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.
MST	Minimum Spanning Tree
MRL	Matryoshka Representation Learning.
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 Overview of the ISIS Research Facility

The STFC Rutherford Appleton Laboratory's ISIS Neutron and Muon research facility is a research centre for physical and life sciences. It is owned and operated by the STFC, a council within the United Kingdom Research and Innovation (UKRI). The UKRI is a public body of the United Kingdom's Government that directs funding for research and innovation through the science budget of the Department for Science, Innovation and Technology. ISIS was designed in the 1970s and early 1980s, with the core of the design being a strong-focusing machine with six radio frequency accelerating cavities to provide an average beam current of $200\mu A$ [81]. According to (Thomason, 2019), with the introduction of ISIS and other neutron sources, rapid development of neutron instrumentation was stimulated. Over the years, the facility has been augmented with numerous components and instruments, such as the second target station. The current schematic representation of the facility can be seen in Figure 1.1.

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

The ISIS facility, described in A Practical Guide to the ISIS Neutron and Muon Source (ISIS Neutron and Muon Source, 2021), is comprised of four major stages: (1) the ion source, (2) the radio-frequency quadrupole (RFQ), (3) linear accelerator (LINAC) and (4) the synchrotron. These components combine to produce a 800 MeV proton beam that is directed to either target station 1 (TS-1) or target station 2 (TS-2). Starting with H^- ions (protons with two orbiting electrons), the path taken through the four stages is shown in Figure 1.2. Throughout the entire H^- or proton acceleration process, the beam must be kept under tight vacuum and, to do so, many tens of vacuum pumps are maintaining the vacuum between 10^{-8} and 10^{-9} of atmospheric pressure. Maintaining this vacuum integrity is a constant operational challenge as leaks may disrupt the beam and potentially damage sensitive components, necessitating numerous pumps and continuous monitoring system [31]. The rest of this section explains the various stages in a more detail with the purpose of highlighting the complex infrastructure in place and critical control required for the reliable operation of the facility.

The first stage of the ISIS machine is the ion source, which generates the negative hydrogen (H^-) ions, which are then accelerated through the RFQ and LINAC. This ion source is a pulsed source that ionises hydrogen gas via an electric discharge between an internal anode and cathode [31]. As detailed in the guide, around 20ml of hydrogen gas per minute is continuously delivered to the ion source from a hydrogen gas bottle. Once the H^- ions emerge from the ion source, they have an energy of 35 keV and are presented to the RFQ via the Low Energy

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

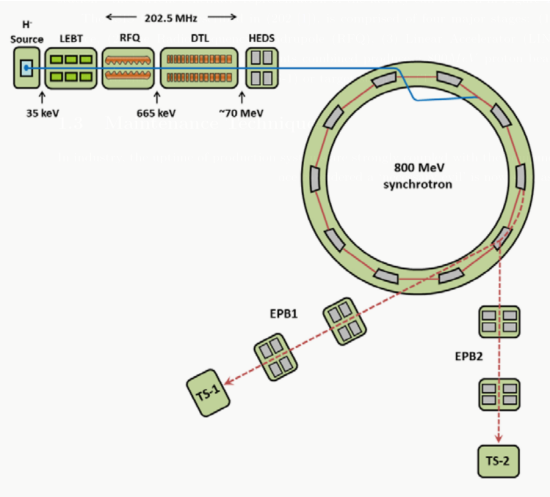


Figure 1.2: ISIS ion source and chain of accelerators, with H^- ions in blue and protons in red. Not to scale. Source: (ISIS Neutron and Muon Source, 2021).

Beam Transport line (LEBT). The LEBT prevents the low-energy H^- beam from increasing in size (due to the mutual repulsion of the ions) and also incorporates a beam stop. This is essentially a remotely removable sheet of metal that physically blocks the beam. Precise control and monitoring of the beam intensity are required here since inconsistencies originating from the ion source affect the overall performance of the machine. This is because these issues can propagate and amplify through the subsequent stages. Additionally, the failure of the source itself, for example due to component wear, requires immediate intervention and is a common maintenance task [31].

Following the ion source and LEBT, the H^- beam enters the RFQ. This section of the machine utilises high-intensity radio frequency electric fields that contain (focus), group (bunch) and begin accelerating the particles in this beam up to an energy of 665 keV. A key function of the RFQ is to isolate downstream accelerators from variations originating in the ion source, thereby improving beam stability in the LINAC, synchrotron and target stations [31]. However, the stability of the RFQ's own high-intensity beam is a primary concern as, again, fluctuations can lead to poorly formed bunches.

The subsequent stage is the LINAC, which increases the beam energy from 665 keV to 70 MeV. This acceleration is achieved via high-intensity radio-frequency fields arranged in structures that possess a very high Q factor [50] (around 50,000). This means their geometry must be exceptionally stable. Therefore the precise temperature regulation of the cooling water tank is critical in ensuring it is significantly under $1°C$ so that the effects of thermal expansion and contraction are countered [31]. Furthermore, maintaining a high vacuum is critical throughout the LINAC. Failure to do so can lead to premature stripping of loosely bound electrons in the H^- ions, consequently leading to an increase in radioactivity and particle loss (i.e. beam loss). To detect and warn about such cases, beam loss monitors are installed throughout the LINAC, which provide an indication whenever there is an excessive

loss in particles. [31].

