
# Benchmarking Self-Supervised Log Embeddings Across Log Modalities and Evaluation Targets

Master's thesis in Data science and AI

Hongliang Zhong

DEPARTMENT OF INDUSTRIAL AND MATERIALS SCIENCE

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

# Benchmarking Self-Supervised Log Embeddings Across Log Modalities and Evaluation Targets

Hongliang Zhong


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Materials Science
Chalmers University of Technology
Gothenburg, Sweden 2026

Benchmarking Self-Supervised Log Embeddings Across Log Modalities and Evaluation Targets

Hongliang Zhong



Supervisor: Silvan Marti, Mechanical Engineering
Examiner: Prof. Björn Johansson, Mechanical Engineering

Master's Thesis 2026
Department of Industrial and Materials Science
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Benchmarking Self-Supervised Log Embeddings Across Log Modalities and Evaluation Targets
Hongliang Zhong
Department of Industrial and Materials Science
Chalmers University of Technology

# Abstract

Log data are increasingly used to support monitoring, troubleshooting, anomaly detection, and operational decision making in software, industrial, and process systems. These records contain heterogeneous information, including structured fields, message text, event sequences, timestamps, and numerical values. Embedding methods can convert such records into fixed-dimensional vectors for downstream analysis, but the usefulness of an embedding depends on which parts of the original input unit remain accessible after representation construction.

Evaluating log embeddings is challenging because log data do not form a single homogeneous modality. Input units can range from individual rows to fixed windows and larger operational traces, and each scale exposes different event-level, sequence-level, numerical, and behavioural structure. At the same time, downstream scores can reflect only one aspect of an embedding space. A classifier score, retrieval result, clustering metric, numerical probe, weak outcome target, or runtime measurement may therefore lead to different conclusions about the same representation.

The aim of this thesis is to benchmark and analyse log embeddings across heterogeneous log settings, with emphasis on what different representation methods preserve. The study compares strong transparent baselines with self-supervised learned embeddings based on reconstruction, contrastive, and hybrid objectives. Four datasets are used to cover structured row-level logs, operational row and window logs, block-level system event sequences, and process-oriented subprocess logs. The embeddings are evaluated using probes, retrieval, clustering, numerical recovery, weak outcome targets, and runtime.

The results show that log embedding quality is conditional on the relation between the input unit, the available observable structure, and the evaluation target. Simple baselines remain strong when the target is close to explicit fields, tokens, counts, or numerical summaries. Learned embeddings provide clearer benefits when the target depends on context or relations across events. The main conclusion is methodological: log embeddings should be evaluated as profiles of preserved information, with method selection grounded in the intended analysis use.

Keywords: log embeddings, representation learning, self-supervised learning, log analysis, benchmark evaluation, operational logs

# Acknowledgements

I would like to express my sincere gratitude to my supervisor, Silvan Marti, for giving me the opportunity to conduct this thesis and for his continuous support throughout the thesis work. I am especially grateful for the many discussions, detailed feedback, practical research tips, and writing suggestions he shared with me. Through his guidance, I learned many useful details about research practice, experiment design, result interpretation, and academic writing.

I am also grateful to my examiner, Prof. Björn Johansson, for his time, feedback, and valuable suggestions during the thesis process.

Finally, my thanks go to my family and close friends for their support and encouragement during this work.

Hongliang Zhong, Gothenburg, 25 June 2026

# AI Declaration

AI tools, including the GPT-5.5 model, were used during this thesis work as supporting tools for writing, programming, and analysis-related assistance. They were used to improve readability and style, discuss thesis structure, support LaTeX and code-related tasks, and assist with checking quality and consistency.

All AI-assisted outputs were critically reviewed, revised, and accepted only after assessment by the author. The final decisions about the research aim, dataset selection, experimental design, method implementation, result interpretation, thesis argument, and written content were made by the author. The author takes full responsibility for the correctness, integrity, originality, and academic quality of the thesis.

Hongliang Zhong, Gothenburg, 25 June 2026

# Contents

# List of Acronyms

Below is the list of acronyms used throughout the document, listed in alphabetical order:

| | |
|---|---|
| ARI | Adjusted Rand Index |
| AUPRC | Area Under the Precision–Recall Curve |
| AUROC | Area Under the Receiver Operating Characteristic Curve |
| CPU | Central Processing Unit |
| CSV | Comma-Separated Values |
| CUDA | Compute Unified Device Architecture |
| F1 | F1 Score |
| GELU | Gaussian Error Linear Unit |
| GRU | Gated Recurrent Unit |
| GPU | Graphics Processing Unit |
| HDFS | Hadoop Distributed File System |
| IoT | Internet of Things |
| JSON | JavaScript Object Notation |
| MAE | Mean Absolute Error |
| MLP | Multilayer Perceptron |
| MSE | Mean Squared Error |
| NMI | Normalised Mutual Information |
| NT-Xent | Normalised Temperature-Scaled Cross-Entropy |
| PLC | Programmable Logic Controller |
| RQ | Research Question |
| SLURM | Simple Linux Utility for Resource Management |
| SVD | Singular Value Decomposition |
| SynIOL | Synthetic Industrial Operational Logs |
| TF-IDF | Term Frequency–Inverse Document Frequency |
| XES | eXtensible Event Stream |

# List of Figures

# List of Tables

# 1

# Introduction

Log embeddings are treated in this thesis as representation choices for heterogeneous operational records. The introduction frames the problem setting, motivates the benchmark, defines the research questions, and outlines the scope and structure of the study.

## 1.1 Background

Modern software, industrial, and cyber-physical systems continuously record operational behaviour. These records include software logs, event traces, alarms, process events, sensor-related values, maintenance information, and other system observations. They are used for monitoring, troubleshooting, anomaly detection, operational analysis, predictive maintenance, and decision support [36, 3, 4]. In industrial settings, such data are often collected for supervision and operation first, and only later reused for machine-learning workflows and analytical decision support [17].

These operational records are not a single data type. Logs appear as structured rows, semi-structured messages, ordered event sequences, textual templates, timestamped records, and process traces. Practical systems often combine several of these forms [19, 36]. Meaningful system behaviour can therefore be expressed through fields and messages, through event order and numerical values, or through temporal patterns across cases, windows, and trajectories.

Representation construction is therefore a central part of log analysis. Once logs are grouped and encoded for machine learning, only part of the recorded behaviour remains directly accessible to downstream models. Prior work on log parsing shows that there is no single parser that performs best across all log datasets [37], and that representation choices can affect downstream anomaly-detection performance [18]. These findings motivate studying log embeddings as representation choices whose usefulness depends on log modality, input unit, and evaluation target.

## 1.2 Problem Framing

The central problem addressed in this thesis is the evaluation of what different log embeddings preserve. Constructing an embedding is only one part of the representation problem; judging its usefulness requires attention to the information that remains accessible after encoding. A representation that is useful for one analysis target may discard information needed for another. For example, a representation can preserve event identity while losing numerical state information. Another representation may support anomaly classification while producing weak neighbourhood structure, or encode local sequence patterns without preserving interpretable global clusters.

This problem is amplified by the heterogeneity of log data. Structured row-level logs, event-sequence logs, process-oriented event logs, and operational logs with numerical values expose different kinds of information. These differences become more pronounced when records are represented at different behavioural scales. A benchmark that uses only one log type or one input unit can therefore make a method appear generally effective when it is only well matched to a specific modality-target combination.

Evaluation is a second part of the problem. In many practical settings, labels are limited, incomplete, or describe only one aspect of the data [19, 3]. A single classifier score or anomaly-detection metric cannot show whether a representation captures reusable structure or only target-specific signals. The thesis therefore treats strong baselines and multiple evaluation views as part of the representation problem, with the related literature discussed further in Chapter 2.

## 1.3 Motivation

The motivation for this thesis is the gap between the availability of log data and the ability to judge whether a representation is useful for analysis. Operational systems already produce records that describe events, states, components, messages, and numerical values. Once these records enter a machine-learning pipeline, however, their usefulness depends on how they are grouped, encoded, and evaluated. A representation that appears successful under one target may still hide information that matters for another form of analysis.

This creates a practical selection problem. In operational log analysis, an analyst must decide whether explicit fields, event counts, sequence summaries, or text features are sufficient, or whether a learned encoder is worth the added training cost and complexity. The answer depends on the target: classification, retrieval, clustering, numerical recovery, weak outcome prediction, and runtime-constrained use place different demands on the same embedding. The benchmark in this thesis is motivated by that decision point and compares simple and learned representations across several evaluation views.

## 1.4 Aim and Research Questions

The aim of this thesis is to benchmark and analyse log embedding methods across multiple log modalities, input units, and evaluation settings.[1] The thesis compares strong baseline representations with self-supervised reconstruction-based, contrastive, and hybrid embedding methods. The focus is on what different embeddings preserve, how their performance depends on modality and target choice, and how evaluation views reflect different aspects of representation quality.

The thesis is guided by the following research questions (RQs):

**RQ1:** To what extent do log embeddings preserve event-level, sequence-level, behavioural, and numerical structure across different log datasets and input units?

**RQ2:** How do strong baseline representations compare with reconstruction-based, contrastive, and hybrid self-supervised embeddings across dataset settings?

**RQ3:** How consistent are probe, retrieval, clustering, numerical, and weak outcome evaluations in ranking log embeddings?

The main contributions are:

- a benchmark pipeline for evaluating log embeddings across heterogeneous log datasets and input units;
- a comparison between strong baseline representations and self-supervised learned embeddings;
- a multi-view evaluation covering probes, retrieval, clustering, numerical recovery, weak outcomes, and runtime.

## 1.5 Scope and Delimitations

This thesis focuses on benchmarking log embedding methods at the representation level. The study assumes that logs are already available in a structured or partially structured form. Log parsing is therefore outside the main experimental scope. Existing event identifiers, templates, categorical fields, timestamps, numerical attributes, and contextual identifiers are used when available. Preprocessing is used to construct suitable embedding inputs; the quality or improvement of parsing algorithms is not evaluated.

[1] The thesis project repository is available at `https://github.com/Lyra-Z/log-to-vec-thesis.git`.

The study also assumes that the prepared benchmark data are sufficiently complete and consistent for representation learning. Missing values, corrupted records, timestamp errors, duplicate handling, data-quality repair, and uncertainty introduced by sensor or logging failures are outside the experimental scope. These issues are important in deployed operational systems, but the present work isolates the representation and evaluation problem after usable log records have been prepared.

The experiments are limited to unsupervised and self-supervised embedding settings. Labels, when available, are used only for evaluation and not for training. The thesis does not aim to develop a new log parser, a production anomaly detector, or a general foundation model for operational data. Both synthetic and real-world datasets are included: synthetic datasets provide controlled structural or latent information, while real-world datasets provide less controlled evaluation conditions. The findings should therefore be interpreted as benchmark evidence within the selected settings, with limited claim to generality across all log embedding tasks.

## 1.6 Thesis Structure

The thesis is organised as follows:

- Chapter 2 introduces the conceptual and methodological background for log representation construction, embedding method families, and multi-view evaluation.
- Chapter 3 presents the benchmark datasets and describes the log modality, input unit, and evaluation role of each dataset.
- Chapter 4 describes the benchmark pipeline, embedding methods, training objectives, evaluation protocol, and experimental settings.
- Chapter 5 reports the empirical results across log representation settings and evaluation focuses.
- Chapter 6 discusses the findings in relation to the research questions, practical implications, and validity threats.
- Chapter 7 concludes the thesis.

# 2

# Background and Related Work

The benchmark builds on three concepts: the unit represented by an embedding, the information used to construct that embedding, and the evaluation view used to inspect it. This chapter defines these concepts and positions the study in relation to log analysis, representation learning, and benchmark evaluation.

## 2.1 Log Units and Representation Choices

The term log modality is used here for the dominant form in which recorded behaviour is made available to a model. Examples include structured rows, event sequences, textual messages, numerical fields, and process-event traces. A single dataset can contain several modalities at once: one record may include an event identifier, a timestamp, a message template, a severity field, and numerical payload values.

The input unit is the object represented by one embedding. It may be a single event record, a fixed window, or a larger operational episode such as a block trace, process case, or subprocess instance. This choice determines the behavioural scale of the representation. A row-level unit emphasises fields attached to one event. A sequence or window exposes local order, repetition, duration, and transition patterns. Sequence-based log analysis methods make this ordering explicit by modelling logs as event sequences and using event context to identify abnormal or meaningful behaviour [10].

A log representation specifies how the selected unit is converted into a machine-learning input. An embedding is a fixed-dimensional vector representation of that unit. The embedding may preserve event identity, local sequence patterns, process context, numerical state, message content, or combinations of these sources. The method families described below differ in which input information they use and in how they shape the resulting embedding space.

## 2.2 Temporal and Numerical Structure in Logs

Operational logs often have a temporal structure even when they are not modelled as conventional time series. A timestamped event log records what happened and when it was recorded. An event sequence records order, repetition, and local context. A process trace or subprocess instance records a bounded operational episode. Numerical payloads, sensor-related values, durations, and counters add state information that can change over time. These properties make time part of the meaning of many log records, extending its role beyond an index used for sorting.

