Building An Agentic System for Explainable Anomaly Detection

Tung Nguyen | tnguyen844@gatech.edu | tungtngyn@gmail.com
Sentinel Devices | Georgia Institute of Technology

1. Project Overview

The primary objective for this project was to develop a system that could meet the following requirements:

1. System should be capable of identifying anomalies in multivariate time-series data.
2. System should be interpretable and should not require technical knowledge to understand outputs and results.
3. System should be able to run on low compute during inference.

The resulting system is described in the diagram below. It leverages two main components: Prophet for anomaly detection and ChatGPT for interpretation. Supporting components include a chatbot interface and a vector database to store documentation.

2. Anomaly Detection

2.1 Models & Model Evaluation

A total of 7 models were evaluated on a 2-month contiguous sample of the MetroPT dataset [1]. Models A-D are traditional, off-the-shelf algorithms and Models E-G were developed for this project. The models considered are listed below:

A. Empirical CDF Outlier Detection (ECOD) [2]
B. Angle-Based Outlier Detection (ABOD) [3]
C. Copula-Based Outlier Detection (COPOD) [4]
D. Deep Isolation Forest (DIF) [5]
E. Prophet, Univariate Approach (PUV)*
F. Prophet, Multivariate Approach (PMV)*
G. Prophet, Multivariate + Multiple Signals (PMV-MS)*

* Refer to the Appendix for a detailed description of the Prophet models

Additionally, due to the nature of the problem, it is not possible to use the recorded failures as a "true" label because the model must be able to find anomalies *before* the failure occurs. Thus, a pseudo-label method was used for evaluation. Models are rewarded for finding anomalies in the 1-week period leading up to a failure and penalized for flagging any other timestamps. Models are not rewarded for flagging timestamps within the failure period.

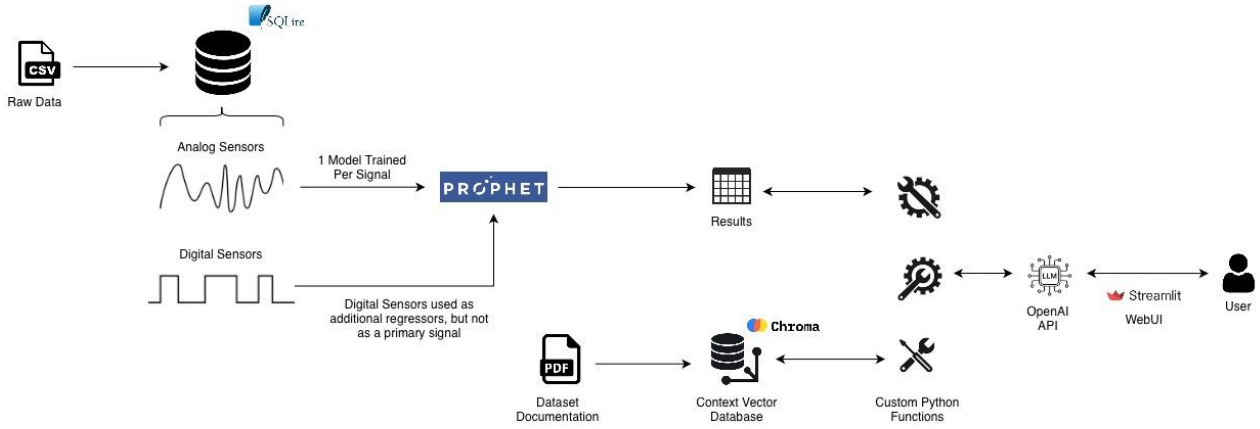

Figure 1: Full System Diagram – From Raw Data to Users

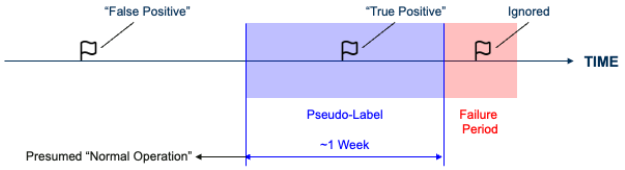

Figure 2: Pseudo-Label Method

Additionally, since all models flag *timestamps* (e.g. 8:54am) and not *timeframes* (e.g. 6:00:00am to 7:00:00am), outputs from all models were filtered for anomaly events lasting 5 minutes or longer. The idea behind this postprocessing is to remove noise; users should only be reviewing a few high-confidence, high-priority. It is also important to note here that the 1-week period is a system hyperparameter and can be tuned based on domain knowledge.