Once the H^- particles leave the LINAC, they then pass through a beam transport line - the High Energy Drift Space (HEDS). The function of this is to mainly reduce the particles spreading and focus the beam to the synchrotron. To accumulate the high intensity beam in the synchrotron, ISIS employs charge-exchange injection. This technique allows incoming H^- beam from the LINAC to be 'layered' into the synchrotron's circumference over many turns. As the H^- ions enter the synchrotron ring, they pass through a thin foil which strips away their two electrons, converting them to protons. The charge-exchange injection process is specifically chosen as it is the most efficient way of taking and wrapping a long string of particles around the circumference of a synchrotron [31, 6]. The integrity of this thin foil is vital for efficient injection and represents a consumable component requiring periodic replacement, which is a scheduled maintenance activity. Damage to the foil will lead directly to beam loss and reduced performance.

The 70 MeV beam is then injected into the synchrotron, which is the primary accelerator ring at ISIS. This ring boosts the (now proton) beam energy to 800 MeV. For an idea of scale, the machine has a circumference of around 163m (corresponding to a radius of approximately 26m). It features a repeating structure of ten 'superperiods'. Each superperoid uses dipole magnets to bend the proton path by 36° and quadrupole magnets to maintain beam focus [31]. The protons gain energy over just under 8000 revolutions, as each turn gives a proton 0.1 MeV. Achieving this final energy requires extreme precision over each and every revolution. While the protons gain energy, the magnetic fields and accelerating radio-frequency voltages must also very precisely be increased in synchronism. Any deviations can cause the beam to oscillate and strike the vacuum vessel walls which results in particle loss and potential component damage. Therefore, continuous precise monitoring of beam position, magnet currents and more via a network of sensors is imperative for controlling the beam orbit and ensuring successful acceleration [31]. The initially continuous ring of injected protons is gathered into two distinct bunches, a little more than $0.1\mu s$ apart, using a low-level radio-frequency voltage before the main acceleration begins [31].

Once accelerated, in order to direct the proton bunches towards a pre-selected target station (TS-1 or TS-2) three kicker magnets are used which deflect the bunches upwards. These bunches are then bent into the respective extracted proton beam lines (EPB1 or EPB2). These lines use further magnets for steering. Safe transport relies on accurate magnet performance as mis-steering could direct the high-energy beam into the pipe walls. Additionally, as a safety precaution, both lines are heavily shielded in thick steel and concrete and beam loss monitors are installed along their length to detect any errant particles and provide crucial, real-time feedback to operators [31].

The primary purpose of the ISIS facility culminates at the target stations. The high-energy proton beam collides with a dense, high atomic number target (tantalum-clad tungsten at ISIS), designed to produce a large number of neutrons via spallation, within a compact region of space [31]. These energetic neutrons are then slowed down (moderated) by surrounding materials (liquid nitrogen, liquid hydrogen and water at ISIS) and reflected back towards instruments by a beryllium reflector [70, 31]. Additionally, ISIS produces muons for experiments by inserting a thin (roughly 1cm thick) graphite target into the proton beam around 20m away from the TS-1 neutron target. Collision in this target generates pions,

which subsequently decay into muons [31]. This entire system needs extremely careful control and monitoring capabilities, as failure due to overheating or material damage is a major operational concern which requires careful management and represents a major maintenance undertaking. Precise monitoring is required of metrics such as incoming beam profile, target temperature, cryogenic temperatures.

This detailed description sourced from the guide highlights some key takeaways. There are an incredible number of points of failure in the facility due to the highly precise and radioactive nature of the task. The ISIS team has taken many precautions such as fault detection mechanisms (i.e. the beam loss monitors) and breakdown counter-measures (i.e. the thick layer of steel and concrete in the EPB1 and EPB2 lines). However, the inherent complexity and harsh operating environment (radiation, high-power) necessitate both robust preventative maintenance schedules and sophisticated diagnostic capabilities that address the component degradation and failures that will inevitably arise.

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 [81]. 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 [81]. 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 [81]. 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.

Add the graph, if you can ever figure out the operating cycles before 2016.

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 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.

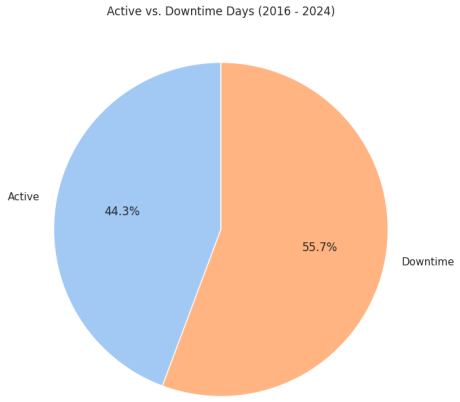


Figure 1.3: A visualisation of the machine downtime as opposed to time operating, according to the ISIS operational cycle. Data source: (ISIS Neutron and Muon Source, 2024)

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 [86, 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 [78, 53, 77]. 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 approach and typically the most costly method as it reduces the facility's availability and requires a complete replacement of parts.

