TODO TITLE

A report submitted to the University of Manchester for the degree of Bachelor of Science in the Faculty of Science and Engineering

Author: Sai Putravu
Student id: 10829976
Supervisor: TODO

2025

School of Computer Science

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

Alphabetically sort this

AE Auto-encoding.

AR Auto-regressive.

BERT Bidirectional Encoder Representation from Transformers.

CBM Condition-based maintenance policy.

CLI Command Line Interface.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

CUDA Compute Unified Device Architecture.

DBSCAN Density-Based Spatial Clustering of Application with Noise.

DL Deep Learning.

ELMo Embeddings from Language Models.

EPB1 Extracted proton beam line 1.

EPB2 Extracted proton beam line 2.

FAP Fault Analysis Pathway.

FLD First-line diagnosis system.

GPT Generative Pre-trained Transformer.

GPU Graphics Processing Unit.

HDBSCAN Hierarchical Density-Based Spatial Clustering of Application with Noise.

HDD Hard-Disk Drive.

HEDS High energy drift space.

IoT Internet of Things.

LEBT Low energy beam transport line.

LINAC Linear Accelerator.

LSA Latent Semantic Analysis.

MCR Main Control Room.

ML Machine Learning.

MLM Masked Language Modelling.

MRL Matryoshka Representation Learning.

MST Minimum Spanning Tree.

MTEB Massive Text Embedding Benchmark.

NLP Natural Language Processing.

Operalog The ISIS Operational Log.

PAM Partitioning Around Medoids.

PCA Principle Component Analysis.

PdM Predictive maintenance policy.

PLM Pre-trained Language Model.

PvM Preventative maintenance policy.

R2F Run-to-failure maintenance policy.

RAM Random Access Memory.

RFQ Radio-frequency quadrupole.

RF Random Forest.

RoPE Rotary Positional Embeddings.

SAFE Supervised Aggregative Feature Extraction.

SMART Self-monitoring and reporting technology.

STFC Science and Technology Facilities Council.

SVD Singular Value Decomposition.

SVM Support Vector Machine.

TS-1 Target Station 1.

TS-2 Target Station 2.

t-SNE t-distributed Stochastic Neighbour Embedding.

UKRI United Kingdom Research and Innovation.

UMAP Uniform Manifold Approximation and Projection.

VRAM Video Random Access Memory.

Abstract

TODO

Chapter 1

Introduction

This project concerns the use of statistical and machine learning models to augment the predictive maintenance process at the STFC Rutherford Appleton Laboratory's ISIS Neutron and Muon research facility. The ISIS Neutron and Muon facility is a research centre for physical and life sciences, owned and operated by the STFC, a council that forms the UK Research and Innovation. In order to produce beams of neutrons and muons, allowing scientists to study materials at the atomic level, large and complex structures and machinery are required. The facility has a wealth of instrumentation taking multitudes of measurements to ensure that proper maintenance is completed in a timely manner.

Finish this off, also mention this is one of two parts. And this aims to focus on the Operalog, a rich textual record of real-world failures of the of the ISIS facility.
Also, mention that this is a very open-ended research project and what we should be judged on? i.e. progressing the space of auto-categorisation of industrial operational logs.
Also mention there is no "industry standard" for this to compare it to.

: Introduce the sections of the paper.

- Refine this after coming back
- Section 2,
- Section 3,
- ...

1.1 Motivation

Motivate the research project.

: The things in this section will include

- Introduce the problem: Auto-categorisation and label inference
- Highlight the research aims. This should highlight the vastly open nature of the project and then hone in on the particular issue I am tackling. (i.e. to progress the state of PdM ...)
- Identify the data input, expected output, data shape and explain why this motivates the project

1.2 The ISIS Facility's Complexity and Operational Challenges

The ISIS Neutron and Muon Source at STFC Rutherford Appleton Laboratory is a large-scale research facility enabling physical and life sciences study through the production of neutron and muon beams [86]. Achieving this requires a complex, multi-stage accelerator system, the operation and maintenance of which present significant challenges relevant to this project.

The core process, described in *A Practical Guide to the ISIS Neutron and Muon Source* (ISIS Neutron and Muon Source, 2021), involves accelerating negative hydrogen (H^-) ions through four major stages: an ion source, a radio-frequency quadrupole (RFQ), a linear accelerator (LINAC), and finally a large synchrotron ring. This chain ultimately produces an 800 MeV proton beam directed towards one of two target stations (TS-1 or TS-2) for neutron or muon generation [32]. A schematic overview is shown in Figure 1.1, with the particle path illustrated in Figure 1.2.

Successful operation across these stages demands extreme precision and stability. Throughout the entire acceleration process, maintaining an ultra-high vacuum (between 10^{-8} and 10^{-9} atmospheric pressure) is crucial, requiring tens of vacuum pumps and continuous monitoring, as leaks can disrupt the beam and damage sensitive components [32]. The initial H^- ion source must provide a stable, consistent beam, as inconsistencies can propagate and amplify downstream; component wear within the source itself is a common maintenance task requiring immediate intervention [32]. Accelerator structures, like those in the LINAC which utilise high-Q radio-frequency fields [53], require exceptional stability achieved through precise temperature regulation (significantly under $1°C$ variation) to mitigate thermal effects of expansion and contraction [32].

Within the 163m circumference synchrotron, magnetic fields and accelerating voltages must be ramped in precise synchronism over nearly 8000 revolutions to boost the beam energy to 800 MeV. Any deviation risks beam oscillation, particle loss, and component damage which necessitates continuous, precise monitoring of beam position and machine parameters via sensor networks [32]. Even the injection mechanism, which uses charge-exchange via a thin foil to convert H^- ions to protons [32, 6], relies on the integrity of this consumable foil, whose replacement is a scheduled maintenance activity and whose damage causes immediate performance loss [32].

The complexity extends to beam transport and target stations. Extracted proton beams must be accurately steered by magnets towards the target stations; mis-steering can damage equipment, hence the need for heavy shielding and beam loss monitors [32]. At the target

stations, the high-energy beam strikes targets to produce neutrons or muons [75, 32]. These target systems operate under extreme conditions and require careful monitoring and management to prevent overheating or material damage, representing another major operational and maintenance concern [32].

This overview highlights the key takeaway: the ISIS facility involves an intricate sequence of precisely controlled stages, creating numerous potential points of failure. While precautions like beam loss monitors and physical shielding exist, the inherent complexity and harsh operating environment (radiation, high-power) mean that component degradation and operational faults are inevitable. Effectively managing the facility therefore necessitates robust maintenance strategies and sophisticated diagnostic capabilities. Documenting fault events, diagnoses, and repairs in operational logs (like the Operalog studied here) is essential, and the technical complexity of the facility directly influences the nature and variety of entries found in these logs, motivating the need for advanced analysis techniques like those explored in this paper.

1.2.1 The end-to-end production of neutrons and muons

1.2.2 Maintenance at ISIS

As detailed in the 33-year historical account of the ISIS facility (Thomason, 2019), the ISIS operations occur in cycles, periods of roughly 30-50 days where the machine runs constantly without breaks. Gaps between cycles typically range from 1 week to 3 months. In addition, typically every four years, shutdowns scheduled for 6-9 months occur for major maintenance and upgrade work. On the ground, day-to-day operations are run from the Main Control Room (MCR) by the ISIS crew which consists of 6 shift teams of the following roles: (1) duty officer, (2) assistant duty officer and (3) duty technician. The expertise and rapid response capabilities of these trained operational crews are vital for both routine operation and initial fault response. Minimising this downtime is crucial for several reasons inherent to operating a large-scale user facility like ISIS [86]. Primarily, unscheduled downtime directly impacts the international researchers allocated beam time, potentially jeopardising experiments planned months or years in advance and wasting valuable research opportunities. Furthermore, interruptions disrupt the tightly packed operational schedule, reduce the overall scientific output, and incur significant operational costs without delivering the facility's core research product [86]. Therefore, understanding the factors that contribute to downtime and implementing strategies to mitigate it are paramount. Many factors affect downtime such as having a robust plan, the day-to-day operators' knowledge of the system and adequate inventories of spares [86]. Figure 1.3 shows the ratio of downtime to active machine operation since 2016, indicating this machine has been down for maintenance over half the time since 2016. In other words, over half the time, the machine is not active and is undergoing maintenance. This highlights the trade-off between maximising operational availability and minimizing failures through planned maintenance. The major maintenance strategies and their trade-offs are further explored in Section 1.3.

To help reduce downtime, over the last few years, a first-line diagnosis (FLD) system was introduced, presented in (Asim Yaqoob, 2017). The FLD system helps reduce downtime by providing expert guidance on fault diagnosis and resolution, which has been shown to improve

Figure 1.1: The schematic representation of the physical layout of ISIS. The light grey areas are footprints of the buildings. Source: (Thomason, 2019).

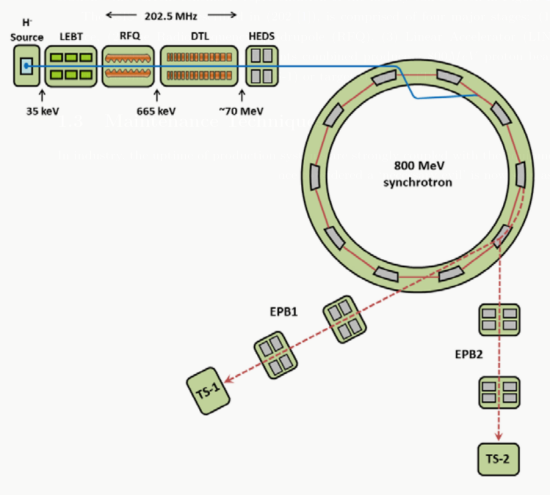


Figure 1.2: ISIS ion source and chain of accelerators, with H^- ions in blue and protons in red. Not to scale. Source: (ISIS Neutron and Muon Source, 2021).

the dissemination of knowledge from experts to operations. The FLD utilises Fault Analysis Pathways (FAPs), which provide structural links between ISIS subsystems. This allows users of the system to access granular subsystems' local documentation minimising file hunting and saving time and effort. An example FAP can be seen in Figure 1.4.

1.3 Maintenance Techniques

In industry, the uptime of production systems is strongly coupled with the equipment maintenance. So much so that what was once considered a 'necessary evil' is now seen as a 'profit contributor' to be able to maintain a competitive edge [91, 25]. For facilities aiming to provide systems for research, maintenance impacts the downtime and cost of running. As a result, both to minimise unexpected downtime and provide a competitive edge, many industrial applications collect vast quantities of data during the entire life cycle of the system. This large amount of data may include information about processes, events and alarms [17] which occur along the industrial production line, collected by different equipment. The equipment may be located in different locations in the sub-components of the larger system or even different sub-components themselves.

In the literature, various terms and categories of maintenance arise each with differing strategies [83, 56, 82]. Thus, while there exists some disagreement in nomenclature, we consider the four categories presented in (Susto et al., 2012). The four maintenance policy categories are as follows, noting that each policy has, uniquely, their own benefits and drawbacks:

1. Run-to-failure (R2F) maintenance: Continual usage of the system until failure. Restoration is performed at the point of noticing failure condition. This is the simplest ap-

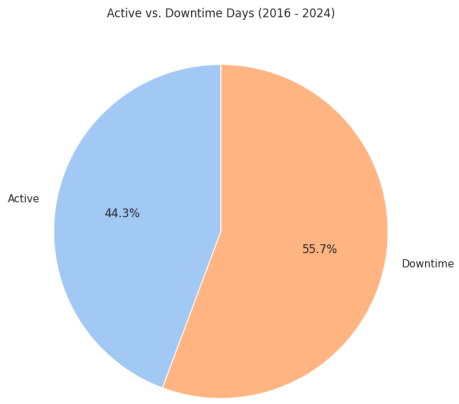


Figure 1.3: A visualisation of the machine downtime as opposed to time operating, according to the ISIS operational cycle. Data source: (ISIS Neutron and Muon Source, 2024)

proach and typically the most costly method as it reduces the facility's availability and requires a complete replacement of parts.

2. Preventative maintenance (PvM): Otherwise referred to as scheduled maintenance, performing maintenance at regular intervals to increase longevity of the component or in anticipation of the end of expected life of the component. While this typically prevents many errors, it wastes maintenance cycles when systems are perfectly healthy. Hence, causing unnecessary downtime and cost.

3. Condition-based maintenance (CBM): Taking the action to perform maintenance on equipment through monitoring various health characteristics and metrics of the components of the system. This approach requires continuous monitoring and, thus, allows for close to instant response on maintenance only when required. However, a drawback of this policy is that one cannot plan maintenances in advance.

4. Predictive maintenance (PdM): Otherwise referred to as statistical-based maintenance, only performs maintenance actions when determined necessary. Prediction tools are utilised to implement forward-planning and scheduling systems, using statistical inference methods. However, if these statistical inferences are not accurate, the whole system suffers which inevitably leads to additional downtime and costs.

It should be noted that several sources conflate CBM and PdM [56]. As in (Susto et al., 2012), where they are given as separate categories, we follow suit.

The PdM strategy stands out in the four categories presented as, given a statistical inference model that is able to detect faults accurately, this policy optimises the trade-off between improving equipment condition, reduce failure rates for equipment and minimising maintenance costs [17]. This technique enables one to apply foresight for pre-emptive scheduling

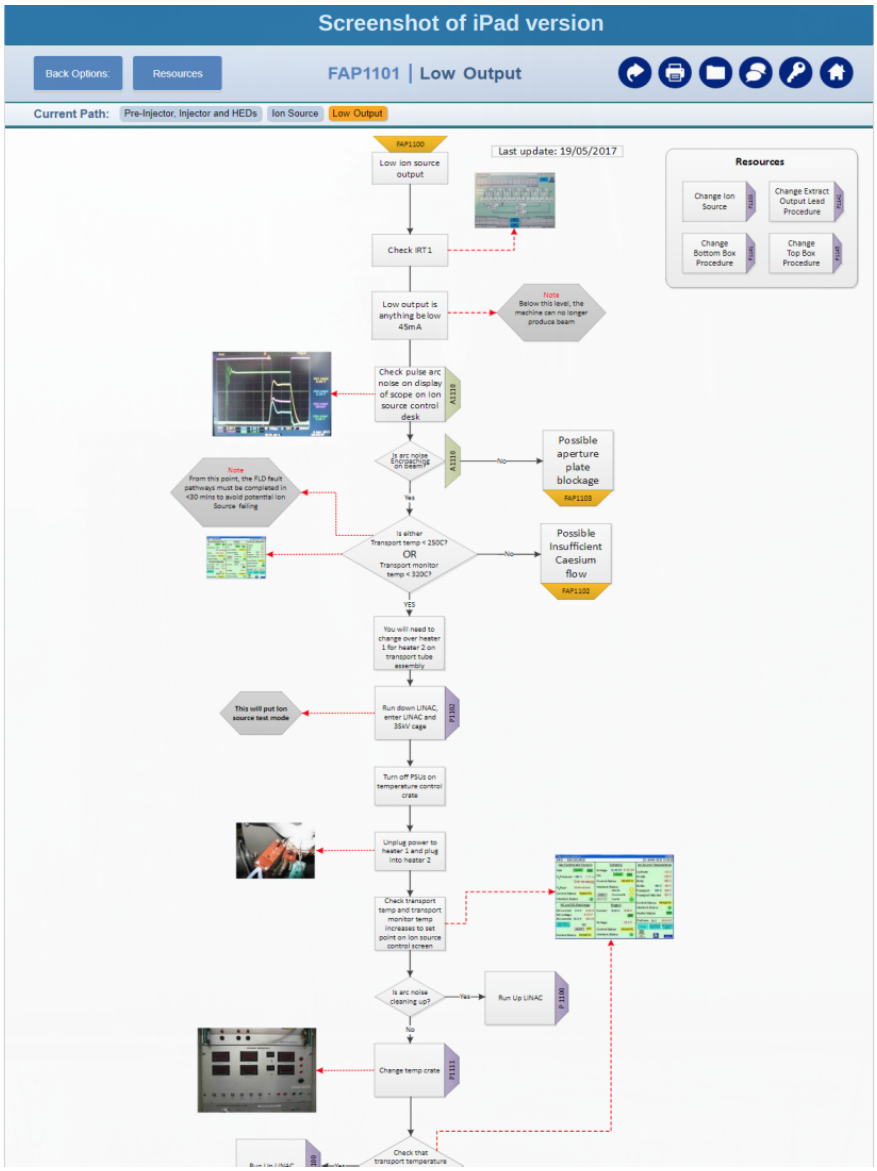


Figure 1.4: Example FAP1101, screenshotted from an iPAD version of FLD version 2.2, which shows the FAP for pre-injector, injector and High Energy Drift Space. Source: (Asim Yaqoob, 2017)

of large-scale maintenance. As pointed out in Section 1.2, the ISIS facility aims to strike a balance between PvM, CBM and PdM through periods of large-scaled scheduled maintenance and collection of high quantities of metrics. This balance is achieved through the careful coordination between cycle scheduling, day-to-day crew-based monitoring and the FLD [86].

