Benchmarking Machine Learning Anomaly Detection Methods on the Tennessee Eastman Process Dataset

Naixin Lyu, Suraj Botcha, Eesha Kulkarni, Shreya Pagaria, Victor Alves, Ethan M. Sunshine, and John R. Kitchin*

Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, United States

E-mail: jkitchin@andrew.cmu.edu

Abstract

The Tennessee Eastman Process (TEP) dataset is a standard benchmark for process control and fault diagnosis due to its complex, multivariate dynamics. We evaluate a broad range of machine learning and deep learning methods for multivariate time-series anomaly detection on TEP, covering 17 of its 20 fault types. After preprocessing the 55-variable dataset, we benchmark classical models such as XGBoost and sequence models including LSTM classifiers, LSTM-FCN, a convolutional transformer, and our proposed Transformer-Kalman (TransKal) classifier. We also assess hybrid LSTM and convolutional autoencoder approaches. Supervised models were trained on combined normal and faulty data, while semi-supervised models used only normal data. Finally, we evaluate the models using an independent dataset to assess generalizability. Several models achieved 99%+ multiclass accuracy, but showed fragility on the independent dataset.

Keywords

Anomaly Detection, Fault Detection, Process Monitoring, Machine Learning, Industry 4.0

Introduction

The advent of Industry 4.0, marked by widespread sensor deployment and enhanced data generation capabilities, has fundamentally reshaped chemical manufacturing. This transformation enables enhanced operational safety, efficiency, and asset reliability via advanced analytics.[1] Among these developments, Fault Detection and Diagnosis (FDD) systems have become a cornerstone of predictive maintenance strategies in large-scale chemical facilities. Effective FDD frameworks can ideally identify early-stage process deviations before they develop into major operational disruptions, thereby mitigating unplanned downtime and associated economic impacts.[2,3]

To evaluate FDD methodologies, researchers frequently employ complex industrial benchmarks such as the Tennessee Eastman Process (TEP), which simulates the nonlinear dynamics and feedback control loops of a chemical plant.[4,5] The choice of methodology is dictated by the learning paradigm, which in turn depends on the availability of labeled data and the desired diagnostic goal. At one end of this spectrum, supervised learning relies on comprehensive datasets containing accurately labeled instances for both normal and faulty operation, enabling multi-class fault diagnosis (i.e., identifying which specific fault is occurring). At the opposite end lies semi-supervised learning, where models are trained solely on normal operating data and tasked with detecting deviations that indicate potential faults.[6] This paradigm is particularly practical for industrial settings, where exhaustive fault labeling is rarely feasible.

Historically, FDD research began with Multivariate Statistical Process Monitoring techniques including Shewhart control charts, as well as Principal Component Analysis (PCA), in which the latter models the variance of normal data to establish a statistical baseline.[7,8]

However, such linear methods proved inadequate in modeling the intricate systems that present challenging nonlinear dynamics, including TEP, often failing to detect a significant portion of faults and required extensive manual feature engineering.[9,10] The field then shifted to classical machine learning (e.g., Support Vector Machines (SVMs)[11] and XGBoost[12]), which demonstrated superior classification but struggled with temporal dependencies and scalability.[13]

The introduction of deep learning enabled automatic hierarchical feature extraction from raw sensor data. This advanced FDD on both fronts. For the semi-supervised detection task, models like Autoencoders (AEs) gained prominence.[14] By learning to reconstruct only normal data, they can flag any data they fail to reconstruct as an anomaly. For the supervised diagnosis task, researchers turned to models designed to process sequential data, as a fault's unique signature often unfolds over time. Models such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and, more recently, transformer architectures have proven exceptionally effective at learning the distinct temporal patterns required to diagnose the specific fault type.[15–17]

Despite this proliferation of advanced models, a critical gap persists in the literature: it is virtually impossible to fairly compare their performance. Published results are often inconsistent and incomplete. For example, while one study[18] on a supervised LSTM model reports an exceptional accuracy of 99.2% , and second one[1] highlights a recall of 99.88% using a Gated Recurrent Unit (GRU), a third study leveraging image processing techniques reports a leading F1-score of 98.81%.[19] These results are not comparable as they are derived from different data splits and, crucially, potentially inadequately balanced dataset subsets. Many traditional TEP benchmarks are derived from single simulation runs, where normal operating data vastly outweighs the limited samples available for each fault class, skewing model evaluation. An operator cannot know if the model with the highest accuracy also suffers from low recall on a critical fault, or if the model with the highest recall generates an unacceptable number of false alarms.

Considering this gap, our study makes two primary contributions. First, we establish a unified framework for reproducible and fair comparison of a set of models. This begins with the creation of a new, curated TEP dataset derived from hundreds of independent simulation runs rather than a single continuous dynamic runs. This approach allows us to construct distinct training, validation, and test sets that strike a deliberate balance between normal and faulty data, avoiding the biases of traditional single-run benchmarks. We then apply a standardized evaluation protocol (i.e., consistent data splits and a comprehensive set of metrics) to all models. Second, we introduce and benchmark novel hybrid architectures against established baselines within this rigorous framework. By systematically evaluating models for both multi-class diagnosis and binary detection, our research provides a clear and comprehensive benchmark to guide the design of robust and reliable FDD systems for demanding industrial environments.

Methods

Benchmark Dataset: The Tennessee Eastman Process (TEP)

The Tennessee Eastman Process (TEP) serves as the benchmark for this study. The typical process flow diagram (PFD) and a typical control structure[20,21] is depicted in Figure 1. TEP has been extensively used in the process control community for plantwide control studies,[22–25] being considered by researchers and practitioners one of the first successful and complete benchmarks of control structure studies of plantwide systems. In addition, its use as a benchmark for fault diagnosis algorithms became common in the previous two decades as reported in the literature.[26,27] The system is described in Figure 1 with a typical control structure, including regulatory and supervisory control layers, as well as the enumerated measurements typically described in the TEP benchmarks found in the literature.[20,21] TEP simulates a chemical plant with dynamics and interactions representative of industrial settings,[4] while simulating a chemical process that produces two desired liquid products (G and H) from four gaseous reactants (A, C, D, and E), with an inert component (B) and byproduct (F), following the reaction network described in Eq. 1.

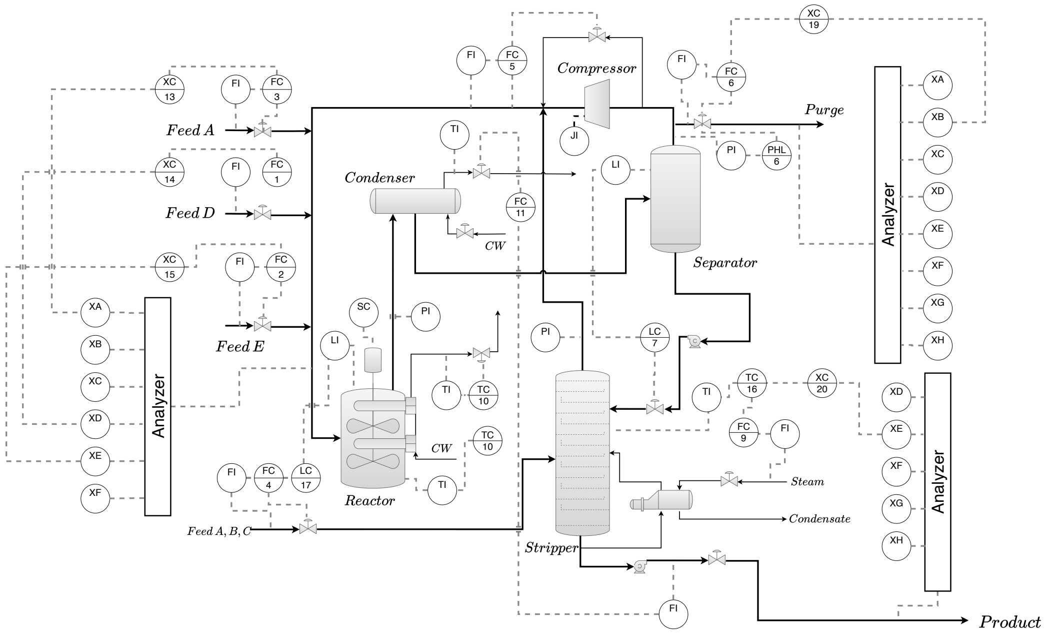


Figure 1: TEP process schematic. Bolder streams represent main process streams. Grey dashed lines represent control signals. Recycles and utility streams are represented by solid, non-bolded lines. The schematic presented here is based on Ref.[20] with control structure # 2 as reported in this reference, originally available in.[21]

