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

**Ivannia Gomez Moreno***, Xiaofan Yu**, Tajana Rosing**
* CETYS University, ** University of California San Diego

System Energy Efficiency Lab
seelab.ucsd.edu

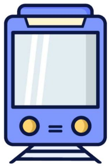
# IoT Forecasting

IoT has widespread applications including:

Healthcare

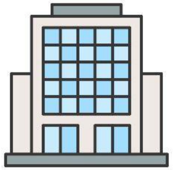
[SN Applied Sciences ‘22]

Transportation



[Future Internet ‘19]

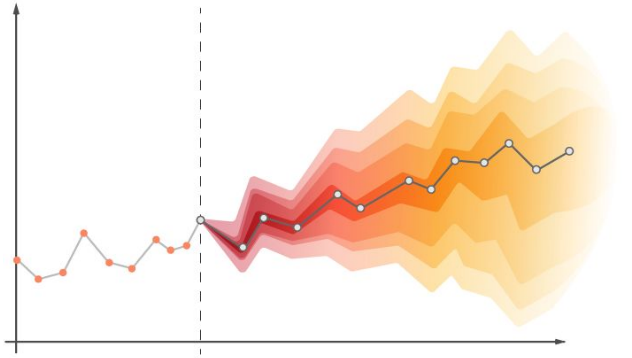
Smart Cities



[EEEIC ‘16]

Data obtained in IoT are **time series** obtained from sensors:

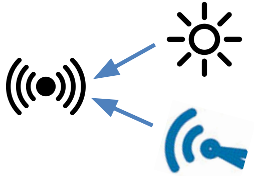
- **Forecasting**: Predicting future values based on historical data
- **Challenges**: Noise from sensor due to electrical disruptions or power issues



# Problem Definition

## Types of Noise

**Gaussian noise**
from sensor disturbances



- Random values from Gaussian curve
- Mean: 0
- Standard deviation: (0.1,1)

[SECON ‘21]

**Missing values**
from power supply failures or infrequent sampling

- Dataset is partitioned in segments
- If segment is missing then replaced with 0s

[ICCIDS ‘19]

**Poisson noise**
from electrical charge disturbances

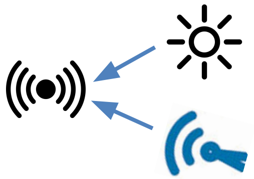
- Random values from Poisson distribution
- λ: (0.1, 0.5)

[SECON ‘21]

# Problem Definition

## Types of Noise

**Gaussian noise**
from sensor disturbances



- Random values from Gaussian curve
- Mean: 0
- Standard deviation: (0.1,1)

[SECON ‘21]

**Missing values**
from power supply failures or infrequent sampling

- Dataset is partitioned in segments
- If segment is missing then replaced with 0s

[ICCIDS ‘19]

**Poisson noise**
from electrical charge disturbances

- Random values from Poisson distribution
- $\lambda$: (0.1, 0.5)

[SECON ‘21]

- IoT involves many inexpensive sensors at low sampling frequencies
- Datasets have multiple times series: 1 from each sensor
- **Input of regression**: *p* consecutive samples
- **Output of regression**: single-step forecasting
- Training is single pass (online training)

# Edge Computing

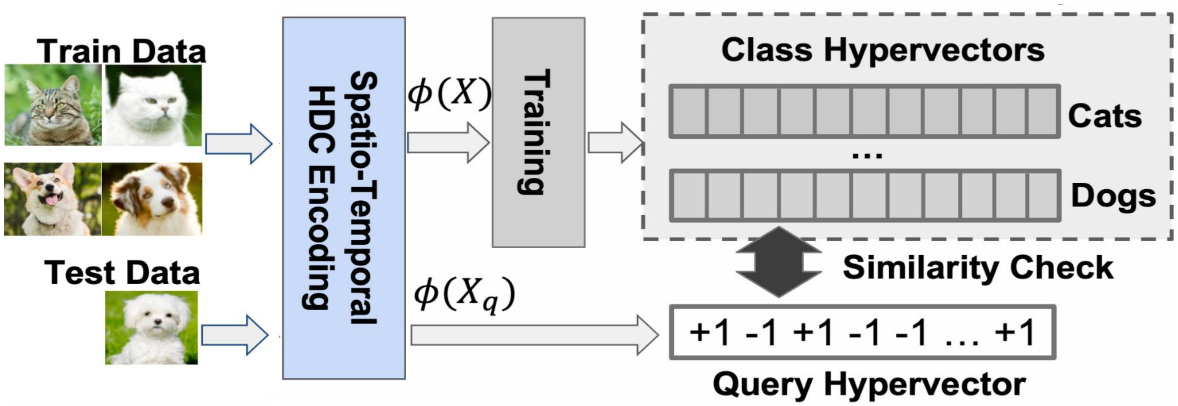
- Edge computing brings real-time training and inference performed at edge devices
- **Benefits**:
  - Timely decision-making
  - Saving communications costs
  - Supporting operation in remote areas
- **Drawback:**
  - Limited computational storage and energy resources

# Current Approaches

- **Statistical Models** (ARIMA [AAAI ‘20]) and **Linear approaches** (SVR [Neurocomputing ‘14] and RF [IS ‘14])
  - Good for limited samples
  - Require multiple iterations, which is not adequate for streaming input in edge settings
- **Kalman Filter**
  - Good for limited samples
  - Robust to only Gaussian noise
  - Computational complexity increases as the number of previous values increases
- **Neural Networks**:
  - Designed to be robust to noise and adaptable
  - Require large volumes of data, which leads to resource-intensive and slow training process
  - Novel models:
    - E-Sense [SECON ‘21]: Mixture of Experts techniques, combining CNN + LSTM
    - PFVAE [Mathematics ‘22]: LSTM as auto-encoder + Variational auto-encoder

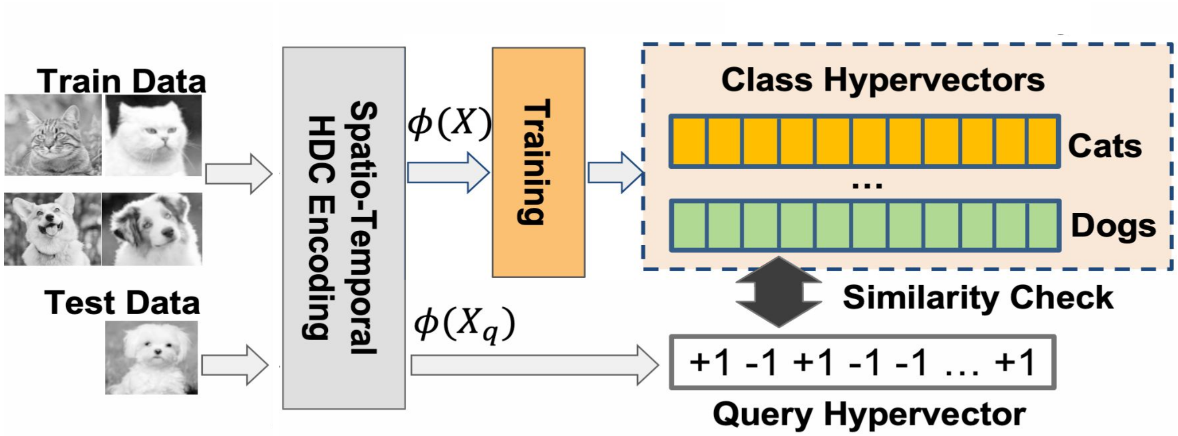
# Hyperdimensional Computing (HDC)