In industry, many maintenance strategies prefer using PdM whilst experimenting with a variety of statistical inference and artificial intelligence modelling approaches [56, 35]. Some examples from [17] are listed in Table 1.1 which highlights the trend in the industry towards more accurate, ML-based approaches.

Table 1.1: Examples of applications of PdM for industrial maintenance strategies.

Type	**Description**	**Reference**
Statistical	Application of SAFE to deal with PdM problems characterised by time-series data. The approach is tested on a real-life dataset of the semiconductor ion implantation process.	(Susto and Beghi, 2016)
ML	Application of SVM classification for fault prediction of rail networks, with discussion on using the model in optimising trade-offs related to maintenance schedule and costs.	(Li et al., 2014)
ML	Audio analysis on IoT devices, enabling acoustic event recognition for machine diagnosis. This paper describes designing an end-to-end system, utilising CNN-based classification.	(Pan et al., 2017)
ML	Utilisation of RF decision trees trained on SMART data to predict reliability of HDD in real-time.	(Su and Huang, 2018)

Chapter 2

Natural Language Processing Background

This chapter delves into the technical background required for the methodology proposed in Chapter 3. Firstly, we discuss various measures of text similarity in Section 2.1, which motivates text embeddings and categorises similarity measures into different generations. Section 2.2 introduces the third-generation model BERT [22] and the fourth-generation universal-embedding architecture XLNet [99]. We specifically discuss the technical details of the BERT model and further iterations on it, then detail one improvement over BERT (XLNet) and outline the design principles that underpin contemporary fourth-generation models. Afterwards, we cover two methods of dimensionality reduction, PCA [65, 31] and UMAP [52], motivated by the need to visualise samples from the high-dimensional embedding spaces of the aforementioned models, in Section 2.3. Finally, we present three clustering algorithms - one partitioning-based (k-Medoids [37]) and two density-based (DBSCAN [24] and HDBSCAN [15]).

2.1 Sentence Similarity

Sentence similarity, otherwise referred to as document similarity, is the (NLP) task of computing the quantification of the similarities between two sentences, documents or texts. This task is motivated by the increasingly large amount of digitisation of human languages (and data, in general), calling for the need to understand similarity between various texts [69]. Examples of the use-cases of sentence similarity include: detection of academic malpractice via plagiarism [50, 8], and text summarisation [5, 39, 36]. According to [69], there are two main types of sentence similarities: (1) lexical similarity and (2) semantic similarity. The former is a computation of the equality between the lexicon of two sentences (i.e. a purely syntactical view). This contrasts with semantic similarity, which compares meaning. Further, the type we focus on, semantic similarity can be split into three types:

- String-based similarity: Measures similarity directly between two strings, accounting for string sequences and character composition. These can be fine-grained, i.e.

character-based; coarse-grained, i.e. term-based; or a hybrid mixture of both [100].

- Knowledge-based similarity: Measures the degree to which two sentences are related, utilising semantic networks (i.e. knowledge graphs). Examples of Knowledge-based similarity approaches include WordNet [12], the most popular type of approach.

- Corpus-based similarity: Premised on a provided corpus, a large database of text to derive inferences from. Methods of this type require the development statistical or DL models that train on the provided corpus and estimate the similarity between two sentence-pair inputs. Popular examples include traditional statistical models, such as LSA [41] and SVD [79] as well as word embedding models (utilising ML), such as Word2Vec [11], GloVe [67] and fastText [54].

Most of the models mentioned above require some numerical representation of the text to be able to apply mathematical procedures for similarity calculation. Computing this representation involves converting unstructured textual data into one or more vectors. Typically, this process includes (1) general natural language pre-processing steps such as stop-word removal, case normalisation, parts-of-speech tagging, lemmatisation, and tokenisation [84]; and (2) applying an embedding model, either to a single token (word embedding) or to a sequence of tokens (sentence embedding). Step (1) can be seen as a feature extraction step applied on the unstructured textual data, where feature extraction is the process of extracting the most useful components of the data [72]. For example, part-of-speech tagging can be seen as introducing non-trivial features to some token through extracting the surrounding context.

This representation is known as an embedding, with the span of the possible vectors referred to as the embedding space. The dimension of the span is termed the embedding dimension. This is an important concept because the characteristics of the embedding space influence the model's ability to capture syntactic and semantic meaning in text, as the embedding space itself encodes this information. This can be seen in the Word2Vec model, described in (Bojanowski et al., 2017), which shows different embedding dimensions produce different results. Another conclusion that can be drawn from this paper is that, if embedding space is not constructed to maximise the meaning of texts, the accuracy of model predictions tends to deteriorate.

Therefore, the problem of sentence similarity can be directly mapped from the problem of sentence embedding (otherwise referred to as text embedding), where text embedding is the (NLP) task of learning a high-dimensional embedding space representation. Various aspects of text embedding are more thoroughly covered in Section 2.2. However, with the advent of the transformer architecture [89] and rise of the large language models, text embedding has been increasingly solved using DL models with high parameter counts [16] and considering extremely large token sequences. Nowadays, word embedding models are considered obsolete with (Cao, 2024) only considering these models second-generation. Further, the paper states newer generations fall into the following categories:

- Third-generation: contextualised embeddings. These models dynamically account for contexts, encoding them into the embedding space. Examples of models include ELMo [73], GPT [68] and BERT [22]. As these models are trained to both understand some

embedding space and generate natural language text, they are canonically referred to as language models.

- Fourth-generation: universal text embeddings. The generation which is currently state-of-the-art, with the aim of developing a unified model which is able to address multiple downstream tasks. Examples of models in this generation, making progress towards unification include Gecko [42], Multilingual e5 text embeddings [94], Nomic [60] and many more.

Second-, third- and fourth-generation text embedding models are used frequently in PdM for applications such as insight extraction [3, 88] and clustering intents from unstructured text data. Sources of natural language datasets, in industrial applications typically arise from operational or managerial log files which document aspects such as failures, resolutions and comments. Advanced text embedding models enable for semi- or fully automatic insight retrieval and auto-categorisation, enabling intuitive understanding of the textual datasets potentially highlighting patterns in failure [59].

2.2 Sentence Embedding

Briefly touched on in Section 2.1, sentence embedding (otherwise known as text embedding) is the NLP task of computing some high-dimensional embedding vector-space representation for unstructured text data. This vector-based representation should encode the semantic and syntactic meaning of the text and establish meaningful relationships. For example, the sentence 'I like dogs' should have the opposite representation to 'I hate dogs'. However these sentences should be more related than to the sentence 'My house was destroyed in an earthquake'. For a naive illustration of this, see Figure 2.1, which shows how similar sentences should be grouped together. As Nomic requires the task explicitly in input text (more on this later), 'clustering:' is appended to all sentences. Deep learning sentence embedding models are now seen to be state-of-the-art as they are able to extract features automatically and more effectively than manual efforts, when supported with large quantities of data [48].

2.2.1 Transformers and Foundational Pre-trained Language Models

An important advancement in modern NLP is the use of Pre-trained Language Models (PLMs). Models like BERT [22] demonstrate success and improvements in performance across various NLP tasks when adapted through a process called fine-tuning [23, 55]. Fine-tuning involves taking a general-purpose (termed 'base') PLM, already trained on vast amounts of text data, and further training it on a smaller, task-specific dataset. This leverages the model's pre-existing knowledge of language structure and semantics, allowing for efficient adaptation to specialised tasks like the Operalog analysis in this paper. These PLMs effectively learn an embedding space for language and can often generate text, providing a powerful foundation [22]. The accessibility of these models has been greatly enhanced by platforms like HuggingFace, which host pre-trained model parameters (or 'weights'), enabling researchers to download and utilise or fine-tune them without undertaking the computationally prohibitive

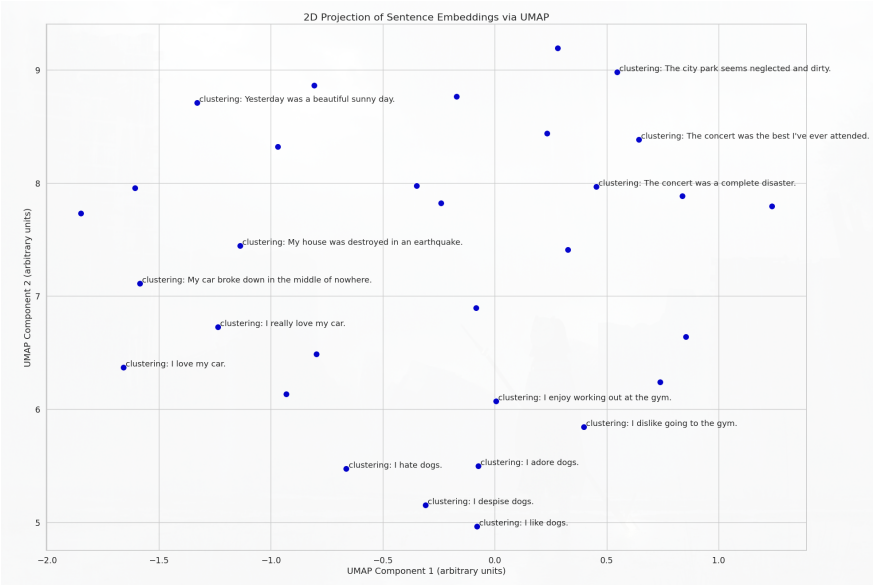


Figure 2.1: Example text embedding with Nomic Text Embedding v1.5 [60] of random sentences generated using OpenAI's ChatGPT o3-mini-high and projected onto 2 dimensions using UMAP [52] with a random seed of 42 and minimum neighbours of 15. Only some of the labels are highlighted for visual clarity.

initial pre-training [97]. This project leverages such pre-trained models, specifically focusing on the Transformer family.

The Transformer architecture, introduced by (Vaswani et al., 2017), employ attention mechanisms, to compute pairwise token significance. That is, conceptually, attention allows the model - when processing a word in a sequence - to weigh the importance of other words in the sequence regardless of their distance to each other. This enables the model to capture long-range dependencies and understand context more effectively than earlier sequential models. While the detailed architecture involves components like multi-head attention and layer normalisation, the core idea is this context-aware weighting mechanism [89]. Given its foundational nature and detailed coverage in the literature, we refer the reader to the original paper for architectural specifics.

Building upon the Transformer, several influential PLMs were developed. We briefly discuss two foundational models, BERT and XLNet, as their concepts and limitations motivate the subsequent choice of models like MPNet used later in this study.

BERT

The Bidirectional Encoder Representations from Transformers (BERT) model was a landmark development by Google AI [22]. Its key innovation was learning deep *bidirectional* representations by considering both the left and right context simultaneously in all layers. This contrasts with earlier models that typically processed text in only one direction. BERT

achieves this primarily through its Masked Language Modelling (MLM) pre-training objective. During MLM, a percentage of input tokens (15%) are randomly masked (replaced with a special `[MASK]` token), and the model learns to predict these original tokens based on the surrounding unmasked context [22]. This forces the model to develop a deep understanding of word relationships and context from both directions. While BERT also used a Next Sentence Prediction task, MLM is its core contextual learning mechanism [77]. BERT demonstrated state-of-the-art performance on numerous NLP tasks after fine-tuning [22].

However, BERT has limitations. A notable issue is the pre-train-to-fine-tune discrepancy: the artificial `[MASK]` token used during pre-training is absent in downstream tasks (i.e. during fine-tuning), potentially hindering performance. Furthermore, by predicting masked tokens independently, BERT doesn't explicitly model the dependencies between the masked tokens themselves, which can be an oversimplification for complex language understanding, as pointed out by (Schank, 1972) [99].

XLNet

XLNet was proposed to address BERT's limitations while retaining the benefits of bidirectional context [99]. It employs an autoregressive (AR) approach, meaning it predicts tokens sequentially, but introduces Permutation Language Modelling. Conceptually, instead of processing the sentence in its original order, XLNet considers all possible permutations (shuffled orders) of the input sequence. For each permutation, it predicts a token based only on the tokens that came before it in that specific shuffled order. By maximizing the prediction likelihood across all permutations, the model implicitly learns context from both original directions (left and right) for every token [99]. This approach avoids the artificial `[MASK]` token and explicitly models dependencies between predicted tokens within a given permutation. It leverages the Transformer-XL architecture for improved handling of longer sequences [19].

While XLNet offered improvements, the development of PLMs continued, leading to models like MPNet [77], which attempts to unify the strengths of both masked language modelling (like BERT) and permutation-based autoregressive modelling (like XLNet), motivating its consideration in this project (see Chapter 3).

Considerations for Using Pre-trained Models

Finally, while leveraging PLMs significantly accelerates research, it introduces reliance on third-party models. As noted previously, users must trust that the pre-trained parameters are sound and haven't been subject to issues like data contamination or malicious alterations. Therefore, favouring models with transparently documented data sources and training procedures is generally preferred when possible.

2.3 Dimensionality Reduction

The sentence embedding techniques discussed previously often produce high-dimensional vector representations, frequently on the scale of hundreds or even thousands of dimensions (i.e. 768 dimensions for BERT base [22]). Directly analysing or visualising data in such high-dimensional spaces is challenging. Furthermore, many data analysis methods, including the

clustering algorithms discussed later (Section 2.4), can suffer from performance degradation in high dimensions due to phenomena collectively known as the 'curse of dimensionality' [90, 29]. This 'curse' refers to counter-intuitive geometric properties of high-dimensional spaces that can make concepts like distance and density less meaningful [90].