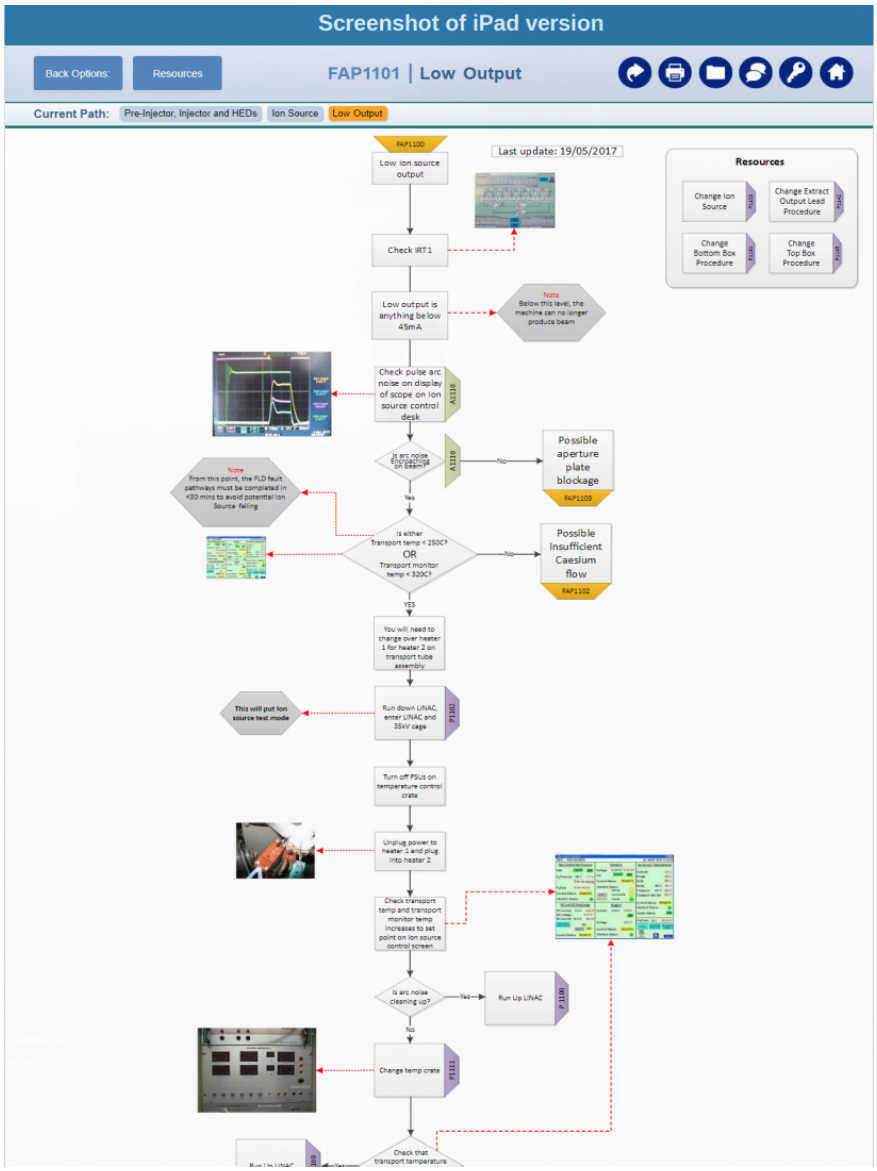


Figure 1.4: Example FAP1101, screenshotted from an iPAD version of FLD version 2.2, which shows the FAP for pre-injector, injector and High Energy Drift Space. Source: (Asim Yaqoob, 2017)

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 [53]. 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 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 [81].

In industry, many maintenance strategies prefer using PdM whilst experimenting with a variety of statistical inference and artificial intelligence modelling approaches [53, 34]. 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 both third-generation model BERT [22] and the more recent fourth-generation universal-embedding architecture XLNet [94]. 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 [62, 30] and UMAP [49], 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 []) and two density-based (DBSCAN [24] and HDBSCAN [15])—together with four clustering-evaluation metrics: (1) Inertia [], (2) Silhouette [66], (3) Davies–Bouldin Index [21], and (4) Calinski–Harabasz Index [14].

Fill this

find the reference for this

maybe talk about optuna, if we use it.

double check all citations here are not empty

: Describe the various technical factors required before attempting to understand the methodology. The things in this section will include

- Discuss sentence embedding, similarity measures: BERT, RoBERTA, MPNet, XLNet, NOMIC.
- Dimensional reduction techniques and need for them (UMAP, PCA, t-SNE).
- Clustering methods: kmedoids, DBSCAN, DBSCAN*/HDBCAN
- Clustering evaluation methods: todo I don't remember these off the top of my head
- Maybe briefly touch on Optuna?

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 [65]. Examples of the use-cases of sentence similarity include: detection of academic malpractice via plagiarism [47, 8], and text summarisation [5, 38, 35]. According to [65], 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 [95].
- 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 [40] and SVD [74] as well as word embedding models (utilising ML), such as Word2Vec [11], GloVe [63] and fastText [51].

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 [79]; 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 [67]. 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 [84] 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 [68], GPT [64] 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 [41], Multilingual e5 text embeddings [89], Nomic [57] and many more.

Second-, third- and fourth-generation text embedding models are used frequently in PdM for applications such as insight extraction [3, 83] 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 [56].

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 [45].

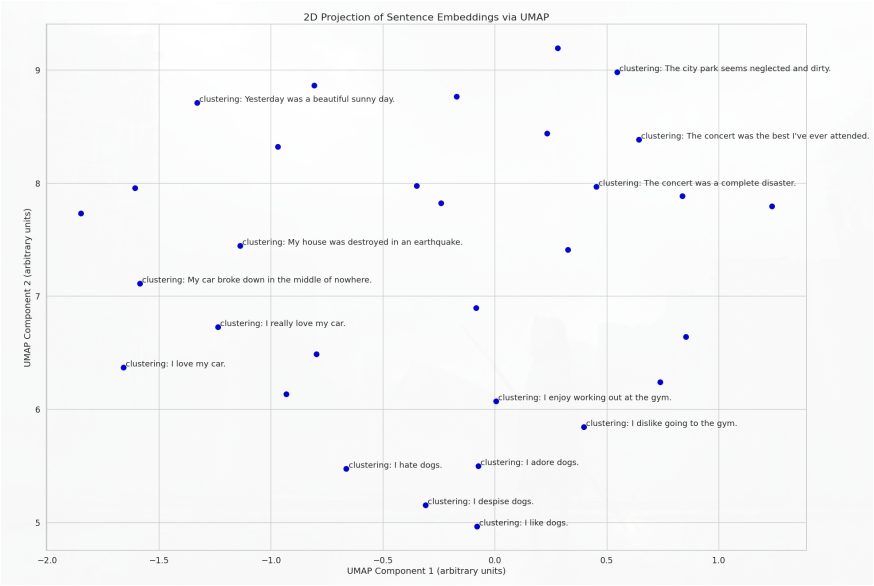


Figure 2.1: Example text embedding with Nomic Text Embedding v1.5 [57] of random sentences generated using OpenAI's ChatGPT o3-mini-high and projected onto 2 dimensions using UMAP [49] with a random seed of 42 and minimum neighbours of 15. Only some of the labels are highlighted for visual clarity.

