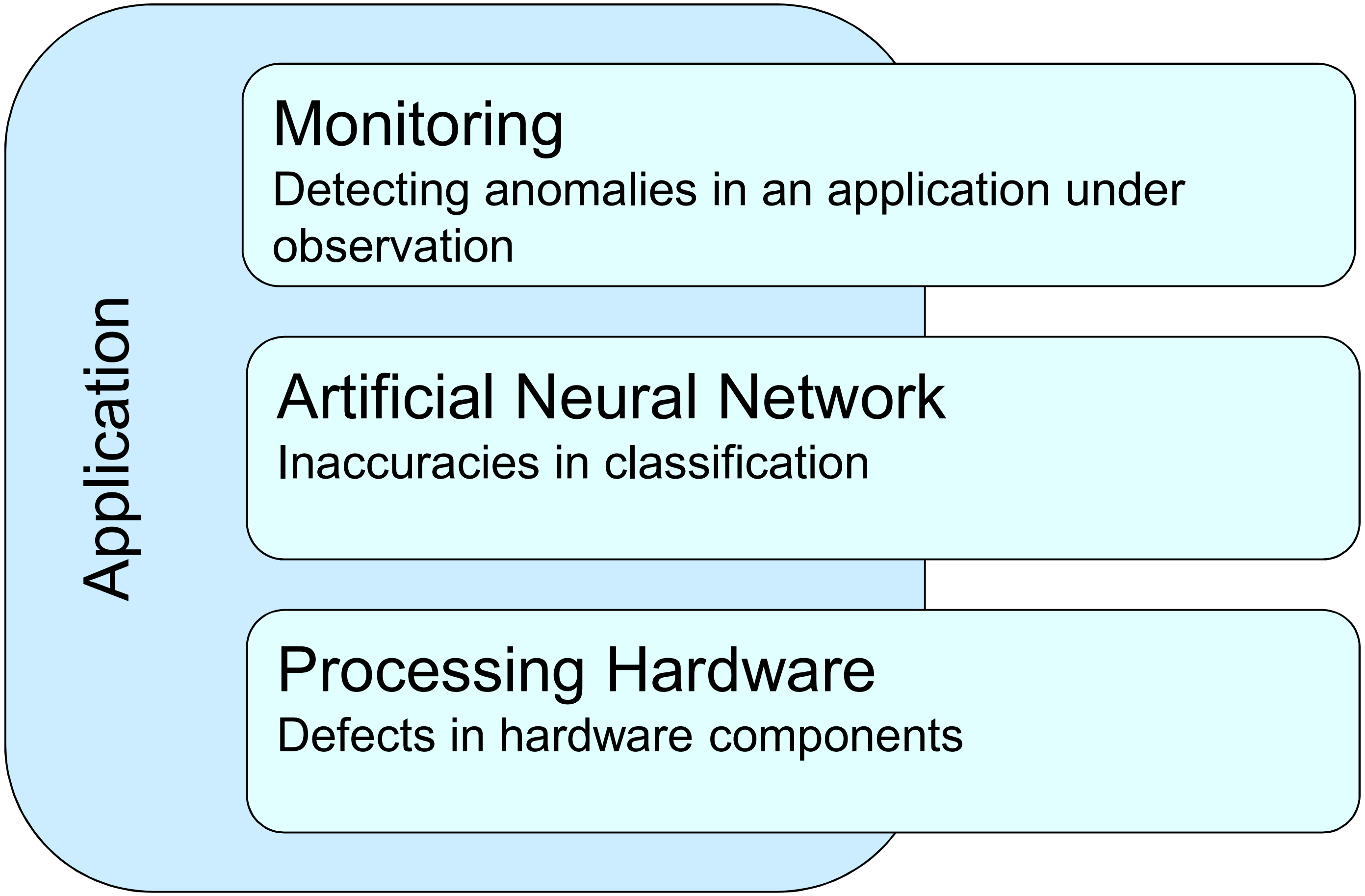
Artificial Neuronal Networks and Faults

Görschwin Fey

Institute of Embedded Systems
Electrical Engineering, Computer Science and Mathematics
Hamburg University of Technology

Credits to Fin Hendrik Bahnsen, Jan Kaiser

Neuronal Networks and Faults



Neuronal Networks and Faults

Monitoring

Detecting anomalies in an application under observation

Fin Hendrik Bahnsen and Goerschwin Fey. Local monitoring of embedded applications and devices using artificial neural networks. In EUROMICRO Symposium on Digital System Design (DSD), pages 485–491, 2019.

Fin Hendrik Bahnsen, Jan Kaiser and Goerschwin Fey. Designing recurrent neural networks for monitoring embedded devices. In IEEE European Test Symposium (ETS), 2021.

Internet of Things

- Devices in our lives are getting smart
- Everything is connected and has an IP address
- Inhomogeneous and unique systems, e.g. smart homes

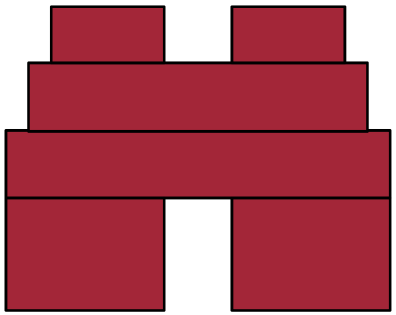
Internet of Things

- Devices in our lives are getting smart
- Everything is connected and has an IP address
- Inhomogeneous and unique systems, e.g. smart homes
- New security breaches and system dependability must be considered

Monitoring

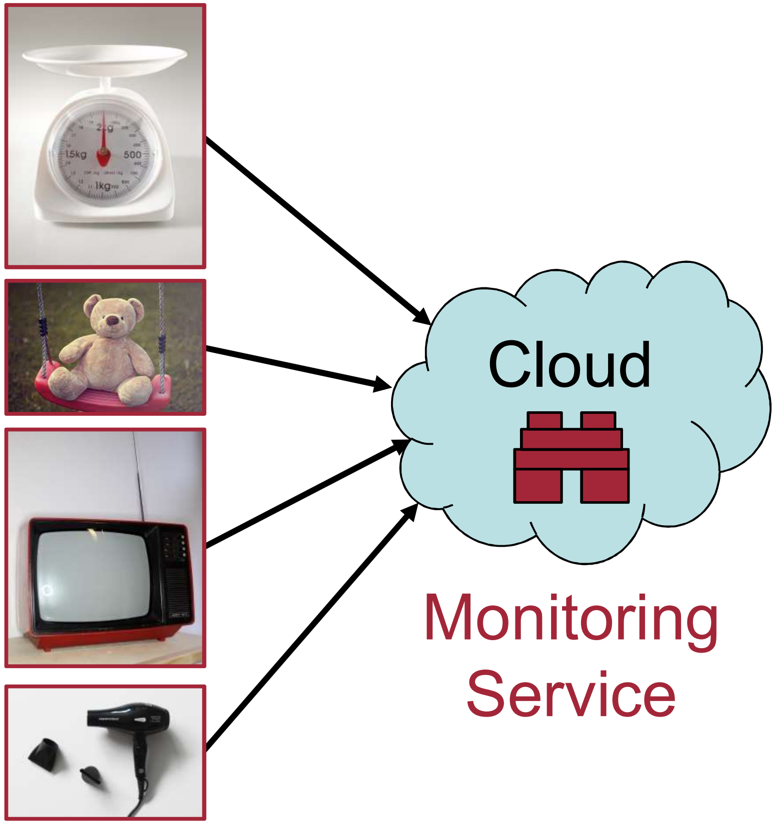
1. Detect a fault/attack
2. Handle the problem

- Monitoring to detect faults
- Single device or whole system



Cloud Based Monitoring

+ monitoring may benefit from other connected systems
+ high performance computing on the cloud server
- limited connectivity and observability
- connection latency added on top
- design effort due to heterogeneous systems



