space, which may impact its performance in IoT cases.

RegHD [14] is the only work that combines regression algorithms with HDC to transform the model into a hypervector representation, suitable for handling IoT data with hardware noise. However, it requires multiple iterations through the training data, making it more suitable for offline rather than online training.

III. Background of HDC and Kalman Filters

A. HDC Primitives

Hyperdimensional Computing (HDC) is inspired from the neuroscience community, seeking to emulate human brain functioning [18]. HDC maps raw signal values into high-dimensional vectors, known as *hypervectors*, which often consist of tens of thousands of dimensions. The fundamental idea behind HDC is that intricate patterns within the original space can potentially be linearly separable when projected into a sparse, high-dimensional space [23]. We next describe the three main steps in HDC.

Encoding: Encoding is the first and foremost step to map raw data into hypervectors. In KalmanHD, we adopt Random Projection [31] followed by non-linear operations for encoding, which has achieved state-of-the-art results in time series analysis [14]. Let the raw values to be encoded be represented as $X \in \mathbb{R}^p$. Random Projection is then applied to the input

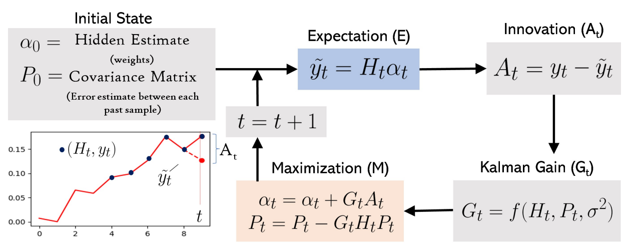
Initial State
$\alpha_0 =$ Hidden Estimate
(weights)
$P_0 =$ Covariance Matrix
(Error estimate between each past sample)
Expectation (E)
$\tilde{y}_t = H_t\alpha_t$
Innovation (A_t)
$A_t = y_t - \tilde{y}_t$
$t = t + 1$
Maximization (M)
$\alpha_t = \alpha_t + G_tA_t$
$P_t = P_t - G_tH_tP_t$
Kalman Gain (G_t)
$G_t = f(H_t, P_t, \sigma^2)$
(H_t, y_t)
$\tilde{y}_t$
A_t
t
0.15
0.10
0.05
0.00
0
2
4
6
8

Fig. 1. The pipeline of Kalman Filter [17].

data, where each value is element-wise multiplied by a random hypervector $h \in \mathbb{R}^d$. Each h_i is chosen from $\mathcal{N}(0,1)$, followed by an L_2 normalization. The result of this process is a set of d-dimensional hypervectors, which are added together to create a single hypervector hv. We then use the non-linear encoding $\cos(hv+b)\cdot\sin(hv)$, where the bias b is chosen from $\mathcal{U}(0,2\pi)$. Finally, we project the hypervector into the bipolar form (each element is from $\{-1,1\}$) using the sign function. Mathematically, the encoding process can be expressed as:

$$hv = \sum_{i}^{p} X_i \cdot h_i, \quad hv = \frac{hv}{\max(\|hv\|_2, \epsilon)} \tag{1}$$

$$hv = \cos(hv + b) \cdot \sin(hv) \tag{2}$$

$$\phi(X) = \text{sign}(hv) \tag{3}$$

Training: During the training phase, representative hypervectors are created for each class. By adding the samples that belong to the same class together, a hypervector closely related to all the samples in that class is formed. Given the hypervectors X^j represent the samples from class j, the class hypervectors are computed as $\phi(c_j) = \bigoplus_i \phi(X_i^j)$.

Inference: During inference, similarities between the query input X_q and all the class hypervectors are calculated. The sample is then assigned to the class with the highest similarity score $\hat{y}$, i.e., $\hat{y} = \text{argmax}_j \cos(\phi(X_q), \phi(c_j))$.