$$
\begin{aligned}
A(g) + C(g) + D(g) &\rightarrow G(liq), \\
A(g) + C(g) + E(g) &\rightarrow H(liq), \\
A(g) + E(g) &\rightarrow F(liq), \\
3D(g) &\rightarrow 2\ F(liq).
\end{aligned}
\tag{1}
$$

In addition, the process comprises five primary operational units: a reactor, a product condenser, a vapor-liquid separator, a product stripper, and a recycle compressor. The reactor converts multiple gaseous and liquid reactants into the desired products (G and H) through an exothermic reaction network as described in Eq 1. Afterward, the reactor products are sent to the condenser, which cools the reactor dowstream and condenses overhead vapors. This resulting two-phase stream is sent to the separator, where non-condensable gases are recycled while the condensed liquid stream proceeds to the product stripper. In the stripper, light components are removed to achieve target product purity specifications, and the overhead vapors are also recompressed and recycled back to the reactor.

Table 1: Description of Tennessee Eastman Process disturbance variables (IDVs), based on Ref.[20]

Variable	Description	Type
IDV1	A/C Feed Ratio	Step
IDV2	B Composition	Step
IDV3	D Feed Temperature	Step
IDV4	Cooling Water Inlet Temperature (Reactor)	Step
IDV5	Cooling Water Inlet Temperature (Condenser)	Step
IDV6	A Feed	Step
IDV7	C Header Pressure Loss	Step
IDV8	A, B, C Feed Composition	Random Variation
IDV9	D Feed Temperature	Random Variation
IDV10	C Feed Temperature	Random Variation
IDV11	Cooling Water Inlet Temperature (Reactor)	Random Variation
IDV12	Cooling Water Inlet Temperature (Condenser)	Random Variation
IDV13	Reaction Network Kinetics	Slow Drift
IDV14	Cooling Water Valve (Reactor)	Sticking
IDV15	Cooling Water Valve (Condenser)	Sticking
IDV16	Unknown	
IDV17	Unknown	
IDV18	Unknown	
IDV19	Unknown	
IDV20	Unknown	
IDV21	Valve stuck at steady state operating point	Constant Position

The TEP simulation includes 20 programmed process faults (IDV1-IDV20) in addition to the normal operating condition (IDV0), described in Table 1, which was based directly on the descriptions detailed in the literature.[20] These faults represent a variety of disturbance types, such as step changes, random variations, slow drifts, and sticking valves. Following common practice in TEP-based FDD research,[3] this study focuses on 17 of the 20 faults, excluding faults IDV3, IDV9, and IDV15 due to their minimal deviation from normal process behavior, resulting in longer detection times.[28] The multiclass classification task in this work, therefore, involves identifying the normal state and the remaining 17 distinct fault types.

Data Preparation

The TEP process is monitored through 52 variables, consisting of 11 manipulated variables (e.g., valve positions, flow rates) and 41 measured variables (e.g., temperature, pressure, level, and component concentrations). A sampling interval of 3 minutes was used for most process measurements.

While standard TEP datasets often consist of a single simulation run for training and testing, this approach provides insufficient data for training robust deep learning models and may lead to biased performance evaluation.[1] To overcome this limitation, this study utilized the enriched TEP dataset developed by Rieth et al.[5] This extended dataset was generated by running the simulation with different random seeds, providing hundreds of independent runs for the normal state and for each fault type. This data-rich environment allows for the creation of distinct training, validation, and testing sets, reducing the bias between sample statistics and true population parameters and enabling a more rigorous evaluation of FDD models. We established a standardized preprocessing pipeline to prepare the raw TEP data for ingestion by temporal deep learning models. This pipeline involves data scaling, sequence generation, and a data splitting strategy.

In the following subsections we provide a narrative description of each of these, but note that all of the notebooks are available in the supporting information for complete details of the implementations.

Data Scaling

The data were normalized using z-score normalization. Importantly, we calculated the mean and variance parameters exclusively from the training dataset and then applied to the validation and test sets. This prevents data leakage from the hold-out sets into the training process, ensuring an unbiased evaluation of the model's generalization capabilities.

Sequence Generation for Temporal Models

The raw time-series data were transformed into supervised learning samples using a sliding window approach with a stride of one. This technique segments a continuous time-series into overlapping sequences of a fixed window length w. Each window, containing w consecutive time steps of the 52 process variables, becomes a single input sample, and the corresponding label is assigned based on the fault state at the final time step of that window. Importantly, windows were generated independently within each simulation run to prevent temporal information leakage across independent experiments. The optimal window length for each model was determined through Bayesian hyperparameter optimization using Optuna, searching over the range of 20–40 time steps. Given the 3-minute sampling interval, this corresponds to 60–120 minutes of process history. The resulting window lengths ranged from 38 to 40 time steps depending on the model architecture. This procedure transforms the original data into a 3D tensor of shape (samples, timesteps, features), which is the required input format for temporal models such as LSTMs and Transformers.[3]

Train-Validation-Test Split

While the enriched dataset by Rieth et al.[29] provides a large pool of simulation runs, we sampled from this pool to create balanced datasets for training, validation, and testing. Faults 3, 9, and 15 were excluded as they are too subtle to detect reliably, leaving 17 fault types for analysis. For supervised multiclass classification, datasets were constructed to contain equal numbers of samples per class (18 classes: normal plus 17 faults). The training set comprises 100 simulation runs per class (1,800 total runs; 864,000 samples), and the validation set comprises 50 runs per class (900 total runs; 432,000 samples). The test set contains 200 runs per class (3,600 total runs; 2,880,000 samples). For semi-supervised anomaly detection, following the standard paradigm where models learn only normal behavior,[6] the training and validation sets contain exclusively fault-free data (100 and 50 runs, respectively). A separate binary test set evaluates anomaly detection performance using 120 normal runs plus 50 faulty runs per fault type (970 total runs; 795,200 samples). In the test simulations, the process operates under normal conditions for eight hours before a fault is introduced at sample 161.

Hyperparameter tuning

Hyperparameters for all models were optimized using Bayesian optimization via the Optuna framework[30] with a Tree-structured Parzen Estimator (TPE) sampler. Each model was tuned over 50 trials with the objective of maximizing the weighted F1 score on the validation set. To reduce computational cost, tuning was performed on 50 of the training data, sampled by complete simulation runs to preserve temporal structure. A MedianPruner terminated unpromising trials early based on intermediate validation performance. For deep learning models, early stopping with a patience of 5 epochs was applied within each trial. The optimal hyperparameters from tuning were then used for final model training on the complete dataset.

Metrics

Multiple evaluation metrics were employed to assess model performance. For multiclass fault classification, we report Accuracy, Balanced Accuracy, Precision, Recall, and the F1-Score. All metrics except Accuracy were computed using weighted averaging, where each class's contribution is weighted by its support (the number of true instances for that class). We also report macro-averaged F1 scores, which compute an unweighted average treating all classes equally. Per-class F1 scores are provided to identify which fault types are most challenging to classify. Because our datasets are perfectly balanced across all 18 classes, the weighted and macro F1 scores are equivalent, and Accuracy equals Balanced Accuracy.