The transition from record to representation determines which temporal relations remain available to a model. Row-level representations expose the fields of a single event but provide limited context about what came before or after it. Window-level and sequence-level representations expose local order, transitions, and repeated patterns. Block-level and subprocess-level units describe larger operational episodes, where event counts, activity sets, duration, and outcome labels can all become relevant. Input-unit construction is therefore a temporal representation choice as well as a data-preparation step.

Time-series representation learning provides useful concepts for analysing temporal neighbourhoods, temporal dependencies, and numerical dynamics [30, 35]. These ideas are relevant to log embeddings because many input units contain ordered events, durations, counters, or sensor-related values. In a log-embedding benchmark, such structure can be evaluated by asking whether the embedding preserves event order, transition information, duration, sequence length, or numerical state.

## 2.3 Log Embedding Method Families

Embedding methods differ in two main respects: the input information available to the representation and the objective used to organise the embedding space. Each method family carries an implicit assumption about what should be preserved. Baseline methods assume that observable fields, counts, lexical content, or summary statistics already contain recoverable structure. Learned methods train an encoder to compress or organise the selected input unit through a self-supervised objective. Hybrid methods combine objectives when both input preservation and neighbourhood organisation are intended.

### 2.3.1 Baseline Embeddings

Baseline embeddings are non-neural representations built from observable input features. They provide a reference for how much structure is already available before a learned encoder is introduced. This role is important in log data because many targets are closely connected to explicit identifiers, repeated event templates, structured numerical values, or informative message tokens.

Baseline features usually come from three sources: structured fields, sequence summaries, and text. For structured log rows, categorical fields such as event identifiers, components, severity levels, states, and resources can be represented with one-hot or count-based encodings. Numerical fields can be scaled directly or summarised over a sequence or window.

For sequence-like input units, event counts describe which events occur, n-gram counts describe short local patterns, and transition counts describe adjacent event changes. For text fields and message templates, sparse lexical features represent the words or tokens present in the sample. Term frequency–inverse document frequency (TF-IDF) weighting increases the weight of terms that are frequent in a sample but less common across the dataset [27].

Sparse categorical, count, n-gram, and TF-IDF features can be high-dimensional. Singular value decomposition (SVD) is used to produce dense fixed-dimensional embeddings [8]. These baselines are transparent because their information source is explicit, which makes them useful for interpreting whether a learned method adds information beyond observable structure.

### 2.3.2 Neural Encoder and Decoder

Learned embedding methods are organised around an encoder, with optional heads depending on the training objective. The encoder maps the selected input unit to a fixed-dimensional vector. The decoder or prediction head maps the encoder output back to reconstruction targets when reconstruction is used. A projection head can be added during contrastive training to compute the contrastive loss while leaving the final embedding source unchanged.

The encoder choice depends on the input unit and on the relations that should be expressible. A multilayer perceptron (MLP) is suitable for a single structured row represented by categorical, textual, and numerical feature vectors [26]. Recurrent encoders such as the gated recurrent unit (GRU) process ordered inputs step by step and maintain a hidden state over the sequence [5]. Transformer encoders use self-attention to model relationships between positions in a sequence [32].

These architectures support different log inputs. MLPs suit compact row representations, recurrent encoders suit ordered sequences, and Transformers can be used for event-token sequences, field-token sequences, and window-level log representations. The architecture helps determine whether the embedding behaves as a row representation, a sequence representation, or a representation of interactions between fields.

### 2.3.3 Reconstruction-based Embeddings

Reconstruction-based embeddings use an encoder–decoder architecture. The encoder maps an input sample to a latent representation, and the decoder reconstructs

the input from this representation. The encoder output is used as the embedding. This follows the autoencoder principle of learning compact representations through reconstruction [12].

For log data, reconstruction can target categorical fields, numerical values, message tokens, event identifiers, or ordered event sequences. Masked reconstruction hides selected parts of the input and trains the model to recover them from the remaining context. This objective is related to denoising autoencoders and masked language modelling [33, 9].

Reconstruction encourages the embedding to retain information needed to reproduce the observable record. Its relevance depends on the relation between the reconstruction target and the later evaluation target. When evaluation is close to reconstructed fields, reconstruction is well aligned with the downstream view. When evaluation depends on unobserved behaviour or weak outcomes, reconstruction can preserve input detail without fully organising the embedding around that target.

### 2.3.4 Contrastive Embeddings

Contrastive embeddings are learned by comparing related and unrelated samples. A common design uses an encoder and a projection head. Two augmented views of the same input are treated as a positive pair, while views from different inputs are treated as negative examples. Training encourages positive pairs to be close and negative examples to be separated in the projected space. This principle is used in contrastive predictive coding and later contrastive representation learning methods [24, 2].

For log data, contrastive views can be created through weak perturbations such as token dropout, field masking, event dropout, span masking, or sequence cropping. For sequential data, temporal neighbourhoods and local subsequences can also provide self-supervised signals [30, 35]. Contrastive learning places emphasis on relative similarity. Its effectiveness depends on whether the augmentations preserve the meaning of the input unit while still creating a training signal that separates useful neighbourhoods. In log settings, augmentation design becomes part of the representation assumption: dropping an event, masking a field, or cropping a sequence implies that the remaining view should still represent the same underlying unit.

### 2.3.5 Hybrid Embeddings

Hybrid embeddings combine reconstruction-based and contrastive objectives in one model. The reconstruction objective encourages the representation to preserve recoverable input information, while the contrastive objective shapes similarity relations between augmented views. The final embedding is normally taken from the shared representation before the objective-specific heads. Hybrid learning is useful when information preservation and neighbourhood organisation are both relevant.

The two objectives can emphasise different information, so their effect needs to be evaluated empirically.

## 2.4 Evaluation of Log Embeddings

After an embedding has been constructed, evaluation asks what information remains accessible and how the embedding space is organised. No single view captures all properties of a representation. A probe tests target recovery, retrieval examines local neighbourhoods, clustering describes global grouping, and numerical probes assess retained continuous state information. The benchmark therefore treats evaluation as a profile of representation properties, with each view defining one part of the later metric design.

### 2.4.1 Probe Evaluation

Probe evaluation asks whether target information is accessible from frozen embeddings. A simple supervised model is trained on top of the embeddings, while the embedding model itself remains fixed. Classification probes are used for categorical targets, such as event categories, anomaly status, behaviour classes, or process activities. Regression probes are used for continuous targets, such as duration, risk level, sequence length, or numerical payload values.

Linear classifier probes are common because they test whether a target can be recovered through a simple decision boundary [1]. A high probe score indicates that the target information is available to a lightweight downstream model. It does not imply that samples with the same target form clear clusters or that the embedding is organised around that target.

### 2.4.2 Retrieval Evaluation

Retrieval evaluation asks whether local neighbourhoods in the embedding space correspond to meaningful similarity. Given a query embedding, the nearest embeddings are retrieved from a reference set, and their labels or attributes are compared with the query. If the embedding space is organised around the evaluated target, similar samples should appear among each other's nearest neighbours.

Precision at $k$ measures the fraction of the $k$ retrieved neighbours that match the query target [23]. Jaccard overlap can also be used when the target is a set, such as a set of subprocess activities. Higher values indicate better neighbourhood agreement.

### 2.4.3 Clustering Evaluation

Clustering evaluation asks whether the embedding space contains global groups that correspond to known categories. A clustering algorithm is applied without using the target labels, and the resulting cluster assignments are compared with the labels.

Adjusted Rand index (ARI) compares cluster assignments with reference labels while correcting for chance [13]. A value of 1 indicates perfect agreement, values near 0 indicate chance-level agreement, and negative values can occur. Normalised mutual information (NMI) measures shared information between cluster assignments and labels [29]. Silhouette score measures how close samples are to their assigned cluster compared with other clusters [25]. Higher values are better for all three metrics.

Clustering scores need careful interpretation for log embeddings. A representation may preserve probe-accessible or local-neighbourhood information even when global clusters are weak. This is especially likely when labels are imbalanced, factors overlap, or behaviour changes continuously.

### 2.4.4 Classification and Regression Metrics

Classification and regression metrics quantify the targets used in probes and task-oriented evaluations. Classification probes and task-oriented evaluations use F1 score, precision, recall, area under the receiver operating characteristic curve (AUROC), and area under the precision–recall curve (AUPRC). Precision measures the fraction of predicted positive samples that are correct, while recall measures the fraction of true positive samples that are found. F1 is the harmonic mean of precision and recall. AUROC measures how well a score ranks positive samples above negative samples across classification thresholds [11]. AUPRC summarises the precision–recall curve and is often informative when the positive class is rare [7]. For these metrics, higher values are better.

Numerical probes use regression metrics to assess whether continuous information remains recoverable from the embedding. $R^2$ measures how much target variance is explained by the prediction. A value of 1 indicates perfect prediction, 0 corresponds to predicting no better than the target mean, and negative values indicate worse performance than this baseline. Mean absolute error (MAE) measures the average absolute prediction error, so lower values are better. Spearman correlation measures rank association between predicted and true values; higher positive values indicate better preservation of numerical ordering.

### 2.4.5 Task-oriented and Weak Outcome Evaluation

Task-oriented outcome evaluation asks whether an embedding supports labels connected to practical analysis goals, such as anomaly detection, failure prediction, or operational response classification. These labels connect representation quality to possible use cases, while still giving only a partial view of the embedding.

In log datasets, an outcome label may be attached to a session, case, process, or system state, while the embedded sample may describe only part of that context. The label may also be coarse, noisy, or influenced by information that is not available in the embedded sample. Such labels are treated as weak outcome targets.

They provide auxiliary evidence and are interpreted together with probes, retrieval, clustering, numerical recovery, and runtime.

## 2.5 Related Work and Benchmark Positioning

Prior work on log analysis has mainly developed along three connected lines. The first concerns log parsing and dataset construction. Benchmark resources such as Loghub and Loghub-2.0 provide structured datasets and parsing results for reproducible log analysis [36, 16], while LogAI provides a toolkit for log analytics workflows [4].

The second line concerns task-oriented log analysis, especially anomaly detection and system behaviour modelling. DeepLog is an early example of using event sequences for anomaly detection [10]. Broader surveys identify preprocessing, model architecture, input representation, and evaluation design as recurring factors in deep log anomaly detection [19].

A third line concerns benchmark validity and the role of comparison methods. Landauer et al. report that common log anomaly-detection datasets may contain signals detectable by simple techniques [20]. Similar concerns have been raised in other machine-learning settings, where learned models can appear stronger when competitive simple methods are absent [6]. These findings are relevant here because baseline representations provide evidence about observable log structure. They help determine how much information is already available before more complex embedding methods are introduced.

Time-series representation learning provides related background for temporal context, numerical state, and sequence-level representation [30, 35]. This thesis uses that connection to interpret logs with order, duration, windows, and numerical fields. The study remains positioned in log embedding evaluation. It evaluates frozen embeddings before task-specific modelling, compares baseline and self-supervised representations, and uses several evaluation views to separate representation behaviour from the performance of one downstream model.

# 3

# Datasets

This chapter describes the datasets used in the benchmark and defines their experimental roles. The emphasis is on the form of each log source, the input unit constructed from it, and the evaluation targets supported by the available fields.

## 3.1 Dataset Selection and Benchmark Role

Four datasets are selected to cover different forms of log representation. The resulting benchmark includes input units ranging from individual event records and fixed windows to block traces and subprocess instances. Table 3.1 summarises the source, input unit, scale, and role of each dataset.

The two synthetic datasets provide controlled settings in which the relationship between observable log records and evaluation targets is known by design. The PLC-like Log dataset represents row-level structured logs with coherent relationships between event types, components, severities, payload fields, and rendered messages. SynIOL represents operational event logs generated from latent behaviour dynamics, which makes it possible to evaluate behaviour labels, transition structure, and numerical state variables. These datasets support controlled comparison, but they are not intended to replace evidence from real operational logs.

The two real datasets provide complementary benchmark settings. HDFS represents a widely used system-log dataset with block-level event sequences and coarse anomaly labels. The IoT-Enriched Event Log represents a process-oriented event log in which subprocess activity sequences are connected to operation context and weak process outcomes. Together, the four datasets allow embeddings to be evaluated under different input-unit constructions, from individual records to grouped windows and trace-like instances. Figure 3.1 summarises how these sources map to the input units and evaluation structure used in the benchmark.

The overview shows that the benchmark is not organised around dataset size alone. Each dataset contributes a different input-unit structure and evaluation role, which is why later chapters report results by representation setting.