B. Kalman Filters

An effective single-pass approach, designed for limited number of samples and noise, is Kalman Filter (KF) [8]. The objective of KF is to ascertain a hidden state through observations governed by the underlying system. For IoT time series forecasting, the past samples H_t are known, while the coefficients (hidden state α_t) required to predict the future values based on these samples, remain unknown. The accuracy of these coefficients can only be verified once the real next step is obtained, which serves as the observed state. KF utilizes the EM (Expectation-Maximization) algorithm, which iteratively refines the model to approximate the true hidden state until convergence, guided by the observed state [8].

As illustrated in Fig. 1, KF begins with randomly initialized weights denoted as $\alpha_0 \in \mathbb{R}^p$, forming the vector of coefficients necessary for forecasting. Additionally, a covariance matrix P_0 is created to represent the relationships between these weights.

At each time stamp t, the algorithm performs the E step, computing a prediction using the current weights $\tilde{y}_t = H_t\alpha_t$. This prediction estimates the next value in the time series, as observed by the dotted line in the bottom left corner of Fig. 1. The innovation term (A_t) signifies the difference between the predicted value and the actual future value $(y_t - \tilde{y}_t)$.

Next, the Kalman Gain G_t is computed using the formula $f(H_t, P_t, \sigma^2) = \frac{P_tH_t^T}{H_tP_tH_t^T+\sigma^2}$. This calculation involves the previous samples H_t, the covariance matrix P_t, and the variance of the data σ^2. The covariance matrix, captures the uncertainty or variance in the estimates of the hidden state variables and represents the correlations between each pair of values in the sample. By considering the variance, the importance of each sample in the training process is weighted accordingly. During the M step, both the weights α_t and the covariance matrix P_t are updated based on H_t, P_t and G_t to maximize the alignment between the predicted and actual future values [8]. The EM process iterates when new samples come in to refine the forecsting model.

IV. KalmanHD Overview

In this section, we provide a rigorous problem formulation for online IoT forecasting, including the typical noise types as well as model inputs and outputs. We then introduce KalmanHD, our proposed approach that integrates HDC techniques to Kalman Filter for enhanced robustness against sensor noise while addressing energy constraints through binary operations.

A. Problem Definition

IoT scenarios often involve a considerable number of inexpensive sensors that often operate at low sampling frequencies. Consequently, datasets are characterized by multiple time series, each containing a subset of relatively noisy samples [22], [32]. This results in highly noisy data. Building on this observation, we address three major types of noise:

- **Normal Gaussian noise**, represents the standard errors that may arise from sensor disturbances. To simulate this, random values are added from a Gaussian curve with a mean of 0 and standard deviations of $0.1, 0.2, 0.5, 1$.
- **Missing values**, occur when the sensors' power supply fails or there's a low sampling rate. To model this, the time series is partitioned into segments of s samples, each has a certain probability of being missing. If a segment is missing, all of its values are replaced with 0.
- **Poisson Noise**, representative of shot noise or disturbances arising from electrical charges in IoT sensor data, which results in irregular fluctuations in the data. To simulate this, random values are added according to the Poisson distribution with λ of $0.1, 0.2, 0.4, 0.5$.

In our forecasting problem, we consider the input $X_{t:t+p} \in \mathbb{R}^p$ which comprises p consecutive samples from time stamp t. The output is a single-step forecasting $\tilde{y} = \tilde{X}_{t+p+1} \in \mathbb{R}$, that minimizes the discrepancy with the actual future value $y = X_{t+p+1} \in \mathbb{R}$. Training is done in single pass and the model is updated with each new incoming sample. This online setting aligns well with the real-time nature of IoT and the resource constraints imposed by edge computing environments.