2.2.1 Transformers

Pre-trained language models (PLMs), such as BERT [22], have been widely successful in a wide range of NLP tasks through fine-tuning [23, 52]. Fine-tuning is the process of re-training a PLM on specialised tasks, leveraging the model's base knowledge, by applying perturbations to the pre-trained model parameters through gradient descent learning algorithms. These language models, trained on finding an embedding space for natural language as well as stochastically generating tokens to mimic natural language, often provide a great platform to perform task-specific model fine-tuning. Platforms such as HuggingFace [91], allow authors to upload these pre-trained model parameters which in-turn allows researchers to download them for them fine-tuning. Moreover, task-specific fine-tuned models parameters are uploaded, downloaded and shared on these platforms. These fine-tuned models are useful for researchers whose focus lies outside of optimising these model parameters. In this section, we talk about the transformer-family of language models.

fix wording

BERT

The Bidirectional Encoder Representations from Transformer (BERT) language model introduced in 2018 by Google AI Language team in (Devlin et al., 2019) was designed to learn deep bidirectional representations from unlabelled text. It achieved this by learning the left and right context in every layer of the model. The model implementation can be split into two phases: (1) the pre-training phase; and (2) the fine-tuning phase. This allows the model archi-

sprinkle some images in here.

tecture to remain common, with many down-stream NLP tasks benefiting from a single PLM [22]. An example illustration can be seen in Figure 2.2, where a single model can be fine-tuned on many down-stream tasks by simply replacing the output layer. The model architecture is a multi-layer transformer encoder based on the original paper (Vaswani et al., 2017) which is bidirectional by nature. As the transformer architecture is a very well researched architecture, well represented in the literature and slightly out-of-scope for this paper, we refer the reader to the original paper.

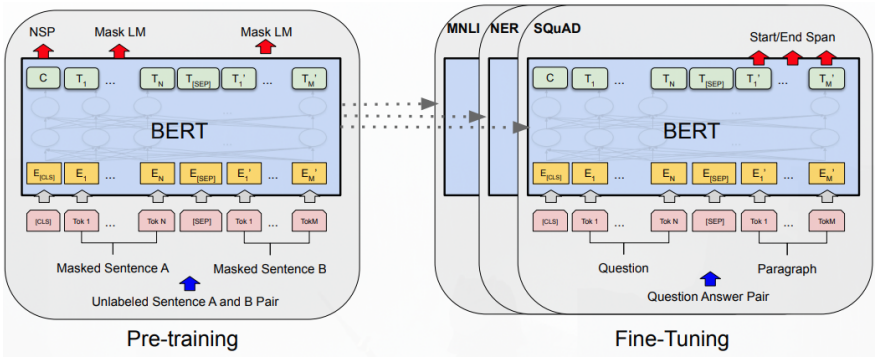


Figure 2.2: The overall pre-training and fine-tuning phases for BERT. For different down-stream tasks, notice the same architecture, except output layer, is used. Source: (Devlin et al., 2019).

The pre-training phase can be split into two further training tasks (otherwise referred to as objectives):

- 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 [96]. In the literature, this is referred to as a Cloze task [80]. In the experiments conducted by (Devlin et al., 2019) 15% of tokens in each sequence are randomly selected for masking. However, to mitigate the mismatch between pre-training and fine-tuning phases, a training data generator replaces each chosen masked tokens 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 prediction 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].

The Google AI Language team further fine-tune BERT across 11 NLP tasks, showing that BERT out-performed the state-of-the-art models at its point in time.

As previously mentioned, a benefit of using PLMs such as BERT for NLP tasks (such as text embedding) is the ability to off-load the computationally intensive pre-training and

any initial fine-tuning stages to another party. Although, this means that there now exists an implicit trust in that third-party's model parameters. For example, the third-party could be motivated by factors to censor certain phrases, artificially injecting testing data into training to increase performance metrics or maliciously change the embedding space. These are all factors which influence our decision on using a PLM and, thus, PLMs with transparently documented data sources, pre-training and fine-tuning should be preferred.

Since the release of the Google AI Language team's BERT, there have been some developments on further improving the performance of PLMs based on techniques presented in the paper. Namely, we touch on two: XLNet [94] and MPNet [72]. XLNet's architecture is backed by the Transfomer-XL model [19]. However, it is still briefly covered here to motivate MPNet. MPNet is a hybrid model that inherits XLNet's key feature, permutation language modelling, and BERT's masked language modelling, producing a 'unified view'.

