Network Anomaly Detection Based on Logs

Team Ember

Directory

PART 1

Analysis

Network Anomaly Detection

Data

Complex network devices produce large amount of log data

Detection

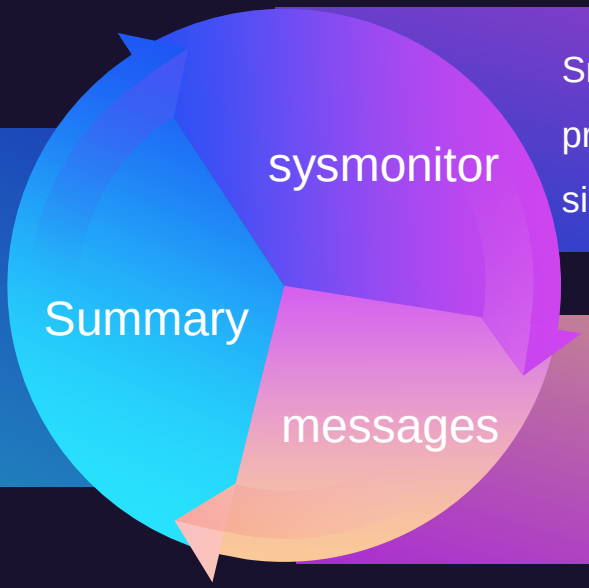
Network or device faults information buried inside data

Intelligence

Fast and smart way to analysis and detect anomaly is required

Dataset

- Two different datasets
- More than 1M lines of unstructured log data
- Trainset only contains normal data



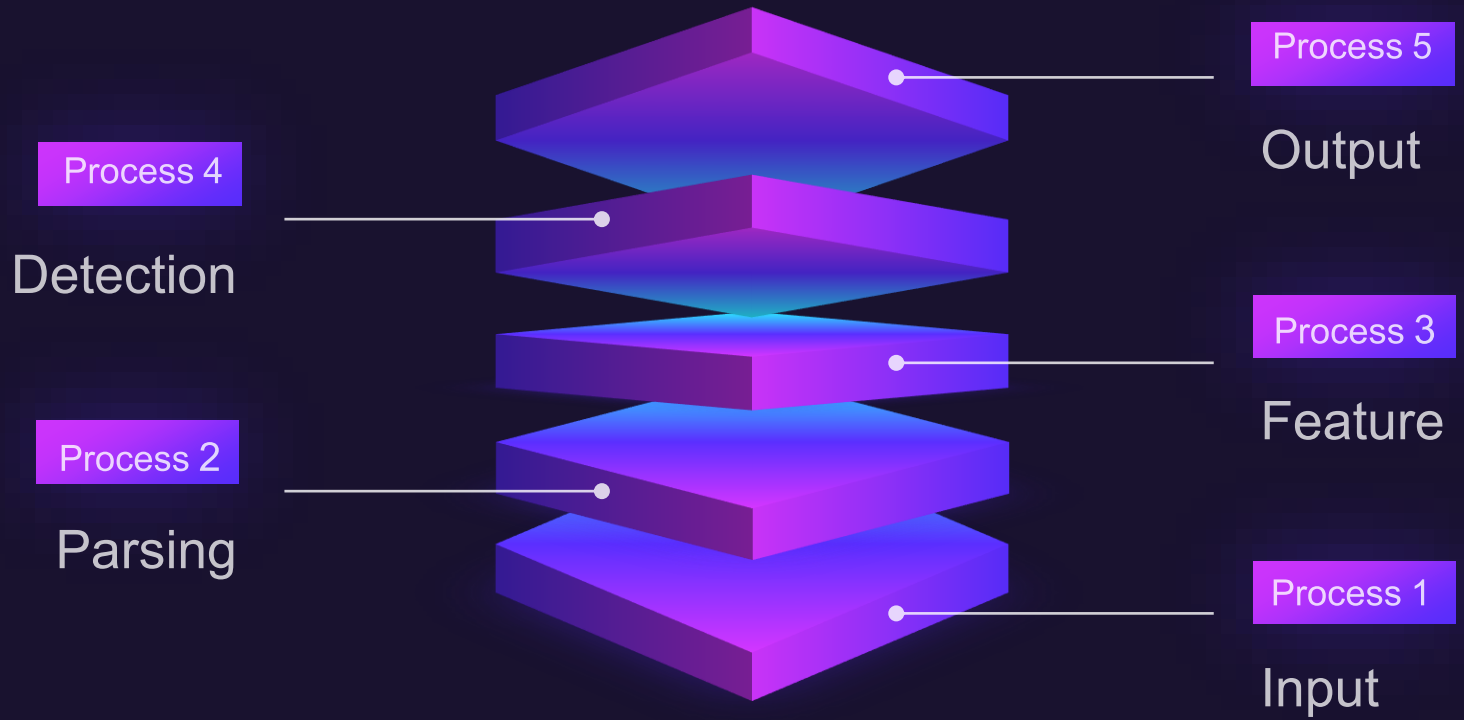
Smaller dataset. Contains log produced by sysmonitor, e.g., signals, processes start/stop.