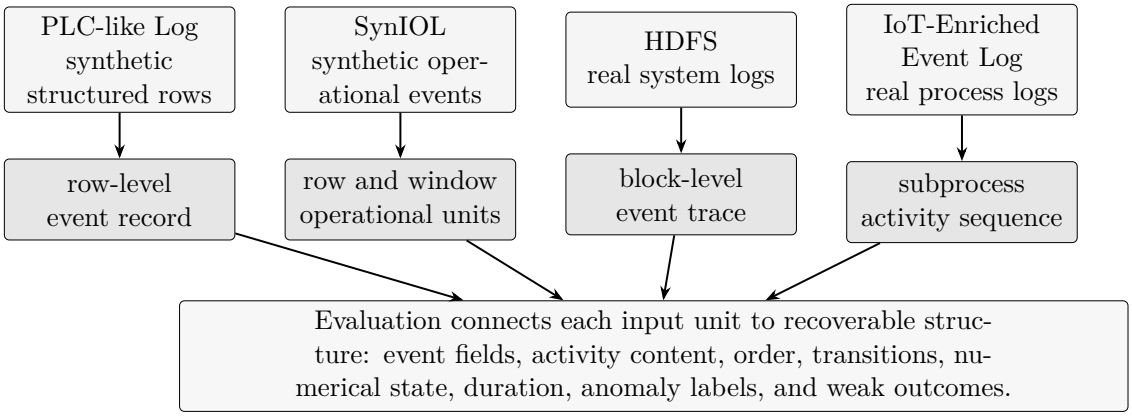


**Figure 3.1:** Relationship between benchmark datasets, input units, and evaluation structure.

**Table 3.1:** Overview of the benchmark datasets.

| Dataset | Source | Input unit | Scale | Role |
|---|---|---|---|---|
| PLC-like Log | Synthetic | Structured log row | 50,000 rows | Row-level semantic and anomaly-related structure |
| SynIOL | Synthetic | Event row and event window | 76,800 events | Latent behaviour, transition, and numerical structure |
| HDFS | Real system log | Block-level event sequence | 575,061 blocks | Anomaly-related structure in system logs |
| IoT-Enriched Event Log | Real process log | Subprocess activity sequence | 3,117 subprocesses | Operation-related and weak outcome structure |

## 3.2 PLC-like Log

A programmable logic controller (PLC) is used in industrial automation and control systems [14]. The PLC-like Log dataset is a synthetic structured log dataset generated for the benchmark. It resembles local log events that may be produced by a programmable controller or an industrial control environment. Each sample is one timestamped structured log row, with identifiers, timestamp information, event descriptors, severity level, rendered message text, structured JavaScript Object Notation (JSON) payload, and labels. The dataset supports the evaluation of row-level semantic and anomaly-related structure.

The event-level structure is defined through a fixed vocabulary covering controller state, process measurement, actuation, alarm handling, communication, watchdog monitoring, configuration, maintenance, and production-cycle events. The complete event vocabulary and the corresponding payload fields are provided in Appendix A.1 and Appendix A.2, respectively. The dataset uses four severity levels: `INFO`, `WARNING`, `ERROR`, and `CRITICAL`. Severity and alarm-related event types are motivated by industrial alarm-management settings, where alarms and events indicate abnormal conditions, equipment-related issues, or situations requiring operational attention [15].

The categorical fields are not generated independently. To make the generated rows structurally coherent, the generator defines relationships between event types, components, severity levels, and payload content. As a result, event types are associated with compatible components and event-specific payload structures. The structured

payload is stored as compact JSON, and the message field is generated from event-specific templates using the payload and row-level context. Each event therefore contains both machine-readable structured fields and human-readable log text.

Table 3.2 summarises the generated dataset. Normal rows are labelled as normal, while selected event types define anomaly-related labels. Temperature alarms, pressure alarms, communication errors, and unsafe controller stops are mapped to anomaly categories such as thermal alarm, pressure alarm, communication fault, and unsafe stop.

Within the benchmark, this dataset serves as a row-level representation setting. It tests whether embeddings capture the semantic relations present in structured fields and rendered messages. Since the dataset is synthetic, the results are interpreted as evidence from a controlled industrial-control-inspired setting, not as a direct measurement of deployment performance in a real PLC environment.

**Table 3.2:** Descriptive statistics of the PLC-like Log dataset.

| **Statistic** | **Count** | **Percentage** |
|---|---|---|
| Log rows | 50,000 | – |
| Event types | 14 | – |
| Message templates | 42 | – |
| Components | 12 | – |
| Normal rows | 43,318 | 86.64% |
| Anomalous rows | 6,682 | 13.36% |

## 3.3 SynIOL

Synthetic Industrial Operational Logs (SynIOL) is a synthetic industrial operational log dataset generated from controlled latent operational dynamics. The underlying behaviour structure is known, while each observable event is represented as a structured log record. This design supports analysis of whether embedding methods can recover behaviour-related structure from observable event sequences.

The dataset has a two-level design. At the row level, each event is treated as one input unit. At the window level, consecutive events within a trajectory are grouped into log windows. These two views separate individual-event representation from behaviour- and transition-oriented sequence representation. They also expose different forms of information relevant to temporal analysis, including event-local fields, ordering relations between events, and behaviour patterns across a window.

Each SynIOL trajectory represents one simulated operational episode and provides the main sequential unit of the dataset. A trajectory consists of multiple behaviour segments, and adjacent segments may be connected by transition regions. Each

segment is generated from one latent behaviour family. The dataset contains five behaviour families. These behaviours are summarised in Table 3.3.

**Table 3.3:** Latent behaviour families in SynIOL.

| Behaviour family | Operational interpretation | Main pattern |
|---|---|---|
| Fast Stable Recovery | A disturbed system returns to a stable condition relatively quickly. | Fast recovery |
| Delayed Recovery | The system recovers more slowly or with partial interruption. | Slow recovery |
| Oscillating Instability | The system alternates between improvement and regression. | Oscillation |
| Cascading Failure | Degradation propagates across components over time. | Component cascade |
| Normal Stable Operation | The system remains in a low-risk state with generally healthy components. | Normal operation |

SynIOL separates fields available to embedding methods from fields retained only for evaluation. The observable part of each event is designed as a structured operational log record. This follows the general event-log perspective, in which events are recorded with attributes such as case identifiers, activities, timestamps, resources, and additional data fields [31]. In SynIOL, the observable fields include the trajectory identifier, event index, timestamp, event type, component, severity, observed state, message, sensor value, and control value.

The event vocabulary covers monitoring events, control-related events, warning and error events, communication-related events, diagnostic events, recovery actions, retry operations, maintenance checks, and healthy-status reports. The full event vocabulary and its functional interpretation are provided in Appendix A.3.

The hidden evaluation fields describe the latent process state used to generate the observable events. These fields include the behaviour label, behaviour variant, segment identifier, transition flag, source and target behaviour during transitions, transition progress, risk level, load level, recovery progress, component health values, and the primary affected component. These fields are not used as model input, but they are retained so that row-level and window-level evaluation targets can be derived consistently.

The generation process includes noise and observation uncertainty. The observed severity and state are not deterministic copies of the hidden operational state. They may be delayed, noisy, or affected by false-positive and false-negative observations. The generator also supports irrelevant event noise, numerical noise, component noise, and multiple noise configurations. The benchmark uses three versions: clean, moderate noise, and high noise. They share the same latent design but differ in the amount of observational noise. Table 3.4 summarises these versions, and Table 3.5 summarises the generated dataset scale.

SynIOL provides the main setting for analysing temporal, behavioural, and numerical structure. Behaviour labels and transition flags test whether embeddings retain latent process organisation. Numerical targets, such as risk level, load level, recovery progress, and component health values, test whether continuous operational state remains recoverable from the representation. The noise versions allow the same

embedding methods to be compared under progressively weaker correspondence between the hidden state and the observed log record.

**Table 3.4:** SynIOL dataset versions.

| Version | Description |
|---|---|
| Clean | Low observational noise and clear correspondence between latent state and observed logs |
| Moderate noise | Increased irrelevant events, numerical noise, and observation uncertainty |
| High noise | Stronger noise and weaker correspondence between latent state and observed logs |

**Table 3.5:** Descriptive statistics of the SynIOL dataset.

| Statistic | Count | Percentage |
|---|---|---|
| Events | 76,800 | – |
| Trajectories | 120 | – |
| Events per trajectory | 640 | – |
| Transition events | 9,107 | 11.86% |

## 3.4 HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed for large-scale data storage and processing [28]. HDFS logs have been extensively used in log analysis and anomaly detection research [36]. The HDFS log dataset was originally generated in a private cloud environment using benchmark workloads, and anomaly labels were assigned using handcrafted rules [34].

A key property of this dataset is its block-level structure. Log messages can be grouped by block identifier, and each block trace is associated with a binary ground-truth label indicating whether the block is normal or anomalous. This makes the dataset suitable for evaluating whether embeddings preserve anomaly-related structure in structured system logs. However, the binary label is coarse and does not describe fine-grained operational behaviours or latent failure mechanisms.

The structured HDFS logs are obtained from Loghub-2.0, which provides improved log templates compared with the earlier Loghub release [16]. Since Loghub-2.0 does not directly include the original anomaly labels, the block-level labels from the original Loghub release are used together with the structured Loghub-2.0 records in this benchmark.

Table 3.6 summarises the scale and label imbalance of the HDFS dataset. According to the Loghub-2.0 parsing results, the structured records contain 46 unique event

templates, represented as `EventId` values in the benchmark. These event identifiers form the main symbolic vocabulary for the block-level event sequences. The benchmark uses a 100,000-sample block-level subset because of the scale of the full dataset. The construction of this subset is described in Chapter 4.

HDFS contributes a real system-log sequence setting. Its main role is to test whether embeddings preserve anomaly-related information in block-level event sequences. The strong class imbalance and coarse binary labels limit the interpretation of the results. A high score on this dataset indicates that the representation supports the available anomaly label, but it does not by itself explain the underlying failure mechanism.

**Table 3.6:** Descriptive statistics of the HDFS dataset.

| **Statistic** | **Count** | **Percentage** |
|---|---|---|
| Structured log lines | 11,167,740 | – |
| Unique block IDs | 575,061 | – |
| Normal blocks | 558,223 | 97.07% |
| Anomalous blocks | 16,838 | 2.93% |
| Normal log lines | 10,884,910 | 97.47% |
| Anomalous log lines | 282,830 | 2.53% |
| `INFO` log lines | 10,805,922 | 96.76% |
| `WARN` log lines | 361,818 | 3.24% |

## 3.5 IoT-Enriched Event Log

The IoT-Enriched Event Log is a process-mining event log generated from a physical smart factory model. The log was created using the DataStream/SensorStream eXtensible Event Stream (XES) extension and contains process events enriched with Internet of Things (IoT)-related information [21, 22]. The original release is provided as a collection of XES event-log files and includes both a native version with data quality issues and a cleaned version in which these issues are removed. The benchmark uses the cleaned event log.

The dataset has a hierarchical process-log structure. It contains top-level process events, subprocess events, and IoT-enriched contextual information. The main process log describes high-level operations executed by machine resources, while subprocess instances describe lower-level activities associated with these operations. Each subprocess instance contains an ordered activity sequence, the corresponding parent activity, resource information, subprocess duration, and a success/failure label derived from the parent operation state.

The original dataset also includes sensor-related information and semantic descriptions based on a domain ontology. The structured version used here focuses on

event-log embedding, so sensor time series and ontology-based reasoning are outside the scope of the benchmark input.

The structured version is the smallest dataset in the benchmark. Despite its limited size, it provides an industrial process-oriented setting in which each sample is represented by an ordered subprocess activity sequence. Table 3.7 summarises the dataset statistics.

**Table 3.7:** Descriptive statistics of the IoT-Enriched Event Log dataset.

| **Statistic** | **Count** | **Percentage** |
|---|---|---|
| Subprocess samples | 3,117 | – |
| Unique subprocess activity templates | 236 | – |
| Templates linked to one parent activity | 211 | 89.41% |

The template distribution indicates a strong template-driven structure, while the dataset still includes cases where similar subprocess sequences appear in different operational contexts. This makes the dataset useful for evaluating whether embeddings organise subprocess sequences according to operation- and template-related structure. The success/failure label is treated as a weak process-level outcome. It is not used as the primary definition of subprocess-level similarity, since successful and failed subprocesses are not always clearly separable from their activity sequences alone.

# 4

# Study Design

The experimental design separates representation construction from downstream evaluation. This chapter describes the pipeline, input units, embedding methods, training objectives, evaluation protocol, and shared settings.

## 4.1 Benchmarking Pipeline

The study is designed as a frozen-embedding benchmark. Each method first maps a prepared input unit to a fixed-dimensional vector, and the resulting embeddings are then evaluated without updating the embedding model. This separation keeps representation learning distinct from downstream evaluation and allows baseline and learned methods to be compared under the same protocol.

The benchmark is implemented as a modular pipeline with separate stages for dataset preparation, baseline embedding construction, learned-model training, embedding extraction, and embedding evaluation. Figure 4.1 illustrates the main workflow of the benchmarking. Auxiliary stages for synthetic data generation, orchestration, reporting, and seed aggregation support the experiments, but they are not treated as separate design components.

The pipeline follows the same comparison rules across all dataset settings. Dataset preparation writes training, validation, and test files in a common artefact structure. Baseline feature extractors, scalers, vectorisers, dimensionality-reduction models, token vocabularies, and learned encoders are fitted only on the training split. Validation and test splits are transformed with the fitted objects. Evaluation labels, hidden variables, identifiers, and target fields are excluded from embedding construction and learned-model training. These rules reduce the risk of leakage and ensure that evaluation targets are used only after embeddings have been produced.