For binary anomaly detection using autoencoder models, we additionally report the Area Under the Receiver Operating Characteristic curve (ROC-AUC) and the Area Under the Precision-Recall curve (PR-AUC). These threshold-independent metrics evaluate the models' ability to rank anomalous samples higher than normal samples across all possible decision thresholds. The anomaly score for autoencoders is the reconstruction error, with higher errors indicating potential faults.

Models

We evaluate five supervised classifiers for multiclass fault diagnosis and two semi-supervised autoencoder architectures for binary anomaly detection.

Multiclass Classification Models

XGBoost XGBoost (eXtreme Gradient Boosting)[31] is a gradient boosted decision tree ensemble that serves as our classical machine learning baseline. Unlike the deep learning approaches, XGBoost treats each time step independently without explicit temporal modeling. The model uses a sequential ensemble approach where each tree corrects the errors of previous trees, with regularization terms to prevent overfitting.

LSTM The Long Short-Term Memory (LSTM) network[15] is a recurrent neural network designed to capture long-term dependencies in sequential data. Our architecture consists of stacked LSTM layers followed by dropout regularization and a fully connected classification head. The model processes sliding windows of sensor measurements and outputs fault class predictions based on the learned temporal representations.

LSTM-FCN The LSTM-FCN architecture[32] combines an LSTM branch for sequential modeling with a Fully Convolutional Network (FCN) branch for local pattern detection. The FCN branch employs three 1D convolutional layers with increasing dilation rates (1, 2, 4) to capture multi-scale temporal patterns, followed by global average pooling. The outputs of both branches are concatenated before the final classification layer, allowing the model to leverage both recurrent and convolutional feature representations.

CNN-Transformer This hybrid architecture combines 1D convolutional layers for local feature extraction with a Transformer encoder[33] for global temporal attention. The input sequence first passes through two 1D convolutional layers with batch normalization, then positional encodings are added before the Transformer encoder layers. Multi-head self-attention enables the model to attend to relevant time steps across the entire sequence. Global average pooling aggregates the temporal features before classification.

TransKal The Transformer-Kalman (TransKal) model extends the Transformer classifier with an adaptive Kalman filter for temporal smoothing of predictions. The base architecture consists of a linear embedding layer with layer normalization, positional encoding, Transformer encoder layers, and a two-layer classification head. During inference, a Kalman filter operates on the softmax probability distributions (not class indices) to smooth predictions over time, with adaptive process and measurement noise parameters.

Binary Anomaly Detection Models

The following autoencoder models are trained exclusively on normal operating data and detect anomalies based on reconstruction error. Samples with reconstruction error exceeding a learned threshold are classified as faults.

LSTM Autoencoder The LSTM Autoencoder[34] learns to compress and reconstruct sequences of normal process behavior. The encoder consists of LSTM layers that compress input sequences into a fixed-size latent representation. The decoder, also composed of LSTM layers, reconstructs the original sequence from this latent vector. Anomalies are detected when the mean squared error between the input and reconstruction exceeds a threshold determined from the training data distribution.

Convolutional Autoencoder The Convolutional Autoencoder uses 1D convolutional layers for both encoding and decoding. The encoder applies strided convolutions to progressively reduce the temporal dimension while increasing feature channels, compressing the input into a latent representation. The decoder uses transposed convolutions to reconstruct the original sequence. This architecture efficiently captures local temporal patterns and trains significantly faster than the LSTM-based alternative while achieving competitive anomaly detection performance.

Hidden Markov Model Classification Filter

To exploit the temporal consistency inherent in process fault conditions, we applied a Hidden Markov Model (HMM) classification filter as a post-processing step to smooth the predictions of trained classifiers. This approach is motivated by the observation that industrial faults typically persist over time rather than appearing and disappearing instantaneously, yet instantaneous classifiers may produce temporally inconsistent predictions. The HMM filter treats the true fault state as a hidden variable and the classifier's predictions as noisy observations of this state. The filter performs sequential Bayesian updates to estimate the most likely fault state given the history of observations. Specifically, the filter comprises two components:

Emission Model The emission probability $P(\hat{y}_t|y_t)$, representing the probability of observing prediction $\hat{y}_t$ given true state y_t, is derived from the classifier's confusion matrix on the validation set. This captures the characteristic error patterns of each classifier.

Transition Model The transition probability $P(y_t|y_{t-1})$ models the temporal dynamics of fault states. We employ a "sticky" transition model controlled by a stickiness parameter $\gamma \in [0, 1]$:

$$P(y_t = j|y_{t-1} = i) = \begin{cases} \gamma & \text{if } i = j \\ \frac{1-\gamma}{K-1} & \text{if } i \neq j \end{cases} \tag{2}$$

where K is the number of fault classes. Higher values of γ increase the model's resistance to state changes, providing stronger temporal smoothing. We evaluated $\gamma \in \{0.70, 0.85, 0.95\}$ to identify the optimal smoothing level for each classifier.

At each time step, the filter updates the belief state using Bayes' rule, combining the prior (based on the previous belief and transition model) with the likelihood (based on the current observation and emission model). The filtered prediction is the class with maximum posterior probability. The filter state is reset at the beginning of each simulation run to prevent information leakage across independent trajectories.

Generalization Evaluation on Independent Data

To assess the generalization capability of the trained models beyond the original test set, we generated an independent evaluation dataset using the TEP simulator `tep-sim`.[35] This software package wraps the original Fortan code in a Python module for ease of data generation. This evaluation addresses a critical question for industrial deployment: how well do models trained on historical data perform when process conditions exhibit natural variation?

The independent dataset was generated with the following characteristics: 1) Different random seeds: All simulation runs used random seeds distinct from those in the training, validation, and test sets, ensuring statistical independence; 2) Identical fault scenarios: The same 18 fault classes (normal operation plus 17 fault types) were simulated to enable direct comparison; 3) Consistent simulation protocol: Each simulation ran for 48 hours with faults injected at the 8-hour mark, matching the test set configuration.

The resulting dataset comprises 1,978,215 samples for multiclass evaluation and 1,157,585 samples for binary anomaly detection. All models were evaluated using their trained weights without any retraining or threshold adjustment, providing a rigorous test of out-of-distribution generalization. This evaluation reveals which architectural choices lead to robust feature learning versus overfitting to dataset-specific characteristics.

Use of Generative AI

The initial experimental design, model architecture selection, and research direction for this work were developed by the authors. Claude Code, an AI-powered coding assistant developed by Anthropic, was subsequently used to orchestrate consistent hyperparameter tuning, model training, and evaluation workflows across all seven model architectures. This included generating standardized notebook templates, ensuring reproducible experimental protocols, and maintaining consistent evaluation metrics throughout the study. Claude Code also assisted in drafting and revising portions of the manuscript text. All numerical results, figures, and performance metrics reported in this paper are generated directly from the Jupyter notebooks provided in the Supporting Information, which can be executed to reproduce the complete analysis.

Additionally, we provide a CLAUDE.md file in the Supporting Information. This is a markdown file that contains details about the repository structure, and the file is used by Claude Code to "learn" about the project. This facilitates others using Claude Code (or other LLMs) to interact with the work.

Results and Discussion

This section presents and discusses the performance of the developed FDD models. We first present results from hyperparameter tuning and then final training. We show the results from the Hidden Markov Model filtering, and finally results from the independent dataset.

Hyperparameter Optimization Results

Table 2 summarizes the hyperparameter optimization results for all models. The best validation F1 scores ranged from 0.924 for XGBoost to 0.994 for the Convolutional Autoencoder, with deep learning models consistently outperforming the gradient boosting baseline during tuning. The optimal hyperparameters for each model are presented in Tables 3 and 4.