- Superior **energy efficiency and smaller training time** than Neural Networks (NNs)
- Three main steps:
  1) **Encoding**: Mapping the data to HD space



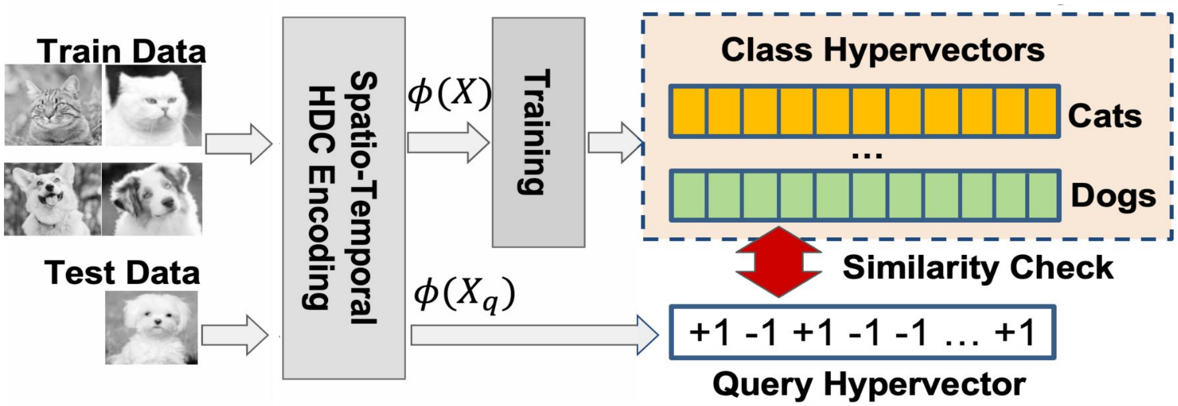
# Hyperdimensional Computing (HDC)

- Superior **energy efficiency and smaller training time** than Neural Networks (NNs)
- Three main steps:
  1) **Encoding**: Mapping the data to HD space
  2) **Training**: Create class hypervectors representative of each class



# Hyperdimensional Computing (HDC)

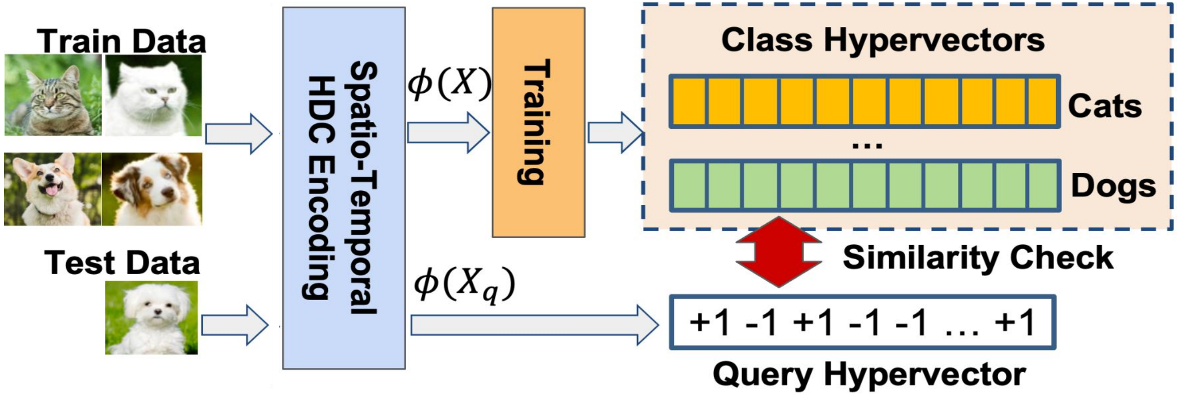
- Superior **energy efficiency and smaller training time** than Neural Networks (NNs)
- Three main steps:
  1) **Encoding**: Mapping the data to HD space
  2) **Training**: Create class hypervectors representative of each class
  3) **Inference**: Assigned to the class with the highest similarity to query hypervector



# Hyperdimensional Computing (HDC)

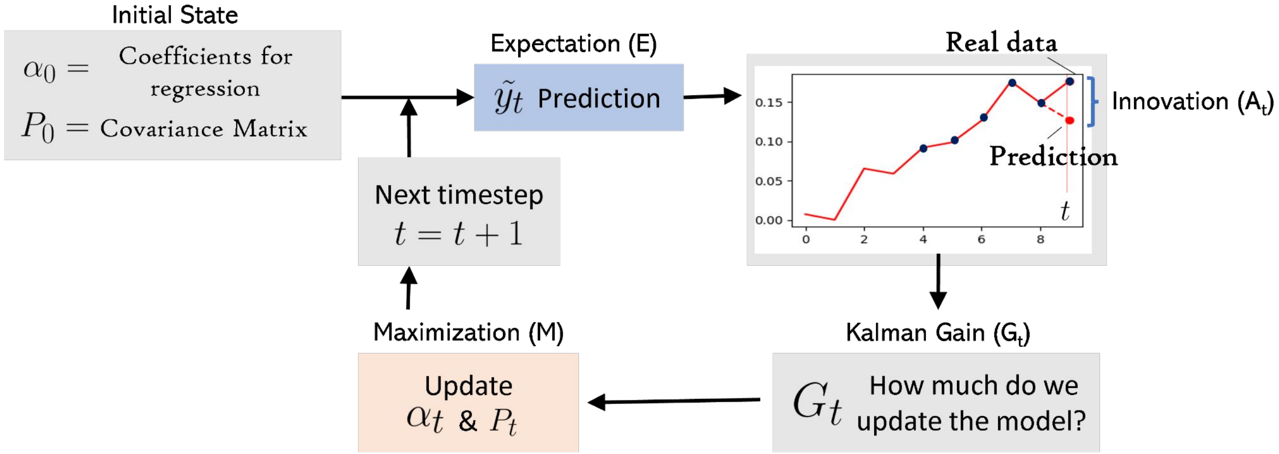
- Superior **energy efficiency and smaller training time** than Neural Networks (NNs)
- Three main steps:
  1) **Encoding**: Mapping the data to HD space
  2) **Training**: Create class hypervectors representative of each class
  3) **Inference**: Assigned to the class with the highest similarity to query hypervector