Larger dataset. Contains log produced by different sources, e.g., system, kernel.

PART 2

Algorithms

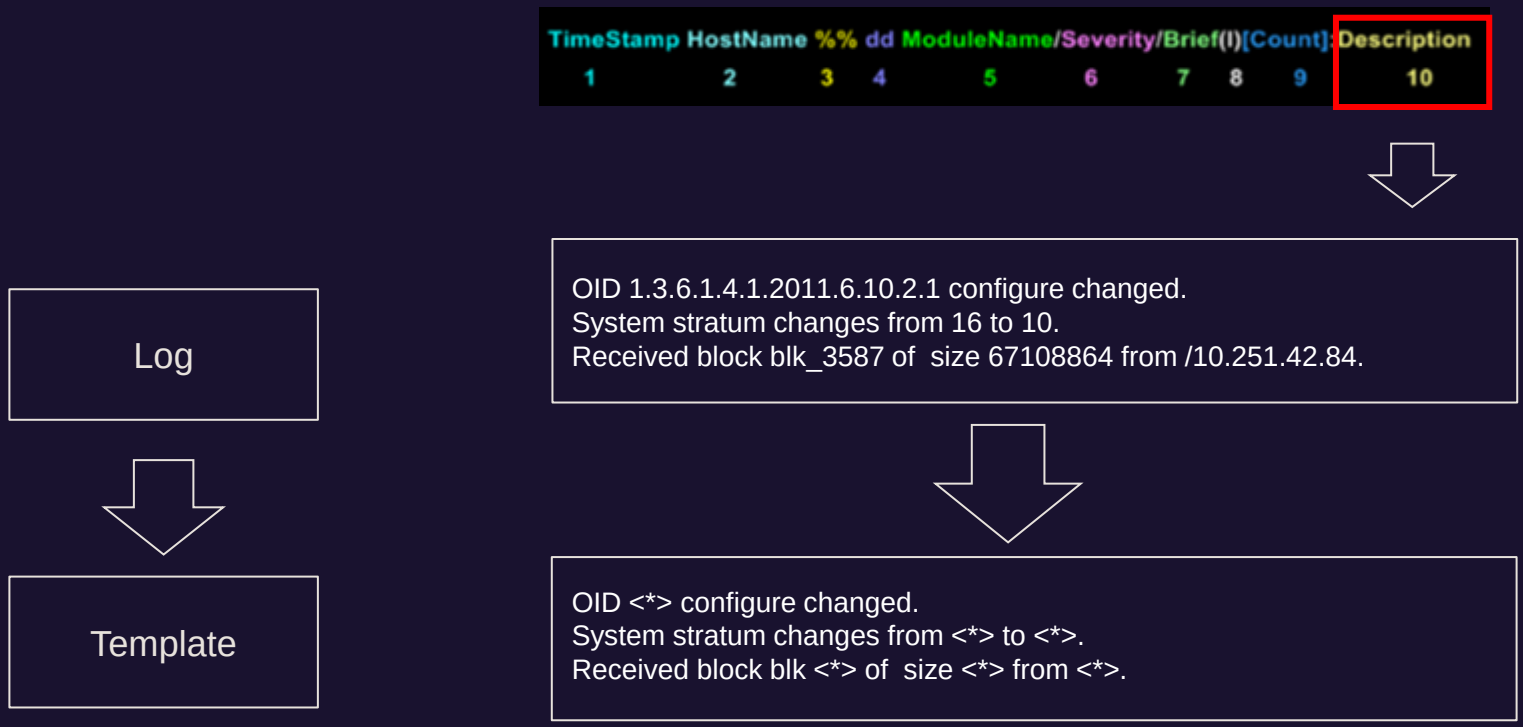
Process Structure



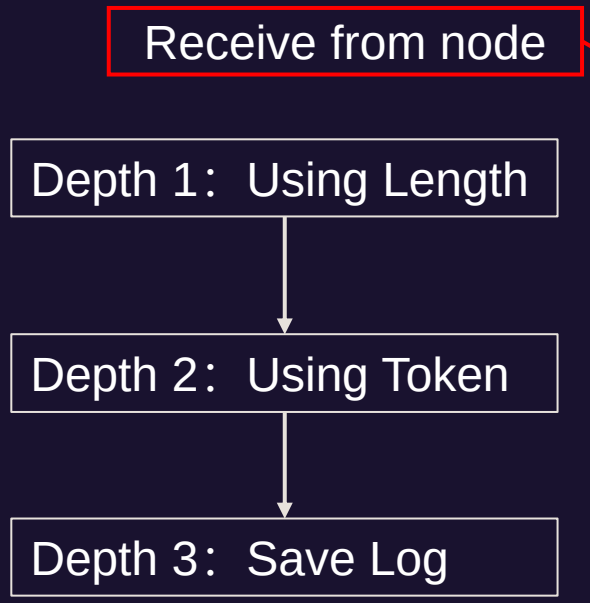
Traditional Log Parsing

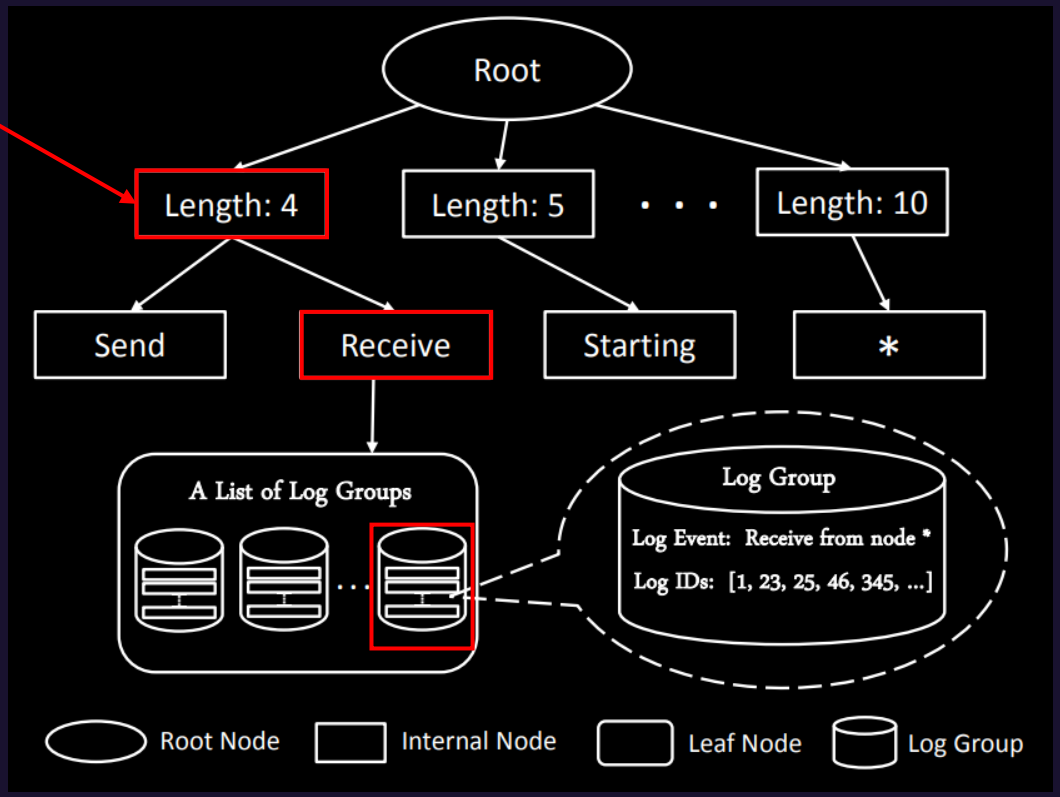
- Manually Analysis
  - Depend on individuals' experience
  - Time consuming and costly to company
  - Prone to human error
- Keywords/Regular Expression Matching
  - High chance to be inaccurate
  - Need manually construct keywords or regular expression
  - Only applicable to certain dataset