Therefore, dimensionality reduction techniques are often employed. These methods aim to find a lower-dimensional representation (typically 2-dimensional or 3-dimensional for visualisation) that captures the most important structure or variance present in the original high-dimensional data [78]. By exploiting redundancy in the data, they project points into a smaller feature space suitable for visualisation or input to subsequent algorithms like clustering [78, 29]. While numerous techniques exist as shown by (Sorzano et al., 2014), this section focuses on two widely used methods employed in this project: Principal Component Analysis (PCA) [65, 31] and Uniform Manifold Approximation and Projection (UMAP) [52].

2.3.1 PCA

Principal Component Analysis (PCA), first introduced by Pearson, 1901 and later developed by Hotelling, 1933, is a linear dimensionality reduction technique. The core idea is to identify new axes, called 'principal components', in the direction of maximum variance in the data [2]. The first principal component captures the largest amount of variance, the second captures the largest remaining variance while being orthogonal (that is, statistically uncorrelated) to the first, and so on. Projecting the original data onto the first few principal components creates a lower-dimensional representation that retains most of the original data's variance [2].

Conceptually, PCA seeks a linear projection that either maximizes the variance of the projected data or, equivalently, minimises the 'reconstruction error' which is the squared distance between original points and their projections [87]. While the formal solution involves finding eigenvectors of the data's covariance matrix, a standard and efficient method to compute the principal components and their associated variance relies on the Singular Value Decomposition (SVD) of the (mean centred) data matrix X [79, 87]. Here, mean centred refers to $X - \frac{1}{nm}\sum_i^n \sum_j^m X_{ij}$ for $X \in \mathbb{R}^{n\times m}$. The SVD factorizes X as $X = USV^T$, where the columns of V represent the principal component directions, and the squared diagonal elements of S represent the variance captured by each component [87]. By keeping only the components associated with the largest singular values (a truncated SVD), PCA achieves dimensionality reduction while preserving maximal variance in a linear sense. A formal treatment of this as an optimisation problem can be found at Appendix A.2.3.

2.3.2 UMAP

Uniform Manifold Approximation and Projection (UMAP) is a more recent, non-linear dimensionality reduction technique based on manifold learning principles [52]. Unlike PCA's linear projections, UMAP aims to capture potentially complex, non-linear structures in the data. It operates under the assumption that the high-dimensional data lies on or near a lower-dimensional manifold embedded within the higher-dimensional space [52]. The goal is to find a low-dimensional embedding that faithfully represents the topological structure of

this presumed manifold. While grounded in advanced mathematical concepts [43, 44, 96], the intuition behind the algorithm can be understood through its two main conceptual stages [52]:

1. High-dimensional Graph Construction - UMAP first builds a weighted graph representation of the data in the original high-dimensional space. It identifies nearest neighbours for each data point and assigns edge weights based on the likelihood that points are closely connected on the underlying manifold. This step primarily focuses on capturing the local connectivity and structure of the data.

2. Low-dimensional Layout Optimisation - UMAP then seeks a corresponding low-dimensional embedding where the graph structure is as similar in structure as possible to the high-dimensional graph. It iteratively adjusts the positions of points in the low-dimensional space to minimise the difference (often measured using cross-entropy, a measure of divergence between probability distributions [102]) between the connectivity probabilities in the high- and low-dimensional representations. This optimisation aims to keep points connected in the low-dimensional space if they were connected in the high-dimensional space, and separated if they were not.

The resulting low-dimensional embedding aims to preserve both local neighbourhood structure and, importantly, aspects of the data's broader global structure [52]. UMAP's output is influenced by several key hyperparameters:

- `n_neighbors`: Controls the size of the local neighbourhood considered. Lower values emphasize local structure, potentially revealing fine clusters, while higher values capture more global structure, potentially merging related groups [52].

- `min_dist`: Affects the minimum separation between points in the low-dimensional embedding. Lower values create tighter, denser clusters, while higher values produce more spread out embeddings where intra-cluster structure might be more visible [52].

- `n_components`: Defines the dimensionality of the output embedding (typically 2- or 3-dimensions for visualisation).

- `metric`: Specifies the distance measure used in the high-dimensional space (i.e. 'euclidean', 'cosine'). The choice is crucial: 'cosine' distance is often preferred for high-dimensional text embeddings as it measures orientation rather than magnitude [52].

Selecting appropriate hyperparameters often requires experimentation (as explored in Section 3.7). UMAP implementations are generally computationally efficient compared to other non-linear methods like t-SNE [13, 52], making it practical for many applications. Figure 2.2 provides a visual intuition of UMAP applied to structured data (sampled points from the skeletal structure of an elephant). Notably, structure preservation can be visually observed, as points from a tusk represent a shape which mirrors a tusk shadow in the 2-dimensional down-projection. However, while powerful for revealing non-linear structures, the axes of a UMAP projection do not have the direct variance interpretation of PCA's principal components.

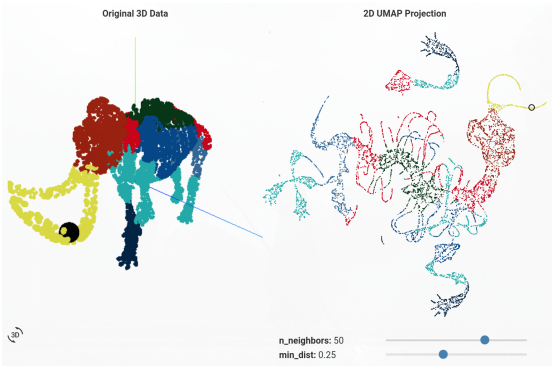


Figure 2.2: An illustration of UMAP applied (right) on samples taken from the skeletal structure of an elephant in 3-dimensions (left) with the UMAP hyperparameter values displayed. Highlighted, a black sphere (left) and circle (right), shows a chosen point mapped from 3-dimensions to 2. Source: (Coenen, Andy and Pearce, Adam, 2020).

2.4 Clustering

Clustering is an unsupervised machine learning task that aims to group data points such that items within a group (a cluster) are more similar to each other than to those in other clusters. Common approaches include partitioning methods, which divide the data into a predefined number of clusters, density-based methods, which identify clusters based on regions of high data point density, and hierarchical methods, which build a tree-like structure of nested clusters [38].

This section provides the necessary background for the clustering techniques employed later in this project. We will cover k-Medoids [37], a partitioning algorithm; Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [24], a density-based algorithm; and Hierarchical DBSCAN (HDBSCAN) [15], which builds upon density and hierarchical concepts.

2.4.1 k-Medoids

The k-Medoids algorithm is a partitioning-based clustering method that aims to divide a dataset into k distinct clusters. The k-Medoids algorithm uses actual data points, known as medoids, as the centres of its clusters. It aims to minimise a cost function based on the sum of pairwise dissimilarities (that is, distances) between each point and the medoid of the cluster it is labelled under [37].

A classical algorithm for implementing k-Medoids is Partitioning Around Medoids (PAM), introduced by (Kaufman and Rousseeuw, 1990). PAM operates in two main phases:

1. The build phase - An initial set of k data points are selected as the starting medoids. This selection aims to establish a reasonable initial clustering by choosing points that are relatively central within potential groups that exists - minimising the initial total dissimilarity.

2. The swap phase - This phase iteratively refines the set of medoids. For each currently

selected medoid m and each non-medoid point data point x, the algorithm computes the change in total dissimilarity that would result from swapping m and x. That is, setting x as that centroid of m's cluster. The swap that yields the greatest reduction across all possible pairs is finally performed. This process repeats until no swap can further reduce the total dissimilarity which indicates a local optimum has been reached.

The PAM algorithm employs a greedy search strategy during the swap phase, which means it may converge to a local rather than a global optimum [37]. The computational complexity of the swap phase is often cited as $O(k(n-k)^2)$ per iteration, where n is the number of data points and k is the number of clusters [37].

Appendix A.4 provides a high-level pseudocode description of the relatively straightforward PAM process. It takes the desired number of clusters k and the dataset X as input, and outputs the final set of medoids M and set of clusters X'. The fact that k is a parameter to the algorithm means that either the number of clusters must be known beforehand or must be found through hyperparameter optimisation.

2.4.2 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is density-based clustering algorithm, unlike partitioning methods like k-Medoids [24]. Instead of partitioning data around centres, DBSCAN groups together points that are closely packed in high-density regions, separated by sparser areas. A key characteristic of this approach is its ability to discover clusters of arbitrary shapes and to automatically determine the number of clusters present, rather than requiring it as a predefined input [24]. The core intuition is that points belonging to the same conceptual group or category tend to lie close to each other, forming denser regions than considered noise or outliers.

The algorithm's behaviour and the concept of density is controlled by two hyperparameters [24]:

- `Eps` - This parameter defines the maximum distance between two points for one to be considered as in the neighbourhood of the other. It essentially sets the radius for identifying neighbours. A smaller `Eps` value requires points to be closer together to form a dense region, potentially leading to smaller, tighter clusters and classifying more points as noise. Conversely, a larger `Eps` value allows clusters to span sparser regions and potentially merge groups that might otherwise be distinct.

- `MinPts` - This parameter specifies the minimum number of points (including the point itself) required within a point's Eps-neighbourhood for that point to be considered a core point (i.e. an interior point of a dense region). MinPts defines the threshold for density. A higher value requires more points to constitute a dense core, potentially resulting in fewer clusters and more points being classified as noise. A lower value allows sparser groupings to qualify as clusters, potentially resulting in more clusters.

Determining appropriate values for `Eps` and `MinPts` often requires a heuristic (such as the one described by (Ester et al., 1996)) or domain knowledge, as DBSCAN is an unsupervised machine learning algorithm [24].

To understand the algorithm's operation, several core concepts are essential [24]

- Eps-neighbourhood - The Eps-neighbourhood of a point is the area within a radius of length `Eps` from that point.
- Density-reachable - A point p is density reachable from another core point, if there is a sequence of points from p to q where the distance between any two points is less than or equal to `Eps` and every point in this sequence except q must be a core point. This defines a chain of density connecting p to q.

Additionally, based on the hyperparameters, DBSCAN categorises each point in the dataset into one of the three types [24].

- Core point - A point whose Eps-neighbourhood contains at least `MinPts` points, including itself. Core points are considered to be in the interior of a dense cluster.
- Border point - A point that is not a core point itself, but falls within the Eps-neighbourhood of at least one core point. Border points lie on the border of a cluster.
- Noise point - A point that is neither a core point nor a border point. These points typically reside in low-density regions and are not assigned to any cluster.

The DBSCAN algorithm, formally defined in (Ester et al., 1996), iterates through the data points. When it encounters an unvisited point, it checks if that point qualifies as a core point by examining the set of points Eps-neighbourhood. If it is indeed a core point, then a new cluster is formed. This cluster is then expanded by finding all density-reachable points, starting from this initial core points. This involves recursively exploring the Eps-neighbourhood of all core points found during the expansion process. Any point reachable through a chain of core points connected by their Eps-neighbourhood becomes part of the cluster. If an unvisited point examined is not a core point, it is initially marked as noise, although it may be later reclassified as a border point, if found within the Eps-neighbourhood of a core point belonging to an expanding cluster. Points identified as density-reachable but not core points become border points assigned to this cluster. This procedure continues until all points in the dataset have been visited and assigned to a cluster or designated as noise.

An important consideration arises when a border point falls within the Eps-neighbourhood of core points belonging to different, simultaneously expanding clusters. Standard DBSCAN implementations typically resolve this based on the order of processing: the border point is assigned to the cluster associated with the first core point that 'discovers' it during expansion [24]. Once assigned, it is generally not reassigned, even if subsequently found to be reachable from another cluster's core point. Thus, while the core cluster structures are stable, the assignment of some border points can be implementation-dependent.

In general, DBSCAN offers significant advantages: it can identify clusters of arbitrary shape, is inherently robust to noise (which it explicitly labels) and does not require the number of clusters to be specified beforehand. However, it also has limitations. The algorithm's performance is sensitive to the choice of `Eps` and `MinPts` and it can struggle to correctly identify clusters of significantly varying densities within the same dataset. Furthermore, the average runtime complexity is $O(n \log n)$ where n is the number of points in the dataset.

[24, 15]. Like many distance-based algorithms, its effectiveness can degrade in very high-dimensional spaces due to the 'curse of dimensionality' (see Section 2.3) [29, 90].

2.4.3 HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN), proposed by Campello et al., 2013, is an advancement over DBSCAN that generates a hierarchical clustering structure. This approach addresses a key limitation of DBSCAN: its reliance on a single global density threshold ϵ (equivalent to `Eps` in DBSCAN), which makes it difficult to identify clusters of varying densities simultaneously. HDBSCAN overcomes this by effectively exploring all possible density thresholds ($\lambda = \frac{1}{\epsilon}$, for all $\epsilon \in \left[0, \infty\right)$) to build a cluster hierarchy [15].

The method introduces two main contributions: (1) the HDBSCAN algorithm itself, which constructs this complete density-based hierarchy, represented as a dendrogram (a hierarchical tree-based representation of the dataset points); and (2) a novel measure of cluster stability used to extract a simplified, flat partition containing only the most significant and persistent clusters from this hierarchy [15]. This extraction is framed as an optimisation problem maximising the total stability of the selected clusters. A significant advantage of HDBSCAN is its reduced sensitivity to hyperparameters as it primarily requires only m_{pts} (equivalent to `MinPts` in DBSCAN), a parameter whose effect as a density smoothing factor is generally well-understood [15].

Importantly, the remaining portion of this section assumes a reasonably basic level of understanding of graph theory. For a great primer, the reader is referred to (Harris, 2008).

To establish a formal link between the flat clustering of DBSCAN and the hierarchy of HDBSCAN, the authors first revisit the DBSCAN concept (referred to as DBSCAN in the paper for clarity). In this view, clusters are defined as connected components of core objects within a graph where edges connect mutually reachable core points based on ϵ and m_{pts}. Non-core objects are considered noise, which includes border points. This perspective allows density-based clusters to be seen as connected components of a level set of density (i.e. here, given a density threshold $\lambda = \frac{1}{\epsilon}$, it can be thought of as a set with points with density above that value) [15].

HDBSCAN builds upon the defining concepts relative to m_{pts}, only allowing ϵ (or λ) to vary [15]:

- Core distance ($d_{\text{core}}(x)$) - For a data point x, this is the distance to its m_{pts}-th nearest neighbour (including itself). It serves as a measure of the local density around x. A smaller core distance implies a denser region.

- Mutual reachability distance ($d_{\text{mreach}}(x_p, x_q)$) - Defined for two data points x_p, x_q, this is the maximum of their respective core distances and their pairwise distance defined as:

  $$d_{\text{mreach}}(x_p, x_q) = \max\{d_{\text{core}}(x_p), d_{\text{core}}(x_q), d(x_p, x_q)\}$$

  This adjusted distance measure ensures that points in sparse regions are effectively pushed further apart, reflecting density differences.

The HDBSCAN algorithm constructs the cluster hierarchy based on these concepts. Conceptually, it first computes the core distance for all points. Then, it builds a graph where all points are vertices, and the weight of the edge between any two points is their mutual reachability distance. The algorithm finds the Minimum Spanning Tree (MST) of this complete graph. The cluster hierarchy (dendrogram) is then extracted from this MST by considering edges in decreasing order of weight (distance). As edges are removed (corresponding to decreasing the density threshold λ or increasing the distance threshold ϵ), connected components in the remaining graph represent clusters at that specific density level. The hierarchy captures how these components merge as the density threshold decreases [15]. This process can be implemented efficiently, with a typical time complexity of $O(n^2)$ if pairwise distances are precomputed or provided, or potentially $O(dn^2)$ depending on MST construction details if distances are computed on the fly from d-dimensional data [15].