Table 2: Hyperparameter optimization summary. All models were tuned using 50 Optuna trials with weighted F1 as the objective.

Model	Task	Best Val F1	Tuning Time
XGBoost	Multiclass	0.924	7h 28m
LSTM	Multiclass	0.988	6h 37m
LSTM-FCN	Multiclass	0.990	11h 1m
CNN-Transformer	Multiclass	0.989	8h 22m
TransKal	Multiclass	0.988	12h 6m
LSTM-AE	Binary	0.982	2h 34m
Conv-AE	Binary	0.994	1h 11m

Table 3: Optimal hyperparameters for multiclass classification models.

Model	Hyperparameters
XGBoost	n_estimators=499, max_depth=6, learning_rate=0.196, subsample=0.967, colsample_bytree=0.841, min_child_weight=8, gamma=0.564, reg_alpha=0.329, reg_lambda=0.063
LSTM	sequence_length=39, hidden_size=32, num_layers=3, dropout=0.066, learning_rate=0.00196, batch_size=32
LSTM-FCN	sequence_length=40, lstm_hidden=24, lstm_layers=1, dropout=0.437, learning_rate=0.00139, batch_size=64
CNN-Transformer	sequence_length=39, conv_filters=32, kernel_size=3, d_model=32, nhead=4, num_encoder_layers=1, dim_feedforward=128, dropout=0.240, learning_rate=0.00241, batch_size=32
TransKal	sequence_length=40, d_model=32, nhead=4, num_layers=2, dropout=0.195, learning_rate=0.000428, batch_size=64, kalman_Q=5.04×10^{-5}, kalman_R=0.0223

Table 4: Optimal hyperparameters for binary anomaly detection models.

Model	Hyperparameters
LSTM-AE	sequence_length=38, hidden_size=64, num_layers=1, latent_size=32, dropout=0.073, learning_rate=0.000320, batch_size=32, threshold_percentile=97.95
Conv-AE	sequence_length=40, conv_filters=64, kernel_size=3, latent_filters=128, use_transformer=False, dropout=0.119, learning_rate=0.00853, batch_size=128, threshold_percentile=98.65

Optimal Sequence Length

A consistent finding across all temporal models was the preference for longer sequence lengths. The optimal window lengths fell between 38–40 time steps for all architectures, corresponding to approximately 114–120 minutes of process history at the 3-minute sampling interval. This suggests that fault signatures in the TEP dataset manifest over extended time horizons, and models benefit from access to this broader temporal context. Shorter windows (20–30 steps) consistently yielded lower validation performance across all model types.

Model Complexity

Contrary to common assumptions, simpler model configurations frequently outperformed more complex alternatives. For the CNN-Transformer, a single encoder layer achieved better validation performance than configurations with two or three layers. Similarly, the LSTM achieved optimal results with a smaller hidden size (32 units) combined with more layers (3 layers), rather than larger hidden sizes with fewer layers. The Convolutional Autoencoder performed best without the optional Transformer layer in its latent space (use_transformer=False). These findings suggest that the fault classification task does not require highly complex models, and that regularization through architectural simplicity is beneficial.

Regularization Requirements

Optimal dropout rates varied substantially across architectures, reflecting their different regularization needs. The LSTM required minimal dropout (0.066), indicating that the model benefits from utilizing its full capacity. In contrast, LSTM-FCN required substantially higher dropout (0.437), likely due to the increased parameter count from its dual-branch architecture. The CNN-Transformer fell between these extremes (0.240). For the autoencoder models, low dropout rates (0.073–0.119) were optimal, as excessive regularization impaired reconstruction fidelity.

Learning Rates

Optimal learning rates spanned two orders of magnitude across models. XGBoost used the highest effective learning rate (0.196), consistent with gradient boosting's sequential correction mechanism. Among deep learning models, the Convolutional Autoencoder converged fastest with a learning rate of 0.00853, while TransKal required the lowest rate (0.000428) to ensure stable integration of the Kalman filter gradients. The remaining models clustered around 0.001–0.002, typical for Adam optimization of neural networks.

Anomaly Detection Thresholds

For the autoencoder-based anomaly detectors, the optimal threshold percentile for classifying reconstruction errors as anomalies fell near the 98th percentile (97.95% for LSTM-AE, 98.65% for Conv-AE). This relatively high threshold reflects the need to minimize false positives while maintaining sensitivity to genuine faults, balancing the trade-off between precision and recall in the binary detection setting.

Computational Efficiency

Total hyperparameter tuning required approximately 49 hours across all seven models. The Convolutional Autoencoder was the most efficient to tune (1.2 hours), benefiting from its simple architecture, small training set (normal data only), and fast convergence with large batch sizes (128). TransKal required the longest tuning time (12.1 hours) due to the additional Kalman filter parameters and the computational overhead of the filtering step during each trial. These tuning times represent a one-time cost, as the optimized hyperparameters were subsequently used for final model training on the complete dataset.

Final Model Performance

All models were trained using the optimized hyperparameters on the complete training dataset and evaluated on the held-out test set. This section presents the performance results for both multiclass fault classification and binary anomaly detection tasks.

Multiclass Fault Classification

Table 5 and Figure 2 present the test set performance for all multiclass classification models. LSTM-FCN achieved the highest accuracy (99.37%) and F1 score (0.9937), followed closely by CNN-Transformer (99.20%), LSTM (99.14%), and TransKal (99.09%). XGBoost, despite being optimized with 499 estimators and extensive regularization, achieved substantially lower performance (93.91%), demonstrating the importance of temporal modeling for this task.

Table 5: Test set performance for multiclass fault classification (18 classes). All deep learning models significantly outperform the XGBoost baseline.

Model	Accuracy	F1 (Weighted)	Precision	Recall	Training Time
LSTM-FCN	**99.37%**	**0.9937**	0.9938	0.9937	26m 14s
CNN-Transformer	99.20%	0.9920	0.9921	0.9920	67m 33s
LSTM	99.14%	0.9914	0.9917	0.9914	20m 5s
TransKal	99.09%	0.9909	0.9910	0.9909	31m 54s
XGBoost	93.91%	0.9416	0.9484	0.9391	40m 0s

The performance gap between deep learning models and XGBoost (approximately 5–6% accuracy) highlights the value of architectures that explicitly model temporal dependencies.

Among the deep learning approaches, the differences are more subtle: LSTM-FCN's combination of recurrent and convolutional processing provides a modest advantage over pure recurrent (LSTM) or attention-based (CNN-Transformer, TransKal) approaches.

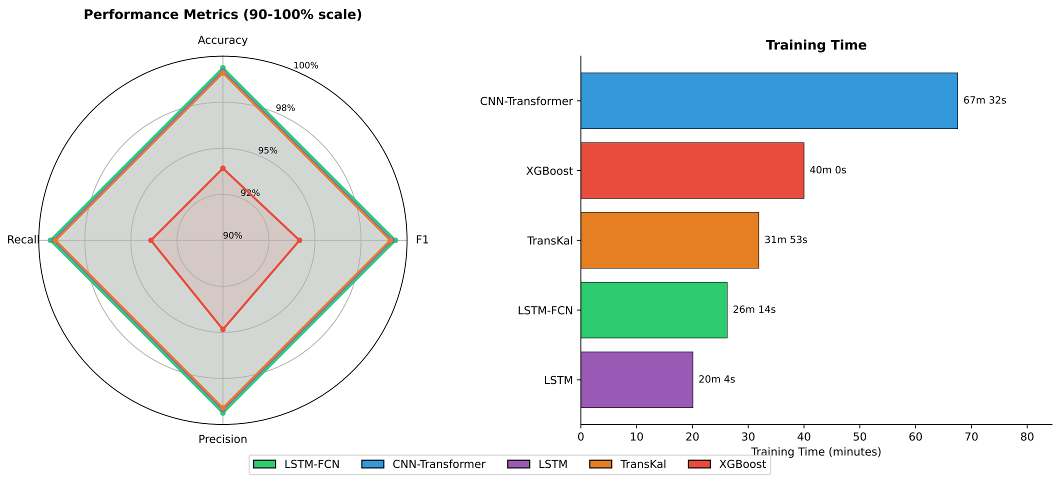


Figure 2: Graphical representation of test set performance for the multiclass models. On the left a radar plot illustrates tradeoffs in each metric. On the right is a comparison of training times.