2.2 Results & Recommendations

Model performance in this study was evaluated primarily using Precision and the ratio of False Positives (FP) to True Positives (TP). Full numerical results are shown in Table 1. Precision values appear low compared to a typical classification problem, primarily due to the pseudo-label approach. To achieve 70%+ precision, as might be expected in standard classification problems, the system would need to mark almost the entire week leading up to a failure as anomalous. In general, this is undesirable and is not helpful to users. To combat this, most insights in the following section rely on *relative* scores between models.

Looking at the results, models can be split into two groups: Models with Precision >10% (3 models) and Models with Precision <10% (4 models). For the latter group, these models can be disregarded due to their low precision. For example, the PMV-MS model has the highest precision out of the 4 at ~6.7%, resulting in TP = ~400 seconds, FP = ~5000 seconds. The ratio of FP/TP for the PMV-MS (FP/TP = ~13) is over 2 times that of the PUV model (FP/TP = ~5). This is an indication that these models would be overly sensitive in a production setting, yielding too many false alarms to be useful.

In the group of models with Precision >10%, the ABOD model marked almost all points as anomalous. This is most likely because the accuracy of angle-based models depends primarily on the $n_{neighbors}$ parameter (model tested with $n_{neighbors}$ = 10). Higher $n_{neighbors}$ values might stabilize the algorithm, however, it also makes it much more expensive – up to $O(n^3)$ complexity. This computational cost alone is enough to rule out angle-based methods from consideration.

The 2 remaining models with Precision >10% are PUV and COPOD. Between these 2 models, PUV is the better model – it has ~6% higher precision and the lower FP/TP ratio of the two. Additionally, the PUV model has many operational benefits:

1. **Interpretable**; PUV's framework is easy to explain because it hides a lot of the complexity behind the uncertainty bound calculation.
2. **Customizable**; Prophet models have a lot of parameters available to tune, which help it adapt to different datasets and requirements (e.g., tune model to be more or less sensitive depending on the system). Other models (ex. COPOD) do not have many tunable parameters.
3. **Algorithmically robust**; Prophet models consider autoregressive relationships that are inherent in time-series data. Other models treat each timestamp as i.i.d as they were designed to identify anomalies in tabular data.
4. **Algorithmically resilient**; Prophet models also gracefully handles both missing data and irregular data (e.g., sensor data that's not sampled at a clean 1 Hz frequency). This is important is production environments as the sensors themselves can be unreliable.
5. **Parallelizable**; PUV fits one model per sensor data with no interaction required between models, thus, training can be done in parallel.

Based on the results above, the recommendation is to use PUV as the anomaly detection model.

Table 1: Model Metrics (Dataset Sample: Jan 1 – Mar 2) *

Model	Accuracy	Precision	Recall	F1	% of Inliers +ve	% of Outliers +ve	True Positive	False Positive	True Negative	False Negative
PUV	88.42%	16.27%	0.09%	0.18%	0.06%	0.09%	449	2310	3862670	503557
ABOD	11.54%	11.53%	99.95%	20.68%	99.99%	99.95%	503747	3864551	429	259
COPOD	88.35%	10.37%	0.13%	0.26%	0.15%	0.13%	655	5664	3859316	503351
PMV-MS	88.36%	6.72%	0.07%	0.15%	0.13%	0.07%	370	5132	3859848	503636
DIF	87.75%	5.97%	0.42%	0.79%	0.87%	0.42%	2129	33507	3831473	501877
ECOD	88.03%	3.20%	0.13%	0.25%	0.51%	0.13%	655	19809	3845171	503351
PMV	78.88%	2.18%	1.89%	2.03%	11.08%	1.89%	9536	428227	3436753	494470

* Failure #1 is roughly from Feb 28 to Mar 2 and is the only failure in this period.

3. Interpretation Interface

This project also developed an interpretation interface built in Streamlit [7] that automatically produces plain-text analyses from model outputs. The entire system, from detection to text, can be broadly divided in 4 components:

1. **Language Model:** Text generation is powered by a Large Language Model (LLM), which keeps client-side inference costs low by offloading compute to the server.
2. **Logistical Utilities:** The system includes functions for interacting with both raw data and model outputs (e.g., querying, aggregating, plotting).
3. **Context Utilities:** In addition to logistical utilities, the system has a Retrieval Augmented Generation (RAG) component, which would allow clients to add their own documentation.
4. **Batch Inference:** The PUV model can be ran with varying values for *n* (training window) and *m* (inference window) on any cadence the user wishes (e.g., daily, weekly, monthly). Results would be written to persistent storage.