To extract a meaningful flat partition from this hierarchy, HDBSCAN introduces the concept of cluster stability. This measure quantifies how persistent a cluster C_i is across different density levels ($\lambda = \frac{1}{\epsilon}$). That is, as you vary the density level λ, this measures the longest period in which the cluster exists without disappearing or splitting. The stability $S(C_i)$ is calculated by summing, for each point x_j belonging to cluster C_i, the range of density levels for which that point remains part of that specific cluster [15]. More formally, using the notation from the paper where $\lambda_{\min}(C_i)$ is the density level at which cluster C_i appears (thought of as is 'born') and $\lambda_{\max}(x_j, C_i)$ is the density at which point x_j leaves cluster C_i, either through becoming noise or part of some child cluster as density increases [15]:

$$S(C_i) = \sum_{x_j \in C_i} (\lambda_{\max}(x_j, C_i) - \lambda_{\min}(C_i)) \tag{2.1}$$

Equation 2.1 adapts the concept of 'excess of mass' [58] to a set of discrete data points and density levels derived from the hierarchy. Clusters where points persist over a larger range of λ values (i.e. are more 'stable' across varying density thresholds) will have higher stability scores.

Using this concept of stability, HDBSCAN selects the optimal flat clustering by solving an optimisation problem: finding a maximal disjoint set of clusters from the hierarchy such that the sum of stabilities $S(C_i)$ is maximised. Here a maximal disjoint set of clusters refers to the fact that no two cluster is an ancestor or descendant of another selected cluster, ensuring a flat partition covering all points down certain branches [15]. The paper provides an efficient algorithm that traverses the simplified cluster tree bottom-up and then top-down to find this optimal set of stable clusters in linear time, relative to the size of the simplified tree. The simplified cluster tree is obtained after applying a minimum cluster size m_{dSize}, typically set equal to m_{pts}. This process effectively performs the optimal 'local cuts' in the cluster dendrogram at different levels of hierarchy, based on this notion of cluster persistence.

Chapter 3

Automatic Categorisation and Label Generation

3.1 Data

Data relevant to the operational state and maintenance of the ISIS facility originates from several sources. The facility publishes open-access datasets, primarily containing instrument calibrations and experimental metrics, via its data portal [34]. Alongside this, the ISIS team maintains proprietary internal datasets detailing specific component performance, which are available upon request. For this research, access was granted to historical logs for the ion source component, with entries dating back to 27 March 2003. Furthermore, the operations crew maintains a daily operational log, the Operalog, documenting facility faults and the corresponding remedial actions taken.

Given the breadth of available data and the project's scope constraints, a decision was made to focus the analysis presented in this paper on the **Operalog**. This dataset provides a rich textual record of real-world failures across the facility, offering valuable insights despite its partially unstructured nature. Data from the specialised ion source logs is the focus of the partner project.

3.1.1 The Operalog

The operational log, otherwise referred to as the 'Operalog', is an Excel spreadsheet with entries documenting moments of machinery failure within the ISIS facility covering the period 1996 - 2023. Table 3.1 documents the important features. Notably, the day-to-day operational crew have developed custom abbreviations, acronyms and terminology which may not immediately be clear. Additionally, different crew members have varying writing styles and levels of depth of information. Thus, this results in an unstructured text dataset which has a huge variance in quality and quantity of information. Furthermore, as noted in Table 3.1, `FaultRepair` was seemingly made redundant post-2017. Figure 3.1a highlights the distribution in the text lengths of both unstructured text fields. Furthermore, around 0.02% of `FaultDescription` fields are empty whereas around 66% of `FaultRepair` fields are empty. The generally shorter

entries in `FaultRepair`, with a large majority having fewer than or equal to 5 characters, suggest lower informational content compared to `FaultDescription`. This is further illustrated in the Wordcloud illustrations [63] (Figure 3.2), where `FaultRepair` only has one extremely large word ('reset'). As the size of the word indicates the frequency, this means `FaultRepair` has the word 'reset' in almost all non-empty entries. With 'reset' being exactly 5 characters and most non-empty entries in `FaultRepair` being at most 5 characters (of which, those that contain reset are over 70% of entries), it is easy to see that the `FaultDescription` field contains richer, more informative context.

Feature Name	Data Type	Description
FaultDate	Date-time	Date the fault occurred, with a precision up to the nearest second.
UserRun	String	Operational cycle that this fault has occurred within. Cycle information up to 2016 can be found at (ISIS Neutron and Muon Source, 2024).
Downtime	Integer	The amount of time, in hours the downtime has occurred for.
Group	String, Fixed Category	The group that the faulty equipment is part of. There are 13 unique equipment groups.
Equipment	String, Fixed Category	The equipment type that failed. There are around 200 unique equipment types that have been logged to fail.
FaultDescription	String, Free-form	This is a free-form, unstructured text field that allows the on-shift operational crew to note details about the problem diagnosis and remediation that has occurred. There is no constraint to the size of this text field.
FaultRepair	String, Free-form	Another free-form, unstructured text field that allows the on-shift operational crew to note only the remediation steps. The crew seems to have stopped using this field after the end of 2017, preferring to put remediation steps in FaultDescription.
ManagersComments	String, Free-form	A very rarely used free-form text field, where the manager in charge of the on-shift crew will input comments.

Table 3.1: Description of important features (columns) in the Operalog.

3.1.2 Preliminary Data Analysis

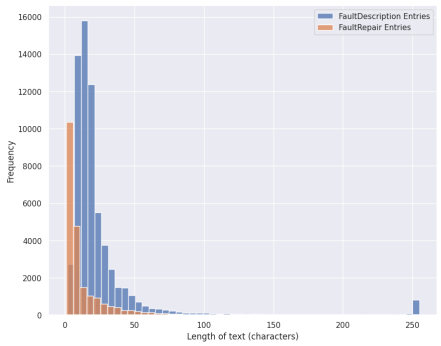
:

- Talk about amount of issues per year
- Talk about the distribution of text lengths across whole df
- Talk about the distribution of text lengths across iondf

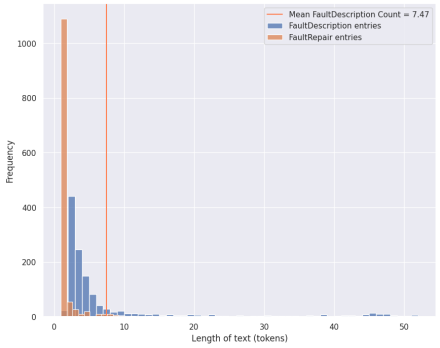
3.2 Overview

Make a diagram that showcases the process (and all the hyperparameter choices)

The process of transforming an unlabelled, uninformative Operalog into an informed, induced-label tagged dataset can be broadly characterised into five major steps. These steps are: (1) initial pre-processing and normalisation of the `FaultDescription` natural language

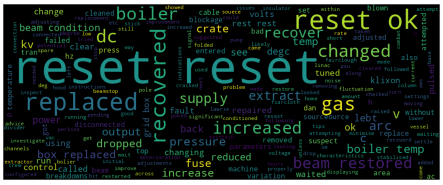


(a) Frequency distribution of the text lengths for non-blank `FaultDescription` and `FaultRepair` fields. Each bar represents a range of 5 characters.

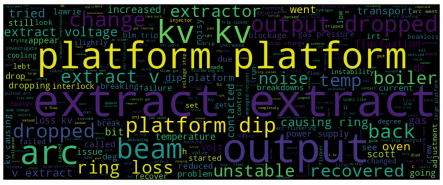


(b) Frequency distribution of the number of tokens for non-blank `FaultDescription` and `FaultRepair` entries. Each bar represents a range of around 2 tokens.

Figure 3.1: Frequency distributions of characters and tokens.



(a) FaultRepair



(b) FaultDescription

Figure 3.2: Comparison of word clouds for the top 300 most common words for the FaultRepair field versus the FaultDescription field. Larger words are more common and collocations are enabled so both words and phrases are shown, leading to some repeating.

text to standardise text and reduce noise; (2) embedding the cleaned text into a high-dimensional embedding space to capture semantic meaning numerically; (3) low-dimensional projection of the embedding space while respecting the higher-dimensional topological manifold (that is, aiming to preserve the structure and relationships present in the original high-dimensional space) to enable effective clustering while preserving structure; (4) performing clustering on the low-dimensional data, optimising clustering using clustering metrics as heuristics for the 'goodness' of a cluster to group similar fault descriptions automatically; and (5) using natural language processing to induce label names for each cluster and tagging each entry with a meaningful label identifying a fault category. Here, we choose to solely consider the `FaultDescription` field as the vast majority of `FaultRepair` fields are either empty or less than or equal to 5 characters long.

Several hyperparameters affect the performance of the label generating process. Examples include model or algorithm specific parameters (i.e. the dimensionality reduction or clustering algorithms) and clustering 'goodness' heuristics used. These hyperparameters are tuned using Optuna [4], an automatic optimisation framework specifically designed for machine learning applications and allows for multi-objective optimisation. This hyperparameter optimisation process determines 'best' induced categories, and thus labels, for each issue type.

Additionally, as this project is an early stage exploration of auto-categorisation for the ISIS Operalog, the major focus is on ion source `Equipment` type unless explicitly specified. This enables the project to focus on refining the approach in a scoped manner with a structured goal in mind - to progress the auto-categorisation effort on the ISIS Operalog. The subsequent sections of this chapter detail each stage of this end-to-end pipeline, developed with the goal of producing an effective auto-categorisation model for the Operalog data.

Clean up wording in the above

3.3 Natural Language Pre-processing

The aim of pre-processing the unstructured, English natural language text data from `FaultDescription` is to normalise and prepare it for text embedding. Normalisation allows many syntactic permutations or augmentations of a text to be collapsed into one standardised form. This allows for consistent results as it ensures texts that are syntactically different but semantically similar will be mapped by embedding model to the same value. For pre-processing, and the rest of the pipeline, each log entry's `FaultDescription` is considered one text.

Clean up wording in the above

Tokenisation is one of the earliest stages of natural language processing. To perform tokenisation, we use Tensorflow's `UnicodeScriptTokenizer` [1]. Tensorflow is 'an end-to-end platform for machine learning', which provides efficient machine learning algorithm implementations. A tokeniser, such as this, splits the text into sub-units called 'tokens' [27]. These tokens are useful to pass to subsequent natural language processing stages which is, in our case, text embedding. UnicodeScriptTokenizer, a specialised version of Tensorflow's `Tokenizer`, tokenises 'UTF-8 strings by splitting where there is a change in Unicode script' ([1], Tensorflow Text).

Once tokenised, stop-word removal is performed. Stop-words are non-informative words such as articles, prepositions and pronouns. Therefore removal of these words tends to help models have access to informative contexts and reduce noise [76]. We consider the standard

English stop-words from NLTK [10] and some context specific stop-words. This includes 'ion source' which does not provide any informative contexts as we have already scoped our dataset to ion source `Equipment` and 'breakdown' which appears in just over 50% of entries.

As mentioned previously, there are a few non-trivial abbreviations used by the operational crew when writing a log entry. These abbreviations are normalised into a standard English word or phrase (see Table 3.2).

Furthermore, punctuation tokens are converted into one of two categories: end-of-sentence (`EOS`) tokens or regular punctuation (`PUNC`) tokens, which are self-explanatory. The goal for this is to normalise punctuation to either signify sentence boundaries or collapse into one token. The majority of the `FaultDescription` text does not typically utilise punctuation to convey additional meaning. However, this is not trivial to see and thus the step was made parameterisable via a flag which decides whether punctuation mapping is enabled. Similarly, text casing normalisation (converting all text to lower-case) follows the same procedure.

Just in terms of ion source `Equipment`, the pre-processing is applied across 1251 log entries, ranging from 2009 - 2023. Tokenisation and operating on tokens is computationally intensive, thus the solution operates on tensors, leveraging Tensorflow capabilities to perform GPU-based compute, which speeds up machine learning based applications [1, 9].

As text pre-processing is computationally intensive, using these flags to perform hyper-parameter optimisation (Section 3.7) is an inefficient use of compute. This is because the pre-processing must be re-computed using these Boolean flags, for each configuration run. As these flags are Boolean, it is relatively easy to manually enumerate each configuration option and, thus, these parameters are instead exposed through the CLI application (Section 3.9) rather than in the hyperparameter optimisation process.

Clean up wording in the above

Table 3.2: Operational crew abbreviation mapping.

Abbreviation Regular Expression	Mapped word or phrase
o/p	output
i/s	ion source
(b/down | break-down | b\down | b/d)	breakdown

3.4 Text Embedding Model Selection and Application

After pre-processing, the cleaned and normalised `FaultDescription` text requires transformation into a numerical format suitable for analysis. Sentence embedding (introduced in Section 2.2) provides this by mapping text into a high-dimensional vector space, where semantic relationships are preserved. This is crucial for the project's goal: automatically categorising log entries by grouping semantically similar entries, which corresponds to finding similar fault types.

3.4.1 Dataset and hardware driven requirements

Effective categorisation relies on an embedding model that can accurately capture the syntactic and semantic nuances of these technical descriptions. Figure 3.1b highlights that, although the average `FaultDescription` entry has roughly 7 tokens, in the dataset, over 60% of entries have less than 5 tokens and around 10% of the 1251 entries have over 20 tokens. This includes entries which stretch to multiple sentences. Additionally, the vocabulary is heavily domain specific (i.e. 'ion source') and a single out-of-place adjective can reverse the meaning of a fault report (i.e. 'beam loss' versus 'no beam loss').

Furthermore, the hardware available for research is detailed in Table 3.3. Some models, such as large language models with a parameter counts in hundreds of millions or billions may exceed the 6GB GPU VRAM limit during inference. Therefore, we restrict consideration to models whose inference memory footprint and compute latency remain feasible on this hardware.

As a result, we require a model that (1) handles sequences that contain both long- and short-term dependencies but still establish semantic relationships effectively; (2) is able to capture the semantic contextual meaning from the entire sequence - that is, bidirectionally; and (3) is able to be feasibly loaded and run on the research hardware specified in Table 3.3.

Hardware	Description
CPU	11th Gen Intel i7-11800H (16 core) @ 4.600GHz
RAM	64GB
GPU	NVIDIA GeForce RTX 3060 Mobile / Max-Q
GPU VRAM	6GB

Table 3.3: Research hardware description.

3.4.2 Candidate Models

With these constraints in mind, we look at the recent state-of-the-art embedding models and highlight models that were not in consideration. Candidate models were compared on the constraints above and selected based on pre-fine-tuned models published on HuggingFace (so we can use them directly, without re-training) [97]. HuggingFace is an open-source platform for machine learning related development, where researchers can upload pre-trained and fine-tuned model parameters. The models that ranked high on clustering tasks on the Massive Text Embedding Benchmark (MTEB) leaderboard, which compares more than 100 text and image embedding models across 132 tasks of 9 categories - 17 of which are clustering tasks [57].