## 4.2 Dataset Preparation and Input Units

Dataset preparation converts each source into the input unit used for embedding construction. The real log datasets require additional processing before entering the

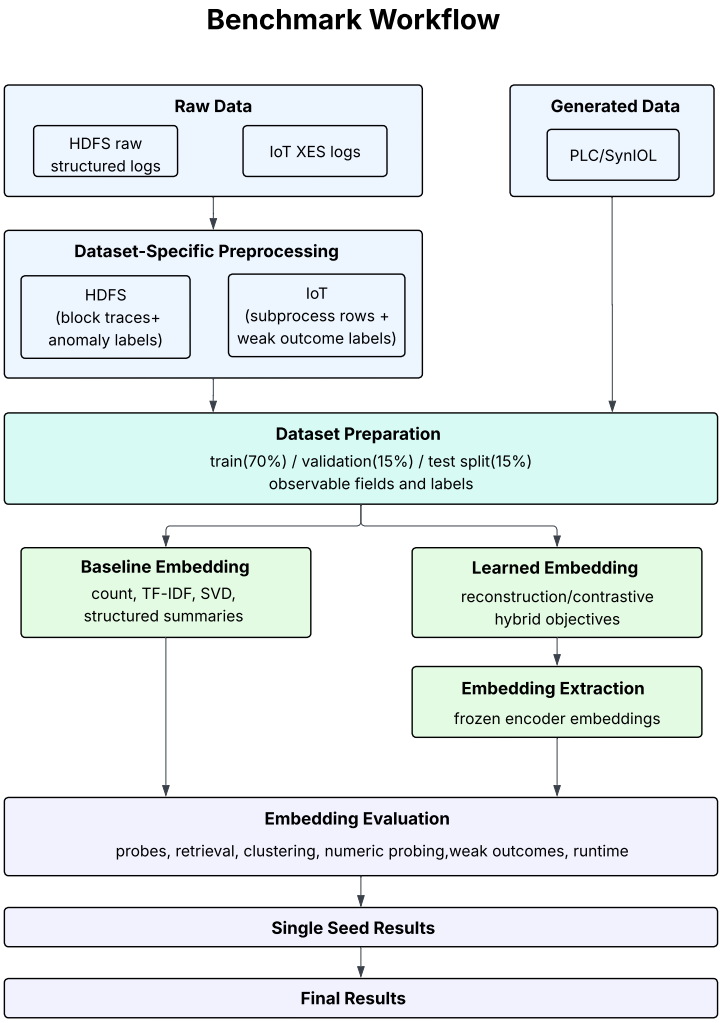


**Figure 4.1:** Benchmark workflow showing the whole experimental process.

main benchmark pipeline, even when the original files already contain structured fields.

For HDFS, structured log lines are grouped by block identifier to construct block-level traces. Each trace stores the ordered event identifiers and the corresponding block-level anomaly label. A 100,000-sample block-level subset is then constructed to keep repeated training and evaluation feasible while retaining the normal/anomaly label distribution.

For the IoT-Enriched Event Log, the raw XES event logs are merged and converted into subprocess-level rows. Each row represents one subprocess instance and stores the ordered subprocess activity sequence, parent-operation metadata, subprocess event count, duration, timestamps, and the weak success/failure outcome. These processed HDFS and IoT files are the input datasets consumed by the main pipeline.

The pipeline preparation stage then applies the common benchmark layout. It writes training, validation, and test splits, separates observable input fields from metadata and evaluation labels, and stores the prepared files used by the embedding stages. The same preparation stage also defines the input units for each dataset setting.

For SynIOL, row-level samples and fixed event windows are created within the pipeline from the generated trajectories. Row-level preparation treats each generated event as one sample. Window-level preparation groups consecutive events within each trajectory using a window size of 32 and a stride of 8. Trajectory-level grouped splitting keeps related rows or windows from the same trajectory in the same split, preventing neighbouring windows from appearing across train and test partitions.

For the remaining datasets, the input unit is selected according to the structure of the log source and the corresponding evaluation target. Table 4.1 summarises the input unit and split rule for each dataset setting.

**Table 4.1:** Prepared input units and split rules.

| Dataset setting | Input unit | Main observable input | Split rule |
|---|---|---|---|
| PLC-like Log | Structured log row | Event fields, message, JSON payload | Stratified by label type |
| HDFS | Block trace | Ordered `EventId` sequence | Stratified by block label |
| IoT-Enriched Event Log | Subprocess instance | Ordered subprocess activity sequence and event count | Stratified by weak failure label |
| SynIOL row | Event row | Event fields, state, message, sensor/control values | Grouped by trajectory |
| SynIOL window | Fixed event window | Field, message, and numeric sequences | Grouped by trajectory |

## 4.3 Baseline Embeddings

Baseline embeddings provide non-neural reference representations. They make the amount of directly recoverable structure visible before self-supervised encoder training is introduced. Strong baseline performance is therefore interpreted as evidence that part of the evaluated structure is already present in explicit observable features. Table 4.2 summarises the baseline families used in each dataset setting.

**Table 4.2:** Baseline embedding methods.

| Dataset setting | Feature source | Baseline methods |
|---|---|---|
| PLC-like Log | Structured fields; message text; payload text and numeric payload values | Structured one-hot; message TF-IDF; payload TF-IDF + numeric; combined strong |
| HDFS | Event identifier sequence; sequence length; rare-event statistics | Event-count SVD; event TF-IDF n-gram SVD; sequence statistics |
| IoT-Enriched Event Log | Subprocess activity sequence; subprocess event count | Subactivity count SVD; subactivity TF-IDF SVD; subactivity n-gram SVD; sequence complexity |
| SynIOL row | Categorical fields; message text; sensor/control values | Categorical SVD; message TF-IDF SVD; numeric summary; combined baseline |
| SynIOL window | Event sequence; component, severity, state, and numeric sequences | Event count SVD; event TF-IDF SVD; event n-gram SVD; structured summary; combined baseline |

All baseline transformations are fitted on the training split. Categorical fields are encoded with one-hot representations. Text and event-sequence features use term frequency–inverse document frequency (TF-IDF) or count vectorisation. Sequence-statistics baselines use numeric summaries such as length, unique-token counts, rare-event counts, or numeric sequence summaries. When the fitted feature space is larger than the target embedding size, truncated singular value decomposition (SVD) is used to produce 64-dimensional dense embeddings. The same fitted transformations are then applied to validation and test samples.

## 4.4 Learned Embedding Methods

The learned embedding design follows the input units defined during dataset preparation. Each learned model maps one prepared input unit to a 64-dimensional embedding. The architecture choices reflect the structure of the corresponding input: row encoders are used for single structured events, while sequence encoders are used for block traces, subprocess instances, and SynIOL windows. Exact input representations, architecture settings, objectives, prediction heads, and embedding sources are listed in Appendix Tables A.4–A.6.

The learned methods represent common self-supervised encoder and objective choices. This keeps the analysis focused on how these choices behave under the same benchmark protocol, with limited architectural confounding. Row encoders

test whether compact structured events can be compressed from explicit fields, text, and numerical values. GRU and Transformer sequence encoders test whether ordered input units benefit from recurrent or attention-based context modelling.

The objective variants cover four representation assumptions. Reconstruction examines input preservation. Masked reconstruction examines contextual recovery. Contrastive training examines neighbourhood organisation under weak perturbations. Hybrid training combines preservation and similarity shaping. These choices keep the learned methods simple and make their behaviour easier to interpret against strong non-neural baselines. Figure 4.2 shows the resulting design in terms of input unit, encoder type, embedding source, and training objective.

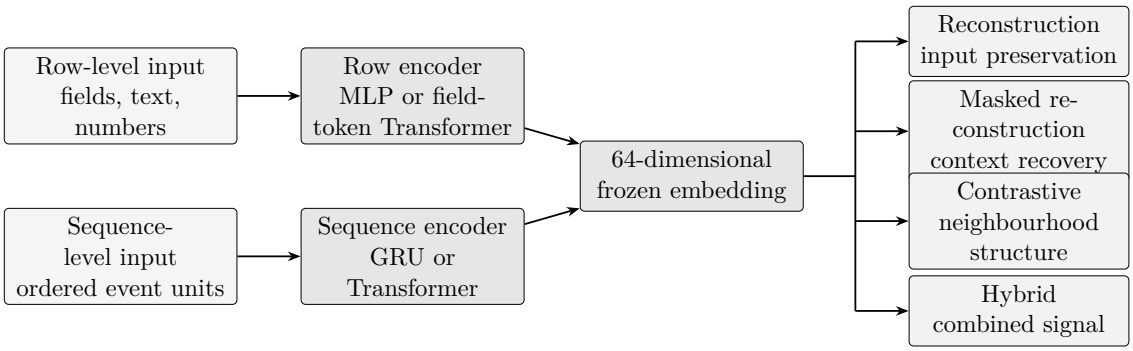


**Figure 4.2:** Reference learned-method design used to compare row and sequence encoders under shared self-supervised objectives.

**PLC-like Log.** The PLC-like setting uses row-level inputs. One encoder combines structured event information, textual content, and numerical payload values before projecting the combined representation to the embedding space. For reconstruction-based training, prediction heads are attached for the observable row components. A Transformer-based row model converts the same PLC-like log row into field and text tokens; the classification-token output of the encoder is used as the row embedding.

**HDFS.** For HDFS, the prepared block trace forms the input sequence. Token and positional embeddings are passed through a Transformer sequence encoder, after which masked mean pooling produces a single block representation. A final projection gives the embedding used in evaluation. The reconstruction objective predicts event tokens from either the full or masked sequence representation.

**IoT-Enriched Event Log.** The IoT setting operates on subprocess instances. A Transformer sequence encoder processes the ordered subprocess activities, and the pooled sequence output is combined with a projected numeric context feature. The final projection produces the subprocess embedding. Masked activity-token prediction supplies the reconstruction signal.

**SynIOL.** SynIOL uses both row-level and window-level learned embeddings. Row models encode one generated event at a time by combining structured fields, message information, and numerical values. Window models first construct an event

representation for each position and then model the ordered window with either a bidirectional GRU or a Transformer. Their reconstruction heads predict observable event-level components at sequence positions.

## 4.5 Training Objectives and Augmentations

The implemented learned methods use four training variants: full reconstruction, masked reconstruction, contrastive training, and masked-contrastive hybrid training. Their benchmark instantiations define the prediction targets, masking rules, augmentation rules, and loss terms used during training. Method-specific masking and augmentation probabilities are reported in Appendix Table A.7.

For reconstruction variants, categorical and token predictions use cross-entropy. Textual row targets are reconstructed as bag-of-words targets with binary cross-entropy. Numerical targets use mean squared error (MSE), weighted by 0.25. The reconstruction loss is:

$$\mathcal{L}_{reconstruction} = \mathcal{L}_{CE} + \mathcal{L}_{BCE} + 0.25\,\mathcal{L}_{MSE},$$

where $\mathcal{L}_{CE}$ is used for categorical fields and event tokens, $\mathcal{L}_{BCE}$ is used for bag-of-words text targets, and $\mathcal{L}_{MSE}$ is used for numerical targets. For masked reconstruction, the loss is computed only on the masked components where applicable.

For contrastive variants, each sample is converted into two augmented views. The normalised temperature-scaled cross-entropy (NT-Xent) loss is used:

$$\mathcal{L}_{contrastive} = -\frac{1}{2B}\sum_{i=1}^{2B}\log\frac{\exp(\mathrm{sim}(z_i, z_{p(i)})/\tau)}{\sum_{j=1,j\neq i}^{2B}\exp(\mathrm{sim}(z_i, z_j)/\tau)},$$

where $B$ is the batch size, $z_{p(i)}$ is the positive view paired with $z_i$, sim is cosine similarity after normalisation, and $\tau = 0.2$. Hybrid methods use:

$$\mathcal{L}_{hybrid} = \mathcal{L}_{reconstruction} + 0.2\,\mathcal{L}_{contrastive}.$$

The loss weights and temperature are fixed across the experiments. They are treated as shared benchmark settings to keep method comparisons consistent across dataset outcomes.

## 4.6 Embedding Extraction

Embeddings are extracted for the training, validation, and test splits after model training. Extraction uses the uncorrupted prepared samples, without masking, augmentation, or sequence cropping. The embedding is always taken from the encoder-side representation before reconstruction heads or contrastive projection heads. Sequence models use the pooled and projected sequence representation. Extracted learned embeddings are normalised to unit length, so the evaluated vector represents the encoder output used for downstream evaluation.

## 4.7 Embedding Evaluation

The evaluation stage applies a common protocol to the extracted embeddings. Probe models are fitted only on top of frozen embeddings, while retrieval and clustering diagnostics are computed directly in the embedding space. The protocol combines supervised probes, nearest-neighbour retrieval, clustering diagnostics, numerical probing, and weak-outcome checks according to the dataset setting, as summarised in Table 4.3.