XLNet

According to (Yang et al., 2019), the most successful pre-training objectives can broadly be split into two categories: (1) auto-regressive (AR) and auto-encoding (AE). AR language modelling, given a sequence of tokens $X = (x_1, ..., x_T)$, aims to factorize likelihood of a token into either a forward (Equation 2.1) or backward (Equation 2.2) product. As an AR language model encodes context in a singular direction (only forward or backward, but not both), it has been shown to not be effective [94]. The notation $X_{<i}$ for some $1 \leq i \leq T$ means to include the first $i-1$ elements (vice versa for $X_{>i}$).

$$p(x) = \prod_{i=1}^{T} p(x_i | X_{<i}) \tag{2.1}$$

$$p(x) = \prod_{i=T}^{i} p(x_i | X_{>i}) \tag{2.2}$$

AE language modelling, such as BERT, on the other hand aims to construct the original data from masked or malformed input, without estimations of the probability densities [94]. One downside of BERT, is the discrepancy between the pre-training and fine-tuning phases. Although BERT utilises bidirectional context for reconstruction of the special `[MASK]` token during pre-training, this special token does not exist at the fine-tuning phase. Additionally, the predicted tokens are masked in the *input* and BERT assumes the predicted tokens are independent to the surrounding context as it is not able to model joint probabilities [94]. This assumption is an gross oversimplification, as longer term dependencies are highly prevalent in natural language applications [69, 19].

Therefore, XLNet proposes an AR language modelling solution, backed by Transfomer-XL architecture [19], that combines the benefits of both AR and AE language modelling whilst avoiding their downsides. It does this by introducing two methods, described below.

- Permutation language modelling - XLNet maximises the pre-training objective, the expected log-likelihood over all possible factorisation orders of a length-T sequence. A

factorisation order is simply one way of shuffling the token positions $1, ..., T$. Imagine picking a shuffle then walking through the sentence in that order - when we come to a token, we predict it only using the tokens that appeared earlier in the very same shuffle even if those tokens lie to its right in the original sequence. Since this procedure is repeated for all shuffles, every token eventually 'sees' information from both sides of the input, which gives XLNet a BERT-like bidirectional context [94]. More formally,

$$\max_{\theta} \ \mathbb{E}_{z \in \mathcal{Z}_T} \Big[\sum_{i=1}^{T} \log p_\theta \left(x_{z_i} \mid x_{z_{<i}} \right) \Big], \tag{2.3}$$

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 that appear before z_i in that permutation. θ denotes the model parameters, and is shared across all orders [94].

- Two-stream self-attention - A naive transformer would expose the model to the very token it is trying to predict. XLNet avoids this by splitting the hidden state at every layer into (1) a content stream h, which encodes both content and position for every input token, and (2) a query stream g which is target-aware. That is, the query stream knows where in the sequence a prediction is to be made, through its positional embedding but never sees the token's content [84, 94]. This helps the model use a different probability density depending on where in the text it is predicting, which BERT was lacking. At layer m and target position z_i, the update sequence for h and g look like the following, using the regular attention function from (Vaswani et al., 2017),

$$g_{z_i}^{(m)} \leftarrow \text{Attention}(Q = g_{z_i}^{(m-1)}, KV = h_{z_{<i}}^{(m-1)}) \tag{2.4}$$

$$h_{z_i}^{(m)} \leftarrow \text{Attention}(Q = h_{z_i}^{(m-1)}, KV = h_{z_{\leq i}}^{(m-1)}) \tag{2.5}$$

  The query-stream vector $g_{(z_i)}^{(M)}$, after M layers, is used to compute the soft-max [26] over some pre-determined vocabulary. This ensures the model predicts $x_{(z_i)}$ without ever having directly observed it. This is while the content stream continues to propagate contextual signals and positional signals. the use of two sets of hidden representations, the content representation and query representation. The content representation encodes both the context and the permutation $x_{(z_i)}$ whereas the query representation only encodes the position z_i and the contextual information $X_{(Z_{<i})}$ [94].

Through integration of permutation language modelling and two-stream self-attention, XLNet inherits the probability density estimation ability of AR models and bidirectional context of AE models. Additionally, it also side-steps BERT's pre-train and fine-tune discrepancy and independence assumption [94]. For further specific details on XLNet, the reader is referred to (Yang et al., 2019).

2.3 Dimensionality Reduction

Embedding spaces can get large, often on the scale of hundreds or even thousands of embedding dimensions. For example, the BERT base architecture uses an embedding dimension of 768 [22]. Therefore, in applications dealing with such embedding spaces, this motivates a technique that reduces the dimensions to 2- or 3-dimensions but reflects the high-dimensional characteristics of the dataset. Dimensionality reduction exploits the redundancy of the dataset by finding a smaller set of new features (variables) that are representative of the original data [73].

It has been shown that many high-dimensional spaces show 'counter-intuitive geometrical properties', impacting data analysis methods, such as clustering. This phenomenon is referred to as the 'curse of dimensionality' [85, 29]. Further, most clustering algorithms (such as those described in Section 2.4) have been shown to be inefficient in high-dimensional spaces, due to this curse [29]. Thus, in order to apply clustering (as we will, later in Chapter 3) dimensionality reduction is once again motivated.

This technique is not new in Statistics, with one of the most widely used dimensionality reduction techniques, Principal Component Analysis (PCA), introduced in (Pearson, 1901). However, with the increase in the quantity of available data and computational resources, researchers in fields such as Machine Learning and Statistics have developed a wide array of techniques aimed at tackling the dimensionality reduction problem [73].

In this section, we cover the background required in understanding techniques used in Chapter 3, Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP).

2.3.1 Principal Component Analysis

Principal Component Analysis (PCA), as mentioned before, was first introduced in (Pearson, 1901) but later independently developed and named in (Hotelling, 1933). PCA computes principal components, or variables which linearly decompose the features of the original dataset. The core idea is to find directions (the principal components) in the data that capture the maximum amount of variance, essentially identifying the axes along which the data spreads out the most [2]. Each subsequent principal component is computed to be orthogonal (that is, statistically uncorrelated) to the previous ones, ensuring that the new components capture different aspects of the data's variance.