Localized Monitoring

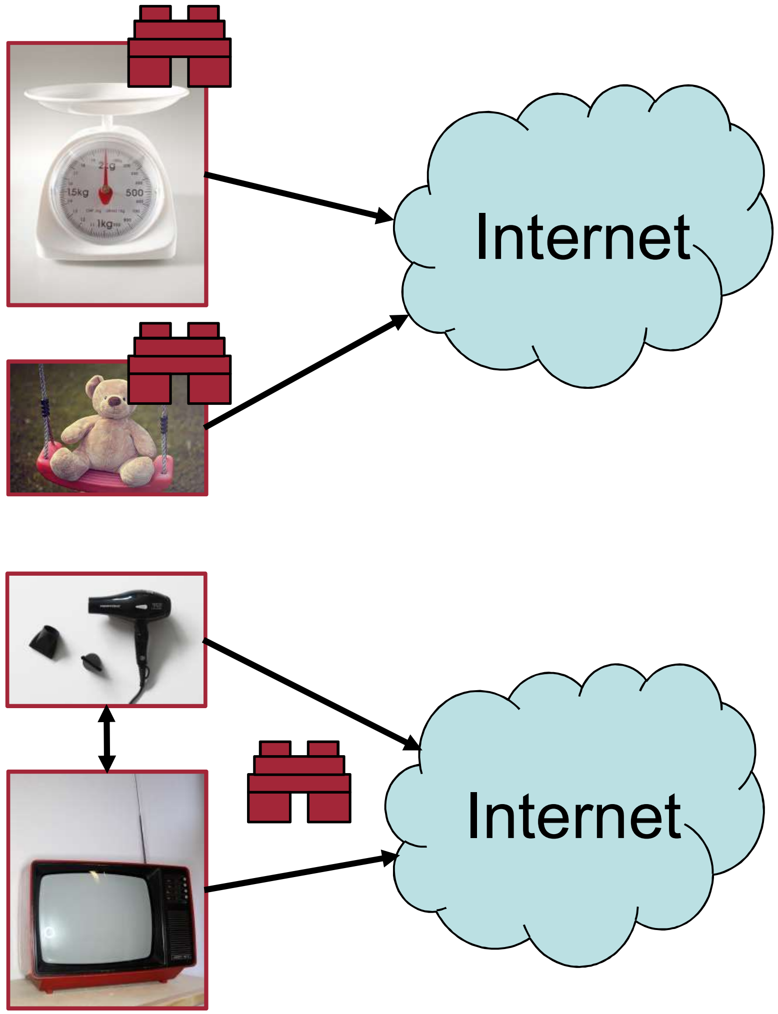
The monitor can be part of an embedded node or be allowed to sniff network traffic

- limited computational resources

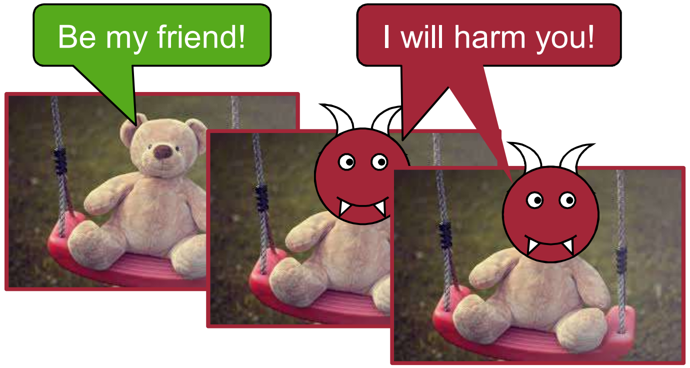
- adaption to unknown environment required

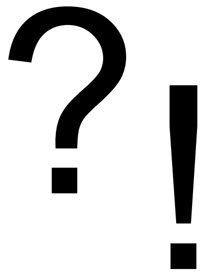
+ low latency for fault detection

+ independent from connectivity or monitoring service availability



"Traditional" Approaches





Double modular redundancy
detects certain faults

Triple modular redundancy
masks certain faults

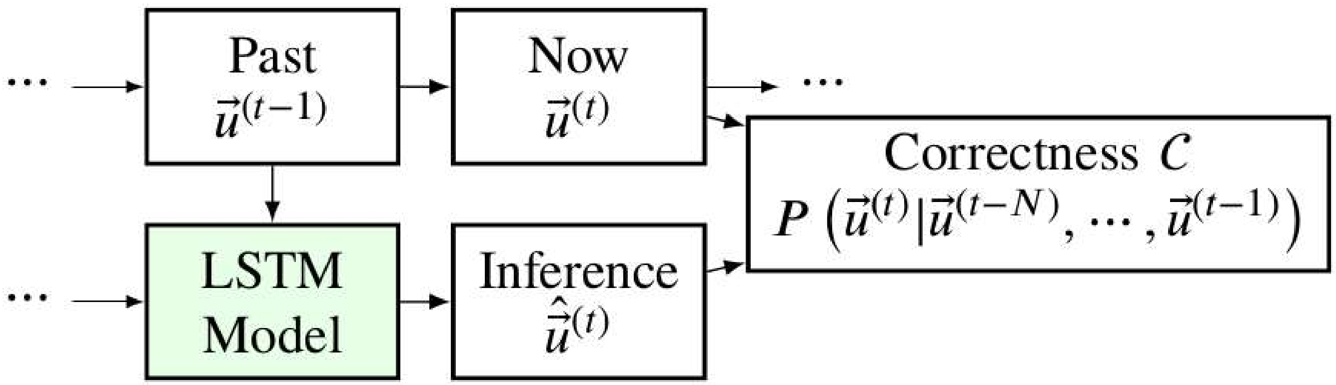
- an attack may corrupt more than one device
- expensive to implement
- Byzantine faults

Assertion based monitoring

- checking at runtime can be expensive
- assertions have to be designed
- in-depth knowledge of system and application is required

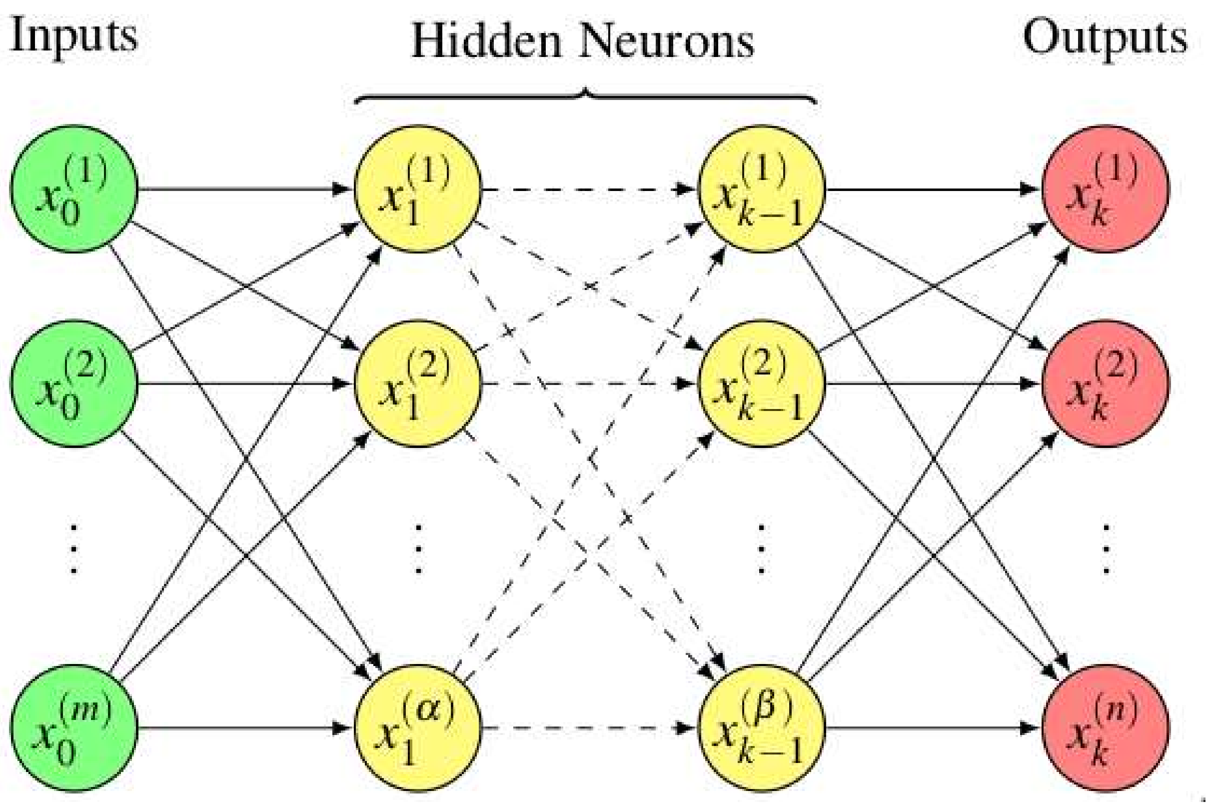
Approach Considered Here

- Specification of acceptable behaviour learned automatically by a neural network