Two BERT-family sentence-embedding models satisfy the constraints above, MPNet and Nomic. Both models were used with their out-of-the-box HuggingFace settings (i.e. maximum token lengths of 384 and 8192, respectively and embedding dimension of 768) [77, 60, 61].

MPNet - an overview 512-token context window with a 768 embedding dimension. Unifies BERT's masked and XLNet's permutation language modelling pre-training objectives. This gives better positional awareness for tokens that are further away from each other [77]. MPNet

has around 109 million parameters, meaning it is able to load onto the research hardware. The HuggingFace model path is: `sentence-transformers/all-mpnet-base-v2` and achieves a clustering score of 40.77, ranking at 98 on the MTEB leaderboard, at time of writing.

Nomic - an overview Current state-of-the-art 8192-token context window with up to a 768 embedding dimension. Utilises rotary positional embeddings and flash attention that provide long-context support while remaining relatively small, with around 137 million parameters. Task prefixes (i.e. `classification:`, `clustering:`, etc.) encourage distinct semantic sub-spaces, thus we prepend all inputs passed to this model with 'clustering:'. Specifically, we use version 1.5 which takes advantage of Matryoshka Representation Learning (MRL) and is able to encode coarse versus fine-grained semantic information in embedding vector sub-spaces [60, 61]. The HuggingFace model path is: `nomic-ai/nomic-embed-text-v1.5` and achieves a clustering score of 41.55, ranking at 54 on the MTEB leaderboard, at time of writing.

Excluded Models The following embedding models were evaluated but ultimately excluded, either because they exceeded our 6GB VRAM budget or failed to meet clustering performance thresholds in pilot runs (see Table 3.4). Excluding these ensures all retained models both fit our hardware and deliver the minimum clustering quality we require.

Model	Parameters	Exclusion Reason	Reference
Qwen2-7B	7 billion	Requires more than 6 GB VRAM at inference.	Yang et al., 2024
Mistral-7B	7 billion	Similar VRAM overrun on consumer GPUs.	Wang et al., 2023
e5-large	440 million	Scored only 32 points on MTEB's clustering benchmark.	Wang et al., 2024

Table 3.4: Models considered but excluded from the embedding pipeline.

Detailed Rationale

In this section, we elaborate on the rationale for selecting MPNet and Nomic as the primary embedding models for evaluation, connecting their specific properties to the requirements identified in Section 3.4.1 and the conceptual background from Chapter 2.

MPNet Justification MPNet [77] was selected as a strong candidate due to its hybrid pre-training approach, designed to capture the benefits of both masked language modelling (like BERT [22]) and permutation language modelling (like XLNet [99]) while mitigating their respective drawbacks. This hybrid nature seemed particularly well-suited to the Operalog dataset for several reasons.

1. The Operalog contains a mixture of short, concise fault entries and longer, multi-sentence descriptions detailing causal factors or sequences of events (as discussed in Section 3.4.1). Effectively representing these requires capturing both local word relationships and broader contextual meaning, potentially involving long-range dependencies. MPNet's unified objective, which learns bidirectional context while explicitly modelling dependencies between tokens (unlike BERT's independence assumption), appeared promising for handling this fault entry length variability [77]. Understanding

the relationship between specific technical terms within the log entries, even if separated by other words (i.e. identifying a fault type and its resolution described later in the entry), is key to correctly categorising the fault, making MPNet's dependency modelling capabilities attractive. The formal definition of MPNet's objective function and an example of its input structuring are provided in Appendix A.2 for completeness (see Equations A.6 and A.7).

2. MPNet addresses potential discrepancies between pre-training and downstream tasks. While XLNet's permutation approach avoids BERT's reliance on the artificial `[MASK]` token, MPNet further refines the mechanism by modifying the two-stream attention masks [77]. These modifications allow the model components responsible for prediction (the query stream) to access information from the entire sequence during pre-training, better mirroring how the model is used during inference for embedding generation [77]. This alignment is advantageous when using an off-the-shelf pre-trained (and further fine-tuned) model directly for embedding, as done in this project. This full sequence visibility aids in capturing long-range dependencies potentially present in Operalog entries where initial symptoms might be reiterated or linked to actions later in the description [77].

3. MPNet demonstrated strong performance on various NLP benchmarks at the time of its release, including the GLUE benchmark [92, 77], suggesting a robust general language understanding capability applicable to the technical language in the Operalog. Its parameter count (approximately 109 million) also made it feasible for the available hardware (Table 3.3).

Nomic Justification Nomic, specifically `nomic-embed-text-v1.5` [60, 61], was selected as a representative of more recent (i.e. fourth-generation), state-of-the-art open-source embedding models, offering several features potentially beneficial for this project.

1. A key advantage is its explicit design for long-context understanding. It utilizes Rotary Positional Embeddings (RoPE) [81] instead of traditional absolute or relative positional embeddings [89]. RoPE has been shown to improve generalisation to sequence lengths not seen during training [81]. Given the significant variation in the length of `FaultDescription` entries in the Operalog (Figure 3.1), Nomic's ability to handle contexts up to 8192 tokens effectively was a major consideration - potentially offering better semantic representation for longer, more detailed entries compared to models with shorter context windows like MPNet (512 tokens).

2. Computational efficiency was another critical factor. Despite its long-context capability, Nomic v1.5 has a relatively low parameter count (approximatly 137 million) and incorporates optimisations like FlashAttention [20], designed to reduce memory usage and latency [60]. This made it feasible to run inference within the project's 6GB GPU VRAM constraint (Table 3.3), unlike significantly larger multi-billion parameter models considered (Table 3.4).

3. Nomic also offers task-specific optimisation via input prefixes [60]. By prepending '`clustering:`' to each `FaultDescription` entry, we aimed to leverage specific semantic subspaces within the model potentially optimised during its pre-training for clustering tasks. This was hypothesised to generate embeddings more suitable with the primary goal of grouping similar fault types compared to using a generic embedding.

4. Nomic v1.5 incorporates Matryoshka Representation Learning (MRL) [40, 61]. While this project utilised the full 768-dimension embeddings, MRL imbues the embedding space with a nested structure where shorter prefixes remain meaningful. This structured property might implicitly aid downstream tasks like clustering by organizing semantic information hierarchically, although exploring different MRL embedding lengths was outside the scope of this project.

5. Nomic's demonstrated strong performance for its size on benchmarks like MTEB, including clustering tasks, provided empirical validation for its consideration [57, 62].

3.4.3 Section Summary

By evaluating both MPNet (representing a mature, unified BERT/XLNet approach) and Nomic (representing recent advancements in long-context efficiency and specialised embeddings), which meet the requirements criteria set in Section 3.4.1, this project aimed to explore different effective strategies for representing the challenging Operalog text data.

3.5 Embedding dimensionality reduction

The high dimensionality (768 dimensions) of the text embeddings necessitates dimensionality reduction, initially to enable visualisation and subsequently to aid clustering (referring to the 'curse of dimensionality' [90], see Section 2.3 for more details). This section explores and justifies the applicability of two techniques for this purpose: Principal Component Analysis (PCA) [65, 31], a linear method, and Uniform Manifold Approximation and Projection (UMAP) [52], a non-linear manifold learning approach. While another non-linear technique, t-distributed Stochastic Neighbour Embedding (t-SNE), was initially considered, we focused on the evaluation of PCA and UMAP. This decision was based on UMAP's reported advantages in preserving global data structure more effectively than t-SNE, alongside its generally greater computational efficiency and run-to-run consistency [52]. Given the project's scope, these practical benefits drew us away from t-SNE. The following subsections present a comparison between PCA and UMAP based on visualisations, justify the subsequent selection of UMAP for our pipeline; and detail its hyperparameter application to the Operalog dataset.

3.5.1 Visualisation

An example visualisation comparing the 3-dimensional projection, obtained using both PCA and UMAP in both MPNet and Nomic embedding spaces, is shown in Figure 3.4. Since the original data resides in a 768-dimension embedding space, direct visualisation is impossible. Projecting this data down to the three dimensions enables visualisation but potentially loses information, making purely visual comparisons between the method outputs subjective.

To supplement the visual assessment with a quantitative measure, we evaluate the spread within these 3-dimensional projections using normalised variance. Normalised variance, defined for a data matrix X in Equation 3.1, serves as a metric to quantify the overall dispersion of the points in the low-dimensional space. Observing Figure 3.4, the UMAP projections visually appear to exhibit more defined structures, such as clearer separations and groupings of points. Comparatively, the corresponding PCA projections look more dispersed and do not show any 'clusters' or groupings of points. This visual impression is supported quantitatively by the normalised variance metric, calculated for these specific runs. The values for the UMAP projections (0.90 for MPNet, 2.27 for Nomic) are substantially lower and closer to 1 than those for PCA (approximately 1.0×10^7 for MPNet, -2.4×10^9 for Nomic). In this context, the lower normalised variance for UMAP suggests a projection that arranges the data more compactly while potentially preserving meaningful neighbourhood relationships, consistent with UMAP's goal of modelling the data manifold.

$$\text{Normalised Variance}(X) = \frac{\text{Var}(X)}{\text{Mean}(X)} \tag{3.1}$$

However, as discussed in Section 2.3.2, UMAP is a stochastic algorithm. That is, due to an element of algorithmic randomness, its output naturally varies between runs [52]. To account for this variability, Figure 3.3 visualises the distribution of normalised variance results obtained over 100 independent UMAP runs for both embedding spaces. While variation exists, these distributions reaffirm the concept that UMAP consistently produces projections with significantly lower normalised variance than those produced by PCA. The distributions exhibit some right-skewness, particularly for the MPNet embeddings but remain concentrated in a range indicative of more structured projections than those from PCA.

Based on the combination of visually apparent characteristics in the example projections Figure 3.4 and the consistently lower (although variable) normalised variance scores, indicative of the better preservation of manifold structure, in Figure 3.3, UMAP is selected as the dimensionality reduction technique for the remainder of this pipeline. It appears more likely, than PCA, to produce a low-dimensional representation that retains structural information beneficial for the next clustering stage.

3.5.2 UMAP Hyperparameters

As discussed in Section 2.3.2, tuning UMAP's hyperparameters is crucial for obtaining a meaningful low-dimensional representation of the high-dimensional Operalog text embeddings. The choice of parameters like `n_neighbors` and `min_dist`, along with the distance `metric`, significantly influences the resulting structure.

Based on the nature of the Operalog data, adjusting `n_neighbors` involves a trade-off. Lower values might isolate highly specific or rare fault types but could cause fragmentation of broader categories. Higher values might group related issues more globally but risk merging unique sub-types. Given the mix of common and rare faults, exploring a range of neighbour values is necessary.

Similarly, `min_dist` affects the visual density. Here, lower values may create tightly

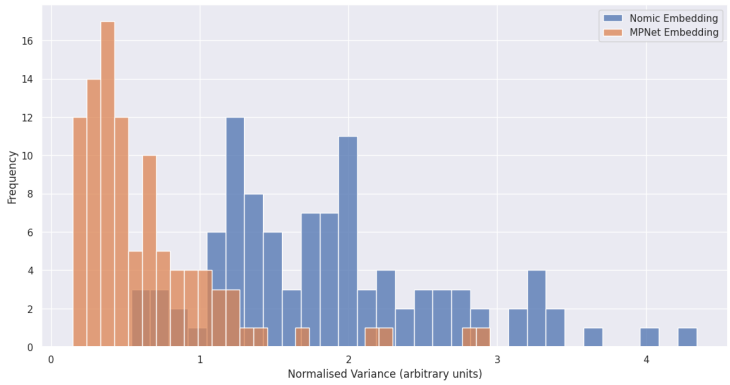


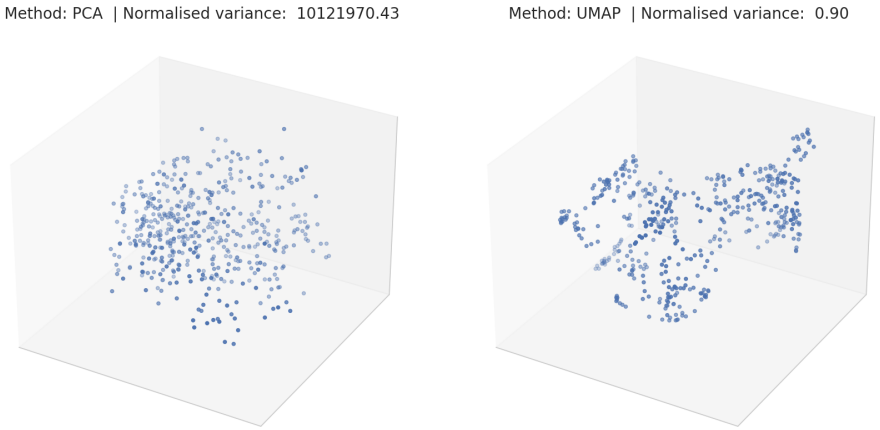
Figure 3.3: Histogram plot of 100 UMAP runs' normalised variance (Equation 3.1) is shown for both MPNet and Nomic embedding spaces of the Operalog `FaultDescription` entries. UMAP parameters: `min_dist` = 0.1, `n_neighbors` = 15, `metric` = cosine.

packed, visually distinct clusters suitable for identifying major categories. However, higher values might reveal finer intra-cluster variations relevant for understanding subtypes, but could make overall separation less clear. This parameter will affect clustering algorithms, such as DBSCAN as it essentially determines potential cluster density [24]. The choice of `metric` (i.e. 'cosine' vs. 'euclidean') is also critical, as 'cosine' is often better suited for high-dimensional text embeddings like those used here [8, 16].

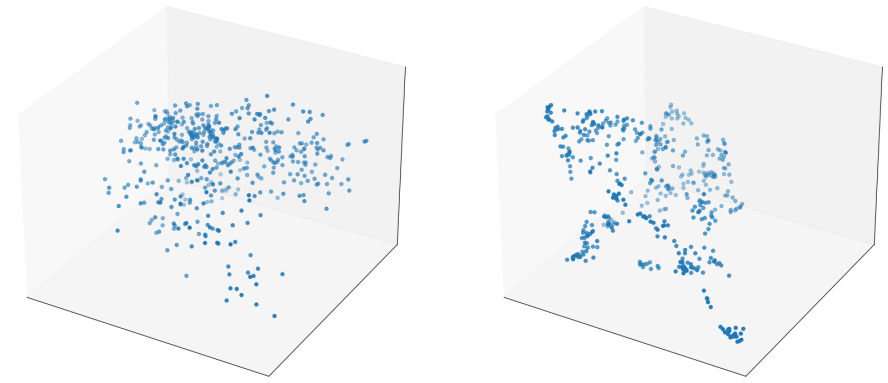
To gain an initial understanding of suitable UMAP parameters for the Operalog data, a naive grid search was performed using both MPNet and Nomic embeddings across the parameter ranges displayed in Table 3.5. The resulting visualisations for each parameter combination are presented in Appendix A.3. However, visual inspection of these results highlighted a surprising finding: contrary to common practice for text embeddings, cosine distance did not appear to yield projections demonstrably better at preserving structure more faithfully than Euclidean distance in this preliminary exploration. These inconclusive and unexpected outcomes underscore the need for a more systematic hyperparameter optimisation phase, which is later detailed in Section 3.7.