This process can be described as an optimisation problem. Given a data matrix $X \in \mathbb{R}^{m \times n}$ (where m is the number of data samples and n is the number of original features), we want to find a new, lower-dimensional linear subspace, defined by the principal components, such that when the data X is projected onto this subspace, the variance of the projected data is maximised. Equivalently, this can be viewed as finding the linear subspace that minimises the reconstruction error, specifically, the sum of squared distances (least-squares distance) between the original data points and their projections onto the subspace [82].

More formally, 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. 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) [82]:

$$\min_C ||X - XCC^T||_F^2 \tag{2.6}$$

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 [74, 82]. SVD is a fundamental matrix factorisation technique that decomposes X into three matrices:

$$X = USV^T \tag{2.7}$$

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 diagonal 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 2.6, 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^TX, assuming X is centered). The decomposition of the covariance matrix is given by:

$$\begin{aligned} X^TX &= (USV^T)^T(USV^T) && (2.8)\\ &= (VS^TU^T)(USV^T) && (2.9)\\ &= VS^T(U^TU)SV^T && (2.10)\\ &= VS^TISV^T \quad (\text{since } U^TU = I) && (2.11)\\ &= V(S^TS)V^T && (2.12)\\ &= VS^2V^T \quad (\text{since } S \text{ is a diagonal matrix}) && (2.13) \end{aligned}$$

This is the eigen-decomposition of the covariance matrix X^TX. 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 [82]. 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.

2.3.2 Uniform Manifold Approximation and Projection

Unlike PCA, Uniform Manifold Approximation and Projection (UMAP) is generally considered a less directly interpretable dimensionality reduction method in terms of linear com-

ponents. UMAP is a non-linear, stochastic 'manifold learning technique' designed for dimensionality reduction, with theoretical foundations in Riemannian geometry and algebraic topology [49]. As it places no constraints on the dimensionality of the input space, it is viable as a general-purpose technique in machine learning, often used for visualisation or as a pre-processing step for other algorithms like clustering. For the reader interested in the rigorous mathematics underpinning these topics, we refer them to the following textbooks: (1) (Lee, 2003) for an introduction to differentiable and Riemannian manifolds; (2) (Lee, 2006) for Riemannian geometry in greater detail; and (3) (Willard, 2012) for fundamental concepts of general topology.

The UMAP algorithm operates under the assumption that the high-dimensional data (the input embeddings in our case) lies on or near a lower-dimensional manifold embedded within the high-dimensional space. The algorithm then attempts to find a low-dimensional representation (typically 2- or 3-dimensional) that best models the essential structure of this assumed manifold.

Conceptually, the UMAP algorithm achieves this in two main stages [49]:

1. High-dimensional graph construction - First, UMAP constructs a representation of the relationships between data points in the original high-dimensional space. It does this by finding the nearest neighbours for each point and constructing a weighted graph where edge weights represent the estimated probability or likelihood that two points are closely connected on the underlying manifold. This step focuses on capturing the local structure and connectivity of the data.

2. Low-dimensional layout optimization - Second, UMAP initialises the data points in the target low-dimensional space randomly. It then iteratively adjusts the positions of these low-dimensional points to create a graph structure that is as similar as possible to the high-dimensional graph constructed in the first stage. This optimisation process aims to minimise the difference (often measured using cross-entropy, a measure of divergence between probability distributions [96]) between the edge weights (probabilities of connection) in the high-dimensional and low-dimensional graphs. In other words, it tries to arrange the points in the low dimension such that connected points in the high dimension remain connected, and disconnected points remain disconnected.

The outcome is a low-dimensional embedding that aims to preserve both the local connectivity structure and, to some extent, the broader global structure of the data's manifold.

The output of the UMAP algorithm can be significantly influenced by several hyperparameters, which allow the user to tune the dimensionality reduction process [49]. The two most commonly adjusted parameters are:

- `n_neighbors`: Determines the size of the local neighbourhood UMAP considers when initially modelling the manifold structure in the high-dimensional space. It controls the balance between capturing local detail versus preserving the global structure of the data. Lower values focus UMAP on very local structure, potentially isolating small, distinct groups but risking the loss of broader relationships. Higher values encourage UMAP to consider more points as neighbours, leading to embeddings that better reflect

the overall global structure but may obscure fine-grained details or merge closely related clusters.

- `min_dist`: This parameter controls the minimum distance between points in the final low-dimensional embedding space. It primarily affects the visual density of the resulting plot. Lower values allow points to be packed very tightly together, emphasizing cluster separation and density. Higher values force points further apart, creating more dispersed embeddings where the structure within potential clusters might be more visible, but the clusters themselves may appear less distinct.

- `n_components`: This parameter specifies the target dimensionality of the reduced embedding space. It is commonly set to 2 or 3 for visualisation purposes, but can be set to higher values if the UMAP output is intended as input for subsequent machine learning tasks, such as clustering. Changing this directly alters the number of dimensions in the output array.

- `metric`: This parameter defines the distance metric used to measure similarity or dissimilarity between data points in the original high-dimensional input space when finding nearest neighbours. The default is typically 'euclidean', but other metrics like 'cosine' (cosine distance) or 'manhattan' (Manhattan distance) can be specified. The choice of metric is crucial and depends heavily on the nature of the input data. For instance, 'cosine' distance is often preferred for high-dimensional, sparse data such as text embeddings, as it measures orientation rather than magnitude [49]. Using an inappropriate metric can significantly distort the perceived local structure and thus the final embedding.

Selecting appropriate values for these parameters often involves experimentation based on the dataset and the specific goals of the dimensionality reduction (i.e. visualisation vs. input for clustering) [49]. Implementations of UMAP, such as the widely used `umap-learn` Python library, are noted for being computationally efficient and scalable relative to other non-linear techniques like t-distributed Stochastic Neighbour Embedding (t-SNE) [13], making UMAP a practical choice for many machine learning applications [49].