Automatic Template Parsing



Log Parsing



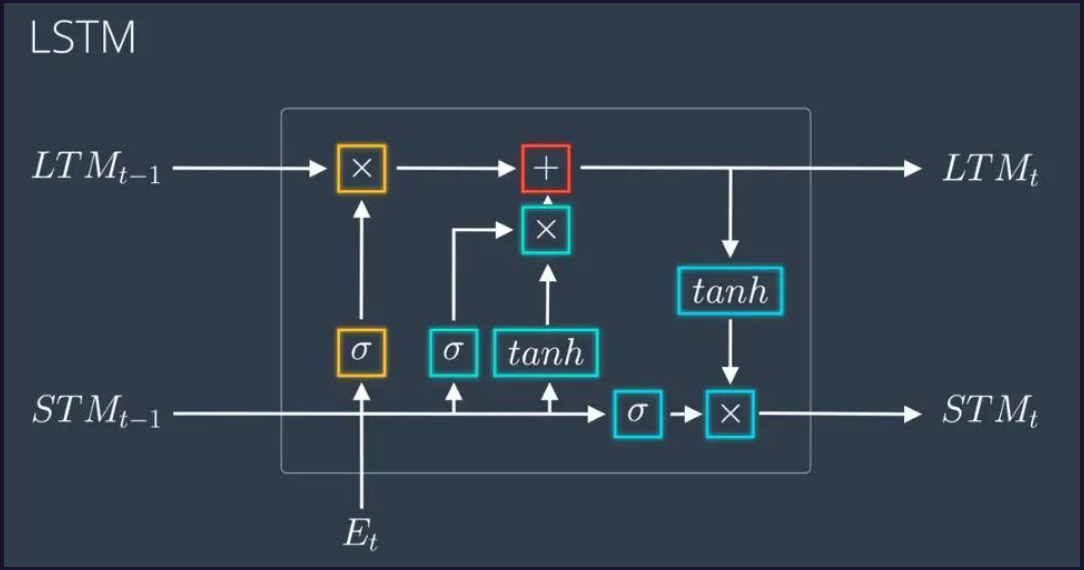


Fixed Depth Tree Parser(Drain)

Traditional Unsupervised Learning

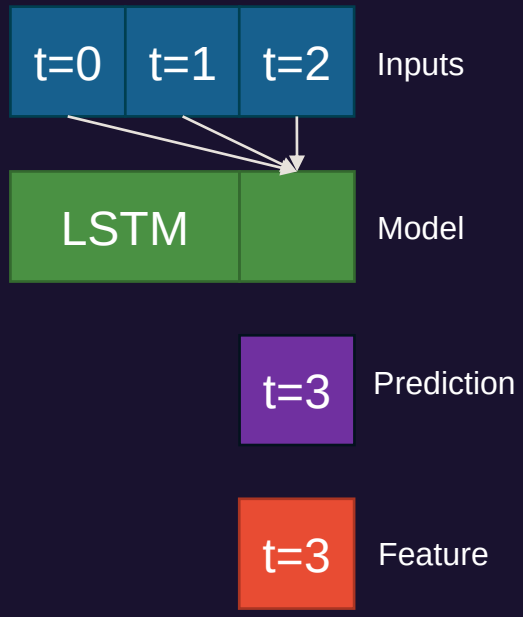
- Clustering
  - Fast at predicting after training when few clusters found
  - Not effective if anomalies don't form significant clusters
- PCA
  - Works better with numeric data
  - Need a lot of resources to scale to very large datasets
- Isolation Forest
  - Fast at predicting after training when tree is built
  - Score depends on the contamination parameter which implies the percentage of the data is anomalous is known beforehand
  - Branching bias may occur depending on how the tree is built