Binary Anomaly Detection

Table 6 presents the test set performance for the autoencoder-based anomaly detectors. The Convolutional Autoencoder achieved superior performance (99.39% accuracy, 0.9964 F1) compared to the LSTM Autoencoder (96.96% accuracy, 0.9820 F1), while requiring dramatically less training time (56 seconds vs. 12 minutes).

Table 6: Test set performance for binary anomaly detection. Both models were trained exclusively on normal operating data.

Model	Accuracy	F1	Precision	Recall	ROC-AUC	Training Time
Conv-AE	**99.39%**	**0.9964**	0.9958	0.9970	0.9988	56s
LSTM-AE	96.96%	0.9820	0.9955	0.9688	0.9925	11m 47s

Both models achieve high ROC-AUC scores (>0.99), indicating excellent separation between normal and faulty conditions across all threshold choices. The Convolutional Autoencoder's higher recall (99.70% vs. 96.88%) indicates fewer missed faults, which is critical for industrial safety applications.

Per-Class Analysis

Table 7 presents per-class F1 scores for the multiclass models, revealing consistent patterns in fault classification difficulty across architectures.

Table 7: Per-class F1 scores for multiclass models. Classes 0 (normal), 12, and 18 are consistently the most challenging across all models.

Fault	XGBoost	LSTM	LSTM-FCN	CNN-Trans.	TransKal
0 (Normal)	0.735	0.969	0.973	0.971	0.971
1	0.994	1.000	0.999	1.000	0.999
2	0.993	1.000	1.000	1.000	0.999
4	0.967	1.000	1.000	1.000	1.000
5	0.991	0.999	0.999	0.992	0.999
6	1.000	0.998	1.000	1.000	0.999
7	1.000	0.999	1.000	1.000	0.999
8	0.957	0.995	0.996	0.997	0.996
10	0.868	0.993	0.994	0.994	0.992
11	0.895	0.999	1.000	1.000	1.000
12	0.938	0.965	0.977	0.969	0.959
13	0.949	0.985	0.991	0.989	0.984
14	0.988	1.000	1.000	1.000	1.000
16	0.895	0.997	0.999	0.999	0.999
17	0.957	0.996	0.996	0.995	0.995
18	0.947	0.958	0.971	0.957	0.950
19	0.942	0.999	0.999	1.000	1.000
20	0.932	0.994	0.994	0.994	0.994

Three fault classes consistently exhibited lower F1 scores across all models. Class 0 (normal operation) proved most challenging, with F1 scores ranging from 0.735 for XGBoost to 0.973 for LSTM-FCN; this difficulty arises because early fault signatures can resemble normal process variability, leading to confusion at class boundaries. Class 12 (condenser cooling water inlet temperature) achieved F1 scores between 0.938 and 0.977, as this fault produces subtle temperature changes that overlap with normal operating variations. Class

18 exhibited F1 scores from 0.947 to 0.971, reflecting high variability in its manifestation pattern.

In contrast, several faults achieved near-perfect classification across all deep learning models. Class 6 (A feed loss) and Class 7 (C header pressure loss) were classified with F1 scores exceeding 0.998 by all models, reflecting the distinct signatures produced by complete feed or pressure loss events. Class 14 (reactor cooling water valve sticking) achieved perfect F1 scores of 1.000 across all deep learning architectures.

Training Efficiency

Training times varied substantially across models, reflecting differences in architecture complexity and dataset size. For multiclass models trained on 864,000 samples, LSTM was the fastest at 20 minutes, while CNN-Transformer required 68 minutes due to the computational cost of self-attention. XGBoost, despite being a non-neural approach, required 40 minutes owing to the large number of estimators (499) and the size of the training set. The Convolutional Autoencoder trained remarkably fast at only 56 seconds, benefiting from its simpler architecture, smaller training set (normal data only, approximately 50,000 samples), efficient 1D convolutions, and large batch size (128).

Summary

These results demonstrate that deep learning substantially outperforms gradient boosting for fault classification in dynamic processes, with all temporal deep learning models achieving greater than 99% accuracy compared to XGBoost's 93.91%. This 5–6% performance gap confirms that explicit temporal modeling is essential when fault signatures evolve over time.

Among the deep learning approaches, LSTM-FCN provided the best multiclass performance, suggesting that combining local pattern detection through convolution with sequential modeling through recurrence offers complementary benefits. For anomaly detection, the Convolutional Autoencoder excelled with 99.39% accuracy despite its architectural simplicity and sub-minute training time, making it highly practical for deployment scenarios requiring rapid model updates.

Notably, the Kalman filter component in TransKal provided minimal benefit, improving accuracy by only 0.006% over the raw Transformer predictions. This suggests that the base Transformer architecture already produces temporally consistent predictions, and the additional complexity of Kalman filtering may not be justified for this application.

Hidden Markov Model Classification Filtering

To evaluate whether post-hoc temporal smoothing could improve classification performance, we applied the HMM classification filter to all multiclass models with stickiness parameters $\gamma \in \{0.70, 0.85, 0.95\}$. Table 8 summarizes the results, comparing raw model predictions to the best HMM-filtered performance for each model.

Table 8: Effect of HMM classification filtering on model performance. Error reduction is calculated as the percentage decrease in misclassification rate.

Model	Raw Acc.	Raw F1	Best γ	Filtered Acc.	Filtered F1	Error Red.
XGBoost	93.91%	0.9416	0.95	**95.90%**	**0.9606**	32.7%
LSTM	99.14%	0.9914	0.95	99.14%	0.9915	0.7%
LSTM-FCN	99.37%	0.9937	0.70	99.37%	0.9937	0.1%
CNN-Transformer	99.20%	0.9920	0.95	99.21%	0.9921	1.2%
TransKal	99.09%	0.9909	0.85	99.08%	0.9909	0.4%

The HMM filter provided substantial benefit for XGBoost, reducing the error rate by 32.7% and improving accuracy from 93.91% to 95.90%. This improvement is expected: XGBoost treats each time step independently without temporal context, making its predictions susceptible to transient misclassifications that the HMM filter can smooth. With the highest stickiness parameter ($\gamma = 0.95$), the filter strongly encourages temporal consistency, correcting isolated erroneous predictions by leveraging the surrounding context.

In contrast, the HMM filter provided negligible benefit for all deep learning models, with error reductions below 1.5% in all cases. The LSTM-FCN showed essentially no improvement

(0.1% error reduction), while LSTM, CNN-Transformer, and TransKal showed marginal gains of 0.7%, 1.2%, and 0.4% respectively. These minimal improvements indicate that the deep learning architectures already produce temporally consistent predictions through their inherent modeling of sequential dependencies.

The optimal stickiness parameter varied across models. XGBoost, LSTM, and CNN-Transformer all performed best with $\gamma = 0.95$, the maximum smoothing level, reflecting their need for (XGBoost) or tolerance of (LSTM, CNN-Transformer) strong temporal regularization. LSTM-FCN performed best with the lowest stickiness ($\gamma = 0.70$), suggesting that excessive smoothing can slightly degrade predictions when the base model is already highly accurate. TransKal performed best at an intermediate level ($\gamma = 0.85$), likely because its built-in Kalman filter already provides temporal smoothing that partially overlaps with the HMM filter's function.