HDC is not robust against noise in the original data space



# Kalman Filter (KF)

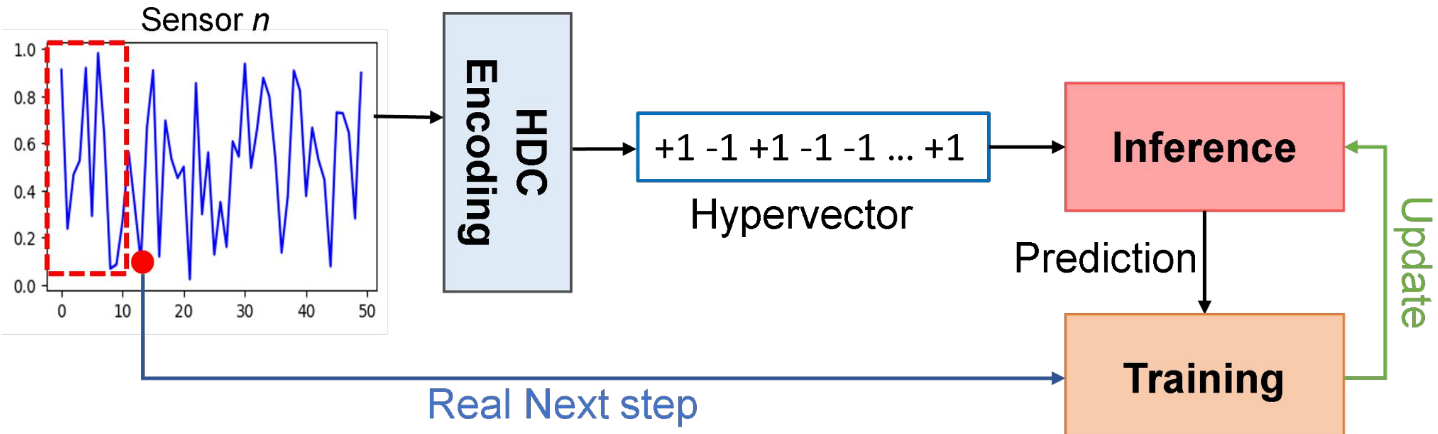
- KF is useful in forecasting when there is **limited number of samples** & **Gaussian noise** within the time series




- **Objective**: Find a **hidden state** (coefficients for the regression) through a different **observed state** (past samples and next-step)
- Considers variance of the samples to determine the importance to training

# Our Contribution: KalmanHD

- Novel, lightweight and robust **forecasting** method for time series
- Integrates:
  - **Kalman Filter** (KF) to increase resilience to noise
  - The lightweightness and single-pass properties of **HDC**



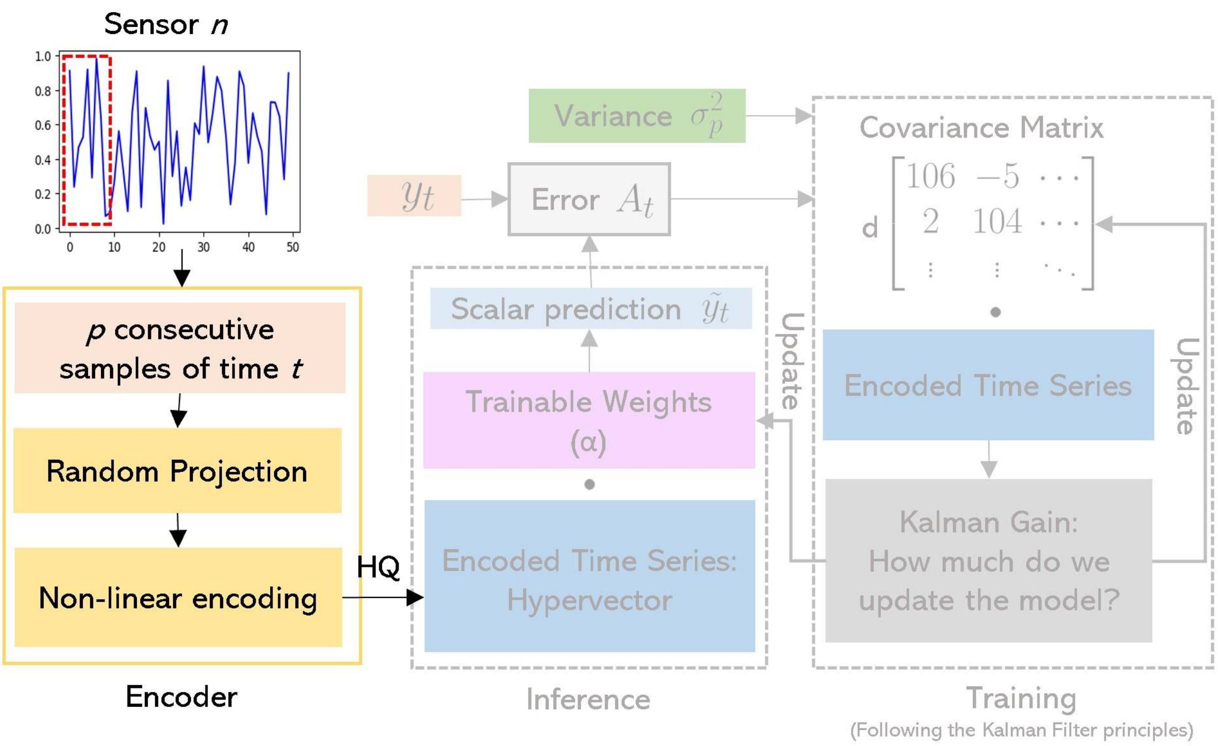
# KalmanHD - Encoding

## Encoding:

- **Input**: past *p* samples
- **Output**: 1 hypervector representative of the samples
- **HQ** is the binarization of the resultant vector

Random Projection:

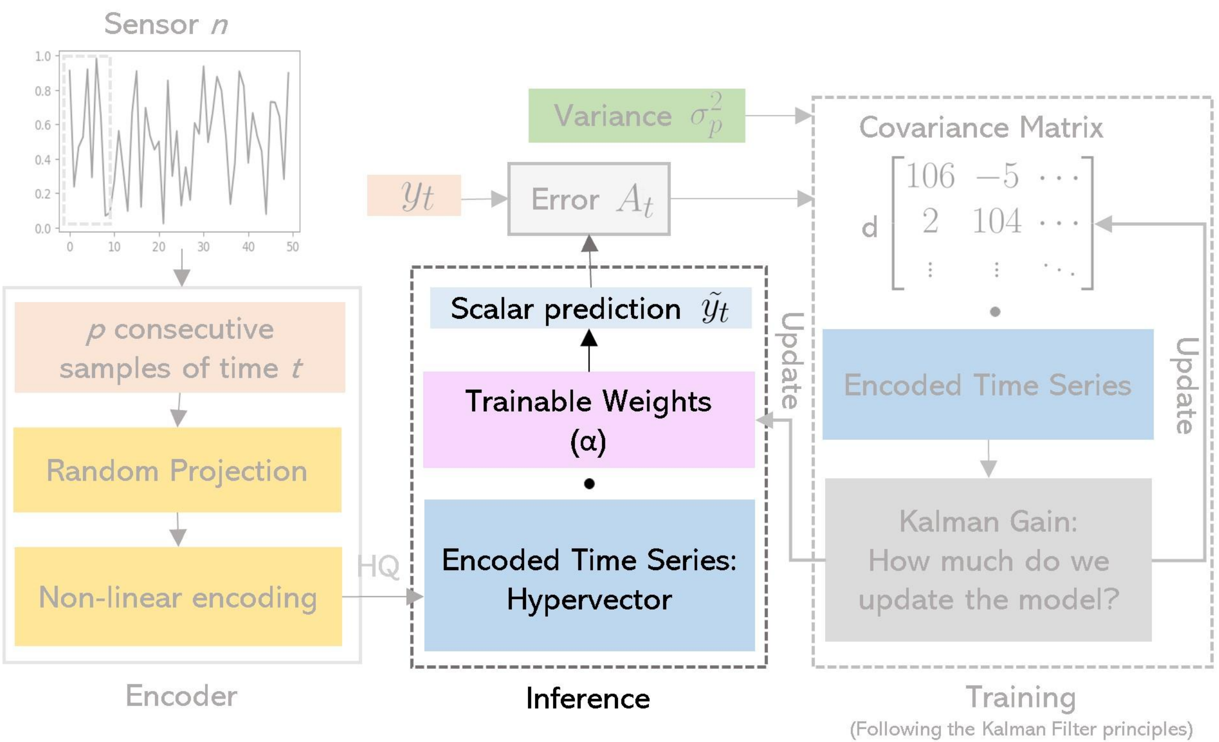
$$hv = \sum_{i}^{p} X_i \cdot h_i$$



# KalmanHD - Inference

## Inference:

- **Input**: hypervector
- **Output**: Next step prediction
- **α** is a d-dimensional vector of coefficients that changes with each sample for a more accurate prediction

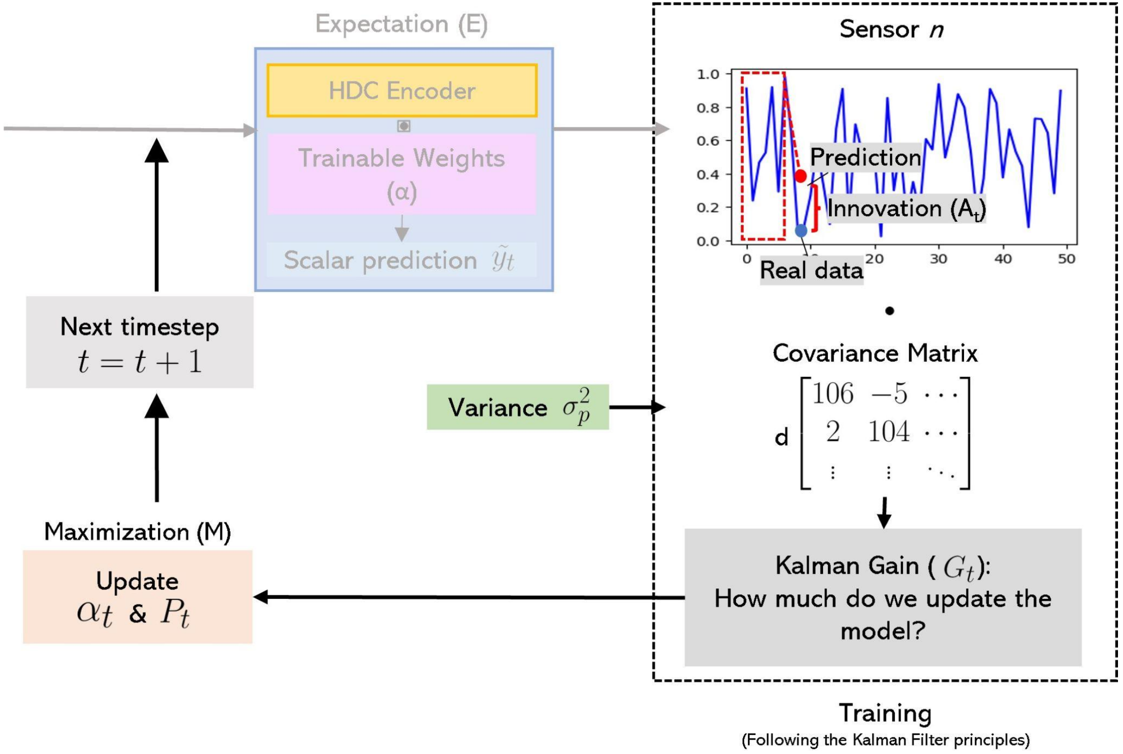


# KalmanHD - Training

## Training:

- **Input**: Error and variance
- **Output**: Kalman Gain
- Updates the model (coefficients and covariance matrix) iteratively
- **Variance**: Inferred based on the previous and current samples:

$$\sigma_p^2 = (\gamma \cdot \sigma_p^2) + (1 - \gamma) \cdot \sigma^2(x)$$

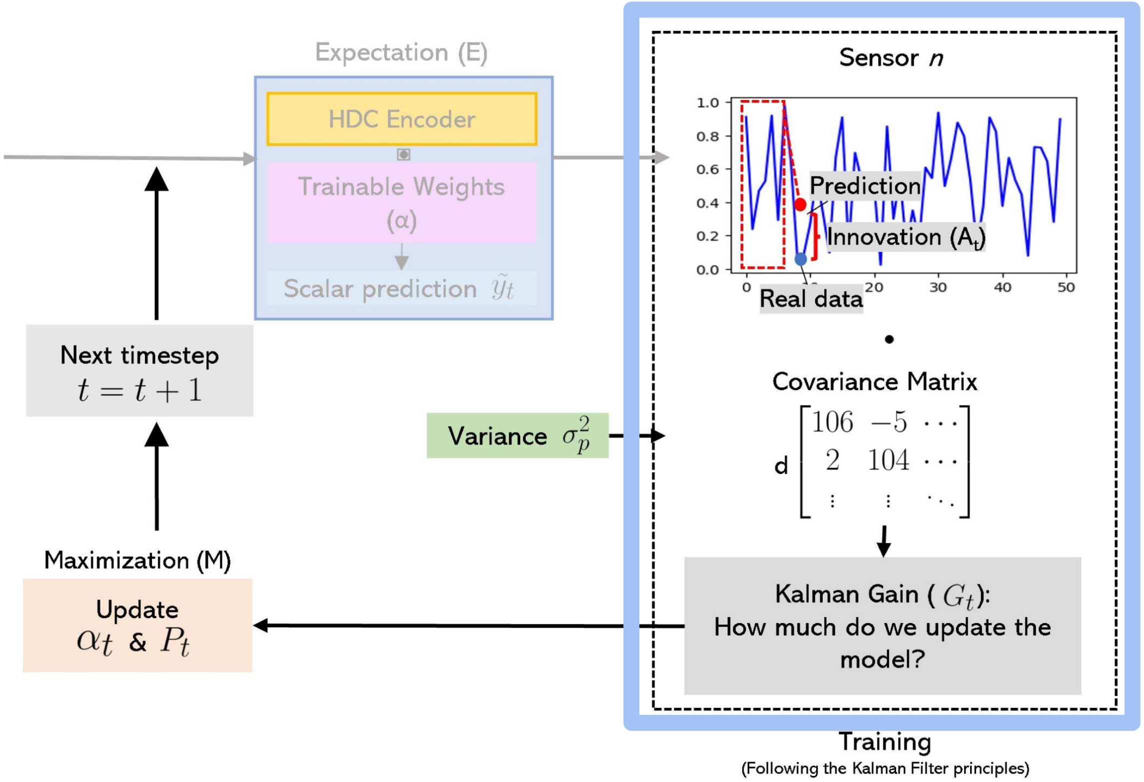


# KalmanHD - Training

## Training:

- **Input**: Error and variance
- **Output**: Kalman Gain
- Updates the model (coefficients and covariance matrix) iteratively
- **Variance**: Inferred based on the previous and current samples:

$$\sigma_p^2 = (\gamma \cdot \sigma_p^2) + (1 - \gamma) \cdot \sigma^2(x)$$

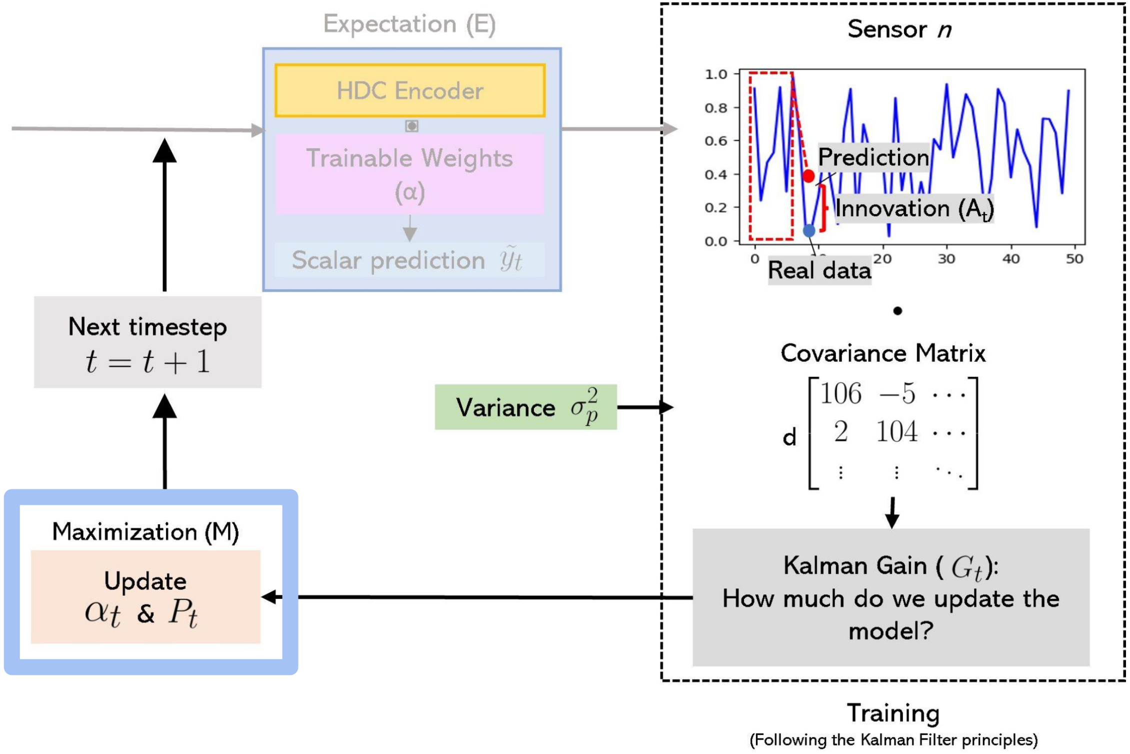


# KalmanHD - Training

## Training:

- **Input**: Error and variance
- **Output**: Kalman Gain
- Updates the model (coefficients and covariance matrix) iteratively
- **Variance**: Inferred based on the previous and current samples:

$$\sigma_p^2 = (\gamma \cdot \sigma_p^2) + (1-\gamma) \cdot \sigma^2(x)$$

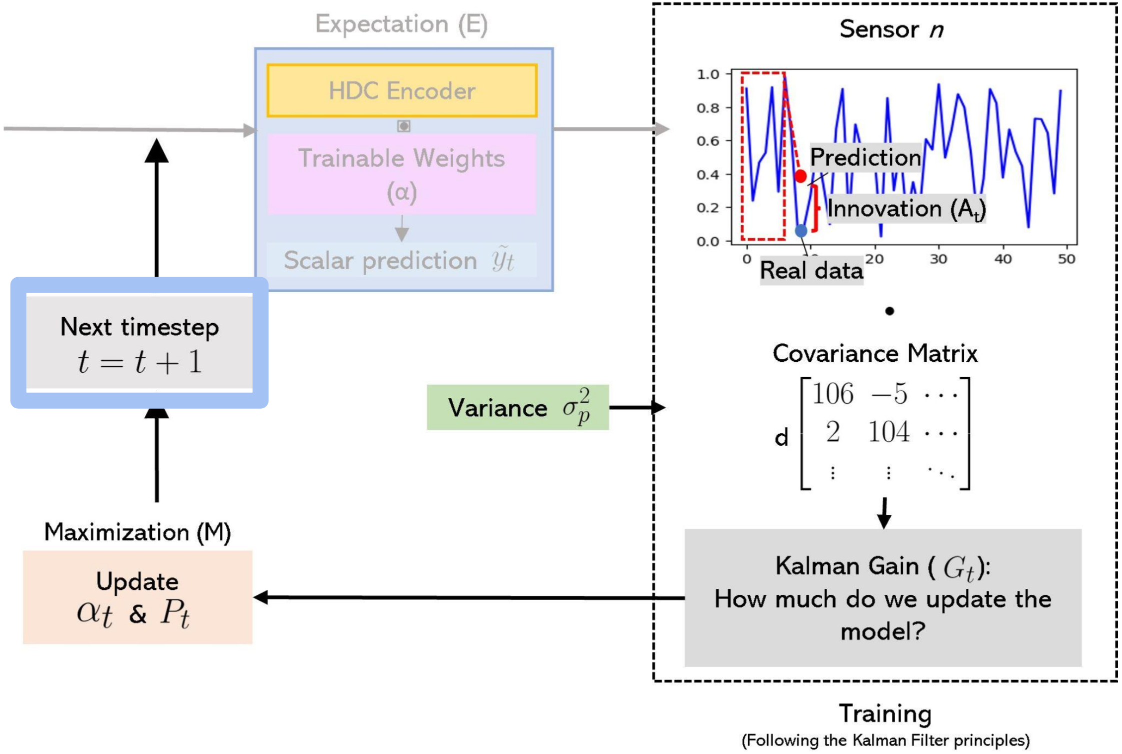


# KalmanHD - Training

## Training:

- **Input**: Error and variance
- **Output**: Kalman Gain
- Updates the model (coefficients and covariance matrix) iteratively
- **Variance**: Inferred based on the previous and current samples:

$$\sigma_p^2 = (\gamma \cdot \sigma_p^2) + (1-\gamma) \cdot \sigma^2(x)$$

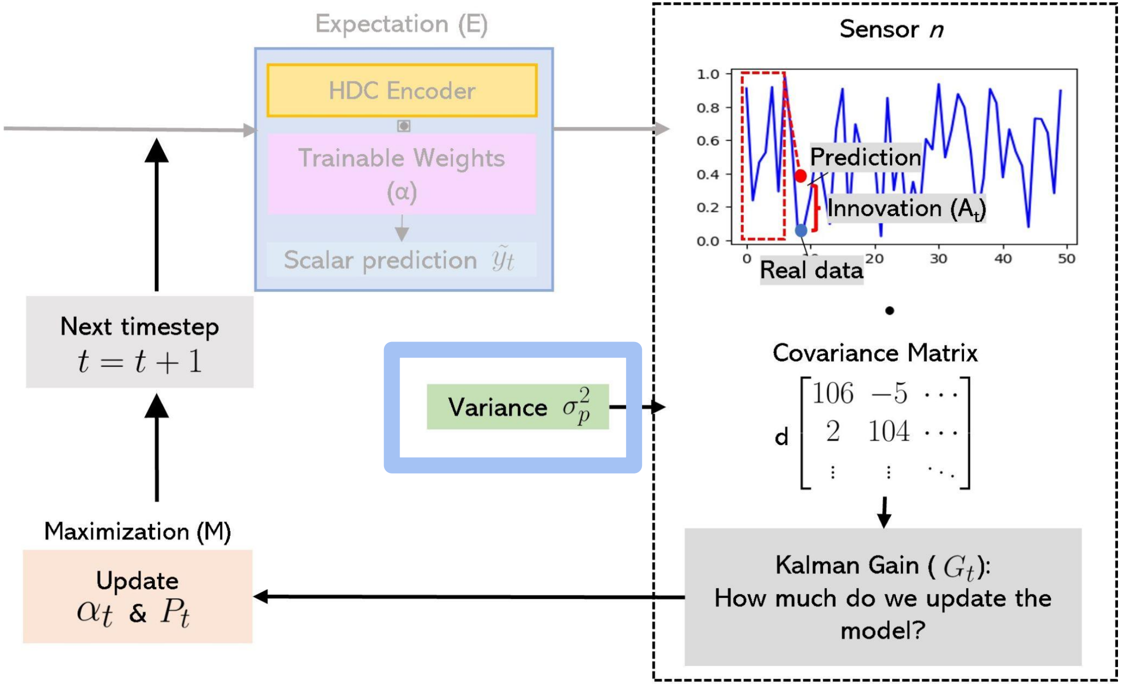


# KalmanHD - Training

## Training:

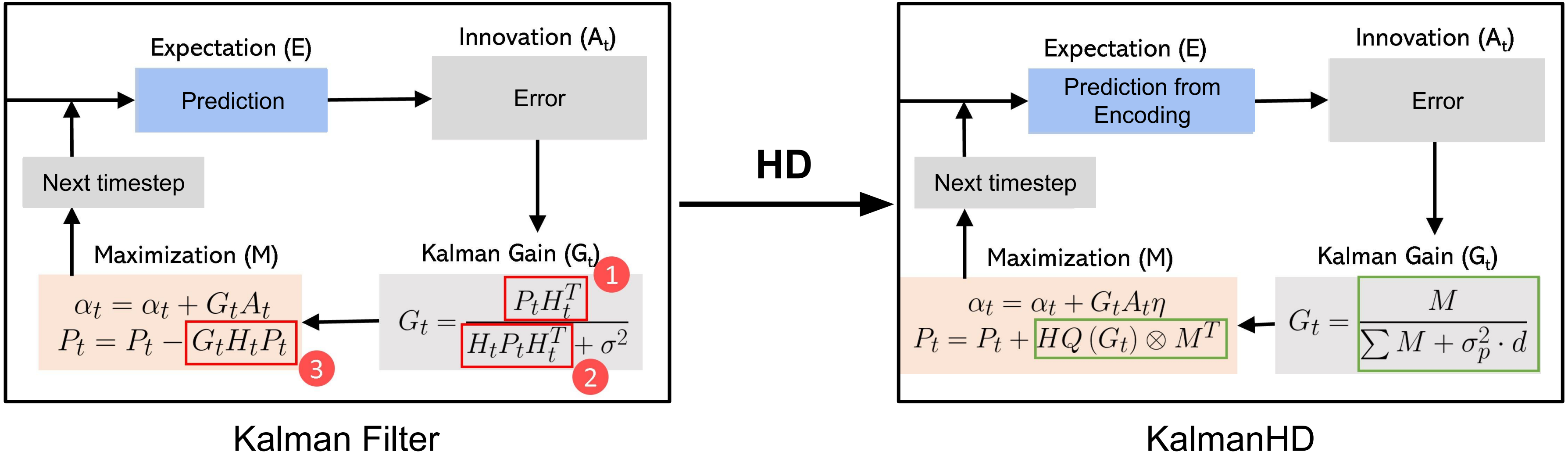
- **Input**: Error and variance
- **Output**: Kalman Gain
- Updates the model (coefficients and covariance matrix) iteratively
- **Variance**: Inferred based on the previous and current samples:

$$\sigma_p^2 = (\gamma \cdot \sigma_p^2) + (1 - \gamma) \cdot \sigma^2(x)$$




Training
(Following the Kalman Filter principles)

# KalmanHD - Reducing Computational Complexity




- Propose M as a **binary** hypervector to reuse operations
- Replacing matrix multiplications with binary operations only decreases accuracy **2.5%**

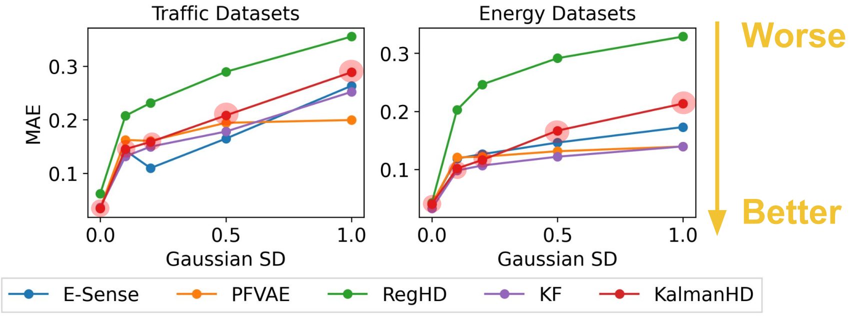
# Experimental Setup

- **Datasets**: Typical IoT data with multiple time series from various sensors and short amount of samples each [EUSIPCO '22].
- **Implementation**: PyTorch and TorchHD
- **Baselines**: E-Sense [SECON '21], PFVAE [Mathematics '22], RegHD [DAC '21], Online Kalman Filter [ARXIV '19].
- **Metric**: Mean Absolute Error (MAE), Execution time (seconds)
- **Devices**:
  - Raspberry Pi 4B with 4GB RAM
  - Intel Core i7-8700 CPU at 3.2 GHz