B. KalmanHD

KalmanHD is a novel, lightweight and robust approach to train single-step forecasting on sensor devices. This method is

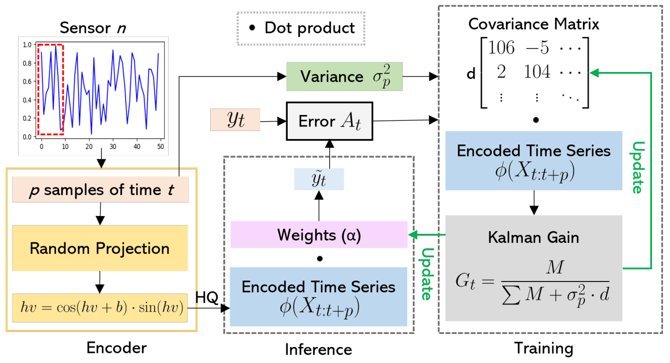
Sensor n
• Dot product
Covariance Matrix
d
$\begin{bmatrix} 106 & -5 & \cdots \\ 2 & 104 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$
Variance σ_p^2
y_t
Error A_t
Encoded Time Series $\phi(X_{t:t+p})$
Update
p samples of time t
$\tilde{y}_t$
Weights (α)
Kalman Gain
$G_t = \frac{M}{\sum M + \sigma_p^2 \cdot d}$
Random Projection
HQ
Encoded Time Series $\phi(X_{t:t+p})$
Update
$hv = \cos(hv + b) \cdot \sin(hv)$
Encoder
Inference
Training

Fig. 2. Step-by-step diagram of KalmanHD.

structured into three phases: encoding, inference, and training, as illustrated in Fig. 2. KalmanHD encodes the past p values $X_{t:t+p}$ from all sensors into a single hypervector $\phi(X_{t:t+p})$. Then KalmanHD adopts the steps of the Kalman Filter to estimate the weights, denoted as $\alpha_t \in \mathbb{R}^d$, by which the encoded past values are multiplied. This process allows the hypervector to be transformed back to the original space, thereby generating the final prediction $\tilde{y}_t = \tilde{X}_{t+p+1}$ during inference. Estimating α_t from noisy observations, the method iteratively updates Kalman Gain G_t and covariance matrix P_t using incoming samples and prediction errors, thereby merging KF's noise resilience with HDC's computational efficiency.

In contrast to RegHD [14], KalmanHD enhances the forecasting model's robustness by considering the variance in its computations. Compared to the traditional Kalman Filter [8], KalmanHD leverages the expressive power and robustness of HDC, thereby further enhancing its noise resilience.

We explain the details of each phase in the following lines:

(a) Encoding. Given the time series readings from multiple sensors, the encoding step in KalmanHD maps the time series from one sensor, $X_{t:t+p}$, into a single hypervector, $\phi(X_{t:t+p})$, using the nonlinear encoding method introduced in Section III. KalmanHD encodes the time series from different sensors separately, while training a global model (the weight hypervector α_t) for forecasting.

(b) Inference. The inference phase predicts the next-step reading $\tilde{y}_t$ based on the p past values of a specific sensor n. In KalmanHD, this is achieved via multiplying the encoded hypervector $\phi(X_{t:t+p})$ by the global model α_t using the dot product, i.e., $\tilde{y}_t = \phi(X_{t:t+p}) \cdot \alpha_t$.

(c) Training. The training iteratively updates the weight vector $\alpha_t \in \mathbb{R}^d$ and the covariance matrix P_t. P_t captures the uncertainty or variance in the estimates of the weights and represents the correlations between each pair of values in the sample $X_{t:t+p}$. When starting the model, α_0 is initialized as zeros while P_0 is filled with ones.

For each incoming sample, the Kalman Gain G_t is computed similar to $f(H_t, P_t, \sigma^2)$ in Section III-B, with the difference being that H_t is replaced by the encoded input samples $\phi(X_{t:t+p})$. As a result, the numerator results in a d-vector, similar to the process in KF. However, the major distinction lies in the calculation of $H_t P_t H_t^T$, which is substituted by $\sum P_t \phi(X_{t:t+p})$. We further optimize the efficiency of KalmanHD by quantizing the matrix multiplication, referred as HQ. This will be elaborated upon in greater detail in Section IV-C. When computing G_t, we scale the variance σ_p^2 by d to ensure it holds significance. σ_p^2 manages the importance and impact of individual samples in the training process and contributing to the overall robustness of the method.