Table 9 shows the per-class improvement from HMM filtering for XGBoost, the only model with meaningful gains. The filter improved F1 scores for 13 of 18 classes, with the largest gains observed for Class 0 (normal operation, +0.066), Class 11 (+0.057), and Class 19 (+0.049). These classes exhibited the most temporally inconsistent predictions in the raw XGBoost output, making them most amenable to smoothing. Five classes showed no change, and no classes degraded, indicating that the filter's conservative sticky transition model avoids introducing new errors.

These results have practical implications for model deployment. For systems using XGBoost or other non-temporal classifiers, HMM filtering offers a computationally inexpensive method to improve accuracy by approximately 2% absolute. The filter adds minimal latency, requiring only a single Bayesian update per time step, and can be applied to any classifier's predictions without retraining. However, for systems using deep learning models with inherent temporal modeling, HMM filtering adds complexity without meaningful performance gains and can therefore be omitted.

Even with HMM filtering, XGBoost (95.90% accuracy) still substantially underperforms

Table 9: Per-class F1 improvement from HMM filtering for XGBoost ($\gamma = 0.95$). Classes with the lowest raw F1 scores show the largest improvements.

Fault Class	Raw F1	Filtered F1	Δ F1
0 (Normal)	0.735	0.801	+0.066
11	0.895	0.952	+0.057
19	0.942	0.991	+0.049
16	0.895	0.942	+0.048
10	0.868	0.905	+0.037
20	0.932	0.955	+0.023
4	0.967	0.983	+0.015
17	0.958	0.972	+0.015
12	0.938	0.948	+0.011
14	0.988	0.998	+0.010
8	0.957	0.962	+0.005
13	0.949	0.953	+0.004
18	0.947	0.950	+0.003

the deep learning models (99.09–99.37% accuracy). This persistent gap of approximately 3.5% demonstrates that post-hoc temporal smoothing cannot fully compensate for the lack of learned temporal representations, reinforcing the value of architectures that natively model sequential dependencies for time series fault classification.

Generalization to Independent Data

To assess model generalization beyond the original test set, all trained models were evaluated on an independently generated dataset using different random seeds in the TEP simulator. Table 10 presents the results for multiclass classification models, comparing performance on the original test set to the new evaluation set.

The CNN-Transformer was the only model to improve on the new dataset, achieving 99.57% accuracy compared to 99.20% on the original test set. This remarkable result demonstrates the superior generalization capability of the Transformer's self-attention mechanism, which appears to learn robust temporal patterns that transfer effectively to unseen data generated from the same underlying process.

Table 10: Generalization performance for multiclass models. Delta indicates the change in accuracy from the original test set to the new evaluation set.

Model	Orig. Acc.	Orig. F1	New Acc.	New F1	Δ Acc.
CNN-Transformer	99.20%	0.9920	**99.57%**	**0.9958**	**+0.38%**
LSTM-FCN	99.37%	0.9937	98.93%	0.9893	−0.44%
LSTM	99.14%	0.9914	98.91%	0.9890	−0.23%
XGBoost	93.91%	0.9416	89.40%	0.8973	−4.51%
TransKal	99.09%	0.9909	87.57%	0.8484	−11.52%

LSTM and LSTM-FCN exhibited robust generalization with modest accuracy drops of 0.23% and 0.44% respectively. Both models maintained accuracy above 98.9%, indicating that recurrent architectures learn transferable representations of fault dynamics. The slightly larger drop for LSTM-FCN compared to LSTM suggests that the additional convolutional branch, while beneficial for the original test set, may capture some dataset-specific patterns that do not generalize as well.

XGBoost showed moderate degradation, with accuracy dropping 4.51% from 93.91% to 89.40%. This degradation is consistent with the model's lack of temporal context: without explicit sequence modeling, XGBoost relies on instantaneous feature distributions that may shift between independently generated datasets. The per-class analysis revealed that Class 5 (−0.21 F1) and Class 20 (−0.14 F1) suffered the largest degradations, suggesting these faults exhibit the greatest variability across different simulation seeds.

TransKal exhibited severe degradation on the new dataset, with accuracy dropping 11.52% from 99.09% to 87.57%. This result is particularly striking given that the model showed no signs of overfitting on the original train/validation/test splits—validation and test performance were nearly identical (98.78% and 99.09% respectively). The failure therefore reflects not overfitting to the training data, but rather a lack of robustness to distribution shift between datasets generated with different random seeds.

Further analysis revealed catastrophic failures on specific fault classes: Class 0 (normal operation) dropped to 0.00 F1, Class 4 dropped to 0.24 F1, and Class 10 dropped to 0.55

F1, while other classes such as 6, 7, and 14 maintained perfect performance. This selective failure pattern suggests that the Kalman filter parameters—process noise $Q = 5.04 \times 10^{-5}$ and measurement noise $R = 0.0223$—were calibrated to temporal dynamics that vary across simulation seeds. On the new dataset, the Kalman filter provided negligible improvement (0.03% accuracy gain over raw Transformer predictions), confirming that the learned filtering dynamics do not transfer.

This finding highlights a critical limitation of hybrid architectures that combine neural networks with classical state estimation: while the neural network component may learn generalizable features, the filter parameters can become tuned to dataset-specific temporal statistics. The base Transformer in TransKal likely learns robust representations, as evidenced by the CNN-Transformer's excellent generalization, but the addition of a Kalman filter with fixed parameters introduces brittleness. For robust deployment, such hybrid approaches may require online adaptation of filter parameters or more conservative parameter choices that are less sensitive to distributional variation.

Table 11 presents the generalization results for the binary anomaly detection models.

Table 11: Generalization performance for binary anomaly detection models.

Model	Orig. Acc.	Orig. F1	New Acc.	New F1	Δ Acc.
LSTM-AE	96.96%	0.9820	93.93%	0.9607	−3.02%
Conv-AE	99.39%	0.9964	76.04%	0.8639	−23.35%

The LSTM Autoencoder showed moderate degradation, dropping 3.02% from 96.96% to 93.93% accuracy. While this represents a meaningful decrease, the model retained reasonable discriminative ability, suggesting that the learned reconstruction patterns for normal operation partially transfer to new data.

The Convolutional Autoencoder exhibited catastrophic failure on the new dataset, with accuracy plummeting 23.35% from 99.39% to 76.04%. As with TransKal, this model showed no overfitting on the original data splits, so the failure reflects sensitivity to distribution shift rather than memorization. Despite the dramatic accuracy drop, the model maintained perfect recall (100%), correctly identifying all faulty samples. The failure mode was a severe increase in false positives: many normal samples were incorrectly flagged as anomalies because their reconstruction errors exceeded the learned threshold. Notably, the ROC-AUC remained high at 0.981, indicating that the model still effectively separates normal from faulty samples in a ranking sense. This confirms that the learned representations retain discriminative power; the failure lies in the threshold calibration. The reconstruction error distribution for normal samples shifted on the new data, rendering the threshold optimized on the original data ineffective. Threshold recalibration on a small sample of new normal data could potentially recover performance, but this requirement limits the model's practical utility for deployment without adaptation mechanisms.

These generalization results reveal a consistent pattern: models with fixed calibrated parameters—whether Kalman filter noise estimates or anomaly detection thresholds—exhibit brittleness to distribution shift, even when they show no signs of overfitting on the original data. In contrast, pure neural architectures (CNN-Transformer, LSTM, LSTM-FCN) that learn end-to-end without such calibrated components demonstrate robust generalization.

Table 12 summarizes the overall generalization ranking across all models.

Table 12: Model ranking by generalization performance on the independent evaluation dataset.