**Table 4.3:** Evaluation views across dataset settings.

| Dataset setting | Probe | Retrieval | Clustering | Numeric probing | Weak outcome |
|---|---|---|---|---|---|
| PLC-like Log | ✓ | ✓ | ✓ | ✓ | Triage and fault category |
| HDFS | ✓ | ✓ | ✓ | Diagnostic sequence length | Anomaly label |
| IoT-Enriched Event Log | ✓ | ✓ | – | ✓ | Failure label |
| SynIOL row | ✓ | ✓ | ✓ | ✓ | – |
| SynIOL window | ✓ | ✓ | ✓ | ✓ | – |

Categorical probes use logistic regression with balanced class weights, and numerical probes use ridge regression. Probe inputs are standardised with training-split statistics. Retrieval uses train embeddings as the reference set and test embeddings as queries, with cosine distance and $k \in \{1, 5, 10\}$. IoT retrieval additionally reports mean subprocess activity-set Jaccard overlap. Clustering uses K-means on the test split, with the number of clusters matched to the evaluated target and 20 initialisations.

Evaluation targets are selected according to the input unit and dataset role. PLC targets cover event, component, triage, anomaly, and numerical payload information. HDFS focuses on the block-level anomaly label, with sequence length used as a diagnostic numerical target. IoT targets cover activity, resource, duration, retrieved activity overlap, and the weak failure outcome. SynIOL targets cover event-level attributes, behaviour identity, transition status, numerical state, and window-level behaviour summaries. Weak outcome targets are interpreted together with the other evaluation views, since they can reflect only part of the information available in an input unit.

## 4.8 Experimental Settings

Experiments are run on the Alvis cluster. Learned-model jobs use GPU nodes, while dataset preparation, baseline construction, evaluation, reporting, and seed aggregation run as CPU-only jobs. Reporting the computing environment supports reproducibility and provides context for runtime comparisons.

The common data, embedding, training, and cluster settings are listed in Tables 4.4, 4.5, and 4.6. The same split ratio, embedding dimension, and random seeds are used across dataset settings unless the input unit requires a specific grouped split.

**Table 4.4:** Experiment configuration.

| Setting | Value |
|---|---|
| Train/validation/test split | 70/15/15 |
| Random seeds | 42, 61, 2026 |
| Embedding dimension | 64 |

**Table 4.5:** Training configuration.

| Setting | Value |
|---|---|
| Optimizer | AdamW |
| Learning rate | $1 \times 10^{-3}$ |
| Weight decay | $1 \times 10^{-4}$ |
| Maximum epochs | 100 |
| Validation frequency | Every 5 epochs |
| Early stopping | 4 validation checks without improvement |
| Minimum improvement threshold | $1 \times 10^{-4}$ |
| Gradient clipping | Maximum norm 5.0 |
| Dropout | 0.1 |
| Row-model batch size | 512 |
| Sequence-model batch size | 256 |
| Extraction batch size | 512 |

**Table 4.6:** Cluster configuration.

| Setting | Value |
|---|---|
| Cluster | Alvis |
| Scheduler | SLURM |
| Learned-model jobs | One NVIDIA T4 GPU per job |
| GPU memory | 16 GB per T4 GPU |
| CPU allocation | 4 CPU cores per GPU |
| System memory allocation | 72 GB or 192 GB per GPU, depending on node type |
| CPU-only stages | Dataset preparation, baselines, evaluation, reporting, seed aggregation |

Together, these settings define the common experimental conditions used when comparing methods. They also make the runtime results interpretable as benchmark measurements under a shared computing environment, not as hardware-independent estimates.

# 5

# Study Results

The empirical results are organised by log representation setting. This organisation makes the main patterns easier to interpret in relation to input units, observable structure, evaluation targets, and runtime.

## 5.1 Result Reporting and Analysis Strategy

Throughout this chapter, table values are reported as means over the evaluated random seeds. Standard deviations are shown after ± and omitted when they round to 0.000.

The analysis focuses on result patterns across settings and evaluation views. The main comparisons concern baseline strength, learned-method gains, temporal context, disagreement between evaluation views, and runtime.

## 5.2 Structured Row-level Logs

The PLC-like Log setting represents structured row-level logs. Each sample contains explicit event fields, message text, payload information, and numerical values. This setting tests whether embeddings add value when much of the evaluated structure is already present in observable row fields.

Tables 5.1 and 5.2 report the task-oriented metrics first, followed by structure, numerical recovery, and runtime.

Several row-level PLC-like embeddings reach near-ceiling task performance. The combined strong baseline, structured one-hot baseline, and MLP autoencoder obtain perfect or near-perfect scores on the task-oriented metrics in Table 5.1. In this structured row-level setting, explicit fields and payload information already contain much of the signal used by the evaluated targets.

The structure-oriented results in Table 5.2 separate the methods more clearly. The

**Table 5.1:** PLC-like Log task-oriented results. Bold indicates representative strong results.

| Family | Method | Triage F1 | Triage P@5 | Event P@5 | Anom. P@5 | Event F1 | Comp. F1 | Fault F1 | Down. Triage F1 |
|---|---|---|---|---|---|---|---|---|---|
| Baseline | **Combined strong** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** |
| Baseline | Structured one-hot | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Baseline | Payload TF-IDF + numeric | 0.937±0.009 | 0.995±0.001 | 1.000 | 0.995±0.001 | 0.999 | 0.680±0.013 | 0.937±0.009 | 0.937±0.009 |
| Baseline | Message TF-IDF + SVD | 0.937±0.009 | 0.995±0.001 | 1.000 | 0.995±0.001 | 1.000 | 0.958±0.008 | 0.937±0.009 | 0.937±0.009 |
| Learned | **MLP autoencoder** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** |
| Learned | Row contrastive encoder | 0.997±0.006 | 0.999±0.001 | 1.000 | 0.999±0.001 | 1.000 | 1.000 | 0.997±0.006 | 0.997±0.006 |
| Learned | Masked row autoencoder | 0.946±0.007 | 0.997±0.001 | 1.000 | 0.997±0.001 | 1.000 | 1.000 | 0.946±0.007 | 0.946±0.007 |
| Learned | Masked-contrastive hybrid | 0.938±0.008 | 0.997±0.001 | 1.000 | 0.997±0.001 | 1.000 | 1.000 | 0.938±0.008 | 0.938±0.008 |
| Learned | Field-token Transformer | 0.937±0.009 | 0.997±0.001 | 1.000 | 0.997±0.001 | 1.000 | 0.991±0.009 | 0.937±0.009 | 0.937±0.009 |

**Table 5.2:** PLC-like Log embedding structure, numerical recovery, and runtime. Bold indicates representative strong results.

| Family | Method | Event-family ARI | Triage ARI | Component ARI | Anom. ARI | Num. $R^2$ | Num. Spearman | Runtime (min) |
|---|---|---|---|---|---|---|---|---|
| Baseline | **Combined strong** | **0.674±0.029** | 0.129±0.102 | 0.417±0.015 | 0.129±0.102 | **1.000** | **1.000** | 0.053 |
| Baseline | Structured one-hot | 0.595±0.023 | 0.124 | **0.730±0.106** | 0.124 | 0.079 | 0.094±0.011 | 0.003 |
| Baseline | Payload TF-IDF + numeric | 0.358±0.035 | **0.274±0.145** | 0.366±0.012 | **0.274±0.145** | **1.000** | **1.000** | 0.035 |
| Baseline | Message TF-IDF + SVD | 0.165±0.020 | -0.081±0.007 | 0.122±0.038 | -0.081±0.007 | 0.096±0.066 | 0.291±0.013 | 0.018 |
| Learned | **MLP autoencoder** | 0.591±0.058 | 0.096±0.019 | 0.592±0.012 | 0.096±0.019 | **0.938±0.018** | **0.965±0.014** | 1.543±0.312 |
| Learned | Row contrastive encoder | 0.105±0.022 | -0.004±0.012 | 0.048±0.002 | -0.004±0.012 | 0.725±0.017 | 0.793±0.009 | 1.627±0.066 |
| Learned | Masked row autoencoder | 0.639±0.072 | 0.076±0.053 | 0.545±0.045 | 0.076±0.053 | 0.614±0.026 | 0.729±0.021 | 1.983±0.847 |
| Learned | Masked-contrastive hybrid | 0.499±0.072 | 0.077±0.064 | 0.344±0.136 | 0.077±0.064 | 0.605±0.013 | 0.709±0.010 | 2.360±0.741 |
| Learned | Field-token Transformer | 0.329±0.081 | 0.070±0.050 | 0.314±0.160 | 0.070±0.050 | 0.440±0.065 | 0.603±0.055 | 19.471±3.148 |

combined strong baseline, payload TF-IDF with numeric features, and MLP autoencoder give the strongest numerical recovery because they preserve the continuous payload directly. The MLP autoencoder performs well here because the PLC-like input is a compact row-level record and the reconstruction objective keeps the numerical fields close to the input. Clustering is less uniform. Structured one-hot gives the highest component clustering, while the combined strong baseline gives the highest event-family clustering.

## 5.3 Operational Row and Window Logs

SynIOL provides a controlled operational-log setting in which row-level and window-level embeddings can be compared under the same latent behaviour design. The row-level view evaluates how much behaviour, transition, and numerical information can be recovered from individual generated events. The window-level view adds local temporal context by grouping consecutive events within a trajectory.

The row-level results in Table 5.3 focus on targets that remain discriminative across the clean, moderate-noise, and high-noise versions. Low-level event-field metrics are not included because they are near ceiling for most methods.

Behaviour and transition scores decrease as noise increases. Numerical recovery stays high for the numeric summary and combined baselines because these baselines retain the relevant continuous fields explicitly. This helps explain why the row autoencoder does not provide the same advantage as the MLP autoencoder in the PLC-like setting. For SynIOL, the numerical target is already captured by simple numeric features, while behaviour and transition targets depend more on combinations of categorical fields and local context. Among learned row-level methods, masked field

**Table 5.3:** SynIOL row-level representative results. Bold indicates representative strong results.

| Family | Method | Behav. F1 | Behav. P@5 | Trans. F1 | Trans. P@5 | Num. $R^2$ | Behav. ARI | Runtime (min) |
|---|---|---|---|---|---|---|---|---|
| **Clean** | | | | | | | | |
| Baseline | Categorical SVD | 0.569±0.003 | 0.498±0.010 | 0.530±0.010 | 0.804±0.009 | 0.831±0.009 | **0.339±0.013** | 0.010 |
| Baseline | **Combined baseline** | **0.644±0.004** | **0.602±0.012** | 0.553±0.008 | **0.834±0.003** | **1.000** | 0.236±0.012 | 0.046±0.001 |
| Baseline | Message TF-IDF SVD | 0.387±0.002 | 0.343±0.004 | 0.528±0.006 | 0.794±0.003 | 0.390±0.005 | 0.000±0.004 | 0.031 |
| Baseline | Numeric summary | 0.463±0.008 | 0.462±0.010 | 0.345±0.005 | 0.819±0.003 | **1.000** | 0.248±0.012 | 0.006 |
| Learned | Row autoencoder | 0.621±0.003 | 0.603±0.012 | 0.560±0.005 | 0.832±0.004 | **0.995±0.001** | 0.126±0.007 | 1.042±0.005 |
| Learned | Row contrastive encoder | 0.627±0.005 | 0.582±0.011 | **0.616±0.006** | 0.830±0.003 | 0.944±0.009 | 0.019±0.003 | 1.074±0.079 |
| Learned | Masked-contrastive hybrid | 0.673±0.005 | 0.624±0.008 | 0.598±0.018 | 0.837±0.003 | 0.970±0.008 | 0.131±0.186 | 1.291±0.225 |
| Learned | **Masked field reconstruction** | **0.680±0.011** | **0.630±0.009** | 0.599±0.021 | **0.839±0.003** | 0.967±0.008 | 0.018±0.001 | 1.079±0.206 |
| **Moderate noise** | | | | | | | | |
| Baseline | Categorical SVD | 0.525±0.010 | 0.442±0.013 | 0.504±0.008 | 0.795±0.007 | 0.693±0.004 | **0.233±0.023** | 0.006 |
| Baseline | **Combined baseline** | **0.617±0.009** | **0.546±0.014** | 0.513±0.008 | **0.825±0.006** | **1.000** | 0.221±0.051 | 0.038±0.001 |
| Baseline | Message TF-IDF SVD | 0.363±0.013 | 0.309±0.006 | 0.514±0.009 | 0.789±0.005 | 0.300±0.008 | 0.000±0.002 | 0.026 |
| Baseline | Numeric summary | 0.453±0.011 | 0.420±0.008 | 0.360±0.009 | 0.814±0.005 | **1.000** | 0.222±0.029 | 0.005 |
| Learned | Row autoencoder | 0.593±0.007 | 0.545±0.014 | 0.527±0.004 | 0.823±0.006 | **0.996±0.001** | 0.077±0.025 | 0.971±0.013 |
| Learned | Row contrastive encoder | 0.596±0.003 | 0.533±0.015 | **0.607±0.013** | 0.822±0.006 | 0.945±0.008 | 0.015±0.011 | 1.021±0.004 |
| Learned | Masked-contrastive hybrid | 0.621±0.007 | 0.554±0.015 | 0.571±0.014 | 0.823±0.005 | 0.961±0.006 | 0.005±0.003 | 1.500±0.189 |
| Learned | **Masked field reconstruction** | **0.625±0.005** | **0.558±0.016** | 0.585±0.002 | **0.825±0.006** | 0.958±0.008 | 0.012±0.001 | 1.112±0.273 |
| **High noise** | | | | | | | | |
| Baseline | Categorical SVD | 0.467±0.011 | 0.390±0.009 | 0.478±0.005 | 0.802±0.005 | 0.558±0.007 | 0.155±0.011 | 0.006 |
| Baseline | **Combined baseline** | **0.566±0.011** | **0.476±0.006** | 0.488±0.007 | **0.815±0.002** | **1.000** | 0.169±0.021 | 0.037±0.001 |
| Baseline | Message TF-IDF SVD | 0.339±0.006 | 0.295±0.002 | 0.504±0.007 | 0.792±0.005 | 0.223±0.011 | -0.004±0.003 | 0.026 |
| Baseline | Numeric summary | 0.435±0.010 | 0.375±0.002 | 0.346±0.009 | 0.808±0.001 | **1.000** | **0.194±0.015** | 0.005 |
| Learned | Row autoencoder | 0.548±0.010 | 0.473±0.005 | 0.504±0.006 | 0.814±0.002 | **0.995** | 0.066±0.011 | 0.971±0.008 |
| Learned | Row contrastive encoder | 0.535±0.012 | 0.462±0.004 | **0.564±0.008** | 0.814±0.003 | 0.951±0.005 | 0.007±0.008 | 1.025±0.011 |
| Learned | Masked-contrastive hybrid | 0.552±0.014 | 0.478±0.005 | 0.543±0.003 | 0.815±0.002 | 0.958±0.007 | 0.007±0.009 | 1.215±0.384 |
| Learned | **Masked field reconstruction** | **0.557±0.008** | **0.481±0.004** | 0.552±0.008 | **0.816±0.002** | 0.947±0.001 | 0.010±0.001 | 1.014±0.169 |