Then, we update both α_t and P_t. The model weights α_t are updated similar to KF ($\alpha_t = \alpha_t + G_t A_t$), but with the inclusion of a learning rate η to accommodate the high dimensionality of the samples. The covariance matrix P_t is updated by multiplying M with the transposed input sample, resulting in a matrix that represents the relationship between each pair of dimensions.

The complete process can be summarized as follows:

$$M = HQ\left(P \cdot \phi(X_{t:t+p})^T\right) \tag{4}$$

$$G_t = \frac{M}{\sum M + \sigma_p^2 \cdot d} \tag{5}$$

$$\alpha_t = \alpha_t + G_t A_t \eta \tag{6}$$

$$P_t = P_t + HQ\left(G_t\right) \otimes M^T \tag{7}$$

Given that the variance is not known beforehand, we infer the variance σ_p^2 of the time series based on the previous and current samples. To tackle this issue, the variance is stored and updated with each new sample according to the formula: $\sigma_p^2 = (\gamma \cdot \sigma_p^2) + (1-\gamma) \cdot \sigma^2(x)$. Here, γ is a coefficient that determines the balance between the importance of the past variance and the new variance from the incoming samples.

C. Optimizing the Efficiency of KalmanHD

In the initial stages of KalmanHD, the training process closely followed the Kalman Filter methodology. However, this approach involved three significant matrix multiplications: two for G_t (in both the numerator and denominator) and another for updating P_t in Fig. 1, which posed computational challenges due to the large size of the matrix ($d \times d$).

To address this issue, we utilize the binary representation of M in Equation (4). Binarizing matrix multiplication simplifies runtime operations at the cost of some accuracy (due to the loss of precision in numbers), especially with a constant dimensionality d. Additionally, we've streamlined P_t updates with binary operations, significantly reducing computational complexity. Specifically, the multiplication is now reduced to XOR operations ($\otimes$). The updated P_t now stores integers, as its updated values are in the range of $\{-1, 1\}$. This optimization drastically improves the efficiency of the KalmanHD training process. To further enhance the efficiency of KalmanHD, we make sure to only train the model when the error A_t is bigger than a threshold (in this scenario $A_t > 0.1$).

V. Evaluation

A. Experimental Setup

Datasets. The datasets, presented in [32], represent typical IoT data with multiple time series from various sensors and limited samples. These datasets encompass two primary

TABLE I
DATASET SETUP.

Dataset	Type	Frequence	Time Series	Time Serie Samples
ECF	Energy	Daily	314	365
SFT	Traffic	Weekly	862	104
MITV	Traffic	Hourly	1	33728
GT	Traffic	Hourly	206	1464
ELD	Energy	Daily	320	1096

TABLE II
HYPERPARAMETER CONFIGURATIONS IN KALMANHD.

Dataset	η	d	γ	p
Energy Consumption Fraunhoufer (ECF)	0.001	500	0.03	20
San Francisco Traffic (SFT)	0.001			
Metro Interstate Traffic Volume (MITV)	0.00001			
Guangzhao Traffic (GT)	0.001			
Electricity Load Diagrams (ELD)	0.0001			

categories: energy consumption and traffic forecasting, each featuring different time resolutions. They are chosen due to the resource constraints of the data-collecting devices, emphasizing the importance of timely forecasting. For more detailed dataset information, please refer to Table I. We partition the samples in a linear fashion, assigning 70% for training, 20% for testing, and 10% for cross-validation for each time series. The input samples are chosen using a sliding window approach with a step size of 1, employing a time window length of p = 20. This sliding window technique, involves utilizing the initial p values of the time series for initial training input and subsequently shifting the training window forward by one step to obtain successive training inputs.

Implementation details. We implement KalmanHD in Python 3.9 using the PyTorch and the TorchHD libraries [13]. All hyperparameters described in the design were carefully selected through validation tests to optimize the model's accuracy while ensuring efficiency including the non-linear encoder. The key hyperparameters are listed in Table II.