Unsupervised Learning

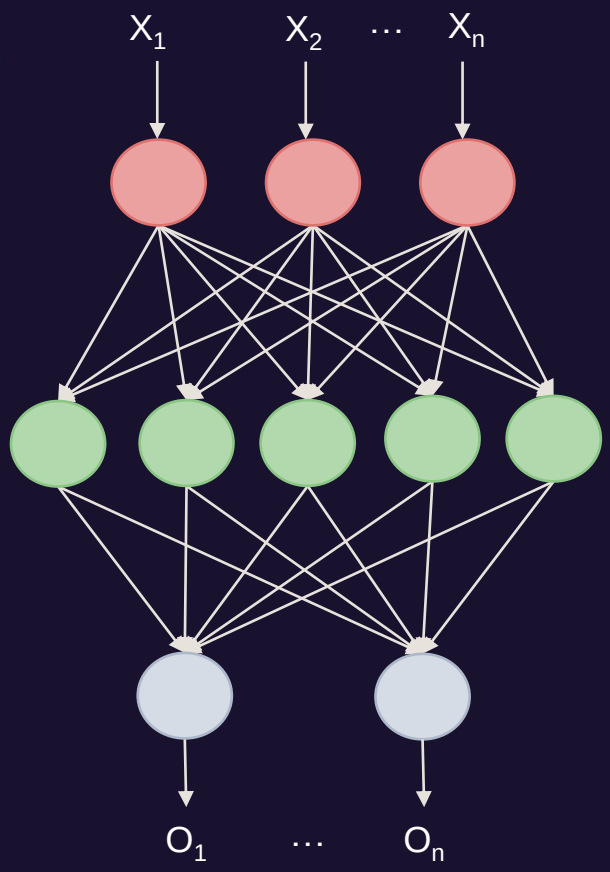


Forget Gate: Decide what information to discard from cell
Input Gate: Decide what information to add to cell
Output Gate: Decide what information in cell to output