A diagram of the interpretation workflow is shown below in Figure 3; it assumes batch inference is completed and documentation has been parsed and chunked into a vector database. The workflow begins with a user question at the *__start__* node, which is then fed into the retrieval (RAG) component where a similarity search is performed using Chroma [8] to pull relevant context. The user's question, context, and a system prompt is then all fed into the language model in the *assistant* node (this project uses ChatGPT [9]). The language model is bound with logistical utilities that allow it to query, aggregate, and plot data as it sees fit. Finally, the language model would formulate a plain-text response (based on the user's question, context, and any data it pulled/plotted) and send it back to the user in the *__end__* node.

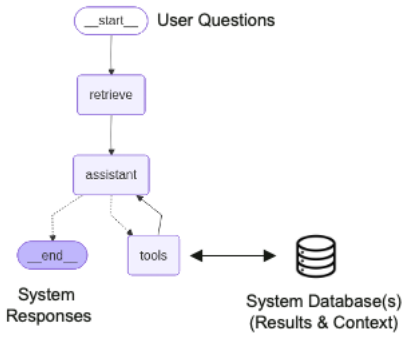

Figure 3: Interpretation Workflow

A video demo of the webapp is provided in Ref. [10]. Full application code (along with the rest of the project code) can be found in Ref. [11] and Ref. [12], respectively. For details on how to run the application, refer to the README.md in the GitHub repository.

4. Supplemental Work

4.1 Parallel Training

As discussed in Section 2.2, the PUV model is highly parallelizable since the 8 models fit do not depend on each other. A proof-of-concept parallel training script is provided in Ref. [13] using Python's multiprocessing library.

4.2 Automated Model Training

One of the main challenges in tuning anomaly detection models is the lack of labeled training data. To address this, a proof-of-concept model tuning strategy was developed. The strategy is as follows: generate synthetic anomalies by multiplying the input signal by some factor, then tune the model using the modified data and a desired performance metric.

Proof-of-concept code for model tuning is provided in Ref. [13]. The code generates synthetic anomalies using the procedure described above, assuming a triangular multiplier distribution capped at 1.25 (see Figure 4). Bayesian optimization was used to tune a sample model, though other optimization algorithms could be substituted in.

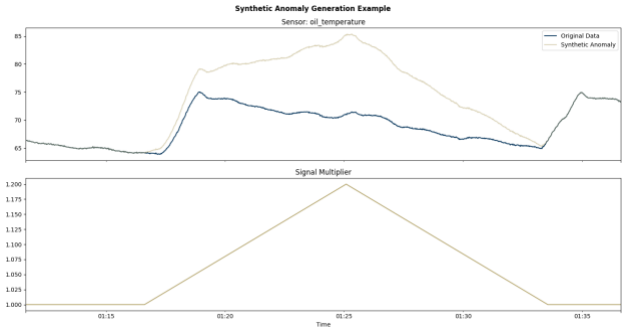

Figure 4: Synthetic Anomaly Generation

Due to computational limits, tuning was performed on a single sensor over a small number of runs with a restricted hyperparameter search space. In a production environment, this process could be scaled to a cluster with broader tuning scope and longer training durations.

5. Future Work

While the proposed system meets its primary objectives, there is still ample room for expansion. With more computational resources, future work could explore adding capabilities such as:

- **Online Learning:** Extend the current algorithm to work with real-time data instead of batched data.
- **Agentic Layouts:** Provide the LLM the capability to change the webapp layout based on the user's questions – multiple plots, different types of plots, interactive charts, etc.
- **Different Anomaly Types:** Extend the current system to handle different types of anomalies, ex. a change in frequency or trend could also be anomalous for certain systems.

Overall, this study lays the groundwork for an interpretable anomaly detection system, which can be expanded into a full-featured platform with continued research.