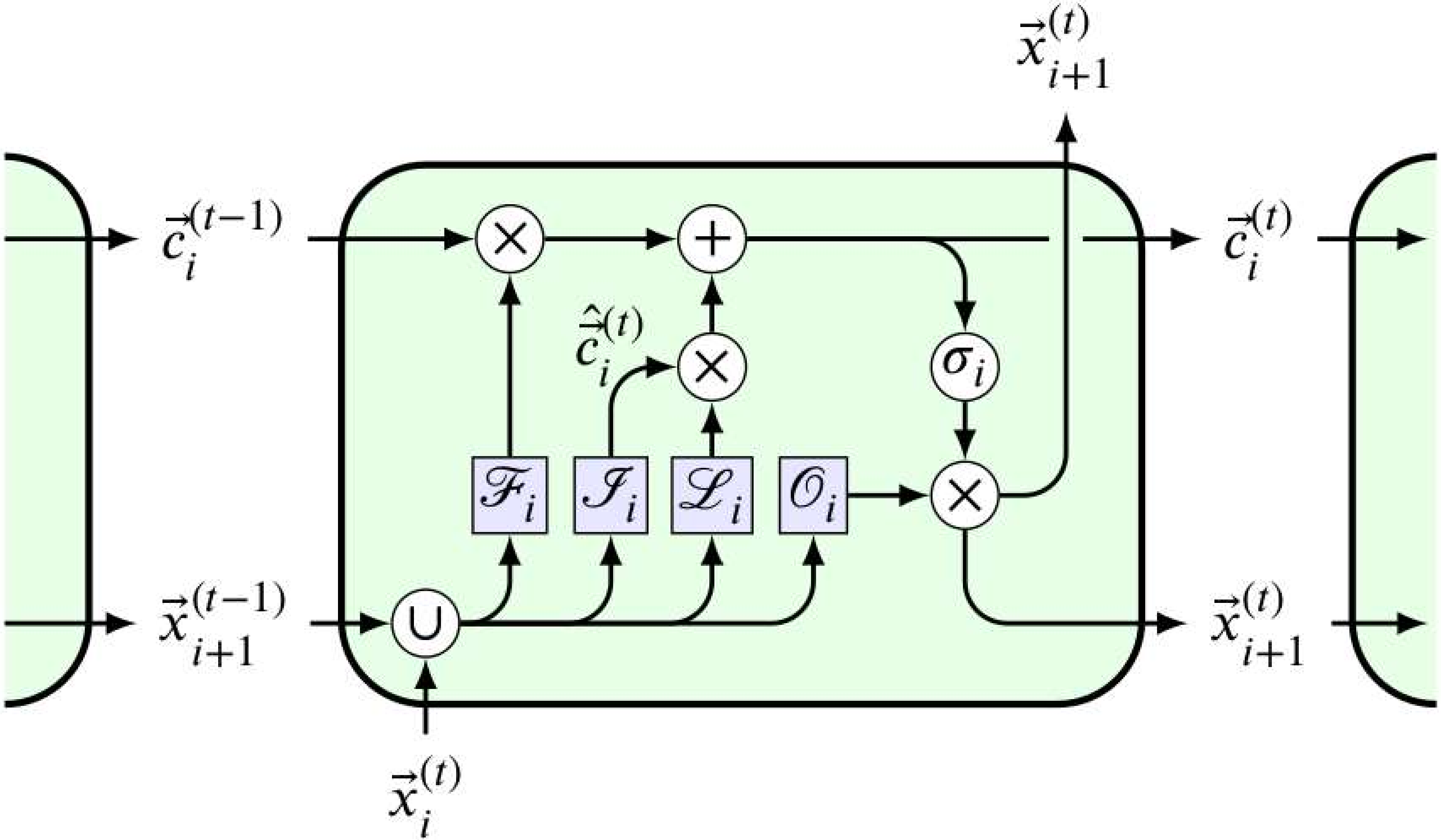
- Training on unstructured high-dimensional data
- No fault model required
- Correctness measure provides a metric for the system health including confidence metric

Multi Layer Perceptron – MLP

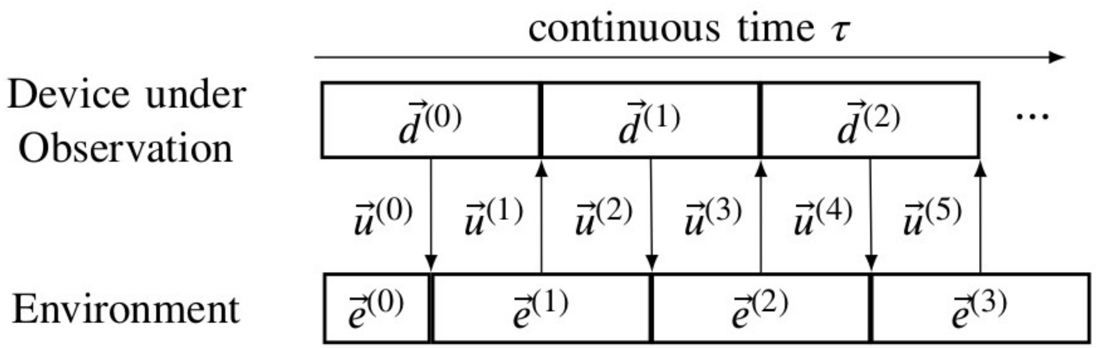


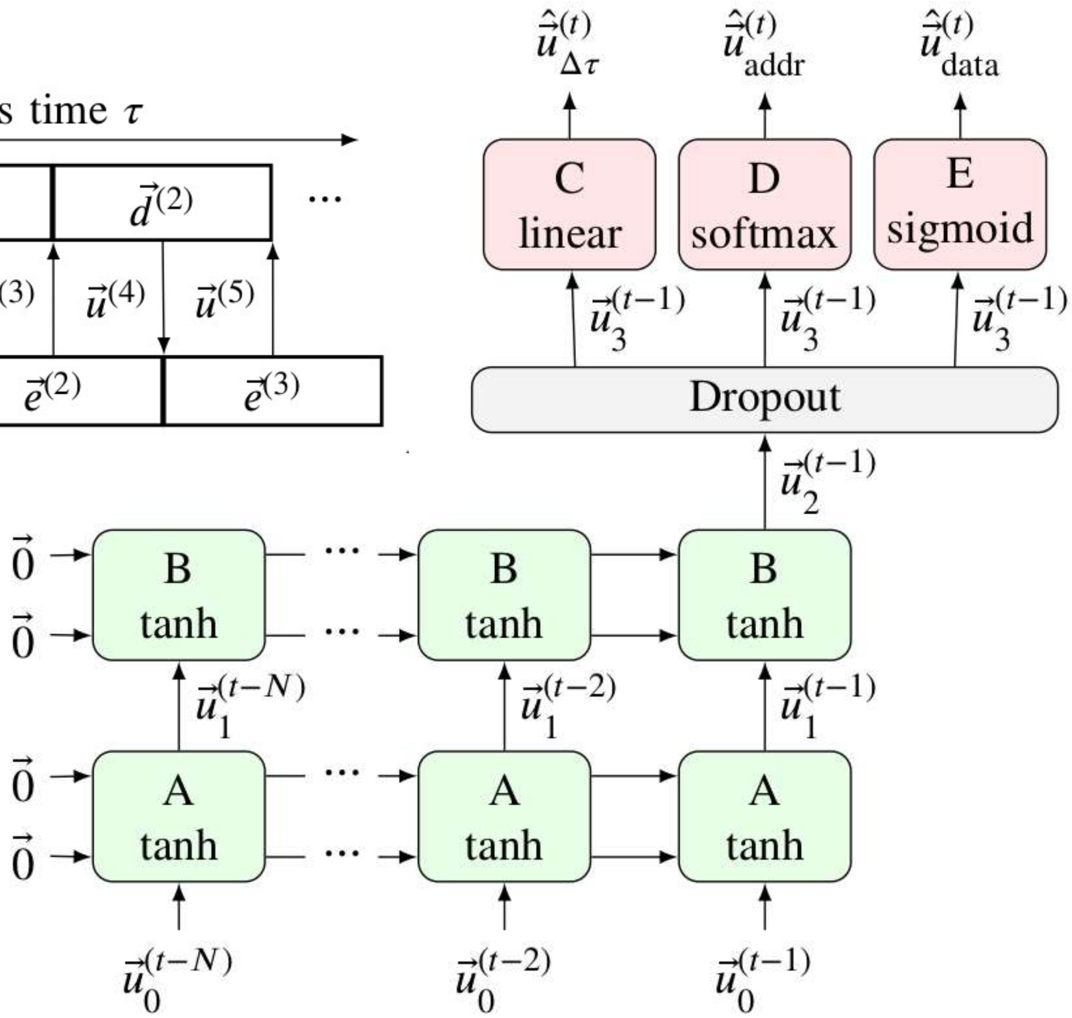
$$\vec{x}_{i+1} = \sigma\left(\boldsymbol{W}_i \cdot \vec{x}_i + \vec{b}_i\right)$$

Long Short-Term Memory (LSTM)



Approach – Observing the “System Bus”