reconstruction gives the strongest behaviour scores, and the row contrastive encoder gives the strongest transition F1.

The window-level results in Table 5.4 use the same representative targets and add primary-component probing. Compared with individual rows, fixed windows expose local order and repeated behaviour patterns. This gives the embedding methods more context for behaviour and transition recovery.

Behaviour and transition scores are generally higher in the window-level setting than in the row-level setting. This indicates that fixed windows contain useful local context that is not available from single rows.

Structured summary and combined baseline are the strongest baseline entries across most window-level targets because they preserve aggregated window information directly. The learned methods vary by metric: masked-contrastive hybrid is strongest on behaviour retrieval in the clean setting, masked event reconstruction is strongest on behaviour clustering, and contrastive or Transformer encoders reach the highest behaviour F1 in some noisy settings. These differences show that the window-level setting does not favour one learned objective uniformly.

## 5.4 Block-level System Event Sequences

HDFS represents block-level system event sequences. This setting tests whether embeddings preserve anomaly-related information in ordered event-template traces,

**Table 5.4:** SynIOL window-level representative results. Bold indicates representative strong results.

| Family | Method | Prim. Comp. F1 | Behav. F1 | Behav. P@5 | Trans. F1 | Trans. P@5 | Num. $R^2$ | Behav. ARI | Runtime (min) |
|---|---|---|---|---|---|---|---|---|---|
| **Clean** | | | | | | | | | |
| Baseline | Combined baseline | 0.680±0.020 | **0.752±0.006** | 0.714±0.002 | **0.800±0.018** | 0.785±0.004 | **0.990** | 0.506±0.006 | 0.070 |
| Baseline | Event count SVD | 0.248±0.019 | 0.533±0.022 | 0.422±0.002 | 0.564±0.003 | 0.608±0.004 | 0.742±0.033 | 0.218±0.022 | 0.006 |
| Baseline | Event n-gram SVD | 0.289±0.018 | 0.513±0.015 | 0.411±0.007 | 0.561±0.018 | 0.607±0.002 | 0.729±0.028 | 0.225±0.012 | 0.014 |
| Baseline | Event TF-IDF SVD | 0.250±0.022 | 0.528±0.021 | 0.421±0.002 | 0.580±0.003 | 0.605±0.007 | 0.755±0.034 | 0.233±0.011 | 0.006 |
| Baseline | **Structured summary** | **0.684±0.019** | **0.750±0.004** | **0.722±0.008** | **0.800±0.016** | **0.801±0.006** | **0.990** | 0.507±0.007 | 0.050 |
| Learned | Contrastive encoder | 0.662±0.024 | 0.764±0.011 | 0.638±0.010 | 0.742±0.019 | 0.749±0.017 | 0.975±0.002 | 0.334±0.025 | 2.356±0.029 |
| Learned | **Masked-contrastive hybrid** | 0.335±0.040 | **0.777±0.018** | **0.700±0.010** | 0.765±0.041 | **0.786±0.013** | 0.973±0.004 | 0.397±0.049 | 2.578±0.201 |
| Learned | **Masked event reconstruction** | 0.452±0.040 | 0.759±0.011 | 0.694±0.003 | **0.792±0.017** | 0.772±0.004 | **0.992±0.001** | **0.529±0.016** | 0.601±0.049 |
| Learned | Sequence autoencoder | **0.679±0.013** | 0.750±0.006 | 0.691±0.003 | 0.744±0.008 | 0.751±0.009 | **0.990±0.002** | 0.516±0.008 | 0.788±0.186 |
| Learned | Transformer encoder | 0.535±0.087 | 0.764±0.014 | 0.682±0.001 | 0.758±0.016 | 0.727±0.014 | 0.970±0.003 | 0.433±0.086 | 2.634±0.040 |
| **Moderate noise** | | | | | | | | | |
| Baseline | Combined baseline | 0.646±0.024 | **0.732±0.023** | 0.684±0.026 | **0.733±0.017** | 0.736±0.008 | **0.981±0.001** | 0.447±0.026 | 0.069 |
| Baseline | Event count SVD | 0.248±0.012 | 0.523±0.018 | 0.411±0.012 | 0.552±0.025 | 0.586±0.007 | 0.723±0.011 | 0.180±0.019 | 0.006 |
| Baseline | Event n-gram SVD | 0.241±0.010 | 0.504±0.021 | 0.386±0.020 | 0.550±0.018 | 0.590±0.007 | 0.704±0.007 | 0.202±0.027 | 0.014 |
| Baseline | Event TF-IDF SVD | 0.252±0.007 | 0.520±0.019 | 0.407±0.011 | 0.569±0.013 | 0.588±0.004 | 0.735±0.010 | 0.188±0.023 | 0.006 |
| Baseline | **Structured summary** | **0.650±0.029** | **0.724±0.021** | **0.687±0.024** | 0.727±0.019 | **0.747±0.007** | **0.981±0.001** | 0.446±0.026 | 0.049 |
| Learned | Contrastive encoder | 0.636±0.026 | 0.758±0.016 | 0.606±0.008 | **0.770±0.019** | 0.738±0.013 | 0.972±0.005 | 0.228±0.144 | 2.324±0.033 |
| Learned | Masked-contrastive hybrid | 0.345±0.057 | 0.761±0.009 | 0.648±0.015 | 0.736±0.032 | **0.757±0.010** | 0.967±0.002 | 0.394±0.041 | 2.613±0.200 |
| Learned | **Masked event reconstruction** | 0.416±0.041 | 0.758±0.021 | 0.656±0.008 | 0.754±0.036 | 0.727±0.004 | **0.988±0.002** | **0.457±0.039** | 0.587±0.023 |
| Learned | Sequence autoencoder | **0.649±0.021** | 0.745±0.036 | **0.662±0.020** | 0.727±0.031 | 0.710±0.010 | **0.988±0.002** | **0.456±0.036** | 0.671±0.084 |
| Learned | **Transformer encoder** | 0.485±0.051 | **0.775±0.012** | 0.659±0.016 | 0.754±0.008 | 0.722±0.011 | 0.971±0.003 | 0.408±0.041 | 2.608±0.025 |
| **High noise** | | | | | | | | | |
| Baseline | Combined baseline | 0.640±0.016 | **0.711±0.018** | **0.621±0.009** | **0.739±0.005** | 0.703±0.003 | **0.966±0.004** | **0.402±0.027** | 0.069±0.001 |
| Baseline | Event count SVD | 0.274±0.016 | 0.487±0.019 | 0.366±0.006 | 0.573±0.014 | 0.582±0.004 | 0.630±0.034 | 0.123±0.018 | 0.006 |
| Baseline | Event n-gram SVD | 0.265±0.001 | 0.468±0.020 | 0.350±0.008 | 0.548±0.009 | 0.588±0.003 | 0.611±0.030 | 0.122±0.024 | 0.014 |
| Baseline | Event TF-IDF SVD | 0.276±0.010 | 0.485±0.014 | 0.364±0.012 | 0.572±0.016 | 0.584±0.006 | 0.640±0.034 | 0.129±0.020 | 0.006 |
| Baseline | **Structured summary** | **0.638±0.014** | **0.711±0.021** | **0.621±0.009** | 0.730±0.008 | **0.706±0.001** | **0.965±0.004** | **0.399±0.025** | 0.049 |
| Learned | **Contrastive encoder** | 0.625±0.028 | 0.737±0.020 | 0.595±0.019 | **0.734±0.017** | 0.724±0.020 | 0.968±0.004 | 0.223±0.012 | 2.368±0.016 |
| Learned | Masked-contrastive hybrid | 0.374±0.005 | 0.721±0.021 | 0.616±0.017 | 0.733±0.045 | **0.749±0.010** | 0.963 | 0.357±0.011 | 2.590±0.210 |
| Learned | **Masked event reconstruction** | 0.426±0.021 | 0.723±0.015 | 0.613±0.015 | 0.728±0.015 | 0.702±0.011 | 0.980±0.001 | **0.407±0.039** | 0.611±0.074 |
| Learned | Sequence autoencoder | **0.646±0.022** | 0.719±0.021 | **0.618±0.015** | 0.706±0.016 | 0.676±0.001 | **0.984±0.003** | 0.399±0.024 | 0.608±0.005 |
| Learned | Transformer encoder | 0.511±0.033 | 0.727±0.012 | 0.603±0.009 | 0.722±0.008 | 0.682±0.007 | 0.968±0.004 | 0.382±0.010 | 2.638±0.019 |

while also showing whether high anomaly scores correspond to coherent embedding structure.

Tables 5.5 and 5.6 separate anomaly-oriented performance from embedding-space structure and sequence-length recovery.

**Table 5.5:** HDFS task-oriented results. Bold indicates representative strong results.

| Family | Method | Anom. F1 | AUROC | AUPRC | Recall | Precision | Overall P@5 | Anom. P@5 |
|---|---|---|---|---|---|---|---|---|
| Baseline | **Event-count SVD** | **0.957±0.003** | 0.973±0.005 | 0.929±0.010 | **0.944±0.010** | **0.891±0.011** | **0.998** | **0.926±0.009** |
| Baseline | Event TF-IDF n-gram + SVD | 0.945±0.004 | 0.977±0.006 | 0.939±0.007 | 0.946±0.011 | 0.846±0.010 | 0.996±0.001 | 0.897±0.010 |
| Baseline | Sequence statistics | 0.550±0.004 | 0.789±0.014 | 0.161±0.017 | 0.816±0.026 | 0.117±0.004 | 0.981±0.001 | 0.627±0.022 |
| Learned | Hybrid masked-contrastive sequence | 0.925±0.016 | **0.979±0.008** | **0.944±0.006** | 0.941±0.013 | 0.783±0.054 | 0.994±0.001 | 0.865±0.011 |
| Learned | Masked event autoencoder | 0.930±0.013 | 0.975±0.004 | 0.936±0.013 | 0.940±0.011 | 0.802±0.046 | 0.996 | 0.890±0.007 |
| Learned | **Sequence autoencoder** | **0.968±0.002** | 0.975±0.006 | 0.934±0.009 | **0.942±0.011** | **0.932±0.011** | **0.996±0.001** | **0.924±0.006** |
| Learned | Sequence contrastive encoder | 0.890±0.029 | 0.975±0.011 | 0.884±0.059 | 0.937±0.009 | 0.682±0.082 | 0.995 | 0.862±0.003 |

For HDFS, the task-oriented metrics in Table 5.5 are high for several embeddings. Event-count SVD and sequence autoencoder have the strongest anomaly F1, while the hybrid masked-contrastive sequence model has the highest AUROC and AUPRC.

The structure-oriented metrics in Table 5.6 give a different ranking. Label ARI remains low across all methods despite high retrieval and anomaly scores. Sequence-length recovery is highest for sequence statistics and event-count SVD because both retain direct information about event counts or block length. The learned sequence models encode event order and reconstruction or contrastive information in a fixed-size vector, but they are not optimised to preserve exact sequence length. This explains why learned methods can perform well on anomaly metrics while losing to simple baselines on the sequence-length probe.