Unsupervised Learning



Time Series Model

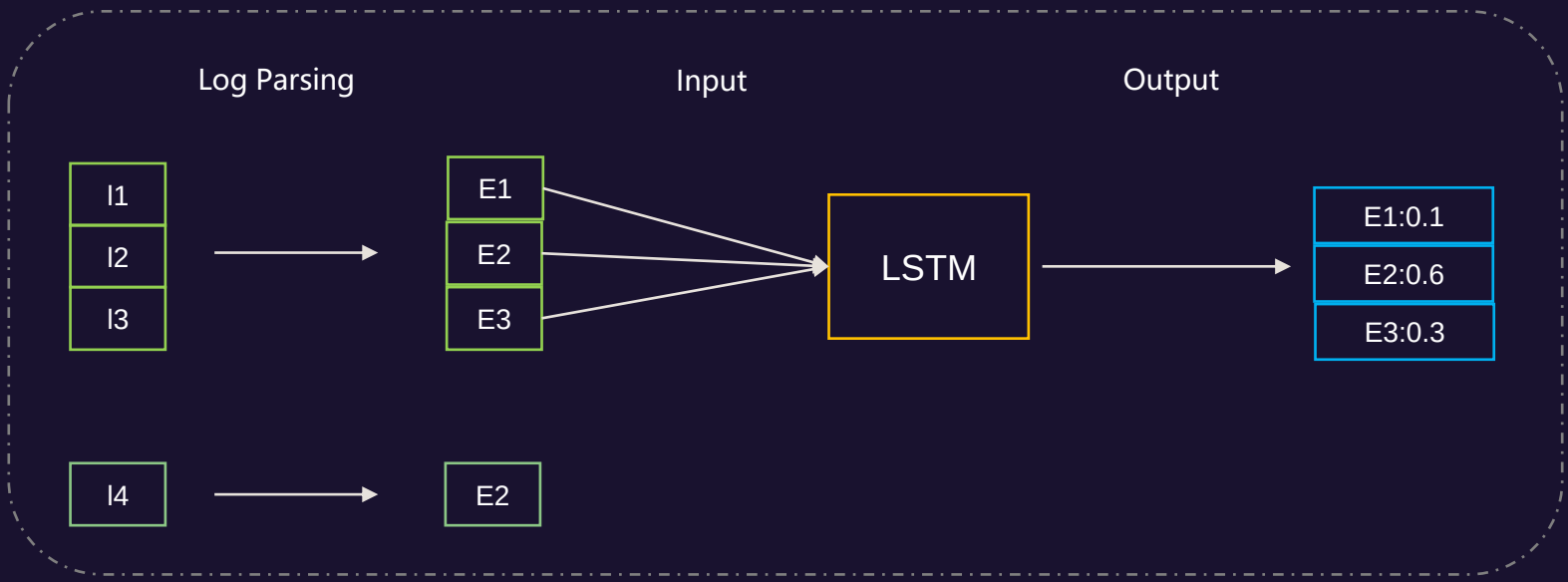


Unsupervised Training: using normal log feature in a time windows as input and use the next feature as output.
Model Predicting: produce predicted feature within a time windows and deicide the feature is matched or not.

PART 3

Implementation

Code Structure



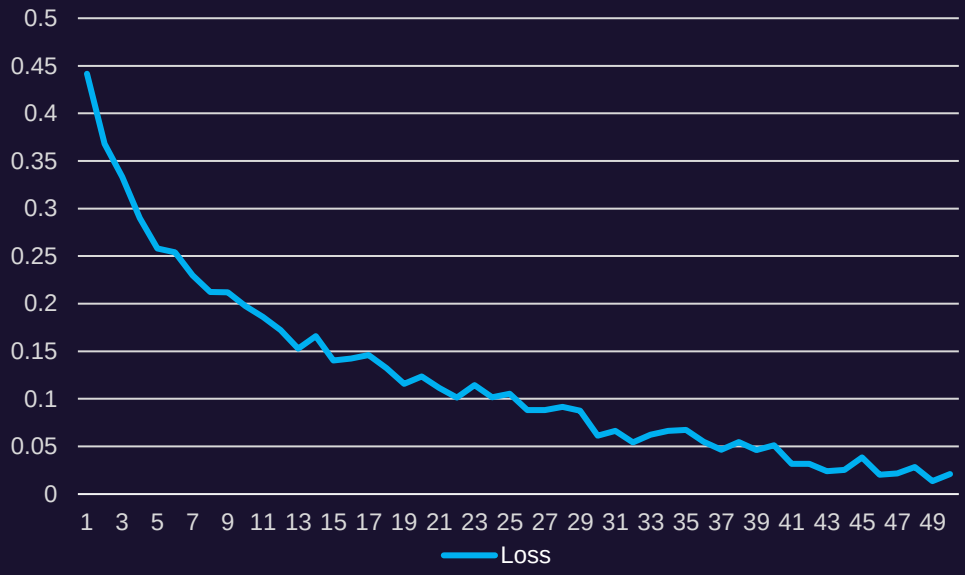
Log Parser **Log Feature** **Log Trainer** **Log Tester**

Code Implementation

- Log Parser
  - Fixed depth parser based on Drain to extract template
- Log Feature
  - Extract template id and assign index
  - Extract time slice and create sequence dataset for LSTM
- Log Trainer
  - Using sequence dataset as input and label at each window as true output
  - Train a LSTM neural network model using extracted feature entries
- Log Tester
  - Take new serial data and predict outcome, if time window label is not in candidates, the series is abnormal

Result

- Parameters
  - window size = 10
  - hidden size = 64
  - num layers = 2
  - batch size = 2048
  - optimizer = adam
- Training
  - Parsing speed = 0.5 millisecond per log
  - Training speed = 12 second per epoch
- Predicting
  - Predict speed = 65 millisecond per time slice
  - F1 score = 0.8205



PART 4

Conclusion

Conclusion

- Achievement
  - Uses AI knowledge to analyze problem and dataset effectively
  - Achieves accurate log parsing using fixed depth parser based on Drain
  - Implements LSTM detection model, resulting SOTA accuracy
  - Optimizes algorithms and parameters to fast execution
  - Create robust and well-organized code that can be deployed easily
- Outlook
  - More model tuning for LSTM model
  - Use template content alongside to provide contextual information
  - Use multiple algorithms to create fusion or boost methods
  - Use network topological information to locate faulty device

Thank You

Team Ember