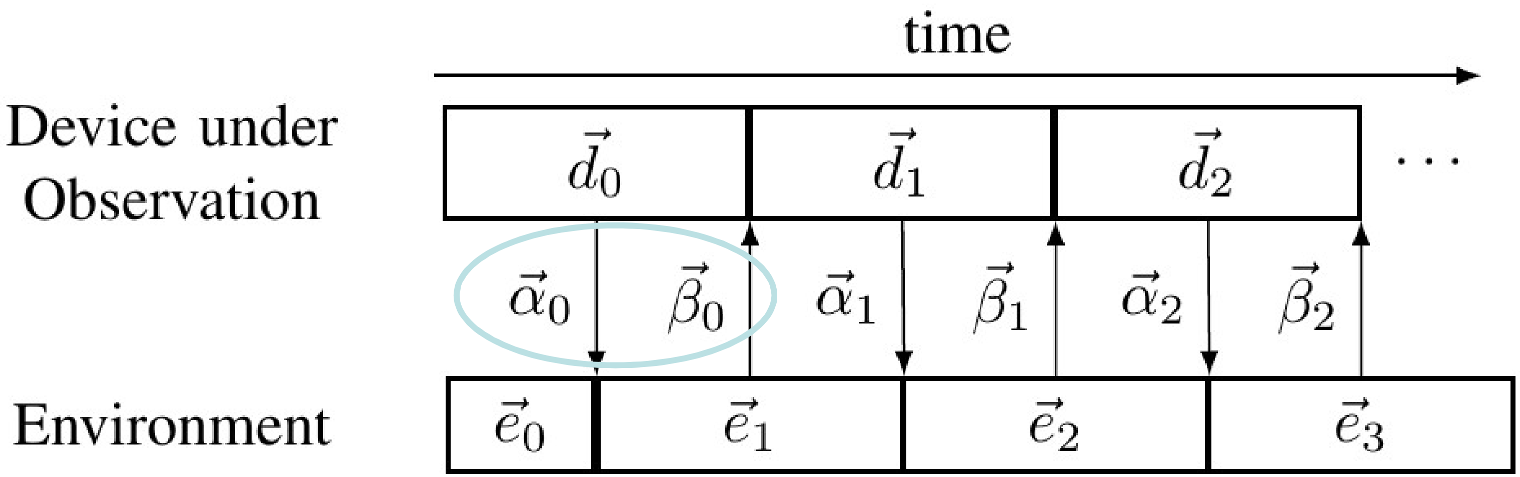


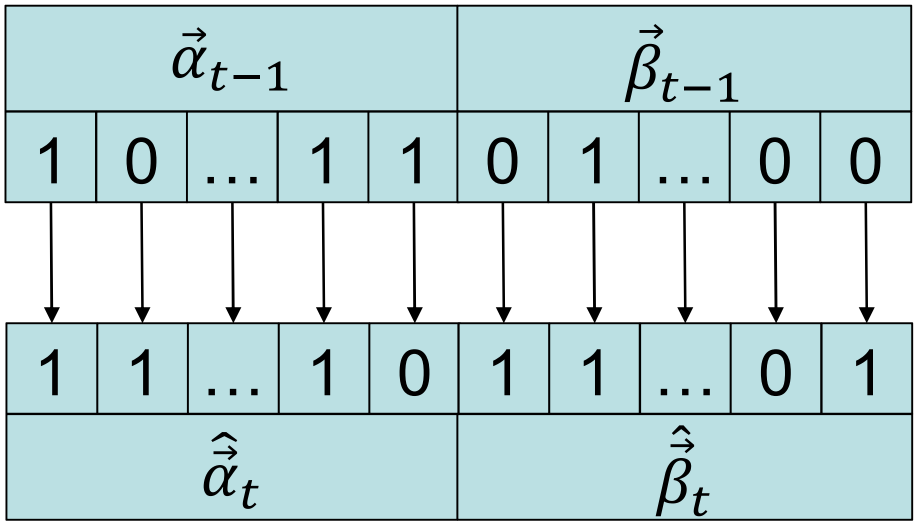
Regression
to predict the time delta between two messages

Single-label classification
to predict a message destination

Multi-label classification
to predict the data per message

Prediction Model – Data Bits

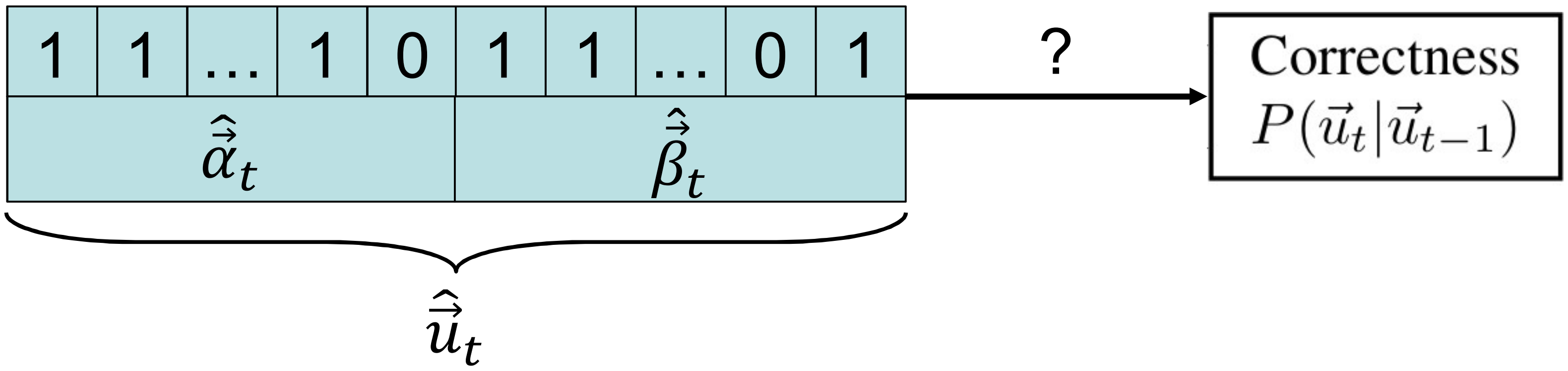




Single-label classification is infeasible for 2^n classes needed for n-bit communication vectors.

Independent prediction per bit

Correctness – Data Bits



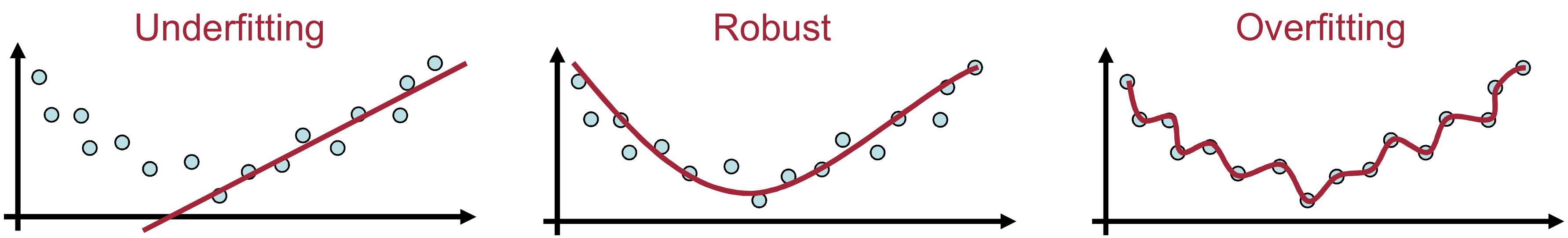
Weighting each bit by the prediction significance (using training data):

$$\vec{s}^{(k)} = 1 - \frac{\sum_t |\hat{\vec{u}}_t^{(k)} - \vec{u}_t^{(k)}|}{T} \qquad \vec{w}^{(k)} = \frac{\exp \vec{s}^{(k)}}{\sum_k \exp \vec{s}^{(k)}}$$

Approximating correctness as weighted sum over absolute similarity:

$$P(\vec{u}_t|\vec{u}_{t-1}) \approx 1 - \sum_k \vec{w}^{(k)} \cdot |\hat{\vec{u}}_t^{(k)} - \vec{u}_t^{(k)}|$$

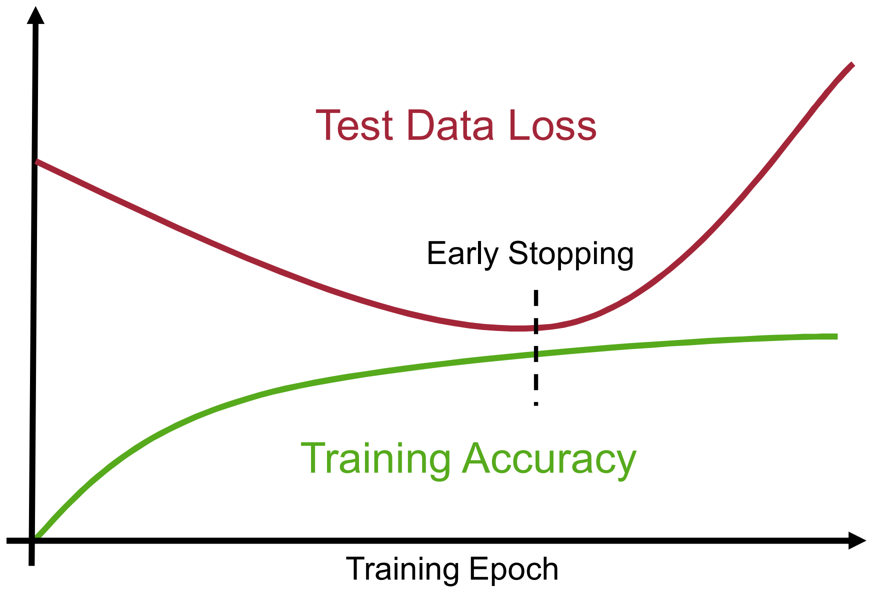
Generalisation



Two different data sets are used to train a neural network:

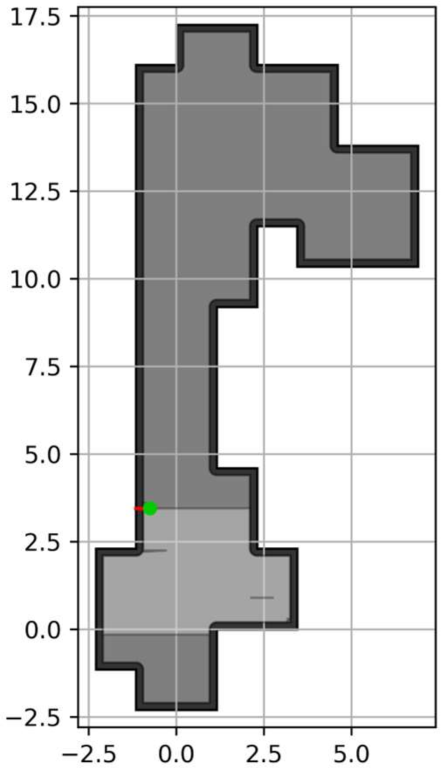
the training set is used to fit the model to the data

the test set is used to evaluate the model during training

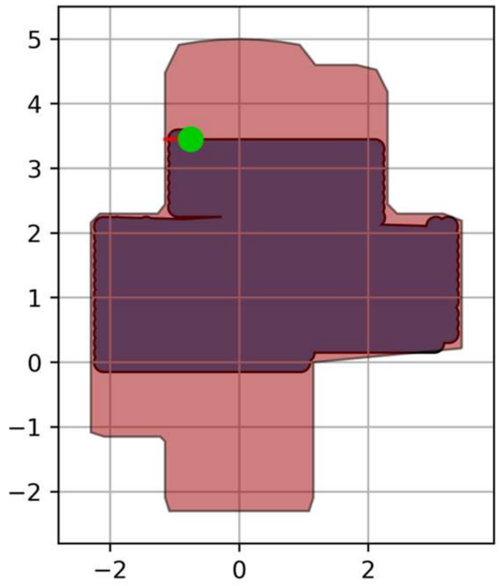


Experiment – Vacuum Cleaning Robot

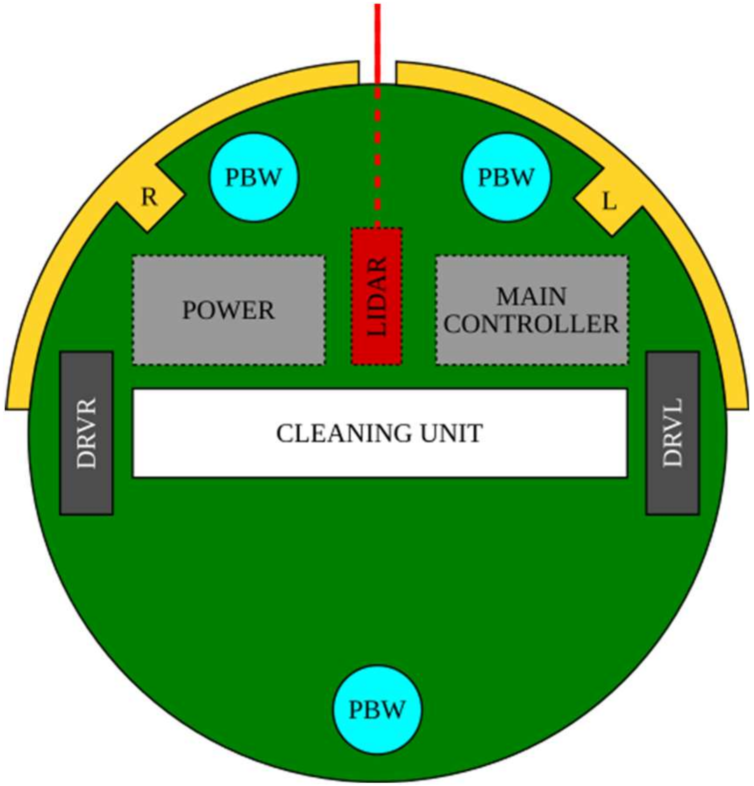
Environment



Controller



Robot Model



- continuously scans its surroundings
- minimizes driving distance for cleaning the whole area

Experiments – Injected Faults in the Robot

Functional Faults

F1: The wires of the left and right motors are swapped.

F2: The LIDAR sensor does not detect the reflected light and always returns the maximum distance of 5 m.

F3: The left touch sensor is broken and is not triggering on hits anymore.

Application Faults

F4: The charging station of the robot is removed from the environment.

F5: The robot is put to a different position in the environment.

Security Faults

F6: A DDoS attack hits the robot controller.

F7: The control algorithm of the robot is replaced by another algorithm.

Results – Vacuum Cleaning Robot

Our approach detected all faults:

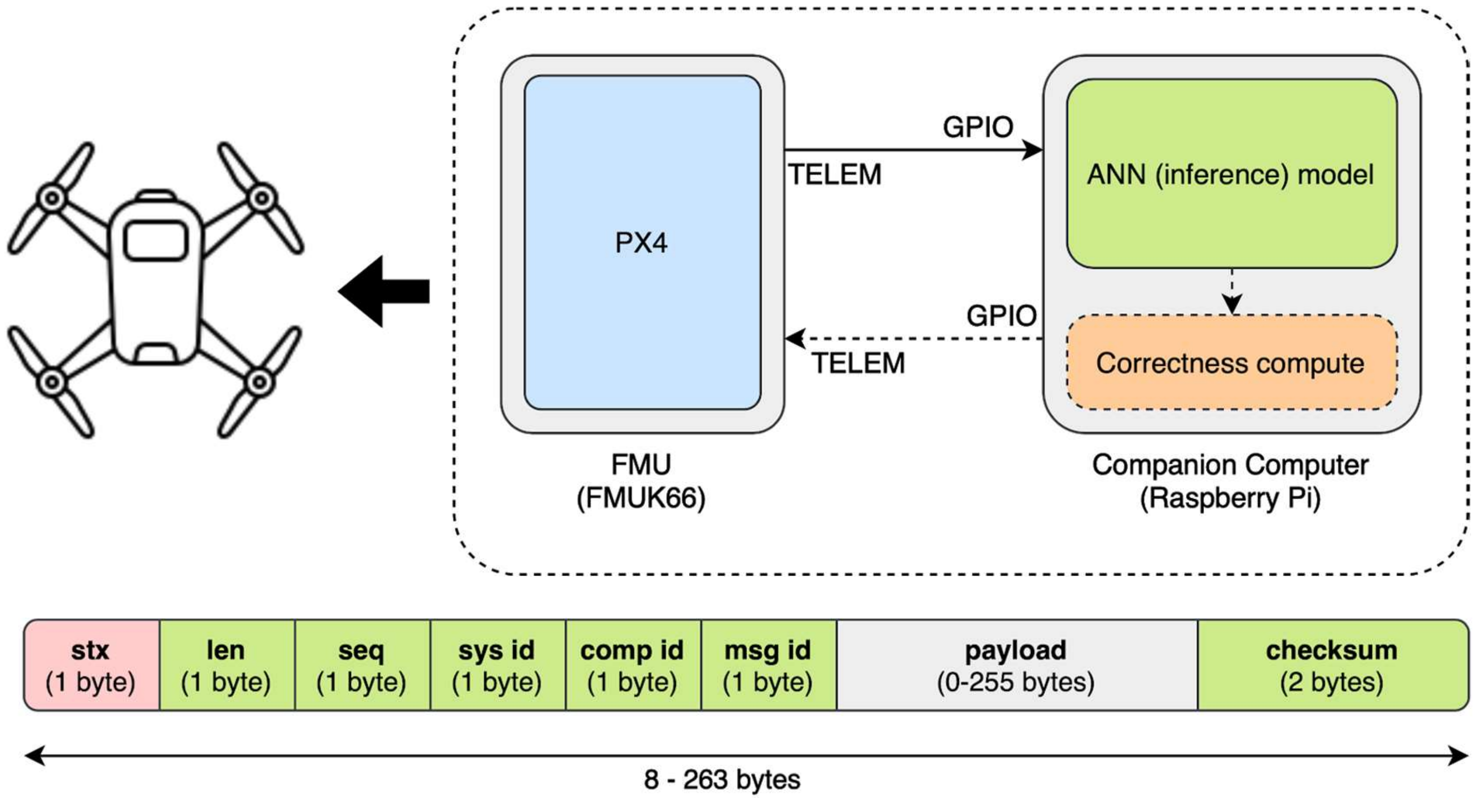
- detection abilities depend on the fault
  - e.g. erroneous sensor data does not change address of a message
  - F5 (position change) and F6 (DDoS attack) were hard to detect
- low latencies – less than the window length

Final Network and Training
- 2 LSTM layers
- 150 neurons
- 1.35*10^6 data points
- 0.1 drop out rate
- sequence length 50
- Hamming window for correctness 12
- Threshold 97.87%