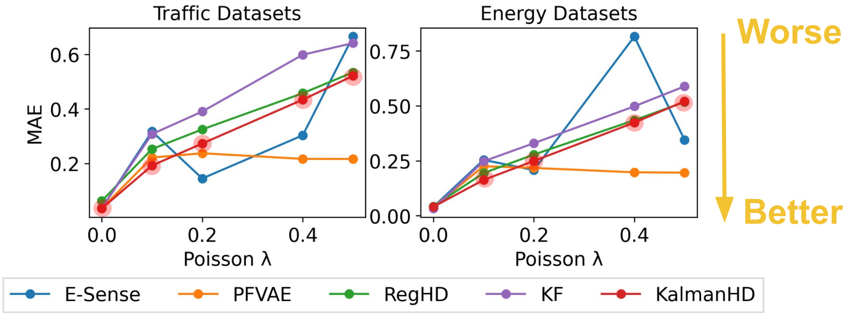
| Dataset | Type | Frequency | Time Series | Time Series Samples |
|---|---|---|---|---|
| Energy Consumption Fraunhoufer | Energy | Daily | 314 | 365 |
| San Francisco Traffic | Traffic | Weekly | 862 | 104 |
| Metro Interstate Traffic Volume | Traffic | Hourly | 1 | 33728 |
| Guangzhou Traffic | Traffic | Hourly | 206 | 1464 |
| Electricity Load Diagrams | Energy | Daily | 320 | 1096 |

# Robustness Results

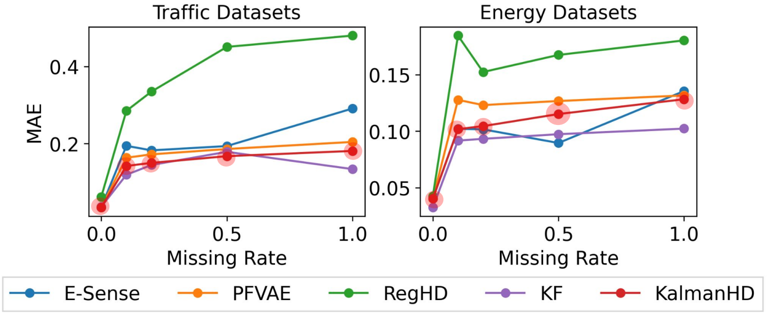
## Gaussian Noise




## Poisson Noise

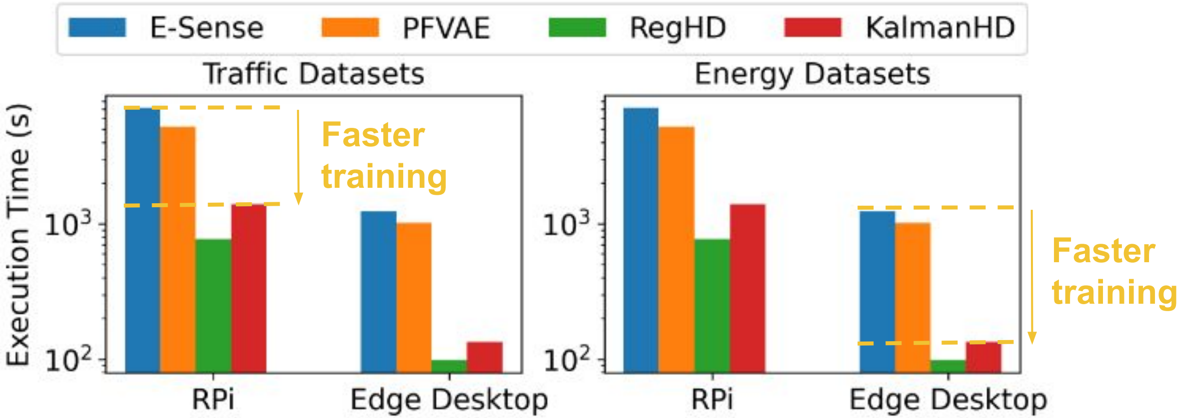



## Missing Values




- KalmanHD has accuracy **on par** with robust NNs (E-Sense [SECON '21] and PFVAE [Mathematics '22]).
- KalmanHD surpasses RegHD's [DAC '21] MAE accuracy by up to **72%**.

# Runtime Results




- KalmanHD is:
  - Up to **5.0x** faster compared to E-Sense [SECON ‘21] on Raspberry Pi.
  - Up to **8.6x** faster compared to E-Sense [SECON ‘21] on edge desktop.

  Than the other NNs like E-Sense [SECON ‘21] and PFVAE [Mathematics ‘22].

- KalmanHD has a **48%** computational overhead compared to RegHD [DAC ‘21] due to additional instructions, but has **72%** better accuracy.

# Conclusion

- **The challenges found in edge time series forecasting are**: Limited energy resources and noise introduced by sensors
- HDC brings efficient computing but lacks robustness **VS** robust models (NNs) can perform well in noise but are slow and resource intensive.
- We propose **KalmanHD**, a novel single-step forecasting approach integrating HDC with Kalman Filter for efficient noise-resistant forecasting at the edge.
- **KalmanHD** achieves comparable accuracy as robust neural networks in online settings while demonstrating **3.6x-8.6x** speedup compared to PFVAE [Mathematics '22] and E-Sense [SECON '21] respectively on typical edge platforms.
- Our model boosts HD regression [DAC '21] accuracy by up to **72%** in noisy environments.
- Code is available at https://github.com/DarthIV02/KalmanHD

# References

[1] Alejandro Hernández-Cano and et al. Reghd: Robust and efficient regression in hyper-dimensional learning system. In DAC ’21. IEEE, 2021.

[2] Christos Tzagkarakis and et al. Evaluating short-term forecasting of multiple time series in iot environments. In EUSIPCO ’22. IEEE, 2022.

[3] Fotios Zantalis and et al. A review of machine learning and iot in smart transportation. Future Internet, 11(4):94, 2019.

[4] Grzegorz Dudek. Short-term load forecasting using random forests. In IS ’2014, pages 821–828. Springer, 2015.

[5] Hamidreza Arasteh and et al. Iot-based smart cities: A survey. In EEEIC ’23, pages 1–6. IEEE, 2016.

[6] Qiquan Shi and et al. Block hankel tensor arima for multiple short time series forecasting. In AAAI, volume 34, pages 5758–5766, 2020.

[7] 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.