6. References

[1]. Veloso, B., Gama, J., Ribeiro, R., & Pereira, P. (2023). MetroPT: A Benchmark dataset for predictive maintenance. Zenodo. https://doi.org/10.5281/zenodo.6854240

[2]. Li, Z., Zhao, Y., Hu, X., Botta, N., Ionescu, C., & Chen, G. H. (2022). ECOD: Unsupervised Outlier Detection using Empirical cumulative distribution functions. IEEE Transactions on Knowledge and Data Engineering, 35(12), 12181–12193. https://doi.org/10.1109/tkde.2022.3159580

[3]. Kriegel, H., Schubert, M., & Zimek, A. (2008). Angle-based outlier detection in high-dimensional data. 14th ACM SIGKDD International Conference, 444–452. https://doi.org/10.1145/1401890.1401946

[4]. Li, Z., Zhao, Y., Botta, N., Ionescu, C., & Hu, X. (2020). COPOD: Copula-Based Outlier Detection. 2020 IEEE International Conference on Data Mining (ICDM), 1118–1123. https://doi.org/10.1109/icdm50108.2020.00135

[5]. Xu, H., Pang, G., Wang, Y., & Wang, Y. (2023). Deep isolation forest for anomaly detection. IEEE Transactions on Knowledge and Data Engineering, 35(12), 12591–12604. https://doi.org/10.1109/tkde.2023.3270293

[6]. Prophet. (n.d.). https://facebook.github.io/prophet/

[7]. Streamlit. (n.d.). https://streamlit.io/

[8]. Chroma. (n.d.). https://www.trychroma.com/

[9]. ChatGPT-4o. (n.d.). https://openai.com/index/hello-gpt-4o/

[10]. Tung Nguyen. (2025, November 12). Streamlit App Demo - ISYE/CSE/MGT 6748 - Sentinel Devices [Video]. YouTube. https://www.youtube.com/watch?v=GOR1r17o70s

[11]. GitHub Folder (n.d.). GitHub. https://github.com/tungtngyn/practicum/tree/main/04-app/

[12]. GitHub Repository (n.d.). GitHub. https://github.com/tungtngyn/practicum

[13]. Jupyter Notebook (n.d.). GitHub. https://github.com/tungtngyn/practicum/blob/main/02_unsupervised_model_tuning.ipynb

APPENDIX – Prophet Model Descriptions

Models E-G were custom designed for this specific project and are all based on the Prophet forecasting algorithm [6] from Meta. Logistically, all models operate using the same offline, batch-inference framework:

1. Train Prophet model(s) with varying parameters on a rolling window comprised of the last *n* days of data (the "training window").
2. Instead of forecasting, predict on training data and extract the uncertainty bounds for the last *m* days (the "inference window).
3. If a data point is *outside* the uncertainty bounds by more than 10%, flag the data point as anomalous.

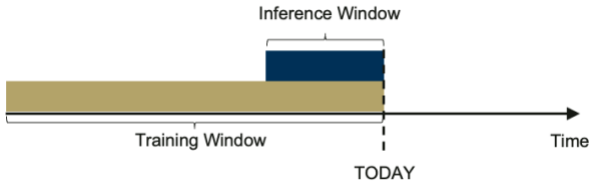

Figure A1: Batch Inference Framework

This inference framework is designed to be ran on a regular cadence. For example, if the inference cadence is weekly, run the inference process to evaluate the last week's data by training on the last month's data. The inference and training windows are system hyperparameters and can be tuned.

In the PUV variant (Model E), 1 model is fit per analog sensor. For a data point to be considered anomalous, 5 or more sensors must be outside of the model's uncertainty bounds by more than 10%. Digital sensor data is ignored in this variation.

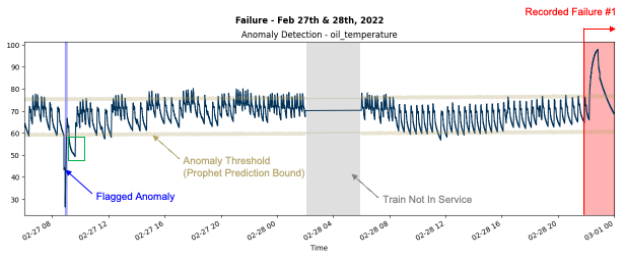

Figure A2: PUV (Model E) Output

The anomalous period is highlighted in blue. Note the green-boxed period is not flagged as an anomaly even though the *oil_temperature* signal is out of bounds because the requirement is 5 or more signals must be out of bounds at the same time.

Table A1: Model E - Prophet Univariate (PUV) Algorithm

set *start*, *end* based on rolling *n*-day window
for each analog (continuous) sensor
  fit Prophet Univariate model
    on sensor data from *start* to *end*

  predict on historical data from *start* to *end*
  record prediction's lower and upper bound

for each second from *start* to *end*:
  for each analog (continuous) sensor:
    if sensor outside of 1.1*(prediction bounds):
      set *pred*(sensor) = 1
    else:
      set *pred*(sensor) = 0

  if sum(*pred*) > 5:
    set *global_pred*(second) = 1
  else:
    set *global_pred*(second) = 0

filter *global_pred* for contiguous events >5 mins.