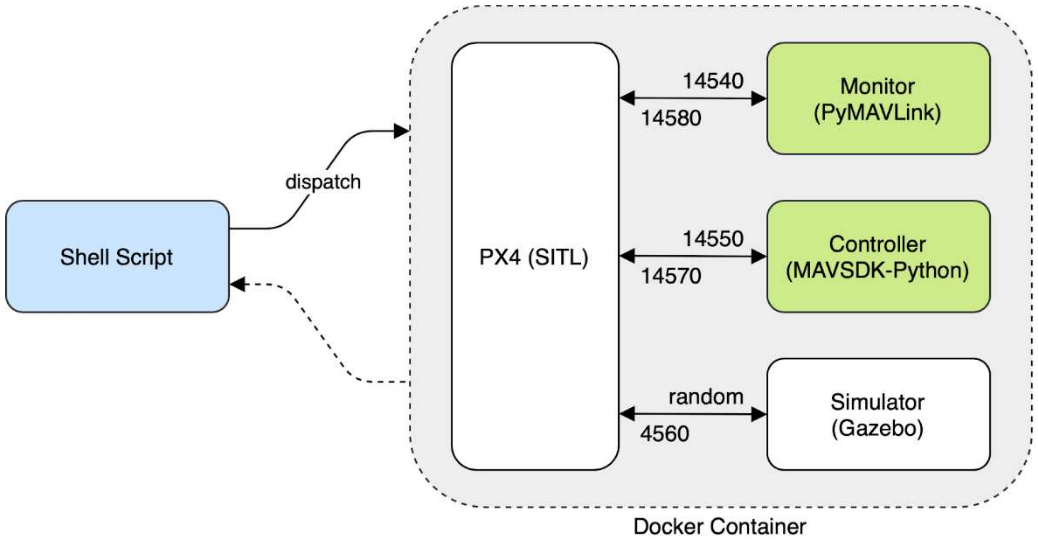
Training takes about 3 hours per model using a machine with:
- Intel Core i7@2.9GHz
- Nvidia GeForce 930 MX
- 16 GB RAM

Performance of sub models:
- Address 99.87 %
- Data 91.35 %
- Time 90.75 %

Monitoring a PX4 UAV

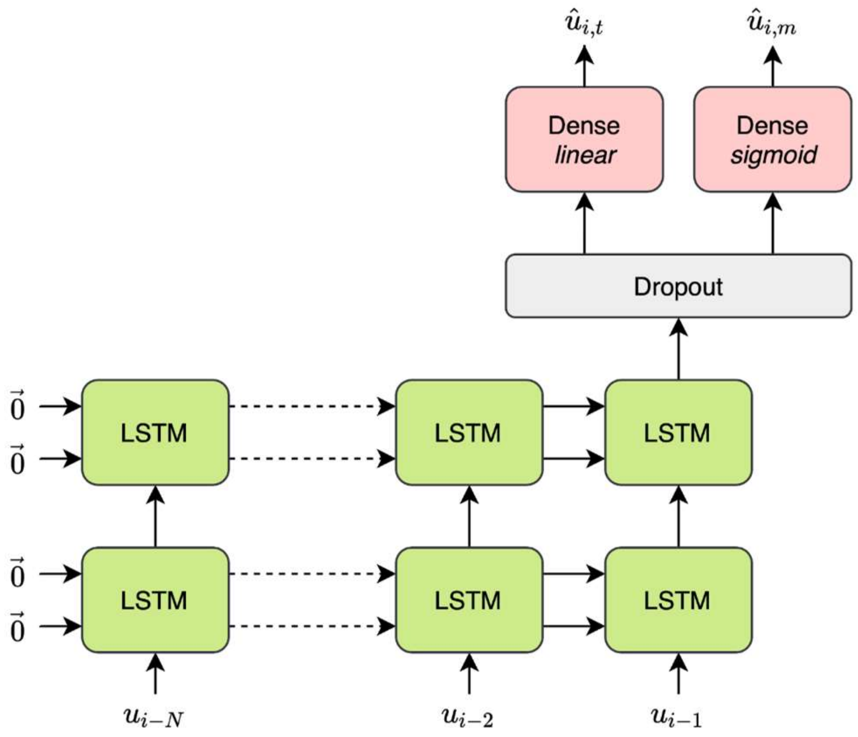


Simulation Environment



- Random flight paths on earth as missions
  - 75 nominal flights
  - 25 (faulty) flights
- Extract sequences (length=50) and targets from overall sequence
- Pad messages to 263 bytes
- Keras Generators implemented to preprocess and feed data on-the-fly
- Hyperparameter tuning using *Hyperband* algorithm in *Keras Tuner*

Statistical Approach – Correctness



$$c_i = P(u_i | u_{i-N}, \ldots, u_{i-1})$$

$$c_{i,t} \approx \frac{\mathcal{N}(\hat{u}_{i,t}, \hat{\sigma}_t^2)(u_{i,t})}{\mathcal{N}(0, \hat{\sigma}_t^2)(0)}$$

$$c_{i,m} \approx 1 - \sum_k w_k |u_{i,m}^k - \hat{u}_{i,m}^k|$$

$$s_k = 1 - \frac{\sum |u_{i,m}^k - \hat{u}_{i,m}^k|}{|U_{\text{test}}|}$$

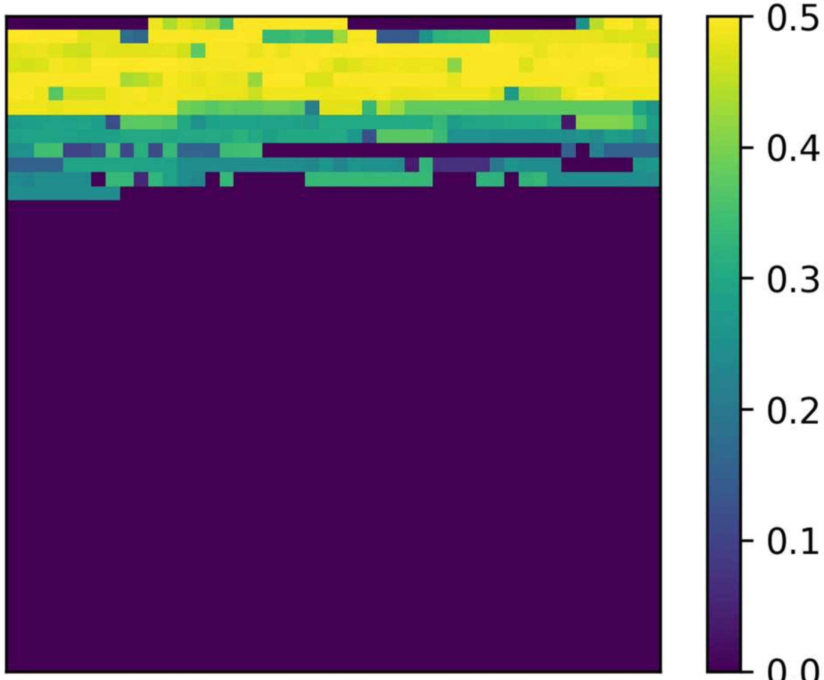
$$w_k = \frac{\exp(s_k)}{\sum_k \exp(s_k)}$$

Tweaking the Approach

How to handle the challenges of a real Device under Test?