While powerful for visualisation and revealing non-linear structures, the resulting axes in the UMAP embedding do not have the same direct interpretation as the variance-ordered principal components derived from PCA. An example illustration of UMAP applied to an elephant skeletal structure, which is considered structured data, can be seen in Figure 2.3.

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 [37].

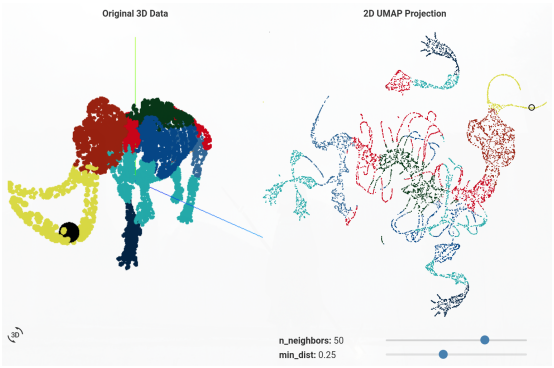


Figure 2.3: An illustration of UMAP applied (right) on samples taken from the skeletal structure of an elephant in 3-dimensions (left) with the UMAP hyperparameter values displayed. Highlighted, a black sphere (left) and circle (right), shows a chosen point mapped from 3-dimensions to 2. Source: (Coenen, Andy and Pearce, Adam, 2020).

This section provides the necessary background for the clustering techniques employed later in this project. We will cover k-Medoids [36], 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 [36].

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 seleceted 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 [36]. 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 [36].

Algorithm 2.1 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.

Algorithm 2.1: 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 minimizes 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

```