Rank	Model	New Accuracy	Δ from Original	Assessment
1	CNN-Transformer	99.57%	+0.38%	Excellent
2	LSTM-FCN	98.93%	−0.44%	Good
3	LSTM	98.91%	−0.23%	Good
4	LSTM-AE	93.93%	−3.02%	Moderate
5	XGBoost	89.40%	−4.51%	Poor
6	TransKal	87.57%	−11.52%	Severe
7	Conv-AE	76.04%	−23.35%	Failed

For practical deployment, the CNN-Transformer emerges as the most robust architecture for multiclass fault classification, uniquely improving on unseen data while maintaining computational tractability. For anomaly detection, the LSTM Autoencoder provides more stable generalization than the Convolutional Autoencoder, despite achieving lower accuracy on the original test set. When high original-test-set accuracy is prioritized over generalization robustness, LSTM-FCN remains an excellent choice with only modest degradation on new data. The severe failures of TransKal and Conv-AE underscore the risks of deploying models with components that are sensitive to dataset-specific statistics without mechanisms for online adaptation or periodic recalibration.

Limitations

Several limitations of this work should be acknowledged. First, the study used a specific balanced subset of the available data, with equal numbers of samples per fault class. While this design choice facilitates fair comparison across fault types and avoids class imbalance artifacts, different sampling strategies could favor other models. An imbalanced dataset reflecting realistic fault frequencies, where normal operation dominates and certain faults are rare, might reveal different performance trade-offs and could benefit from specialized techniques such as oversampling, class weighting, or cost-sensitive learning.

Second, faults 3, 9, and 15 were excluded from analysis. Fault 3 (step change in reactor cooling water inlet temperature) and Fault 15 (valve stiction in condenser cooling water flow) produce subtle signatures that overlap substantially with normal operation and with each other. Fault 9 (random variation in reactor outlet temperature) is inherently stochastic and difficult to distinguish from process noise. Previous work has demonstrated that these faults are too subtle to detect reliably within short time windows, which is why they are commonly excluded from machine learning studies on the TEP dataset.[28] Developing methods capable of detecting these challenging faults remains an open problem.

Third, the generalization evaluation, while informative, was limited to data generated from the same TEP simulator with different random seeds. This tests robustness to stochastic variation but does not address generalization to different operating conditions, equipment degradation, or the sim-to-real gap that would be encountered in actual industrial deployment. The severe degradation observed for TransKal and the Convolutional Autoencoder on even this modest distribution shift raises concerns about deployment robustness that warrant further investigation with more diverse evaluation scenarios.

Fourth, models incorporating calibrated parameters—specifically the Kalman filter noise parameters in TransKal and the anomaly detection thresholds in the autoencoders—exhibited brittleness to distribution shift despite showing no overfitting on the original data splits. This limitation is particularly relevant for industrial applications where process conditions may drift over time. Practical deployment of such models would require mechanisms for online parameter adaptation or periodic recalibration, which were not explored in this work.

Finally, all experiments were conducted on simulated data from a well-characterized benchmark process. While the TEP dataset provides a valuable standardized testbed, real industrial processes exhibit complexities not captured in simulation, including sensor noise, missing data, unmeasured disturbances, and evolving fault modes. Validation on real industrial data would be necessary to confirm the practical utility of these methods.

Conclusion

This work presented a comprehensive evaluation of machine learning and deep learning methods for fault detection and diagnosis on the Tennessee Eastman Process, benchmarking seven model architectures across multiclass fault classification and binary anomaly detection tasks. The study employed rigorous methodology including Bayesian hyperparameter optimization, balanced dataset construction, and evaluation on both standard test sets and independently generated data to assess generalization.

The results demonstrate that deep learning architectures with explicit temporal modeling substantially outperform classical machine learning approaches for fault classification in dynamic processes. All deep learning models achieved greater than 99% accuracy on the original test set, compared to 93.91% for the optimized XGBoost baseline. This 5–6% performance gap persisted even after applying HMM classification filtering to smooth XGBoost's predictions, confirming that post-hoc temporal processing cannot fully compensate for the lack of learned sequential representations. Among the deep learning approaches, LSTM-FCN achieved the highest test set accuracy (99.37%) by combining recurrent processing for temporal dependencies with convolutional processing for local pattern detection.

The generalization evaluation on independently generated data revealed critical insights for practical deployment. The CNN-Transformer was the only model to improve on new data, achieving 99.57% accuracy compared to 99.20% on the original test set, demonstrating that self-attention mechanisms learn particularly robust temporal features. LSTM and LSTM-FCN maintained strong performance with modest degradation below 0.5%. In contrast, models incorporating calibrated parameters—TransKal with its Kalman filter and the Convolutional Autoencoder with its anomaly threshold—exhibited severe degradation despite showing no overfitting on the original data splits. This finding highlights a critical limitation of hybrid architectures: while neural network components may learn generalizable features, calibrated parameters can become tuned to dataset-specific statistics and introduce brittleness to distribution shift.

For binary anomaly detection using only normal training data, the Convolutional Autoencoder achieved the highest original test set performance (99.39% accuracy) with remarkably efficient training (56 seconds), but the LSTM Autoencoder provided more stable generalization to new data. This trade-off between peak performance and robustness has important implications for industrial deployment, where process conditions may drift over time.

Based on these findings, we offer the following practical recommendations. For multiclass fault classification requiring robust generalization, the CNN-Transformer provides the best balance of accuracy and transferability. When maximum accuracy on a fixed data distribution is prioritized, LSTM-FCN offers marginally higher performance. For systems using non-temporal classifiers such as gradient boosting, HMM classification filtering provides a computationally inexpensive method to improve accuracy by approximately 2% absolute.

For anomaly detection, the LSTM Autoencoder is preferred when generalization stability is important, while the Convolutional Autoencoder is suitable when rapid retraining or threshold recalibration is feasible. Hybrid architectures combining neural networks with classical state estimation should be deployed with caution, as their calibrated parameters may require online adaptation mechanisms not explored in this work.

Several directions remain for future investigation. Extending this analysis to include the challenging faults 3, 9, and 15, which were excluded due to their subtle signatures, would provide a more complete picture of fault detection capabilities. Developing methods for online adaptation of Kalman filter parameters and anomaly thresholds could address the generalization limitations observed for TransKal and the Convolutional Autoencoder. Evaluation on real industrial data would validate the practical utility of these methods beyond simulation benchmarks. Finally, exploring uncertainty quantification and interpretability techniques could enhance the trustworthiness of deep learning models for safety-critical industrial applications.

In conclusion, this work establishes that temporal deep learning architectures, particularly those based on attention mechanisms, provide robust and accurate fault detection for complex chemical processes. The comprehensive benchmark results and generalization analysis presented here offer guidance for practitioners selecting fault detection methods for industrial deployment and highlight the importance of evaluating models on independently generated data to assess true generalization capability.

Data Availability

All code, data, and trained models required to reproduce the results presented in this paper are available in a public GitHub repository at `https://github.com/KitchinHUB/tep-manuscript`. The repository contains Jupyter notebooks organized by analysis stage: dataset creation (01-series), hyperparameter tuning (10-series), final model training and evaluation (20-series), HMM classification filtering (30-series), and generalization evaluation on independent data (40-series). Each notebook can be executed to regenerate the corresponding results, figures, and metrics reported in this paper. The repository also includes the optimized hyperparameters, trained model weights, and preprocessed datasets to enable direct reproduction without retraining. The raw Tennessee Eastman Process simulation data originates from the enriched dataset by Rieth et al.,[29] and the independent evaluation dataset was generated using the `tep-sim` simulator.[35]

References

(1) Lomov, I.; Lyubimov, M.; Makarov, I.; Zhukov, L. E. Fault detection in Tennessee Eastman process with temporal deep learning models. *Journal of Industrial Information Integration* **2021**, *23*, 100216.