- Salience-weighted loss
- Message type balancing
- (Gated) Message type bypass

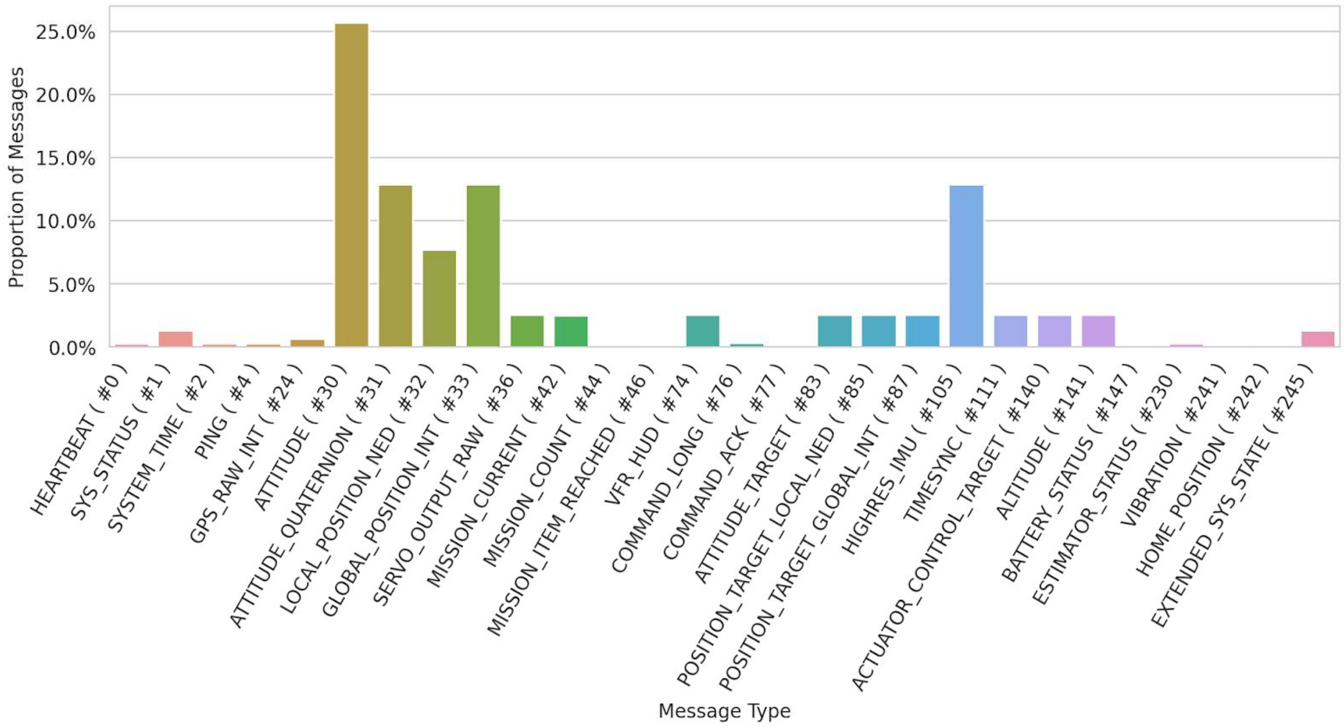
Salience-Weighted Loss



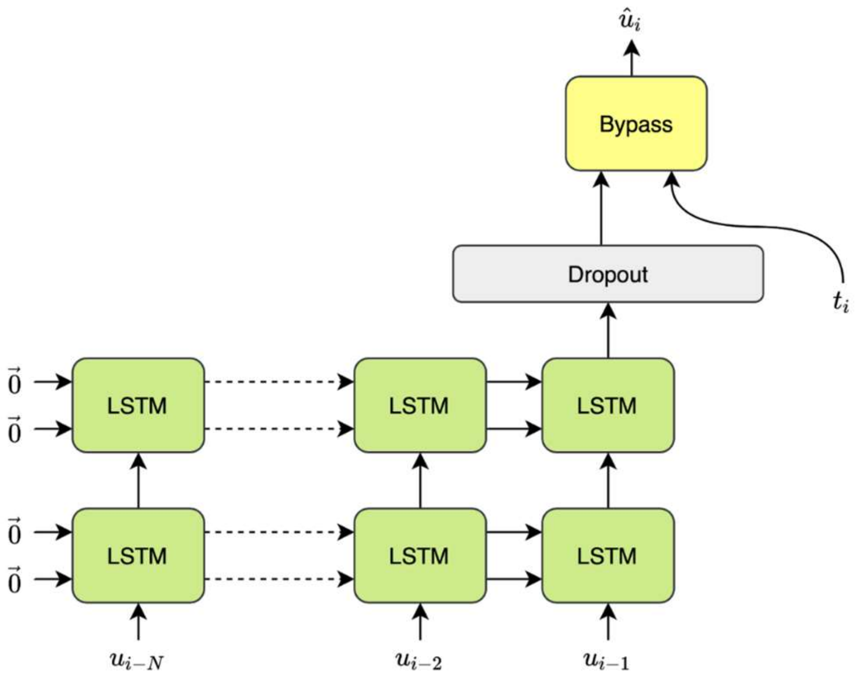
Standard Deviation $\sigma_{\text{train}}^{k}$

$$\mathcal{L}_{\text{SBCE}}(u_i, \hat{u}_i) = \frac{\sum_k \sigma_{\text{train}}^{k} \text{BCE}(u_i^k, \hat{u}_i^k)}{\sum_k \sigma_{\text{train}}^{k}}$$