Table 3.5: UMAP hyperparameter values explored in naive grid search

Hyperparameter	**Values Explored**
`n_neighbors`	{ 5, 10, 13, 15, 20, 25 }
`min_dist`	{ 0.0125, 0.05, 0.1, 0.15, 0.2 }
`metric`	{ `'euclidean'`, `'cosine'` }



(a) MPNet — PCA **(b)** MPNet — UMAP



(c) Nomic — PCA **(d)** Nomic — UMAP

Figure 3.4: Comparison of PCA versus UMAP dimensionality reduction on the MPNet (top row) and Nomic (bottom row) embedding spaces on Operalog `FaultDescription` entries Normalised variance (Equation 3.1) is shown. UMAP parameters: `min_dist` = 0.1, `n_neighbors` = 15, `metric` = cosine.

3.5.3 Section Summary

Based on the comparison in the previous subsections, UMAP was selected over PCA as the preferred dimensionality reduction technique for the MPNet and Nomic embedding spaces. The preliminary visualisations also highlighted that UMAP's output is sensitive to its hyperparameters, indicating that a systematic approach is required to find optimal settings for revealing structure within the Operalog data. Directly evaluating the 'quality' of different UMAP projections is challenging therefore, to assess the effectiveness of various projections (resulting from different hyperparameter settings), we adopt an indirect evaluation strategy by applying clustering algorithms to the low-dimensional UMAP outputs. The quality of the resulting clusters, measured using specific clustering evaluation metrics (introduced in Section **??**), will serve as a proxy for how well the UMAP projection preserved the relevant structure for categorisation. The next section introduces the clustering algorithms and metrics chosen for this purpose

3.6 Unsupervised categorisation through clustering

Following dimensionality reduction via UMAP, the resulting low-dimensional embedding space is prepared for the core task of clustering. As outlined in Section 2.4, clustering is an unsupervised machine learning approach used to group similar data points. Within this project's context, the goal is to group the embeddings of Operalog `FaultDescription` entries such that each resulting cluster corresponds to a distinct, underlying fault category. Ideally, the clustering process should automatically discover these latent categories from the data structure.

To explore different clustering paradigms, three algorithms were selected for evaluation based on their characteristics described in Section 2.4:

- k-Medoids [37] - A partitioning-based algorithm that groups data around actual data points (medoids).
- DBSCAN [24] - A density-based algorithm that identifies clusters based on regions of high point density.
- HDBSCAN [15] - An algorithm combining density-based principles with hierarchical clustering to identify stable clusters across varying densities.

Notably, k-Means [51], another common partitioning algorithm, was considered but excluded due to its known sensitivity to initialisation and potential instability compared to k-Medoids [101].

A fundamental challenge in unsupervised clustering is evaluating the quality of the resulting clusters without ground truth labels. To quantitatively assess and compare the different clustering outcomes, three established internal evaluation metrics (heuristics) are employed: the Silhouette coefficient [70], the Davies-Bouldin index [21], and the Calinski-Harabasz index [14].

This section details the rationale for selecting the specific clustering algorithms and introduces the evaluation heuristics used in the subsequent hyperparameter optimisation phase.

3.6.1 Clustering Candidates

The suitability of each clustering algorithm depends on the structure of the data and the nature of the desired clusters. Below, we discuss the characteristics of each chosen algorithm in the context of the Operalog embedding data.

k-Medoids As described in Section 2.4.1, the k-Medoids algorithm (implemented via the PAM algorithm in common libraries like scikit-learn [66]) requires only one primary hyperparameter: k, the desired number of clusters. While determining the optimal k for the unknown Operalog fault categories is non-trivial, having a single parameter simplifies the optimisation process. Furthermore, k has an intuitive interpretation: higher k might isolate rate but specific ion source faults, while lower k might group broader issues such as 'ion source failure'. However, k-Medoids assumes clusters are roughly spherical and aims to minimise distances to central medoids [37]. Visual inspection of the UMAP projections (Figure 3.4 and Appendix A.3) suggests the Operalog embeddings form complex, non-spherical structures. Therefore, k-Medoids might struggle to accurately capture the boundaries of these potential fault categories.

DBSCAN This algorithm (Section 2.4.2) uses two key hyperparameters, `Eps` and `MinPts`, to define local density. Its main advantages are the ability to find arbitrarily shaped clusters and explicitly identify noise points, which could be valuable for isolating unusual or poorly described Operalog entries [24]. However, DBSCAN's performance relies heavily on selecting appropriate `Eps` and `MinPts` values. Its use of a single, global density threshold makes it potentially sensitive to datasets containing clusters of varying densities [15]. Assessing whether the Operalog embeddings exhibit such density variations is difficult visually. The IsoScore metric [71], which measures the uniformity of space utilisation by the projected points, provides some insight. Figure 3.5 shows average IsoScores of 0.653 (MPNet) and 0.710 (Nomic) for the UMAP projections. While these values indicate reasonable space utilisation, they are not close to 1, suggesting potential density variations that might pose a challenge for DBSCAN's global threshold approach. We expect some very short `FaultDescription` entries which are common, leading to dense clusters and longer entries, which are more rare, leading to sparse clusters. This insight would support the concern about the varying densities.

HDBSCAN As an extension of DBSCAN (Section 2.4.3), HDBSCAN aims to overcome the sensitivity to Eps and the varying density problem. It achieves this by building a cluster hierarchy and using a stability measure to extract clusters that persist across a range of density levels [15]. It requires only one primary hyperparameter, m_{pts} (equivalent to `MinPts` in DBSCAN), making it simpler to tune than DBSCAN. Its ability to handle varying densities and its robustness (consistent results between runs) make it a strong candidate for the Operalog data, where the true number and density of fault categories are unknown. As mentioned before, since we expect a clusters of varied densities, being able to find stable clusters in this space is desirable for discovering potentially diverse fault types in the Operalog.

Based on these characteristics, we hypothesize that HDBSCAN might be best suited for this task, followed by DBSCAN, with k-Medoids likely performing worst due to the non-spherical nature of the projected data. However, empirical evaluation across all three algo-

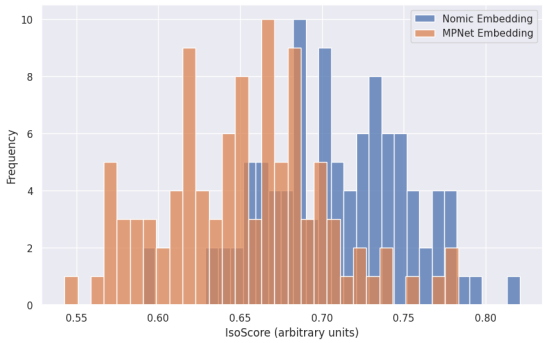


Figure 3.5: IsoScore comparison of MPNet versus Nomic embeddings over 100 runs of UMAP. UMAP parameters: `min_dist` = 0.1, `n_neighbors` = 15, `metric` = cosine.

rithms is necessary to validate these expectations and determine the most effective approach for automatically categorising the Operalog fault descriptions. The subsequent hyperparameter optimisation (Section 3.7) and evaluation will compare their performance quantitatively using the clustering heuristics.

3.6.2 Clustering heuristics

Motivated by the necessity to quantitatively evaluate the quality of different clustering results in this unsupervised setting, three well-understood, internal clustering evaluation metrics (heuristics) are utilised. These metrics assess the 'goodness-of-fit' based on the properties of the clusters themselves:

- Silhouette coefficient - This metric evaluates how well-separated clusters are by comparing each point's average distance to points in its own cluster (cohesion) with its average distance to points in the nearest neighbouring cluster (separation) [70]. Scores range from -1 to 1, where values closer to 1 indicate well-separated, dense clusters, values near 0 suggest overlapping clusters, and negative values indicate potential misclassifications.

- Davies-Bouldin index - This index quantifies the average 'similarity' between each cluster and its most similar neighbour, where similarity is defined as the ratio of within-cluster scatter to between-cluster separation [21]. Lower Davies-Bouldin scores indicate better clustering, implying clusters are compact and well-separated from their nearest neighbours.

- Calinski-Harabasz index - Measures the ratio of the sum of between-cluster dispersion to the sum of within-cluster dispersion for all clusters [14]. Higher Calinski-Harabasz scores generally indicate better defined clusters. That is, clusters are dense and well-separated.

Furthermore, each of these metrics captures a different aspect of clustering quality (compactness, separation and variance ratios). As it is unclear which single metric best reflects the desired clustering structure for a specific dataset and task, particularly in an exploratory

context like identifying Operalog fault categories, all three metrics were used collectively during the hyperparameter optimisation phase (Section 3.7) to provide a more comprehensive assessment of the different clustering solutions.

3.7 Hyperparameter optimisation

As established in previous sections, achieving effective unsupervised categorisation requires careful selection of parameters for both the dimensionality reduction (UMAP, Section 2.3.2) and clustering stages (Section 2.4). Given the interplay between these stages and the sensitivity of algorithms like UMAP and DBSCAN to their settings, a systematic approach to find optimal parameter combinations is necessary. Furthermore, as discussed in Section 3.6.2 different internal clustering evaluation metrics capture distinct aspects of cluster quality, and it is unclear which single metric best reflects the desired outcome for a specific dataset.

To address these challenges, this project employed the Optuna framework for automatic hyperparameter optimisation [4]. Optuna was selected over alternatives like Hyperband [46] primarily due to its robust support for multi-objective optimisation [4]. This capability is crucial here, as the goal was to simultaneously optimise the clustering results based on multiple evaluation heuristics: the Silhouette coefficient, Davies-Bouldin index, and Calinski-Harabasz index. The optimisation aims to maximise the Silhouette and Calinski-Harabasz scores while minimising the Davies-Bouldin score [70, 21, 14]. A practical advantage of Optuna noted during this work was its ability to save and resume optimisation states, improving research efficiency [4].

An Optuna optimisation session, termed a 'study', iteratively evaluates different hyperparameter configurations ('trials') [4]. For each trial, Optuna suggests parameter values based on a chosen sampler algorithm. This study utilised the Tree-structured Parzen Estimator sampler, which is Optuna's default algorithm known for efficiently exploring complex search spaces [95]. The core components of the Optuna study setup were:

- Objective function - A function designed to take a set of suggested hyperparameters, execute the relevant parts of the pipeline (UMAP transformation and clustering), calculate the three chosen clustering heuristic scores, and return these scores to Optuna. This function defines the search space by specifying the range or set of possible values for each hyperparameter using Optuna's library semantics. The specific hyperparameters optimised included shared UMAP settings and algorithm-specific parameters, with ranges informed by preliminary experimentation as shown in Table 3.6. To handle potential errors arising from UMAP's stochastic nature or problematic parameter combinations leading to clustering failures, the objective function incorporated a retry mechanism (up to 10 attempts). If errors persisted, the trial was assigned extremely poor heuristic scores to guide the optimiser away from that configuration region.

- Number of trials - An upper limit defining the maximum number of configurations to evaluate within the study.

Upon completion of the specified number of trials, the study yields a set of optimal configurations, those for which no single objective (i.e. heuristic) score can be improved without

worsening at least one other objective. Since multiple objectives often lead to several such 'best' configurations, a post-optimisation filtering step was applied using rank aggregation [47]. For each heuristic, the optimal configurations were ranked, and each configuration received points equal to its inverse rank. For example, for 10 configurations, rank 1 gets 10 points, rank 2 gets 9, ..., rank 10 gets 1 point. These points were summed across all three heuristics for each configuration. The configuration(s) with the highest total score were selected as the final 'best' performing set of hyperparameters, representing a balanced choice across the different measures of clustering quality. The quantitative results of this optimisation are presented and discussed in Chapter 4.

Table 3.6: Optuna hyperparameter search space

Scope	Hyperparameter	Range / Set
General	`n_components`	Integer range $[2, 6]$
General	`n_neighbors`	Integer range $[3, 15]$
General	`min_dist`	Float range $(0, 0.2)$
General	`metric`	Set { `'euclidean'`, `'cosine'` }
k-Medoids	`k`	Integer range $[3, 30]$
DBSCAN	`Eps`	Float range $[0.1, 1.0]$
DBSCAN	`MinPts`	Integer range $[3, 12]$
HDBSCAN	m_{pts}	Integer range $[3, 30]$

3.8 Label generation

After achieving the 'most optimal' clusters through the hyperparameter optimisation process described in Section 3.6, the final step involves automatically inferring meaningful names or labels for these discovered groups. This automatic label generation was performed leveraging the natural language processing capabilities of the SpaCy library [30]. The main idea is to capture the essence of the text in each cluster into a precise but representative phrase.

The process begins by examining all the `FaultDescription` sentences assigned to a particular cluster. For each sentence, SpaCy is employed to perform detailed linguistic analysis, including part-of-speech tagging and dependency parsing [84]. This helps identify key components of the grammar and determine their relationships within the sentence.

To focus on the most informative words, several filtering (pre-processing) steps are applied during this analysis. Common English stop-words, along with project-specific stop-words (like 'ion source' or 'breakdown', as discussed in Section 3.3), are disregarded [84]. Furthermore, tokens that are non-alphanumeric or consist of fewer than two characters are excluded to reduce noise from punctuation or trivial entries.

From the remaining tokens, the system specifically identifies and collects verbs (particularly the root verb of sentences), direct objects (identified via dependency relations), nouns, and adjectives. To ensure consistency and group related word forms, lemmatisation is applied, reducing words to their base or dictionary form. For example, 'changed' and 'changing'

become 'change' [84].

Building the label then involves constructing a naive phrase by combining the most frequently occurring elements identified across all sentences in the cluster. Specifically, the system determines:

- The single most common verb (identified as the root of sentences).
- The single most common direct object (based on dependency parsing).
- The top two most common nouns (excluding any words already selected as the verb or direct object to ensure variety).

These selected components are then combined into a single, underscore-separated string to form the cluster's generated label. This approach incorporates a mechanism to handle cases where fewer than the desired number of unique terms are available within a cluster's vocabulary, ensuring a label is still generated. The selection process considers a configurable maximum number of top nouns to examine, ensuring efficiency while capturing the primary themes.

Finally, this label generation process is integrated into the data handling workflow. It operates on the clustered data, mapping the generated underscore-separated label back to each original log entry belonging to that cluster.

The intuition behind this heuristic approach is to capture the core semantic meaning of the fault descriptions within a cluster by focusing on the primary action (verb), the entity acted upon (direct object), and the key subjects or items involved (nouns), thereby creating a succinct, machine-generated descriptor for each automatically identified fault category.

3.9 CLI Application

To facilitate the practical application and reproducibility of the methodology developed in this project, a Command Line Interface (CLI) tool was created, accompanying the code submission. This application encapsulates the end-to-end pipeline, providing users with a means to apply the automatic labelling process to Operalog data.

The primary functions offered by the CLI include managing the hyperparameter optimisation process, allowing users to initiate new optimisation runs or resume existing ones, and executing the label inference step on clustered data. Upon completion, the tool generates two key output artefacts: an updated version of the Operalog dataset, augmented with columns indicating the assigned numerical cluster identifier (label) and the inferred natural language label (`real_label`) for each entry, and a structured JSON file. This JSON file contains lists of the original sentences grouped by their assigned cluster, alongside a mapping from each cluster ID to its generated natural language label.

To offer flexibility, the CLI exposes several parameters for user configuration. The primary parameters are listed below:

- Model - The HuggingFace model path.
- Keep punctuation - Boolean toggle which keeps punctuation from being stripped during the text pre-processing stage.