- **E-Sense [28]** combines multiple layers of CNN and a parallel LSTM model to address various types of noise, mainly for Gaussian noise. We adapt the open-source implementation of this model.
- **PFVAE [16]** employs LSTM as an auto-encoder and variational auto-encoder (VAE) for time series prediction to mitigate the effects of Gaussian noise. As the code for PFVAE is not publicly available, we manually implemented PFVAE without incorporating the planar flow section for our experiments.
- **RegHD [14].** The first HDC-based regression method suitable for handling IoT data with hardware noise. We adapt the open-source implementation of RegHD with a default learning rate of $1e-6$.

The evaluation of each test sample uses the mean absolute difference (MAE) between the predicted and actual values. For efficiency experiments, we use two representative edge devices including a Raspberry Pi 4B [1] with 4GB RAM, an edge desktop with Intel Core i7-8700 CPU at 3.2 GHz and 16 GB RAM. We measure the execution time on these devices, which linearly reflects the amount of energy consumed while performing various methods.

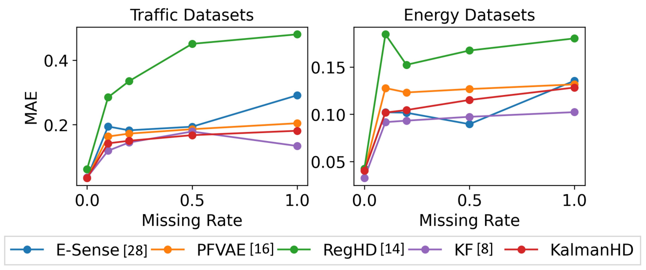

Fig. 3. Average MAE with missing samples on all datasets.

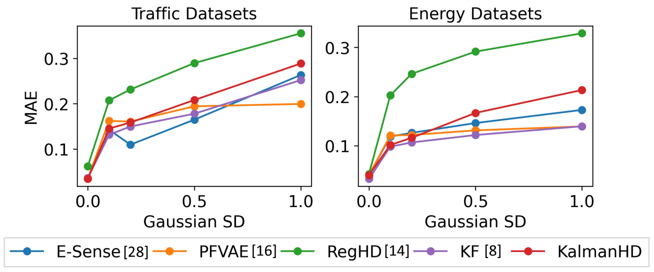

Fig. 4. Average MAE with Gaussian noise on all datasets.

B. MAE / Robustness Results

Fig. 3, Fig. 4 and Fig. 5 present the average MAE of all methods across various missing sample ratios, standard deviation (SD) of Gaussian noise and Poisson noise distributions. Each figure features two plots: the left plot summarizes the average MAE across all traffic datasets, while the right plot focuses on the energy datasets. A lower MAE means a more accurate forecast of actual values. Across all datasets and noise levels, KalmanHD achieves accuracy on par with robust NN benchmarks, specifically E-Sense [28] and PFVAE [16]. RegHD [14] experiences considerable performance deterioration when confronted with sensor noise of any kind. This outcome is anticipated, given that RegHD is not designed to withstand *sensor noises*, despite HDC's inherent resilience to hardware errors due to its high dimensionality. The raw KF [8] exhibits strong performance with Gaussian Noise and missing samples which aligns with its Gaussian noise assumption. However, when applied to other types of noise (like Poisson noise), KF underperforms significantly.Our approach KalmanHD demonstrates the potential to attain robustness in forecasting models utilizing HDC and KF, for Gaussian noise, Poisson noise and missing values. In the cases of extreme noise, KalmanHD surpasses RegHD's MAE performance by up to 72%.