(2) An, D.; Kim, N. H.; Choi, J.-H. Practical options for selecting data-driven or physics-based prognostics algorithms with reviews. *Reliability Engineering & System Safety* **2015**, *133*, 223–236.

(3) Zhao, S.; Duan, Y.; Roy, N.; Zhang, B. A novel fault diagnosis framework empowered by LSTM and attention: A case study on the Tennessee Eastman process. *The Canadian Journal of Chemical Engineering* **2025**, *103*, 1763–1785, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cjce.25460.

(4) Downs, J.; Vogel, E. A plant-wide industrial process control problem. *Computers & Chemical Engineering* **1993**, *17*, 245–255.

(5) Rieth, C. A.; Amsel, B. D.; Tran, R.; Cook, M. B. In *Advances in Human Factors in Robots and Unmanned Systems*; Chen, J., Ed.; Springer International Publishing: Cham, 2018; Vol. 595; pp 52–63, Series Title: Advances in Intelligent Systems and Computing.

(6) Chandola, V.; Banerjee, A.; Kumar, V. Anomaly detection: A survey. *ACM Computing Surveys* **2009**, *41*, 1–58.

(7) Yélamos, I.; Escudero, G.; Graells, M.; Puigjaner, L. Performance assessment of a novel fault diagnosis system based on support vector machines. *Computers & Chemical Engineering* **2009**, *33*, 244–255.

(8) Grbovic, M.; Li, W.; Subrahmanya, N. A.; Usadi; Vucetic, S. Cold Start Approach for Data-Driven Fault Detection. *IEEE Transactions on Industrial Informatics* **2013**, *9*, 2264–2273.

(9) Yin, S.; Ding, S. X.; Haghani, A.; Hao, H.; Zhang, P. A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark Tennessee Eastman process. *Journal of Process Control* **2012**, *22*, 1567–1581.

(10) Gao, X.; Hou, J. An improved SVM integrated GS-PCA fault diagnosis approach of Tennessee Eastman process. *Neurocomputing* **2016**, *174*, 906–911.

(11) Moura, M. D. C.; Zio, E.; Lins, I. D.; Droguett, E. Failure and reliability prediction by support vector machines regression of time series data. *Reliability Engineering & System Safety* **2011**, *96*, 1527–1534.

(12) Harinarayan, R. R. A.; Shalinie, S. M. XFDDC: eXplainable Fault Detection Diagnosis and Correction framework for chemical process systems. *Process Safety and Environmental Protection* **2022**, *165*, 463–474.

(13) Amin, M. T.; Imtiaz, S.; Khan, F. Process system fault detection and diagnosis using a hybrid technique. *Chemical Engineering Science* **2018**, *189*, 191–211.

(14) Cheng, F.; He, Q. P.; Zhao, J. A novel process monitoring approach based on variational recurrent autoencoder. *Computers & Chemical Engineering* **2019**, *129*, 106515.

(15) Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **1997**, *9*, 1735–1780.

(16) Arunthavanathan, R.; Khan, F.; Ahmed, S.; Imtiaz, S. A deep learning model for process fault prognosis. *Process Safety and Environmental Protection* **2021**, *154*, 467–479.

(17) Zhang, L.; Song, Z.; Zhang, Q.; Peng, Z. Generalized transformer in fault diagnosis of Tennessee Eastman process. *Neural Computing and Applications* **2022**, *34*, 8575–8585.

(18) Verma, R.; Yerolla, R.; Besta, C. S. Deep Learning-based Fault Detection in the Tennessee Eastman Process. 2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS). Coimbatore, India, 2022; pp 228–233.

(19) Hajihosseini, P.; Anzehaee, M. M.; Behnam, B. Fault detection and isolation in the challenging Tennessee Eastman process by using image processing techniques. *ISA Transactions* **2018**, *79*, 137–146.

(20) Chiang, L. H.; Russell, E. L.; Braatz, R. D. *Fault Detection and Diagnosis in Industrial Systems*; Springer London: London, 2001; pp 103–112.

(21) Lyman, P. R.; Georgakis, C. Plant-wide control of the Tennessee Eastman problem. *Computers & Chemical Engineering 19*, 321–331.

(22) McAvoy, T. J.; Ye, N. Base control for the Tennessee Eastman problem. *Computers & Chemical Engineering 18*, 383–413.

(23) Ye, N.; Mcavoy, T. J.; Kosanovich, K. A.; Piovoso, M. J. Optimal averaging level control for the Tennessee Eastman problem. *The Canadian Journal of Chemical Engineering 73*, 234–240, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cjce.5450730210.

(24) Larsson, T.; Hestetun, K.; Hovland, E.; Skogestad, S. Self-Optimizing Control of a Large-Scale Plant: The Tennessee Eastman Process. *Industrial & Engineering Chemistry Research 40*, 4889–4901, Publisher: American Chemical Society.

(25) Assali, W. A.; McAvoy, T. Optimal Selection of Dominant Measurements and Manipulated Variables for Production Control. *Industrial & Engineering Chemistry Research 49*, 7832–7842, Publisher: American Chemical Society.

(26) Gertler, J. *Fault Detection and Diagnosis in Engineering Systems*; CRC Press.

(27) Ge, Z. Review on data-driven modeling and monitoring for plant-wide industrial processes. *Chemometrics and Intelligent Laboratory Systems* **2017**, *171*, 16–25.

(28) Shams, M. B.; Budman, H.; Duever, T. Fault detection using CUSUM based techniques with application to the Tennessee Eastman Process. *IFAC Proceedings Volumes* **2010**, *43*, 109–114.

(29) Rieth, C. A.; Amsel, B. D.; Tran, R.; Cook, M. B. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation. 2017.

(30) Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019; pp 2623–2631.

(31) Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco California USA, 2016; pp 785–794.

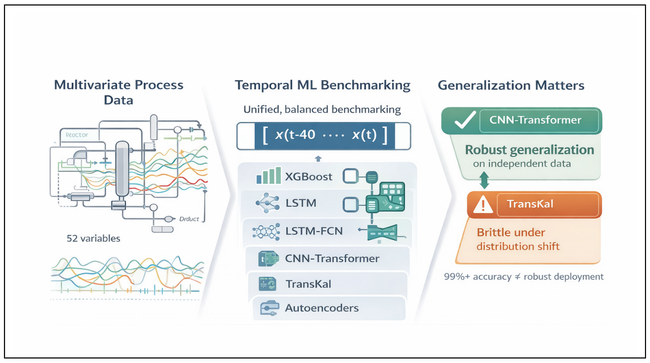
(32) Karim, F.; Majumdar, S.; Darabi, H.; Chen, S. LSTM Fully Convolutional Networks for Time Series Classification. *IEEE Access* **2018**, *6*, 1662–1669.

(33) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; Polosukhin, I. Attention is All You Need. Advances in Neural Information Processing Systems. 2017.

(34) Malhotra, P.; Ramakrishnan, A.; Anand, G.; Vig, L.; Agarwal, P.; Shroff, G. LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. ICML 2016 Anomaly Detection Workshop. 2016.

(35) Kitchin, J. R. Tennessee Eastman Process Dataset from Prof. Braatz. 2024; https://github.com/jkitchin/tennessee-eastman-profbraatz.

TOC Graphic


Multivariate Process Data
Reactor
Product
52 variables
Temporal ML Benchmarking
Unified, balanced benchmarking
[x(t-40 ···· x(t)]
XGBoost
LSTM
LSTM-FCN
CNN-Transformer
TransKal
Autoencoders
Generalization Matters
CNN-Transformer
Robust generalization
on independent data
TransKal
Brittle under distribution shift
99%+ accuracy ≠ robust deployment