- Ignore case - Boolean toggle which ignores casing during the text pre-processing stage.
- Clustering methods - Define one or more clustering methods to use for the hyperparameter optimisation stage.
- Heuristic methods - Define one or more heuristic methods to use for the hyperparameter optimisation stage.

The design of the CLI application and its underlying codebase explicitly considers future development. Its structure allows users to experiment with embedding models beyond those researched in this project by simply providing a different model path. Furthermore, the modular organisation of the codebase aims to simplify the future integration of additional clustering algorithms or evaluation heuristics, enhancing the tool's potential utility for ongoing research or operational use.

Chapter 4

Results and Discussion

:

- Rank aggregation
- plot whiskers (or maybe in results?). acc this should be in results as it makes sense that is quantitative evaluation of results.
- results should also talk about qualitatively and motivate the conclusion, this is a step in the right direction but not 100%.

: Describe the results and analyse the results

- Analyse the sentence embedding results.
- Anaylse the UMAP hyperparameter optimisation qualitatively, mention that we use Optuna.
- Next steps: Fine tuning the PLM (Nomic or MPNet). Using MOE nomic (v2). Rent higher computation, then you can use top performing PLMS. Using t-SNE to see differences.

Chapter 5

Conclusion

:

- Definitely talk about clear and transparent PLMs and malicious stuff (see BERT section).

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[3] PY Abijith, Piyush Patidar, Gaurav Nair, and Rohan Pandya. Large language models trained on equipment maintenance text. In *Abu Dhabi International Petroleum Exhibition and Conference*, page D021S065R003. SPE, 2023.

[4] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[5] Ramiz M Aliguliyev. A new sentence similarity measure and sentence based extractive technique for automatic text summarization. *Expert Systems with Applications*, 36(4):7764–7772, 2009.

[6] C Ankenbrandt, C Curtis, C Hojvat, RP Johnson, C Owen, C Schmidt, L Teng, and RC Webber. H- charge exchange injection systems. In *11th International Conference on High-Energy Accelerators: Geneva, Switzerland, July 7–11, 1980*, pages 260–271. Springer, 1980.

[7] Asim Yaqoob. First line diagnosis at ISIS, Oct 2017. URL https://indico.cern.ch/event/558933/contributions/2724364. Last visited 2025-04-15.

[8] Kensuke Baba, Tetsuya Nakatoh, and Toshiro Minami. Plagiarism detection using document similarity based on distributed representation. *Procedia computer science*, 111:382–387, 2017.

[9] Ioana Baldini, Stephen J Fink, and Erik Altman. Predicting gpu performance from cpu runs using machine learning. In *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, pages 254–261. IEEE, 2014.

[10] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[11] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[12] Alexander Budanitsky and Graeme Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and other lexical resources*, volume 2, pages 2–2, 2001.

[13] T Tony Cai and Rong Ma. Theoretical foundations of t-sne for visualizing high-dimensional clustered data. *Journal of Machine Learning Research*, 23(301):1–54, 2022.

[14] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[15] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer, 2013.

[16] Hongliu Cao. Recent advances in text embedding: A comprehensive review of top-performing methods on the mteb benchmark. *arXiv preprint arXiv:2406.01607*, 2024.

[17] Thyago P Carvalho, Fabrízzio AAMN Soares, Roberto Vita, Roberto da P Francisco, João P Basto, and Symone GS Alcalá. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137:106024, 2019.

[18] Coenen, Andy and Pearce, Adam. Understanding UMAP, 2020. URL https://pair-code.github.io/understanding-umap/. Last visited 2025-04-21.

[19] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[20] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[21] David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[23] Sergey Edunov, Alexei Baevski, and Michael Auli. Pre-trained language model representations for language generation. *arXiv preprint arXiv:1903.09722*, 2019.

[24] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[25] Maurizio Faccio, Alessandro Persona, Fabio Sgarbossa, and Giorgia Zanin. Industrial maintenance policy development: A quantitative framework. *International Journal of Production Economics*, 147: 85–93, 2014.

[26] Bolin Gao and Lacra Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*, 2017.

[27] Gregory Grefenstette. Tokenization. In *Syntactic wordclass tagging*, pages 117–133. Springer, 1999.

[28] John M Harris. *Combinatorics and graph theory*. Springer, 2008.

[29] Alexander Hinneburg and Daniel A Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. 1999.

[30] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. 2017.

[31] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[32] ISIS Neutron and Muon Source. A Practical Guide to the ISIS neutron and Muon source, Apr 2021. URL https://www.isis.stfc.ac.uk/Pages/News21_PracticalGuide.aspx. Last visited 2025-04-15.

[33] ISIS Neutron and Muon Source. ISIS beam operations, Apr 2024. URL https://www.isis.stfc.ac.uk/Pages/beam-status.aspx. Last visited 2025-04-15.

[34] ISIS Neutron and Muon Source. ISIS data gateway, 2025. URL https://data.isis.stfc.ac.uk/datagateway. Last visited 2025-04-16.

[35] Ali Jezzini, Mohammad Ayache, Lina Elkhansa, Bassem Makki, and Maya Zein. Effects of predictive maintenance (pdm), proactive maintenace (pom) & preventive maintenance (pm) on minimizing the faults in medical instruments. In *2013 2nd International conference on advances in biomedical engineering*, pages 53–56. IEEE, 2013.

[36] Taeho Jo. K nearest neighbor for text summarization using feature similarity. In *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, pages 1–5. IEEE, 2017.

[37] Leonard Kaufman and Peter Rousseeuw, J. *Partitioning Around Medoids (Program PAM)*, chapter 2, pages 68–125. John Wiley and Sons, Ltd, 1990.

[38] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[39] Sushil Kumar and Komal Kumar Bhatia. Semantic similarity and text summarization based novelty detection. *SN Applied Sciences*, 2(3):332, 2020.

[40] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

[41] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25 (2-3):259–284, 1998.

[42] Jinhyuk Lee, Zhuyun Dai, Xiaoqi Ren, Blair Chen, Daniel Cer, Jeremy R Cole, Kai Hui, Michael Boratko, Rajvi Kapadia, Wen Ding, et al. Gecko: Versatile text embeddings distilled from large language models. *arXiv preprint arXiv:2403.20327*, 2024.

[43] John M Lee. *Smooth manifolds*. Springer, 2003.

[44] John M Lee. *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media, 2006.

[45] Hongfei Li, Dhaivat Parikh, Qing He, Buyue Qian, Zhiguo Li, Dongping Fang, and Arun Hampapur. Improving rail network velocity: A machine learning approach to predictive maintenance. *Transportation Research Part C: Emerging Technologies*, 45:17–26, 2014.

[46] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[47] Xue Li, Xinlei Wang, and Guanghua Xiao. A comparative study of rank aggregation methods for partial and top ranked lists in genomic applications. *Briefings in bioinformatics*, 20(1):178–189, 2019.

[48] Hong Liang, Xiao Sun, Yunlei Sun, and Yuan Gao. Text feature extraction based on deep learning: a review. *EURASIP journal on wireless communications and networking*, 2017:1–12, 2017.

[49] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[50] Romans Lukashenko, Vita Graudina, and Janis Grundspenkis. Computer-based plagiarism detection methods and tools: an overview. In *Proceedings of the 2007 international conference on Computer systems and technologies*, pages 1–6, 2007.

[51] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, volume 5, pages 281–298. University of California press, 1967.

[52] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[53] HT Michael. Electronic circuits: fundamentals and applications, 2006.

[54] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[55] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023.

[56] R Keith Mobley. *An introduction to predictive maintenance*. Elsevier, 2002.

[57] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022.

[58] Dietrich W Müller and Günther Sawitzki. Excess mass estimates and tests for multimodality. *Journal of the American Statistical Association*, 86(415):738–746, 1991.

[59] Giancarlo Nota, Alberto Postiglione, and Rosario Carvello. Text mining techniques for the management of predictive maintenance. *Procedia Computer Science*, 200: 778–792, 2022.

[60] Zach Nussbaum, John X Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder. *arXiv preprint arXiv:2402.01613*, 2024.

[61] Nussbaum, Zach and Cembalest, Max. Unboxing Nomic Embed v1.5: Resizable Production Embeddings with Matryoshka Representation Learning, Feb 2024. URL https://www.nomic.ai/blog/posts/nomic-embed-matryoshka. Last visited 2025-04-20.

[62] Nussbaum, Zach and Cembalest, Max. Nomic Embed's Surprisingly Good MTEB Arena Elo Score, Aug 2024. URL https://www.nomic.ai/blog/posts/evaluating-embedding-models. Last visited 2025-04-18.

[63] Layla Oesper, Daniele Merico, Ruth Isserlin, and Gary D Bader. Wordcloud: a cytoscape plugin to create a visual semantic summary of networks. *Source code for biology and medicine*, 6(1):7, 2011.

[64] Zhaotai Pan, Yi Ge, Yu Chen Zhou, Jing Chang Huang, Yu Ling Zheng, Ning Zhang, Xiao Xing Liang, Peng Gao, Guan Qun Zhang, Qingyan Wang, et al. Cognitive acoustic analytics service for internet of things. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pages 96–103. IEEE, 2017.

[65] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[67] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[68] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[69] T Nora Raju, PA Rahana, Raichel Moncy, Sreedarsana Ajay, and Sindhya K Nambiar. Sentence similarity-a state of art approaches. In *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)*, pages 1–6. IEEE, 2022.

[70] 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.

[71] William Rudman, Nate Gillman, Taylor Rayne, and Carsten Eickhoff. Isoscore: Measuring the uniformity of embedding space utilization. *arXiv preprint arXiv:2108.07344*, 2021.

[72] Mark Sammons, Christos Christodoulopoulos, Parisa Kordjamshidi, Daniel Khashabi, Vivek Srikumar, and Dan Roth. Edison: Feature extraction for nlp, simplified. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4085–4092, 2016.

[73] Justyna Sarzynska-Wawer, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135, 2021.

[74] Roger C Schank. Conceptual dependency: A theory of natural language understanding. *Cognitive psychology*, 3(4):552–631, 1972.

[75] BP Sharma. Nuclear reactors: Moderator and reflector materials. *Encyclopedia of Materials: Science and Technology*, pages 6365–6369, 2001.

[76] Catarina Silva and Bernardete Ribeiro. The importance of stop word removal on recall values in text categorization. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pages 1661–1666. IEEE, 2003.

[77] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems*, 33:16857–16867, 2020.

[78] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[79] Josef Steinberger and Karel Ježek. Text summarization and singular value decomposition. In *Advances in Information Systems: Third International Conference, ADVIS 2004, Izmir, Turkey, October 20-22, 2004. Proceedings 3*, pages 245–254. Springer, 2005.

[80] Chuan-Jun Su and Shi-Feng Huang. Real-time big data analytics for hard disk drive predictive maintenance. *Computers & Electrical Engineering*, 71:93–101, 2018.

[81] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[82] Gian Antonio Susto and Alessandro Beghi. Dealing with time-series data in predictive maintenance problems. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2016.

[83] Gian Antonio Susto, Alessandro Beghi, and Cristina De Luca. A predictive maintenance system for epitaxy processes based on filtering and prediction techniques. *IEEE Transactions on Semiconductor Manufacturing*, 25(4):638–649, 2012.

[84] Ayisha Tabassum and Rajendra R Patil. A survey on text pre-processing & feature extraction techniques in natural language processing. *International Research Journal of Engineering and Technology (IRJET)*, 7(06):4864–4867, 2020.

[85] Wilson L Taylor. "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

[86] JWG Thomason. The isis spallation neutron and muon source—the first thirty-three years. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 917:61–67, 2019.

[87] Madeleine Udell, Corinne Horn, Reza Zadeh, Stephen Boyd, et al. Generalized low rank models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.

[88] Juan Pablo Usuga-Cadavid, Samir Lamouri, Bernard Grabot, and Arnaud Fortin. Using deep learning to value free-form text data for predictive maintenance. *International Journal of Production Research*, 60(14):4548–4575, 2022.

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[90] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *International work-conference on artificial neural networks*, pages 758–770. Springer, 2005.

[91] Geert Waeyenbergh and Liliane Pintelon. A framework for maintenance concept development. *International journal of production economics*, 77(3):299–313, 2002.

[92] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[93] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*, 2023.

[94] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*, 2024.

[95] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *arXiv preprint arXiv:2304.11127*, 2023.

[96] Stephen Willard. *General topology*. Courier Corporation, 2012.

[97] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[98] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, C Li, D Liu, F Huang, et al. Qwen2 technical report. arxiv 2024. *arXiv preprint arXiv:2407.10671*, 2024.

[99] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[100] Minghe Yu, Guoliang Li, Dong Deng, and Jianhua Feng. String similarity search and join: a survey. *Frontiers of Computer Science*, 10:399–417, 2016.

[101] Zhe Zhang, Junxi Zhang, and Huifeng Xue. Improved k-means clustering algorithm. In *2008 Congress on image and signal processing*, volume 5, pages 169–172. IEEE, 2008.

[102] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.

Appendix A

Supplementary Technical Details

A.1 BERT and XLNet Pre-training

This section provides supplementary technical details on the pre-training objectives and mechanisms of the BERT and XLNet models, as discussed conceptually in Section 2.2.1.

A.1.1 BERT Pre-training Tasks

The BERT model [22] employs two pre-training objectives during its pre-training phase:

- Masked Language Modelling - During training, a portion of the input tokens are 'masked' and the model predicts these masked tokens based on the remaining context, using cross-entropy loss [102]. In the literature, this is referred to as a Cloze task [85]. In the experiments conducted by (Devlin et al., 2019) 15% of tokens in each sequence are randomly selected for masking. To mitigate the mismatch between pre-training and fine-tuning phases where the `[MASK]` token does not appear, the training data generator replaces each chosen masked token with: (1) the special token `[MASK]` 80% of the time; (2) a random token 10% of the time; and (3) the original token itself 10% of the time [22].
- Next Sentence Prediction - In this task, the model is trained on a binary classification task predicting the relationship between sentence pairs. Given two sentences `A` and `B` either (1) sentence `B` is the actual sentence that follows `A` 50% of the time or (2) sentence `B` is randomly selected from the corpus 50% of the time [22]. This task aimed to help BERT understand sentence relationships, though later models like RoBERTa and MPNet found it might not always be beneficial [49, 77].

Figure A.1 illustrates the overall pre-training and fine-tuning workflow for BERT.

A.1.2 XLNet Technical Details

XLNet [99] introduced mechanisms to overcome limitations observed in BERT, particularly concerning the pre-train-to-fine-tune discrepancy and the independence assumption in MLM.