C. Efficiency Results

Fig. 6 illustrates the average execution times of each baseline and our model to finish one pass on the traffic and energy datasets. Benefiting from HDC's efficiency, KalmanHD is 5.0x and 3.6x faster than E-Sense and PFVAE on Raspberry Pi. On the edge desktop, KalmanHD presents 8.6x and 7.1x faster execution speed than E-Sense and PFVAE. RegHD exhibits faster computations, but its forecasting errors quickly accumulate when faced with noise. While KalmanHD adds more sophisticated computation inspired from Kalman Filter,

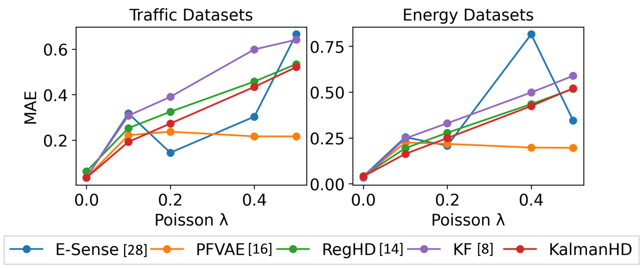

Fig. 5. Average MAE with Poisson noise on all datasets.

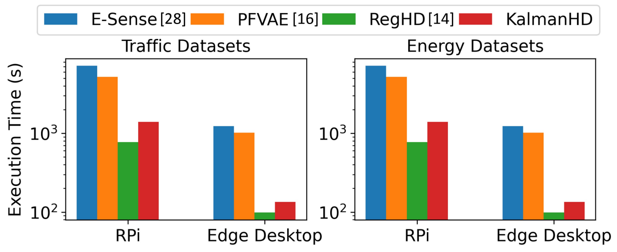

Fig. 6. Execution time results on Raspberry Pi (RPi) and edge desktop.

KalmanHD only takes up to 48% more time than RegHD on the edge desktop. For all datasets, KalmanHD consumes less than 35 and 4 minutes to complete one pass of the whole dataset on Raspberry Pi and edge desktop, respectively.

VI. Conclusion

In IoT forecasting, two primary challenges arise: limited energy resources necessitating efficient models, and noise introduced by sensors due to resource constraints. HDC is a promising area that focuses on efficient computing, offering hardware noise resistance but lacking robustness to sensor noise. We propose a novel single-step forecasting approach called KalmanHD, which integrates HDC with the Kalman Filter. Through extensive experiments on real IoT datasets with varying levels of Gaussian noise, missing samples and Poisson noise KalmanHD achieves comparable accuracy results to robust deep networks in online settings while demonstrating 3.6-8.6x speedups on typical edge platforms.

Acknowledgment

This work was supported in part by National Science Foundation under Grants #2003279, #1826967, #2100237, #2112167, #1911095, #2112665, and in part by SRC under task #3021.001. This work was also supported in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA.