# References

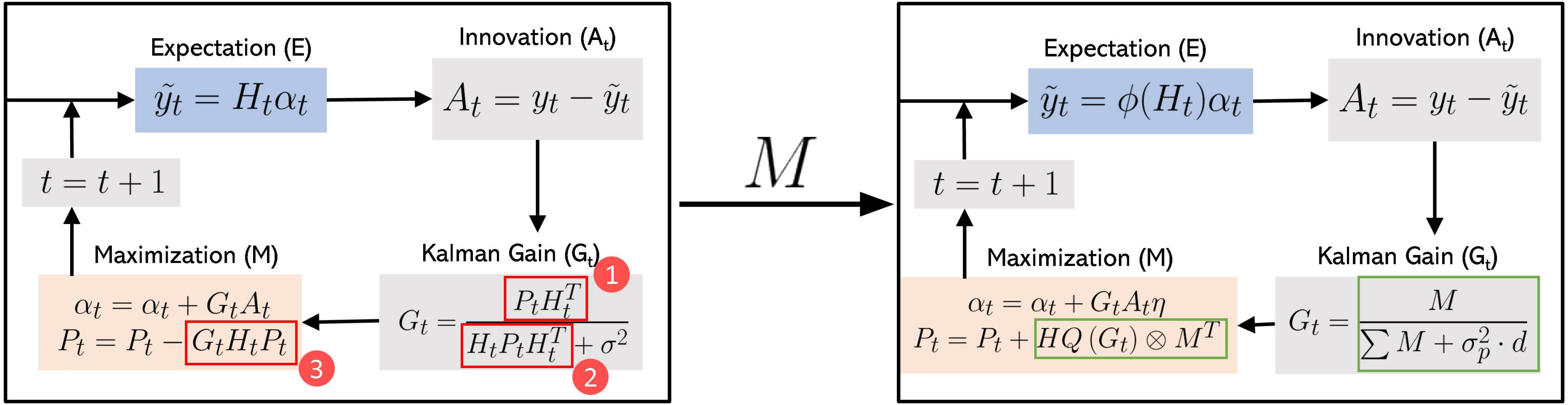
[8] Sureshkumar Selvaraj and et al. Challenges and opportunities in iot healthcare systems: a systematic review. SN Applied Sciences, 2(1):139, 2020.
[9] Xi andet al. Chen. Autoregressive-model-based methods for online time series prediction with missing values: an experimental evaluation. arXivpreprint arXiv:1908.06729, 2019.
[10] 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.
[11] Yukun Bao and et al. Multi-step-ahead time series prediction using multiple-output support vector regression. Neurocomputing, 129:482–493, 2014.
[12] Qiquan Shi and et al. Block hankel tensor arima for multiple short time series forecasting. In AAAI, volume 34, pages 5758–5766, 2020.
[13] Christos Tzagkarakis and et al. Evaluating short-term forecasting of multiple time series in iot environments. In EUSIPCO '22. IEEE, 2022.

# Backup (Hyperparameters)

- Best parameter are chosen via experimentation

| Dataset | η | *d* | y | 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 | | | |

# KalmanHD - Optimization




Kalman Filter | KalmanHD

$$M = HQ\left(P \cdot \phi(H_t)^T\right)$$

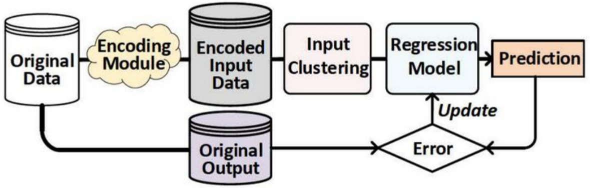
# Hyperdimensional Computing (HDC)

- Seeking to emulate human brain functioning
- **Superior energy efficiency and faster learning rate** then Neural Networks (NNs)
- Main ideas:
  - Mapping inputs to high dimensional sparse binary vectors (hypervectors)
  - Intricate patterns in the original data → linearly separable in HD space
- 3 main steps:
  1) **Encoding**: Mapping the data.

$$\underbrace{[0.5\ \ 0.6\ \ \ldots\ \ 0.8]}_{\text{Time Serie}} \times \begin{bmatrix} -0.5 & 0.6 & \ldots & 0.8 \\ 0.2 & -0.7 & \ldots & 0.9 \\ & & \ldots & \\ 0.1 & -0.4 & \ldots & -0.5 \end{bmatrix} \rightarrow \begin{bmatrix} -0.25 \\ 0.10 \\ \ldots \\ 0.05 \end{bmatrix} + \ldots + \begin{bmatrix} 0.64 \\ 0.72 \\ \ldots \\ -0.40 \end{bmatrix} = \begin{bmatrix} 0.30 \\ -0.11 \\ \ldots \\ 0.25 \end{bmatrix} \xrightarrow{HQ} \begin{bmatrix} 1 \\ -1 \\ \ldots \\ 1 \end{bmatrix}$$

  2) **Training**: Corrects the hypervector based on the error.
  3) **Inference**: Dot product between encoded sample and model hypervector.

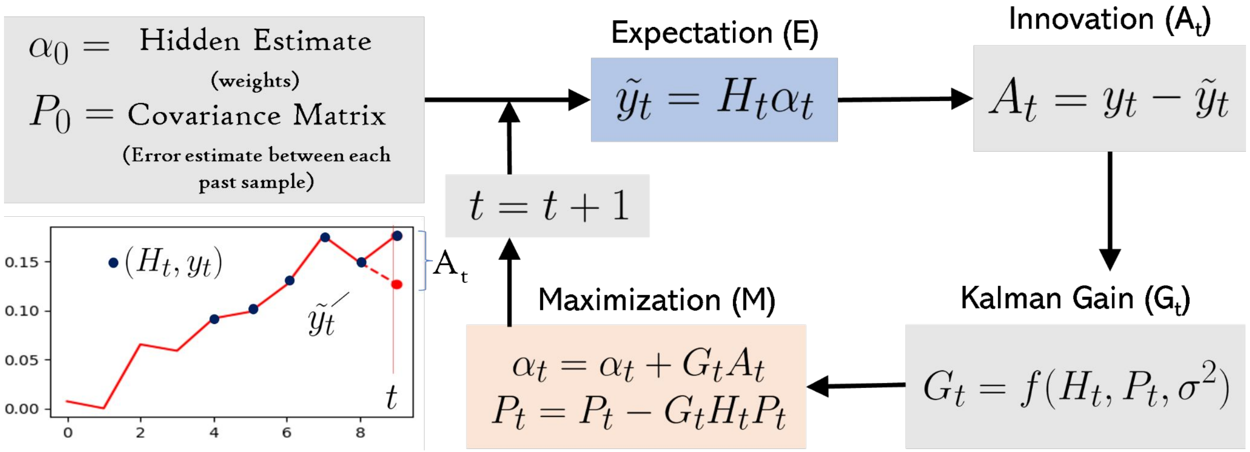
- HDC is not inherently robust against noise in the original data space




[DAC '21]

# Kalman Filter (KF)

- KF is useful for **limited number of samples** & **gaussian noise** within the time series




- **Objective**: Find a **hidden state** (coefficients for the regression) through a different **observed state** (past samples and next-step)
- Considers variance of the samples to determine the importance to training