**Table 5.6:** HDFS embedding structure, sequence recovery, and runtime. Bold indicates representative strong results.

| Family | Method | Label ARI | Label NMI | Silhouette | Seq. $R^2$ | Runtime (min) |
|---|---|---|---|---|---|---|
| Baseline | **Event-count SVD** | -0.001±0.005 | 0.001±0.001 | **0.775±0.079** | **0.986±0.024** | 0.039 |
| Baseline | Event TF-IDF n-gram + SVD | **0.062±0.002** | **0.036±0.003** | 0.450 | 0.810±0.062 | 0.093±0.001 |
| Baseline | Sequence statistics | 0.014±0.002 | 0.009±0.006 | 0.441±0.351 | **1.000** | 0.023 |
| Learned | Hybrid masked-contrastive sequence | 0.002±0.019 | 0.025±0.002 | 0.356±0.014 | 0.836±0.046 | 19.530±1.081 |
| Learned | Masked event autoencoder | -0.009±0.005 | 0.001±0.001 | 0.598±0.030 | 0.831±0.049 | 3.219±1.245 |
| Learned | **Sequence autoencoder** | **0.019±0.048** | 0.013±0.001 | **0.748±0.014** | **0.873±0.029** | **1.433±0.193** |
| Learned | Sequence contrastive encoder | -0.008±0.010 | 0.018±0.015 | 0.311±0.013 | 0.826±0.065 | 22.510±2.033 |

## 5.5 Process-oriented Subprocess Logs

The IoT-Enriched Event Log represents process-oriented subprocess logs. Each input unit is an ordered subprocess activity sequence with operation context, duration, and a weak failure outcome. This setting separates semantic activity recovery from numerical duration recovery and weak outcome prediction.

Tables 5.7 and 5.8 separate semantic retrieval and activity recovery from duration, failure, and runtime metrics.

**Table 5.7:** IoT-Enriched Event Log semantic and retrieval results. Bold indicates representative strong results.

| Family | Method | Jaccard@5 | Jaccard@10 | Activity F1 | Activity P@5 | Resource F1 | Main Act. F1 | Main Act. P@5 |
|---|---|---|---|---|---|---|---|---|
| Baseline | Sequence complexity | 0.179±0.010 | 0.174±0.010 | 0.478±0.009 | 0.442±0.041 | 0.478±0.009 | 0.218±0.008 | 0.361±0.012 |
| Baseline | **Subactivity count SVD** | **0.985±0.002** | **0.980±0.002** | **1.000** | **1.000** | **1.000** | 0.740±0.019 | **0.892±0.006** |
| Baseline | Subactivity n-gram SVD | 0.925±0.005 | 0.918±0.006 | 0.964±0.011 | 0.940±0.002 | 0.964±0.011 | 0.714±0.012 | 0.833±0.007 |
| Baseline | **Subactivity TF-IDF SVD** | **0.985±0.002** | **0.980±0.002** | **1.000** | **1.000** | **1.000** | **0.748±0.009** | **0.892±0.003** |
| Learned | Hybrid masked-contrastive | 0.976±0.003 | 0.967±0.002 | 0.997±0.003 | 0.996±0.002 | 0.997±0.003 | 0.739±0.018 | **0.892±0.011** |
| Learned | **Masked subactivity autoencoder** | 0.975±0.003 | 0.968±0.003 | **1.000** | 0.999±0.001 | **1.000** | **0.749±0.015** | 0.891±0.007 |
| Learned | Subprocess contrastive encoder | 0.976±0.001 | 0.969±0.001 | 0.995±0.003 | 0.994±0.003 | 0.995±0.003 | 0.724±0.017 | **0.892±0.009** |

**Table 5.8:** IoT-Enriched Event Log duration, failure, and runtime. Bold indicates representative strong results.

| Family | Method | Duration $R^2$ | Duration MAE | Duration Spearman | Failure AUROC | Failure AUPRC | Failure F1 | Runtime (s) |
|---|---|---|---|---|---|---|---|---|
| Baseline | Sequence complexity | 0.154±0.096 | 10.335±0.779 | 0.509±0.069 | 0.687±0.116 | 0.089±0.049 | 0.462±0.002 | **0.008** |
| Baseline | **Subactivity count SVD** | **0.562±0.173** | **3.703±0.368** | **0.903±0.015** | 0.763±0.062 | 0.064±0.018 | 0.470±0.018 | 0.100±0.001 |
| Baseline | Subactivity n-gram SVD | 0.534±0.154 | 3.928±0.342 | 0.893±0.020 | 0.756±0.054 | 0.058±0.015 | 0.457±0.027 | 0.117±0.003 |
| Baseline | **Subactivity TF-IDF SVD** | 0.549±0.160 | 3.768±0.343 | **0.903±0.017** | 0.730±0.090 | 0.058±0.025 | 0.462±0.024 | 0.101±0.001 |
| Learned | Hybrid masked-contrastive | 0.545±0.185 | 4.419±0.468 | 0.886±0.023 | 0.790±0.058 | 0.099±0.028 | 0.463±0.015 | 38.267±0.786 |
| Learned | **Masked subactivity autoencoder** | 0.504±0.145 | 4.932±0.162 | 0.873±0.018 | **0.815±0.057** | **0.189±0.048** | **0.480±0.026** | 6.073±0.634 |
| Learned | Subprocess contrastive encoder | **0.552±0.183** | 4.492±0.512 | 0.891±0.021 | 0.799±0.026 | 0.095±0.048 | 0.470±0.005 | 24.126±4.795 |

For the IoT-Enriched Event Log, the strongest semantic and retrieval results come from subactivity-based representations. Subactivity count SVD, subactivity TF-IDF SVD, and masked subactivity autoencoder are the strongest methods in Table 5.7. The ranking is consistent with the dataset structure, where activity-module and resource-type targets are closely tied to the observed subactivity sequence.

The duration and failure targets in Table 5.8 rank methods differently. Duration recovery is strongest for subactivity-based baselines, while masked subactivity autoencoder has the strongest failure-related scores among the reported methods. The difference indicates that semantic activity recovery, duration recovery, and failure prediction use different parts of the representation.

## 5.6 Cross-setting Result Patterns

The setting-level results above show that method rankings change with the evaluated information type. The main observations are summarised in Table 5.9.

**Table 5.9:** Cross-setting summary by evaluation focus.

| Evaluation focus | Relevant settings | Observed pattern |
|---|---|---|
| Explicit field and activity recovery | PLC-like Log, SynIOL row, IoT | Structured, count, TF-IDF, and combined baselines are often strong |
| Numerical and duration recovery | PLC-like Log, SynIOL, HDFS, IoT | Numeric summaries and sequence statistics perform well |
| Context and transition recovery | SynIOL windows, HDFS | Window and sequence models gain from local order and surrounding events |
| Weak outcome prediction | HDFS, IoT | Outcome scores can be high without clear global clusters |
| Neighbourhood and clustering structure | All settings | Probe and retrieval scores do not always match clustering scores |
| Runtime | All settings | Baselines are cheaper; learned models vary more |

This cross-setting view closes the results chapter by collecting the main empirical patterns.

# 6

# Discussion

This chapter connects the benchmark findings to the research questions, practical method selection, and threats to validity.

## 6.1 Answers to the Research Questions

**RQ1: Preserved log structure.** The results show that log embeddings preserve structure selectively. Explicit field, activity, text, and numerical targets are often preserved well by representations that retain observable input structure directly. Contextual targets, such as behaviour, transition, and anomaly-related sequence patterns, benefit more from input units that include surrounding events. Weak outcomes add another layer of uncertainty because they may reflect only part of the recorded behaviour. These patterns support the view that preserved structure depends on the input unit, the target, and the observable sources available to the embedding.

**RQ2: Baseline and self-supervised embeddings.** Strong baselines remain necessary comparison points. Count-based, TF-IDF, structured-field, and numerical-summary representations perform well when the target is close to observable fields, event frequencies, activity tokens, or simple sequence summaries. Learned embeddings are useful in selected settings, especially when reconstruction or sequence modelling helps combine fields, represent local context, or capture relations across events. Their advantage is conditional: model complexity is justified when it improves the evaluation view that matters for the intended use.

**RQ3: Consistency between evaluation views.** The evaluation views do not produce one stable ranking of methods. Probe, retrieval, clustering, numerical probing, weak outcome evaluation, and runtime expose different properties of the same embedding space. A representation can support accurate target recovery while showing weak global clustering, or it can preserve numerical information while giving weaker behavioural separation. This disagreement is a main result of the benchmark. Embedding quality is better interpreted as a profile of properties than as a single score.

## 6.2 Interpretation of the Main Result Patterns

The competitive performance of baselines is itself informative. It indicates that several benchmark targets are strongly connected to observable log structure. Explicit fields, event counts, activity tokens, lexical content, numerical payloads, sequence length, and duration summaries can carry enough information for many evaluation targets. In these cases, a representation does not need to infer hidden structure before it becomes useful; it mainly needs to retain the relevant observable signal in a stable vector form. Similar concerns have been raised in log anomaly detection and broader machine-learning benchmarks, where simple methods can reveal signals that learned models do not need to discover from scratch [20, 6].

The results also clarify where learned embeddings become more useful. Their strongest role appears when the target depends on context or relations across events. Window-level and sequence-level inputs provide evidence about local order, transitions, repeated patterns, and surrounding events that isolated rows cannot fully express. This helps explain why learned methods are more convincing for behaviour, transition, and anomaly-related sequence targets than for targets that are already close to single fields or direct summaries.

Weak outcome targets require careful interpretation. High anomaly or failure scores do not by themselves imply that the embedding space is globally organised around the same label. Outcome labels may reflect only part of the information contained in a block trace, window, or subprocess instance. Evaluating several views makes this limitation visible.

The disagreement between metrics is informative. Probe scores show whether a target can be recovered by a lightweight supervised model. Retrieval scores show whether local neighbourhoods contain similar samples. Clustering scores test whether the same target appears as a broader geometric organisation. Numerical probes and runtime add further constraints. When these views disagree, the result should not be treated as noise in the benchmark; it indicates that the embedding preserves some properties of the input unit more clearly than others.

## 6.3 Implications

For practical log analysis, method selection should start from the input unit, the available fields, and the target of interest. If the target is close to event counts, structured identifiers, activity tokens, text, or numerical summaries, a transparent baseline may be sufficient and easier to inspect. Learned embeddings are more compelling when the target requires combinations of fields, local temporal context, relations across events, or compact sequence representations.

Benchmark design should therefore include strong baselines, frozen-embedding evaluation, and several complementary metrics. A single downstream score can hide

whether the representation preserves numerical state, local neighbourhoods, global groups, or weak outcome information. Runtime should also be reported, since baselines are consistently cheaper while learned models add training cost, configuration choices, and hardware dependence.

## 6.4 Threats to Validity

**Evaluation targets.** The conclusions depend on the selected targets. Alternative probe labels, retrieval labels, numerical variables, or weak outcome definitions could change some method rankings. The benchmark evaluates usefulness for the selected downstream views, not intrinsic embedding quality in general.

**Data quality and temporal irregularity.** The experiments assume that the benchmark inputs have been prepared into usable structured or sequence-level records. Real operational logs may contain missing fields, corrupted messages, duplicated events, inconsistent identifiers, delayed records, clock drift, irregular sampling intervals, or timestamp errors. These issues can strongly affect embedding quality because they change field availability, event order, window construction, duration estimates, and numerical summaries. The reported results therefore describe representation behaviour under prepared benchmark conditions.

**Synthetic datasets.** PLC-like Log and SynIOL are generated datasets. They provide controlled labels and known structure, but their generation processes may favour methods that match the designed fields, noise process, or sequence patterns. HDFS and IoT-Enriched Event Log add real-data settings, but they do not cover all log domains. Dataset scarcity and concentration around a small number of public log datasets remain known issues in log anomaly detection research [19].

**Dataset scale and coverage.** The real datasets differ strongly in scale and structure. HDFS is large and highly imbalanced, while IoT-Enriched Event Log is much smaller and process-oriented. These differences help test multiple log settings, but they also limit how far the results can be generalised.

**Randomness and model configuration.** The learned models depend on random initialisation, batching, sampling, masking, and augmentation. Multiple seeds are reported where applicable, but three seeds cannot cover all training variability. The results also use one main configuration per method, so rankings may change under different learning rates, batch sizes, early-stopping settings, temperatures, hybrid loss weights, or augmentation probabilities. This is especially relevant for contrastive and hybrid methods.

**Fixed representation size.** Most embeddings are compared at a common dimensionality. This supports controlled comparison, but it may not be optimal for every method family. Some baselines may naturally prefer sparse or higher-dimensional forms, while some learned methods may benefit from larger latent spaces.

**Computing environment and runtime.** Runtime is hardware- and configuration-dependent. Different GPUs, CPU allocations, memory limits, batch sizes, and implementation choices may change execution time. The reported runtime values are measurements under the experimental configuration used here, not hardware-independent estimates.

# 7

# Conclusion

This chapter summarises the main findings, gives final remarks, and outlines future work.

## 7.1 Summary of Findings

The benchmark did not identify a universally strongest embedding family. Its main finding is that embedding performance changes with the information made available by the input unit and with the property selected for evaluation. Transparent baselines are often sufficient for targets tied to explicit fields, tokens, counts, or numerical summaries. Learned encoders are more useful when the target depends on contextual or relational structure across events.