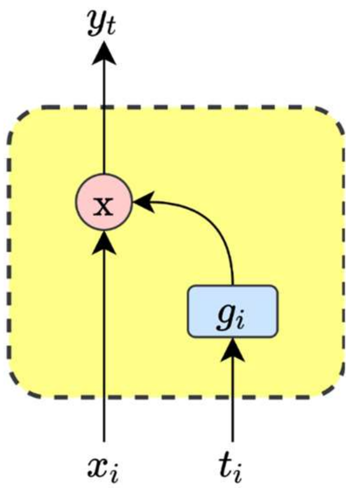
Message Type Balancing



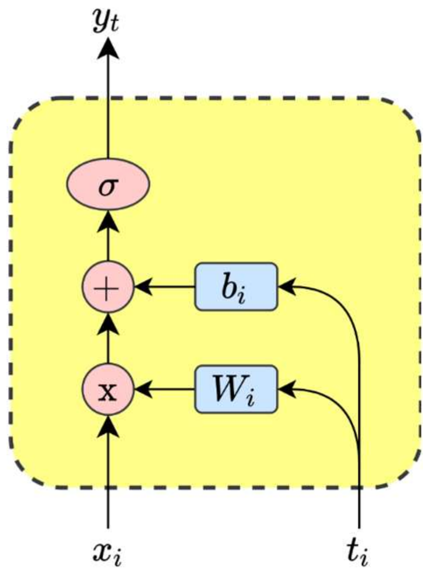
Message Type Bypass Cell



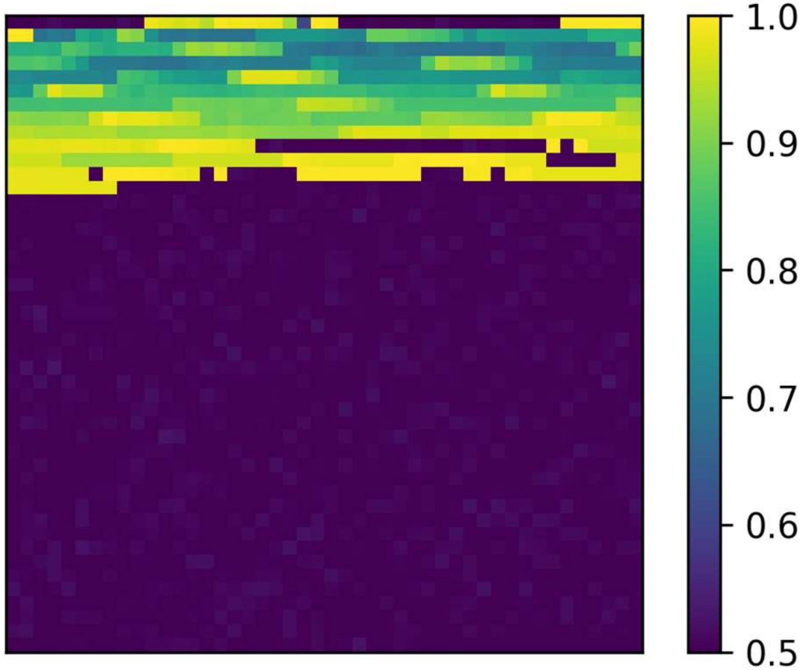
Gated

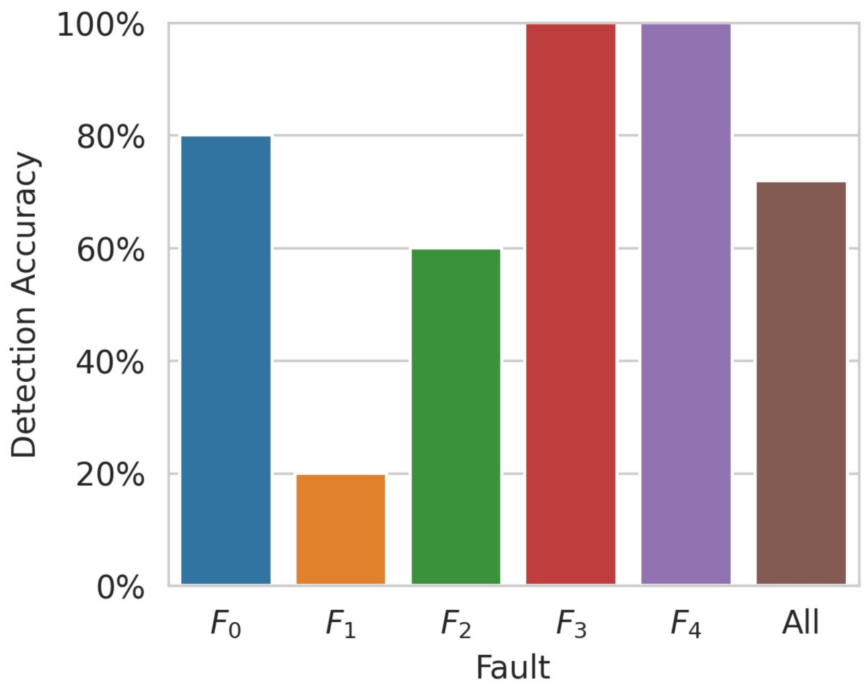


Linear

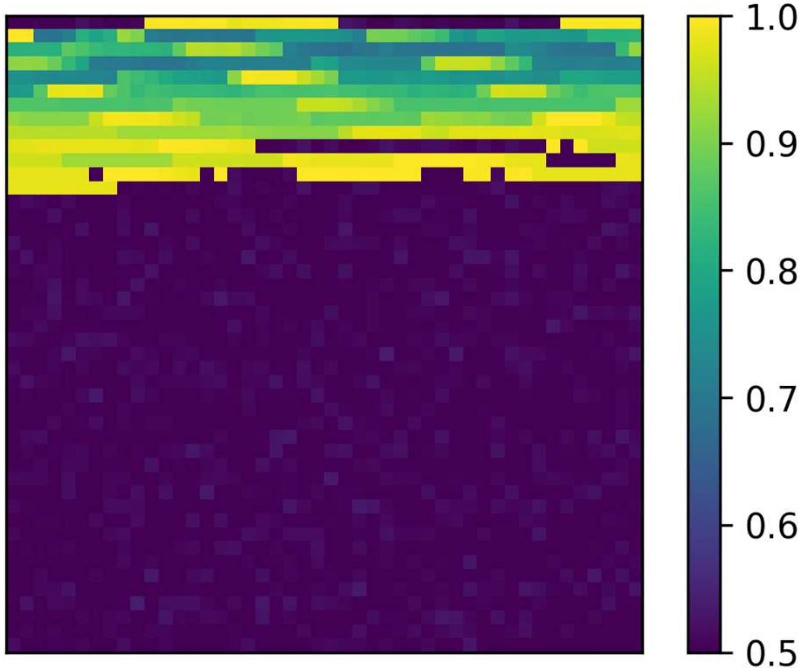


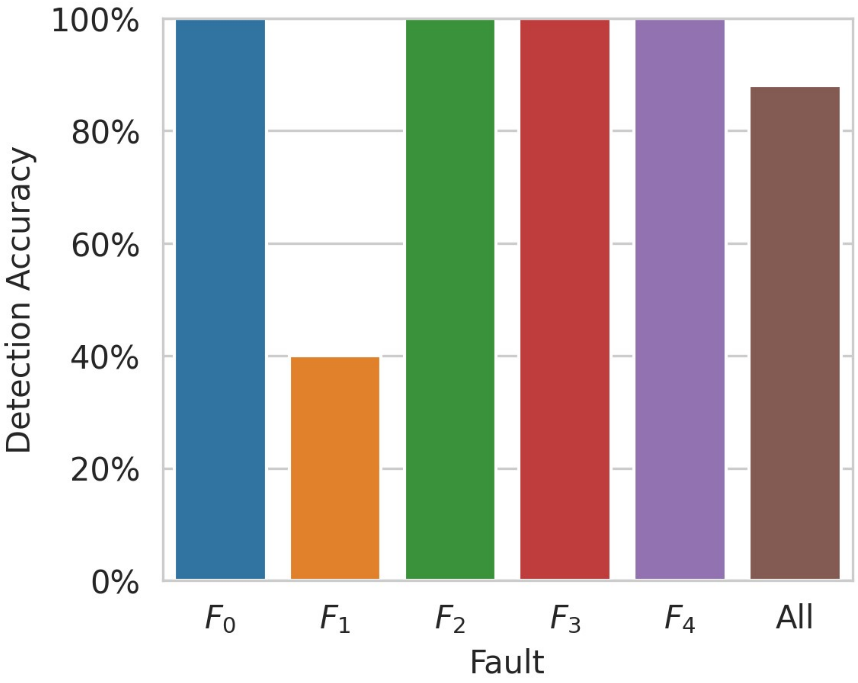
Linear Bypass





Gated Bypass





Results Summary

	M_0	M_{nt}	M_{sal}	M_{bal}	M_{gby}	M_{lby}
L	64	32	32	64	512	512
r	0.1	0.1	0.1	0.1	0.3	0.5
$\mathcal{L}$	0.505	0.086	0.433	0.348	0.208	0.249
h	512	128	256	512	64	512
$\mathcal{T}$	93.5%	94.9%	83.3%	77.1%	83.0%	82.7%
F_0	92.4%	95.0%	83.3%	76.9%	83.2%	82.5%
F_1	94.0%	95.3%	86.5%	78.9%	83.2%	83.6%
F_2	93.9%	95.4%	85.7%	79.1%	82.2%	83.0%
F_3	94.0%	94.6%	82.3%	78.1%	82.4%	82.2%
F_4	92.9%	94.7%	81.6%	76.8%	80.7%	80.9%
$\mathcal{A}$	64.0%	68.0%	64.0%	56.0%	88.0%	72.0%

M_0 : initial model
M_{nt} : no time
M_{sal}: salience-weight-loss
M_{bal}: message type balanced
M_{gby}: gated bypass cell
M_{lby} : linear bypass cell

Green: fault was always detected

Red: fault was sometimes detected

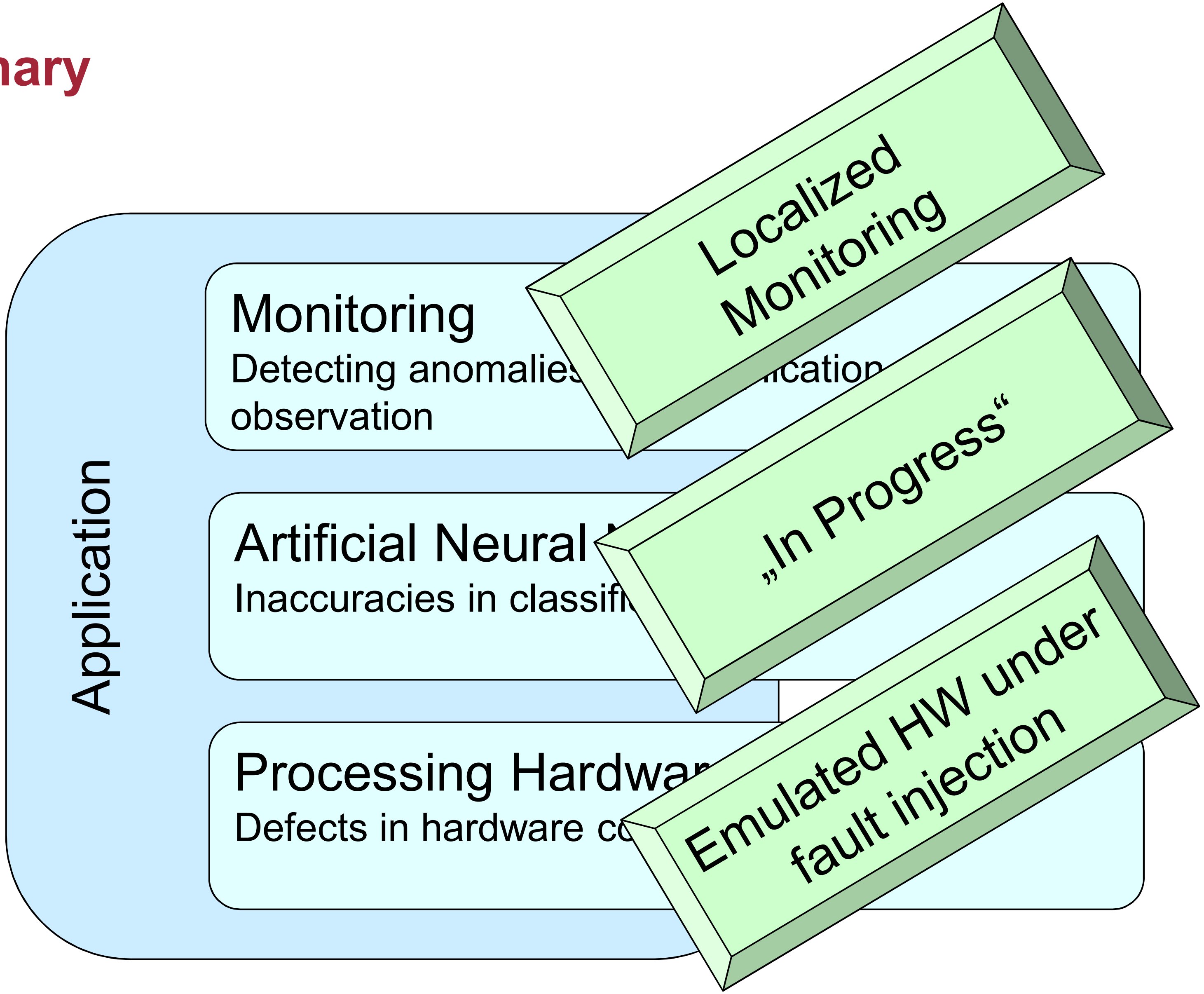
2 LSTM layers, L neurons/layer, r drop out rate, h Hamming window, T threshold
F_i lowest/highest correctness for F_0/$F_{1\text{-}4}$, A accuracy
L,r optimized by hyperparameter tuning

Localized Monitoring

- **Summary**
  - No fault model, no predefined specification needed
  - Empirical results for complex state-based system
  - Original system unchanged

- **Conclusion**
  - Being completely agnostic of the monitored system does not work ((yet))
  - Minor changes in the NN architecture have a significant impact on accuracy and fault detection

Summary



Application
Monitoring
Detecting anomalies
lication
observation
Artificial Neural
Inaccuracies in classifi
Processing Hardwar
Defects in hardware co
Localized Monitoring
„In Progress“
Emulated HW under fault injection