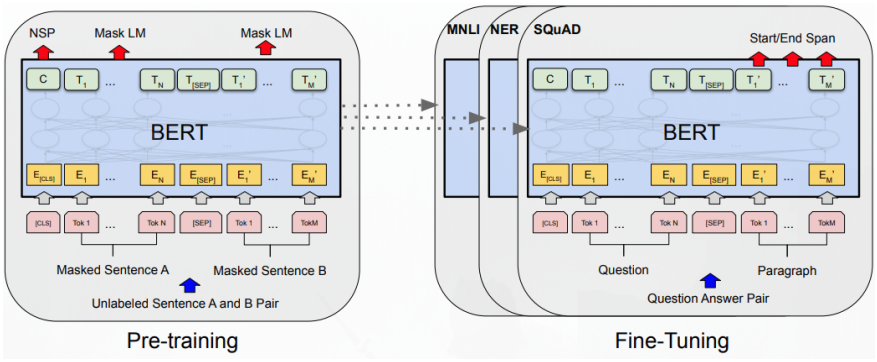


Figure A.1: The overall pre-training and fine-tuning phases for BERT. For different downstream tasks, notice the same architecture, except output layer, is used. Source: (Devlin et al., 2019). (Figure originally appeared as Figure 2.2)

Auto-regressive vs. Auto-encoding Perspectives

According to (Yang et al., 2019), pre-training objectives can be broadly categorized as auto-regressive (AR) or auto-encoding (AE). AR language modelling, given a sequence of tokens $X = (x_1, ..., x_T)$, aims to factorize the likelihood of a token into either a forward or backward product:

$$p(x) = \prod_{i=1}^{T} p(x_i | X_{<i}) \tag{A.1}$$

$$p(x) = \prod_{i=T}^{i} p(x_i | X_{>i}) \tag{A.2}$$

where $X_{<i}$ denotes tokens before position i. As standard AR models only encode context unidirectionally, their effectiveness can be limited [99]. AE models like BERT reconstruct original data from corrupted input (i.e. masked tokens), allowing bidirectional context but potentially suffering from issues like the pretrain-finetune discrepancy and independence assumptions [99].

Permutation Language Modeling Formalism

XLNet employs Permutation Language Modelling as an AR objective that captures bidirectional context. It maximizes the expected log-likelihood over all possible factorisation orders (permutations) of the input sequence [99]. More formally:

$$\max_{\theta} \ \mathbb{E}_{z \in \mathcal{Z}_T} \Big[\sum_{i=1}^{T} \log p_\theta \left(x_{z_i} \mid x_{z_{<i}} \right) \Big], \tag{A.3}$$

where $\mathcal{Z}_\mathcal{T}$ is the set of all $T!$ permutations of the index list $[1, ..., T]$. For a given permutation $z = (z_1, ..., z_T)$, $X_{z_{<i}}$ denotes the tokens appearing before z_i in that permutation. θ represents the shared model parameters across all permutations [99]. By training across all permutations,

each position learns to utilise context from all other positions.

Two-Stream Self-Attention Mechanism

To implement permutation language modelling while preventing the model from seeing the token it needs to predict, XLNet uses a two-stream self-attention mechanism [89, 99]. It maintains two hidden states per position at each layer:

1. Content Stream (h) - Encodes both the content and position of a token, similar to standard transformer hidden states. It can access contextual information up to and including the current position in the permutation.

2. Query Stream (g) - Encodes contextual information and the current position z_i, but, importantly, not the content x_{z_i} of the token at the current position. It is used to gather information from the context $x_{z_{<i}}$ to predict x_{z_i}.

At layer m and target position z_i, the streams are updated using attention mechanisms, conceptually as follows [89, 99]:

$$g_{z_i}^{(m)} \leftarrow \text{Attention}(Q = g_{z_i}^{(m-1)}, KV = h_{z_{<i}}^{(m-1)}) \tag{A.4}$$

$$h_{z_i}^{(m)} \leftarrow \text{Attention}(Q = h_{z_i}^{(m-1)}, KV = h_{z_{\leq i}}^{(m-1)}) \tag{A.5}$$

The query stream $g_{z_i}^{(m)}$ attends only to the content representations h of tokens preceding z_i in the current permutation. The content stream $h_{z_i}^{(m)}$ calculates an attention distribution over tokens up to and including z_i. After the final layer M, the query stream representation $g_{(z_i)}^{(M)}$ is used to compute the probability distribution over the vocabulary for predicting x_{z_i} via a soft-max layer [26, 99]. This mechanism allows XLNet to predict tokens using contextual information gathered auto-regressively based on the permutation, without directly accessing the target token's content representation.

A.2 MPNet Mathematical Formulations

This section provides supplementary mathematical details for the MPNet model discussed in Section **??**.

A.2.1 MPNet Pre-training Objective

The pre-training objective for MPNet aims to maximize the expected log-likelihood of predicting tokens based on a permuted context that includes information about masked tokens later in the sequence. Formally, let $\mathcal{Z}_T$ be the set of all $T!$ permutations of the index list $[1, ..., T]$. For a given permutation $z = (z_1, ..., z_T)$, let x_{z_i} denote the token at original position z_i, and $x_{z<i}$ denote the sequence of tokens preceding z_i in the permutation z. Let M be the set of indices corresponding to masked tokens, and $M_{z>c}$ be the subset of masked token indices appearing after position c in the permutation z. The objective is defined as [77]:

$$\max_{\theta} \mathbb{E}_{z \in \mathcal{Z}_T} \left[\sum_{i=1}^{T} \log p_\theta(x_{z_i} \mid x_{z<i}, M_{z>c}) \right] \tag{A.6}$$

where θ represents the shared model parameters, fixed for each permutation z.

A.2.2 MPNet Input Structuring Example

To implement the objective (Equation A.6), the input sequence is rearranged based on the permutation z and a split point c. The sequence is divided into non-predicted tokens ($x_{z_{\leq c}}$) and predicted tokens ($x_{z_{>c}}$). The input representation fed to the model concatenates the non-predicted part, mask token placeholders corresponding to the predicted part ($M_{z>c}$), and the actual predicted tokens (used only for loss calculation). For an example input sequence $(x_1, ..., x_6)$, a permutation $z = (x_3, x_5, x_2, x_1, x_4, x_6)$, and a split point $c = 3$, the structured input is represented as [77]:

$$\langle x_{z_{\leq c}}; M_{z>c}; x_{z_{>c}} \rangle = \langle x_3, x_5, x_2; [M], [M], [M]; x_1, x_4, x_6 \rangle \tag{A.7}$$

where $[M]$ denotes the mask token placeholder.

A.2.3 PCA Formalism and SVD Derivation

This appendix provides the formal mathematical details underpinning the conceptual description of Principal Component Analysis (PCA) presented in Section 2.3.1, including its optimisation objective and connection to Singular Value Decomposition (SVD) [79].

If we want to reduce the dimensionality from n features down to k principal components (for $k < n$), we seek a matrix $C \in \mathbb{R}^{n \times k}$ whose columns represent these top k orthogonal component directions (i.e. columns are statistically uncorrelated). The projected data in the lower k-dimensional space is given by XC. We can approximate the original data X by projecting back from this lower dimension using XCC^T. PCA aims to find the matrix C that minimises the difference between the original data X and this reconstructed data XCC^T, subject to the constraint that the column-wise components are orthogonal ($C^TC = I$, for the identity matrix I) [87]:

$$\min_{C} \|X - XCC^T\|_F^2 \tag{A.8}$$

A standard and efficient method for solving this optimisation problem and finding the principal components is through the Singular Value Decomposition (SVD) of the data matrix X [79, 87]. SVD is a fundamental matrix factorisation technique that decomposes X into three matrices:

$$X = USV^T \tag{A.9}$$

Here, $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$ are matrices with orthogonal columns, $S \in \mathbb{R}^{k \times k}$ is a diag-

onal matrix (only the diagonal contains non-zero elements) which contains the non-negative singular values (s_i) sorted in descending order, and k is the rank of matrix X.

The crucial link to PCA is that the columns of the matrix V are precisely the principal component directions we seek. That is, the matrix C in Equation A.8, containing the top k principal components, corresponds to the first k columns of V.

Furthermore, SVD connects directly to the covariance matrix of the data ($X^T X$, assuming X is centered). The decomposition of the covariance matrix is given by:

$$X^T X = (USV^T)^T (USV^T) \tag{A.10}$$
$$= (VS^T U^T)(USV^T) \tag{A.11}$$
$$= VS^T (U^T U) SV^T \tag{A.12}$$
$$= VS^T ISV^T \quad (\text{since } U^T U = I) \tag{A.13}$$
$$= V(S^T S)V^T \tag{A.14}$$
$$= VS^2 V^T \quad (\text{since } S \text{ is a diagonal matrix}) \tag{A.15}$$

This is the eigen-decomposition of the covariance matrix $X^T X$. It shows that the columns of V are the eigenvectors (the principal components), and the diagonal entries of S^2 are the corresponding eigenvalues (i.e. $s_i^2 = \lambda_i$). Since the eigenvalues represent the variance of the data along the eigenvectors, the squared singular values (s_i^2) directly quantify the variance captured by each principal component [87]. Therefore, computing the SVD of the data matrix X provides both the principal components (in V) and the measure of variance associated with each component (in S^2) without explicitly forming the covariance matrix. Using a truncated SVD (keeping only the top k components corresponding to the largest singular values) gives us a reduction in the dimensions.

A.3 UMAP Hyperparameter Search

Figures A.2 through A.7 in this appendix illustrate the results of the naive grid search performed over the UMAP hyperparameters listed in Table 3.5, applied to both MPNet and Nomic embeddings of the Operalog `FaultDescription` entries. These visualisations demonstrate the effects of varying parameters like `n_neighbors`, `min_dist` and `metric` as described conceptually in Section 2.3.2. Notably, visual inspection suggests that, for the goal of creating a low-dimensional representation that isolates structural groupings in this specific dataset, the 'cosine' distance metric did not produce significantly different or clearly superior results compared to the 'euclidean' metric. This outcome is somewhat surprising as cosine distance is frequently recommended for text embedding applications in the literature [16, 8]. This observation, combined with the difficulty in determining the optimal settings purely through visual inspection of numerous plots, makes it evident that this naive grid search approach is insufficient. It therefore motivates the need for the systematic hyperparameter optimisation process detailed in Section 3.7.

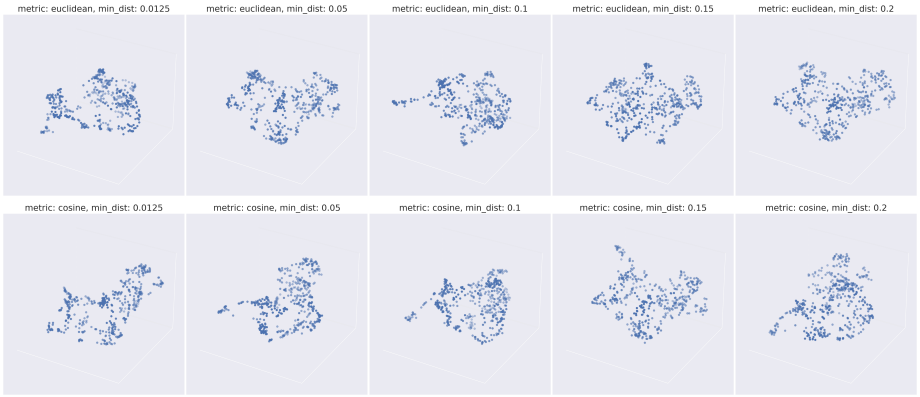


Figure A.2: MPNet, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `metric` and `min_dist`

Figure A.3: MPNet, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `min_dist` and `n_neighbors`

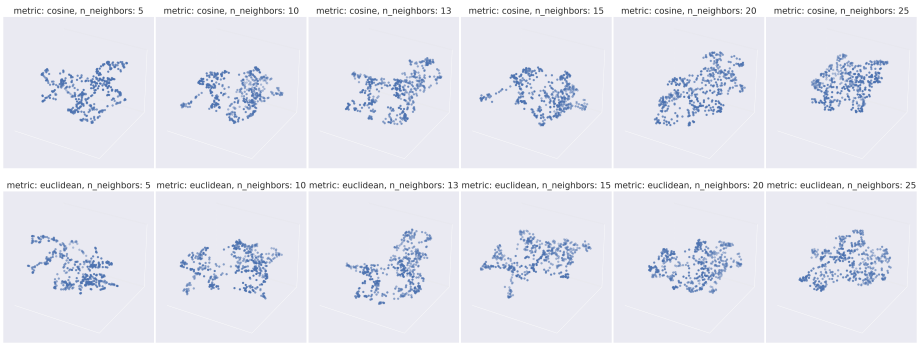


Figure A.4: MPNet, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `metric` and `n_neighbors`

Figure A.5: Nomic, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `metric` and `min_dist`

Figure A.6: Nomic, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `min_dist` and `n_neighbors`

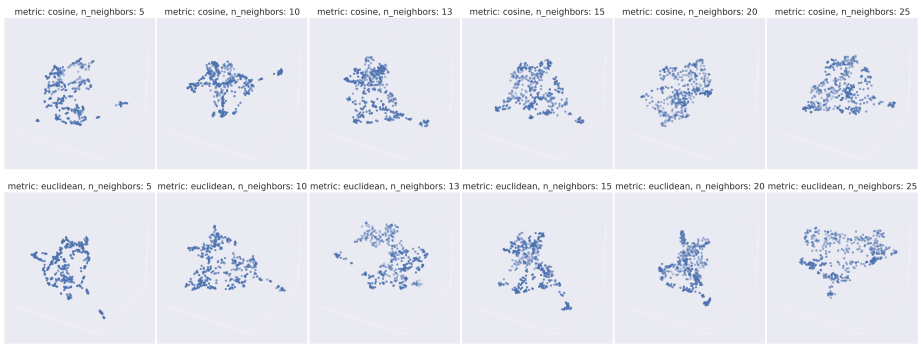


Figure A.7: Nomic, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `metric` and `n_neighbors`

A.4 k-Medoids - The PAM Algorithm

This section provides a high-level pseudocode of the Partitioning Around Medoids (PAM) algorithm, the core algorithm of the k-Medoids clustering method, as discussed in Section 2.4.1 [37].

Algorithm A.1: Partitioning Around Medoids (PAM).

```
input: Dataset X = {x_1, ..., x_n}, integer k (number of clusters)
output: Set of k medoids M, Partition X' = {C_1, ..., C_k} where C_j is the set of points
      assigned to medoid m_j

begin
    # --- Build Phase (Conceptual) ---
    # Select an initial set of k medoids M from X
    # (i.e. using a heuristic that minimises initial total dissimilarity)
    M ← SelectInitialMedoids(X, k)

    # --- Swap Phase ---
    repeat
        best_cost_reduction ← 0
        best_swap ← ∅
        total_cost ← ComputeTotalDissimilarity(X, M) # Cost based on current M

        # Consider swapping each medoid m with each non-medoid x
        foreach m ∈ M do
            foreach x ∈ X \ M do
                # Compute cost if m is replaced by x
                M_swap ← (M \ {m}) ∪ {x}
                cost_after_swap ← ComputeTotalDissimilarity(X, M_swap)
                cost_reduction ← total_cost - cost_after_swap

                # Check if this swap is the best found so far
                if cost_reduction > best_cost_reduction then
                    best_cost_reduction ← cost_reduction
                    best_swap ← (m, x) # Store the potential swap pair
                end
            end
        end

        # Perform the best swap found in this iteration, if any cost reduction
            occurred
        if best_cost_reduction > 0 then
            (m_out, x_in) ← best_swap
            M ← (M \ {m_out}) ∪ {x_in} # Update medoid set
            improved ← true
        else
            improved ← false # No cost reduction possible
        end
    until not improved # Stop when no swap improves the total cost

    # Assign points to final medoids to form clusters X'
    X' ← AssignPointsToClusters(X, M)

    return M, X'
end
```