References

[1] Raspberry Pi 4B. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, 2023. [Online].
[2] Hamidreza Arasteh and et al. Iot-based smart cities: A survey. In *EEEIC '23*, pages 1–6. IEEE, 2016.
[3] Yukun Bao and et al. Multi-step-ahead time series prediction using multiple-output support vector regression. *Neurocomputing*, 129:482–493, 2014.
[4] George EP Box and et al. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
[5] Cheng-Yang Chang and et al. Recent progress and development of hyperdimensional computing (hdc) for edge intelligence. *IEEE J. Emerg. Sel.*, 2023.
[6] Chris Chatfield. *Time-series forecasting*. Chapman and Hall/CRC, 2000.
[7] Ling Chen and Xu Lai. Comparison between arima and ann models used in short-term wind speed forecasting. In *APPEEC '11*. IEEE, 2011.
[8] Xi andet al. Chen. Autoregressive-model-based methods for online time series prediction with missing values: an experimental evaluation. *arXiv preprint arXiv:1908.06729*, 2019.
[9] Gaia Codeluppi and et al. Forecasting air temperature on edge devices with embedded ai. *Sensors*, 21(12):3973, 2021.
[10] Grzegorz Dudek. Short-term load forecasting using random forests. In *IS '2014*, pages 821–828. Springer, 2015.
[11] Lulu Ge and et al. Classification using hyperdimensional computing: A review. *IEEE CAS Magazine*, 20(2):30–47, 2020.
[12] Nastaran Gholizadeh and et al. Federated learning with hyperparameter-based clustering for electrical load forecasting. *Internet of Things*, 17:100470, 2022.
[13] Mike Heddes and et al. Torchhd: An open-source python library to support hyperdimensional computing research. *arXiv preprint arXiv:2205.09208*, 2022.
[14] Alejandro Hernández-Cano and et al. Reghd: Robust and efficient regression in hyper-dimensional learning system. In *DAC '21*. IEEE, 2021.
[15] Mohsen Imani and et al. Semihd: Semi-supervised learning using hyperdimensional computing. In *ICCAD '19*, pages 1–8. IEEE, 2019.
[16] Xue-Bo Jin and et al. Pfvae: a planar flow-based variational auto-encoder prediction model for time series data. *Mathematics*, 10(4):610, 2022.
[17] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.
[18] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1:139–159, 2009.
[19] Raghavendra Kumar and et. al. Time series data prediction using iot and machine learning technique. *Procedia computer science*, 167:373–381, 2020.
[20] Bryan Lim and et al. Time-series forecasting with deep learning: a survey. *Philos. Trans. Royal Soc. A*, 379(2194):20200209, 2021.
[21] Lingling Lv and et al. An edge-ai based forecasting approach for improving smart microgrid efficiency. *IEEE Trans Industr Inform*, 18(11):7946–7954, 2022.
[22] Amer Malki and et al. Machine learning approach of detecting anomalies and forecasting time-series of iot devices. *AEJ*, 61(11):8973–8986, 2022.
[23] Ali Moin and et al. A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition. *Nature Electronics*, 4(1):54–63, 2021.
[24] Yang Ni and et al. Neurally-inspired hyperdimensional classification for efficient and robust biosignal processing. In *ICCAD '22*, 2022.
[25] Kenny Schlegel and et al. Multivariate time series analysis for driving style classification using neural networks and hyperdimensional computing. In *IV '21*, pages 602–609. IEEE, 2021.
[26] Sureshkumar Selvaraj and et al. Challenges and opportunities in iot healthcare systems: a systematic review. *SN Applied Sciences*, 2(1):139, 2020.
[27] Qiquan Shi and et al. Block hankel tensor arima for multiple short time series forecasting. In *AAAI*, volume 34, pages 5758–5766, 2020.
[28] Sudipta Saha Shubha and et al. A diverse noise-resilient dnn ensemble model on edge devices for time-series data. In *SECON '21*, pages 1–9. IEEE, 2021.
[29] Sima Siami-Namini and et al. The performance of lstm and bilstm in forecasting time series. In *Big Data '19*, pages 3285–3292. IEEE, 2019.
[30] Afaf Taïk and et al. Electrical load forecasting using edge computing and federated learning. In *ICC '20*, pages 1–6. IEEE, 2020.
[31] Anthony Thomas and et al. A theoretical perspective on hyperdimensional computing. *JAIR*, 72:215–249, 2021.
[32] Christos Tzagkarakis and et al. Evaluating short-term forecasting of multiple time series in iot environments. In *EUSIPCO '22*. IEEE, 2022.
[33] Billy M Williams. Multivariate vehicular traffic flow prediction: evaluation of arimax modeling. *Transp. Res. Rec.*, 1776(1):194–200, 2001.
[34] Samir Yerpude and et al. Impact of internet of things (iot) data on demand forecasting. *INDJST*, 10(15):1–5, 2017.
[35] Lean Yu and et al. A compressed sensing based ai learning paradigm for crude oil price forecasting. *Energy Economics*, 46:236–245, 2014.
[36] Fotios Zantalis and et al. A review of machine learning and iot in smart transportation. *Future Internet*, 11(4):94, 2019.
[37] Sizhe Zhang and et al. Scalehd: Robust brain-inspired hyperdimensional computing via adapative scaling. In *ICCAD '22*, pages 1–9, 2022.