In the PMV variant (Model F), a single model is trained on the analog signal most correlated with the pseudo-label. Prophet *regressors* (e.g. additional linear terms) are added for every other signal. The correlation matrix for the test period looks like:

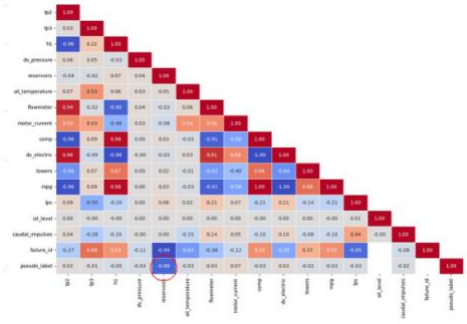

Figure A3: Correlation Matrix

The *reservoirs* sensor (circled in red) is most correlated with the pseudo-label for the given period. Thus, a single Prophet model is fit with this sensor's data. All other sensors are used as additional regressors. This algorithm is most suitable for systems that rely on a single critical sensor, as all other sensors are considered supporting data (modeled as linear terms).

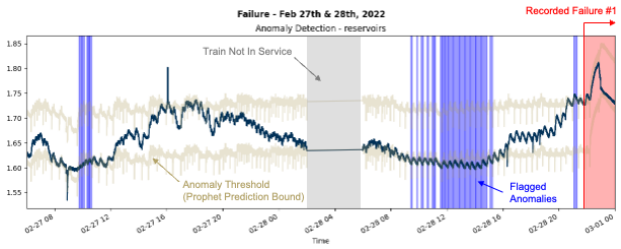

Figure A4: PMV (Model F) Output

As shown above, the PMV model is much more sensitive than the PUV model. This difference is intuitive – PUV's ensemble structure dampens noise from model outputs, while PMV's single-model design does not.

Table A2: Model F - Prophet Multivariate (PMV)

Algorithm

calculate covariance matrix
set *primary_signal* = sensor most correlated with pseudo-label using training data

set *start*, *end* based on rolling *n*-day window
fit Prophet Multivariate Model
    on sensor data from *start* to *end*
    add 1 regressor for every other sensor

predict on historical data from *start* to *end*
record prediction's lower and upper bound

for each second from *start* to *end*:
    if *primary_signal* outside of 1.1*bounds:
        set *pred* = 1
    else:
        set *pred* = 0

filter *pred* for contiguous events >5 mins.

The PMV-MS variant (Model G) is a combination of the 2 variants above: 1 model is trained per analog sensor (similar to PUV), each model trained has additional regressors added for every other sensor (similar to PMV). For example, the *oil_temperature* model from Figure A5 would have additional regressors for the 7 other analog sensors + 8 regressors for the 8 digital sensors, for a total of 15 regressors.

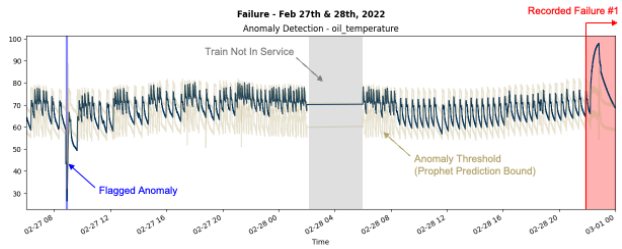

Figure A5: PMV-MS (Model G) Output

The PMV-MS model's output is like the PUV model output, except with noisier uncertainty bounds.

Table A3: Model G
Prophet Multivariate + Multiple Signals (PMV-MS)

Algorithm

set *start*, *end* based on rolling *n*-day window
for each analog (continuous) sensor
    fit Prophet Univariate model
        on sensor data from *start* to *end*
        add 1 regressor for every other sensor

    predict on historical data from *start* to *end*
    record prediction's lower and upper bound

for each second from *start* to *end*:
    for each analog (continuous) sensor:
        if sensor outside of 1.1*(prediction bounds):
            set *pred*(sensor) = 1
        else:
            set *pred*(sensor) = 0

    if sum(*pred*) > 5:
        set *global_pred*(second) = 1
    else:
        set *global_pred*(second) = 0

filter *global_pred* for contiguous events >5 mins.