The study also shows the value of evaluating embeddings through several views. Probe, retrieval, clustering, numerical, weak-outcome, and runtime results describe different aspects of the same representation. Treating these results together gives a clearer basis for method selection than relying on one downstream metric alone.

## 7.2 Concluding Remarks

Taken together, the study frames log embedding as a representation-selection problem. Method choice should follow the intended analysis use and the information that must remain accessible after encoding.

## 7.3 Future Work

Future work should extend the benchmark with more real-world log domains, larger operational datasets, and task-specific downstream workflows. Further experiments should test broader hyperparameter and augmentation settings, study multimodal encoders more explicitly, and design evaluation targets that better separate event-level, sequence-level, numerical, and behavioural structure.

Another direction is to connect frozen-embedding evaluation with deployed analysis pipelines. This would make it possible to study how representation quality affects alert triage, maintenance support, process monitoring, and other operational tasks after task-specific models and human decision processes are introduced. Future benchmarks should also examine data-quality repair, temporal irregularity, and uncertainty handling, since these factors can substantially affect log representations in operational environments.

# Bibliography


[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607, 2020.

[3] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R. Lyu. Experience report: Deep learning-based system log analysis for anomaly detection. *arXiv preprint arXiv:2107.05908*, 2021.

[4] Qian Cheng, Amrita Saha, Wenzhuo Yang, Chenghao Liu, Doyen Sahoo, and Steven C. H. Hoi. Logai: A library for log analytics and intelligence. *arXiv preprint arXiv:2301.13415*, 2023.

[5] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[6] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 101–109, 2019.

[7] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 233–240, 2006.

[8] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[10] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, 2017.

[11] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[12] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[13] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[14] International Electrotechnical Commission. Iec 61131-3:2013, programmable controllers – part 3: Programming languages, 2013.

[15] International Electrotechnical Commission. Iec 62682:2022, management of alarm systems for the process industries, 2022.

[16] Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R. Lyu. A large-scale evaluation for log parsing techniques: How far are we? In *International Symposium on Software Testing and Analysis*, 2024.

[17] Simon Kamm, Sushma Sri Veekati, Timo Müller, Nasser Jazdi, and Michael Weyrich. A survey on machine learning based analysis of heterogeneous data in the industrial automation domain. *Computers in Industry*, 149:103930, 2023.

[18] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel C. Briand. Impact of log parsing on deep learning-based log anomaly detection. *Empirical Software Engineering*, 29(6), 2024.

[19] Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*, 12:100470, 2023.

[20] Max Landauer, Florian Skopik, and Markus Wurzenberger. A critical review of common log data sets used for evaluation of sequence-based anomaly detection

techniques. *Proceedings of the ACM on Software Engineering*, 1(FSE):1354–1375, 2024.

[21] Lukas Malburg, Joscha Grüger, and Ralph Bergmann. Dataset: An iot-enriched event log for process mining in smart factories, 2022.

[22] Lukas Malburg, Joscha Grüger, and Ralph Bergmann. An iot-enriched event log for process mining in smart factories. *arXiv preprint arXiv:2209.02702*, 2022.

[23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[24] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[25] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[26] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[27] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[28] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*, pages 1–10. IEEE, 2010.

[29] Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

[30] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. In *International Conference on Learning Representations*, 2021.

[31] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[33] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 2008.

[34] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, 2009.

[35] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8980–8987, 2022.

[36] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R. Lyu. Loghub: A large collection of system log datasets for ai-driven log analytics. In *IEEE International Symposium on Software Reliability Engineering*, 2023.

[37] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, pages 121–130, 2019.

# A

# Appendix

## A.1 PLC-like Log Event Vocabulary

Table A.1 lists the event vocabulary used in the PLC-like Log dataset. Each event type is associated with a fixed event identifier and a short description of its operational meaning.

**Table A.1:** Event vocabulary used in the PLC-like Log dataset.

| Event type | EventId | Description |
|---|---|---|
| SYSTEM_START | E589964 | PLC runtime or main controller startup after power-on or restart checks. |
| SYSTEM_STOP | E6AC533 | Controller stop request or runtime transition into stopped state. |
| SENSOR_READ | EC1B0CD | Periodic process measurement from a compatible field sensor. |
| ACTUATOR_CMD | E69CAD3 | Command issued to a physical actuator or drive. |
| ALARM_TEMP | ED9214E | Temperature limit violation raised by sensor or safety logic. |
| ALARM_PRESSURE | E023574 | Pressure limit violation raised by sensor or safety logic. |
| ALARM_CLEAR | EC3E297 | Acknowledgement or reset of a previously active alarm. |
| COMMUNICATION_OK | E29878A | Healthy communication heartbeat or restored fieldbus link. |
| COMMUNICATION_ERROR | E72EACE | Fieldbus, rack, or heartbeat communication fault. |
| WATCHDOG_TICK | E22CA18 | Controller scan watchdog service event. |
| CONFIG_CHANGE | E870B56 | Operator or configuration manager changed a process parameter. |
| MAINTENANCE_MODE | EA22941 | Automatic production suspended for manual maintenance or inspection. |
| PRODUCTION_CYCLE_START | E3F8798 | Production cycle entered its active run phase. |
| PRODUCTION_CYCLE_END | EE83056 | Production cycle completed or left its active run phase. |

## A.2 PLC-like Log Payload Fields

Table A.2 summarises the event-specific payload fields used in the PLC-like Log dataset. The payload is stored as compact JSON and provides structured contextual information for each event type.

**Table A.2:** Payload fields used in the PLC-like Log dataset.

| Event type | Payload fields |
|---|---|
| SYSTEM_START | None. |
| SYSTEM_STOP | stop_reason, interlock_status |
| SENSOR_READ | measurement, value, unit, signal_quality |
| ACTUATOR_CMD | command, position, target_position, state |
| ALARM_TEMP | value, threshold, excess, zone |
| ALARM_PRESSURE | value, threshold, excess, line |
| ALARM_CLEAR | alarm_code |
| COMMUNICATION_OK | remote_rack, latency_ms, retry_count, protocol |
| COMMUNICATION_ERROR | remote_rack, latency_ms, retry_count, protocol |
| WATCHDOG_TICK | scan_time_ms, jitter_ms, watchdog_limit_ms |
| CONFIG_CHANGE | parameter_name, old_value, new_value, recipe_id |
| MAINTENANCE_MODE | station_id |
| PRODUCTION_CYCLE_START | station_id, recipe_id |
| PRODUCTION_CYCLE_END | cycle_result, cycle_duration_ms, station_id, recipe_id |

## A.3 SynIOL Event Vocabulary

Table A.3 lists the event vocabulary used in SynIOL. The vocabulary is designed to cover common functional categories in industrial monitoring and control environments. The event names are intentionally abstract and are not tied to specific physical variables such as temperature or pressure. This design reduces the risk that a latent behaviour can be inferred from a single overly specific event token, and instead encourages evaluation based on temporal, contextual, and component-related structure.

## A.4 Learned Embedding Configuration

Table A.4 summarises the input representation and encoder used for each learned embedding setting. Table A.5 lists the corresponding architecture settings. Table A.6 reports the objective, head, and embedding-source settings. Table A.7 reports the masking and augmentation settings.

**Table A.3:** Event vocabulary used in SynIOL.

| **Event type** | **Functional category** | **Interpretation** |
|---|---|---|
| STATUS_CHECK | Monitoring | Periodic status or heartbeat check. |
| VALUE_READ | Monitoring | Process or telemetry value reading. |
| SETPOINT_UPDATE | Control | Update of a control target or setpoint. |
| CONTROL_COMMAND | Control | Dispatch of a control instruction. |
| ACTUATOR_RESPONSE | Control feedback | Response or acknowledgement from an actuator. |
| WARNING_RAISED | Abnormal condition | Warning-level operating deviation. |
| ERROR_DETECTED | Abnormal condition | Detected abnormal condition or process deviation. |
| COMM_DELAY | Communication | Communication latency or delayed packet sequence. |
| NETWORK_TIMEOUT | Communication | Timeout or watchdog expiration in communication. |
| DIAGNOSIS_START | Diagnosis | Start of a diagnostic or fault-isolation routine. |
| RECOVERY_ACTION | Recovery | Execution of a recovery or stabilization action. |
| RETRY_OPERATION | Retry | Repeated operation after a transient issue. |
| MAINTENANCE_CHECK | Maintenance | Scheduled inspection or maintenance validation. |
| HEALTH_OK | Healthy status | Report that the component or subsystem remains healthy. |

**Table A.4:** Learned embedding input and encoder settings.

| **Dataset setting** | **Input representation** | **Encoder** |
|---|---|---|
| PLC-like Log | Row-level structured event fields, message tokens, payload-text tokens, and payload numeric values | Row MLP or field-token Transformer |
| HDFS | Block-level event identifier sequence | Transformer sequence encoder |
| IoT-Enriched Event Log | Subprocess activity sequence plus subprocess event count | Transformer sequence encoder with numeric context |
| SynIOL row | Row-level event fields, message tokens, and sensor/control values | Row MLP |
| SynIOL window | Ordered event representations within a fixed window | Bidirectional GRU or Transformer window encoder |

**Table A.5:** Learned embedding architecture settings.

| **Encoder setting** | **Applies to** | **Configuration** |
|---|---|---|
| Row MLP | PLC-like Log; SynIOL row | Categorical embeddings 16; token embeddings 64; numeric projection 32; ReLU activation; dropout 0.1; embedding dimension 64 |
| PLC row model size | PLC-like Log | Hidden dimension 192; maximum message length 32; maximum payload-text length 24; vocabulary size 4000; optional field-token Transformer with 2 layers, 4 heads, model dimension 128, GELU activation, dropout 0.1 |
| SynIOL row model size | SynIOL row | Hidden dimension 128; maximum message length 24; vocabulary size 2000; embedding dimension 64 |
| Sequence Transformer | HDFS; IoT; SynIOL window Transformer | Token and positional embeddings; masked mean pooling; 1 Transformer layer; 4 attention heads; GELU activation; dropout 0.1; embedding dimension 64 |
| Sequence model size | HDFS; IoT; SynIOL window | Model dimensions 96, 128, and 64, respectively; maximum sequence lengths 128, 96, and 32, respectively; vocabulary size 2000 |
| Projection head | Contrastive sequence models | Two-layer GELU projection head; projection dimension 64 |
| Numeric context | IoT | Normalised subprocess event count projected and concatenated with the pooled sequence representation |

**Table A.6:** Learned embedding objective and head settings.

| Setting | Objectives | Prediction heads | Embedding source |
|---|---|---|---|
| PLC-like Log | Full reconstruction; masked reconstruction; contrastive; hybrid; field-token Transformer hybrid | Categorical heads, numeric head, message bag-of-words head, payload-text bag-of-words head | Row encoder output; CLS-token projection for field-token Transformer |
| HDFS | Full sequence reconstruction; masked event reconstruction; contrastive; hybrid | Event-token head at sequence positions | Masked-mean pooled Transformer output after embedding projection |
| IoT-Enriched Event Log | Masked subactivity reconstruction; contrastive; hybrid | Subactivity-token head at sequence positions | Pooled Transformer output concatenated with projected numeric context, then embedding projection |
| SynIOL row | Full reconstruction; masked field reconstruction; contrastive; hybrid | Categorical heads, numeric head, message-token head | Row encoder output |
| SynIOL window | Full window reconstruction; masked event reconstruction; contrastive; hybrid; Transformer hybrid | Per-position categorical heads, numeric head, message-token head | Masked-mean pooled GRU or Transformer output after embedding projection |

**Table A.7:** Learned embedding masking and augmentation settings.

| **Input type** | **Masked reconstruction** | **Contrastive augmentation** |
|---|---|---|
| PLC rows | Event type 0.10; component 0.05; severity 0.05; payload fields 0.10; message tokens 0.15; payload tokens 0.10; numeric fields 0.10 | Same field and token masking; numeric jitter with standard deviation 0.05 |
| SynIOL rows | Event type 0.10; component 0.05; severity 0.05; state 0.05; message tokens 0.15; numeric fields 0.10 | Message-token masking, one masked categorical field in one view, numeric jitter with standard deviation 0.05 |
| HDFS sequences | Event tokens masked with probability 0.15 | Token masking 0.15; event dropout 0.10; span masking 0.10; crop probability 0.10 |
| IoT subprocesses | Subactivity tokens masked with probability 0.15 | Token masking 0.15; token dropout 0.10; span dropout 0.10; crop probability 0.15 |
| SynIOL windows | Sequence fields masked with probability 0.15; message tokens 0.15; numeric fields 0.10 | Message-token masking 0.15; span masking up to 8 events; numeric jitter with standard deviation 0.05 |

**DEPARTMENT OF INDUSTRIAL AND MATERIALS SCIENCE**
**CHALMERS UNIVERSITY OF TECHNOLOGY**
Gothenburg, Sweden
www.chalmers.se


CHALMERS
UNIVERSITY OF TECHNOLOGY