```
42      # Assign points to final medoids to form clusters X'
43      X' ← AssignPointsToClusters(X, M)
44
45      return M, X'
46  end
```

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, 85].

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, what is the longest period in which the cluster exists without disappear 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]:

$$S(C_i) = \sum_{x_j \in C_i} (\lambda_{\max}(x_j, C_i) - \lambda_{\min}(C_i)) \tag{2.14}$$

Equation 2.14 adapts the concept of 'excess of mass' [55] 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.

2.5 Clustering Evaluation

Maybe this can be moved to the methodology?

Furthermore, as evaluating the quality of unsupervised clustering results is crucial, we will also introduce several common clustering evaluation heuristics used in this work: Inertia, the Silhouette coefficient [66], the Davies-Bouldin Index [21], and the Calinski-Harabasz Index [14].

:
Maybe this should be later on in the methodology?

- Motivate the need for clustering evaluation.
- Just highlight mathematics of each algorithm and mention the need for them.

2.5.1 Inertia

2.5.2 Silhouette

2.5.3 Davies-Bouldin Index

2.5.4 Calinski-Harabasz Index

2.6 Maybe Optuna

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 [33]. 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 [60] (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

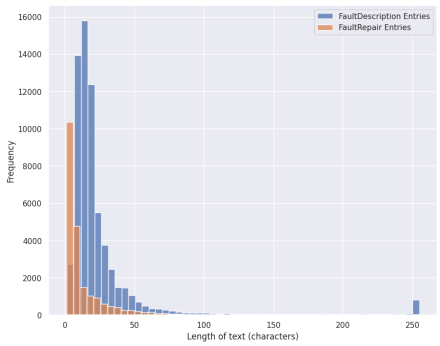
:

- 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

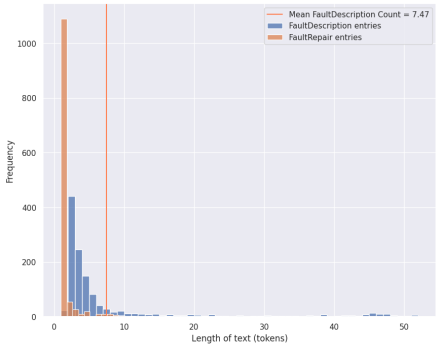
3.2 Overview

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

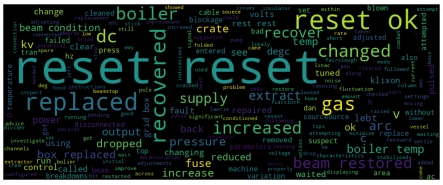


(a) Frequency distribution of the text lengths for non-blank `FaultDescription` and `FaultRepair` fields. Each bar represents a range of 5 characters.

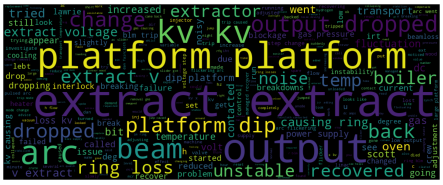


(b) Frequency distribution of the number of tokens for non-blank `FaultDescription` and `FaultRepair` entries. Each bar represents a range of around 2 tokens.

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 [71]. 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) [91]. 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 [54].

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) [72, 57, 58].

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 [72]. 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 and is able to encode coarse versus fine-grained semantic information in embedding vector sub-spaces [57, 58]. 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.

3.4.3 Detailed Rationale

In this section, we cover the selected models in a higher level of technical detail.

MPNet

MPNet aims to combine both BERT's and XLNet's pre-training objectives to achieve the benefits without the drawbacks of each individual models. This hybrid approach is particularly attractive for this project because the Operalog dataset contains a mixture of short, concise entries and longer, more descriptive entries detailing causal factors or sequences of events. Effectively understanding these requires capturing both fine-grained token relationships and broader contextual meaning.

While detailed descriptions of BERT and XLNet are in Section 2.2.1, the key challenge MPNet addresses is that masked language modelling (BERT) assumes independence between masked tokens, potentially losing valuable dependency information, and permutation language modelling (XLNet), while capturing dependencies, can have inconsistencies regarding positional information between pre-training and fine-tuning [72, 94].

MPNet proposes a 'unification' solution that addresses these issues. It achieves this by intelligently permuting the order of input tokens and applying masking in a way that allows

the model to learn bidirectional context (like BERT) while also modelling the dependencies between tokens (like XLNet) [72]. This capability is crucial for the Operalog data, where understanding the relationship between specific technical terms, even if separated by other words, is key to identifying the true nature of a fault. This unified objective function is formally defined in Appendix A.1 (Equation A.1).

To facilitate this objective, MPNet structures the input by rearranging tokens based on the permutation and partitioning them into non-predicted and predicted segments, using appropriate masking (conceptually illustrated in Figure 3.3). This allows the model to condition its predictions effectively. The precise mechanism for structuring the input is detailed with an example in Appendix A.1 (Equation A.2).

Similarly to XLNet, MPNet also employs a two-stream self-attention mechanism (content vs query) but modifies the attention mask to enable the model to see the entire input sequence. The modified attention masks allow the query stream to access the entire sequence during pre-training, which better aligns with downstream (fine-tune) tasks [72]. This also matters in our use-case as, using a pre-fine-tuned model, we want a pre-training objective that reflects inference usage, rather than focus on fine-tuning discrepancies. Because many Operalog entries reiterate initial fault cues later in the same sentence (e.g. describing a corrective action by name), allowing the query stream full sequence visibility helps MPNet embed those long-range dependencies.

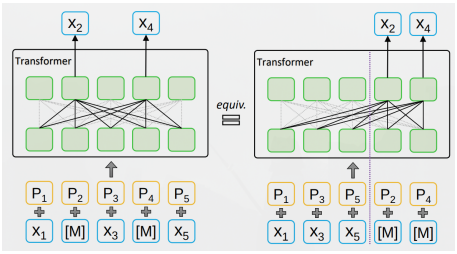
Similarly to XLNet, MPNet employs a two-stream self-attention mechanism (distinguishing content and query information). However, MPNet modifies the attention masks to allow the query stream to access the entire input sequence during pre-training [72]. This modification better aligns the pre-training conditions with how the model is used during inference for downstream tasks like embedding generation. This is relevant for our use-case as Operalog entries sometimes re-iterate initial fault symptoms or mention corrective actions later in the description (i.e. 'ion source unstable ... source change required.'). Allowing the model full sequence visibility helps MPNet embed those long-range dependencies, leading to more accurate embeddings for capturing the overall semantic meaning.

Using these changes, (Song et al., 2020) has quantitatively shown that MPNet leverages 92.5% tokens (as opposed to 85% by BERT) and 100% of positions (as opposed to 92.5% by XLNet) of input sequences (under the assumption they predict the same 15% masked token amounts). In addition, they have also shown that MPNet outperforms the state-of-the-art, at the time of publishing, in almost all GLUE benchmark tasks (an industry standard language model benchmark, with 9 natural language understanding tasks) and various other benchmarks [87].

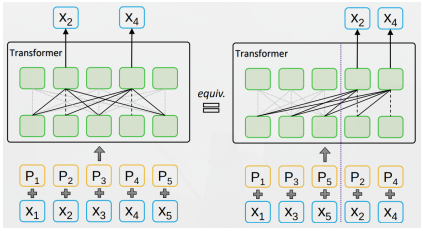
For further information on the MPNet architecture and pre-training, the reader is referred to the original paper by (Song et al., 2020).

Nomic

The second model evaluated is Nomic. This model, released in 2024, represents a state-of-the-art, open-source text embedding model specifically designed for long-context understanding and reproducibility [57, 58]. Version 1.5, used in this project, builds upon the initial release with features like Matryoshka Representation Learning (MRL) and multi-modal support.



(a) Masked language modelling (left) compared to the permuted order (right, unified view).



(b) Permutation language modelling (left) compared to the permuted order (right, unified view).

Figure 3.3: Comparison of both masked and permutation language modelling pre-training objectives. Here, tokens are denoted X_i, masked tokens are denoted `[M]` and positional encoding is denoted P_i. Source: (Song et al., 2020).

The relevant adaptions and justifications are listed below.

- Long context handling via RoPE - Nomic utilise Rotary Positional Embeddings (RoPE) instead of traditional positional embeddings [76, 57, 84]. RoPE encodes positional information using rotations, which has been shown to improve model generalisation to sequences of lengths not seen during pre-training (whether longer or shorter) [76]. This is a significant advantage for the Operalog dataset, where `FaultDescription` entries vary greatly in length (Figure 3.1), helping ensure effective semantic understanding by the model.

- Computational efficiency - Despite its capabilities, Nomic v1.5 maintains a relatively small parameter count of around 137 million compared to other leading models, whose parameter counts typically exceed 1 billion significantly. It also incorporates architectural optimizations like FlashAttention [20], an I/O-aware attention mechanism that reduces memory usage and latency during inference [57]. These factors were significant in selecting Nomic, as they allow the model to run efficiently within the constraints of the available 6GB GPU VRAM (Table 3.3), unlike larger models such as Qwen2 or Mistral-7B (Table 3.4).

- Task-specific optimisation - Nomic supports task-specific prefixes prepended to the input text. For this project, the 'clustering:' prefix was used for all `FaultDescription` entries fed into the Nomic model. This technique enables the model to produce embeddings specifically optimized for clustering tasks, by leveraging distinct semantic embedding subspaces learned during pre-training [57]. Utilising this prefix aims to enhance the quality of the embedding space specifically for the goal of grouping similar fault types, potentially leading to more coherent and meaningful clusters.

- Matryoshka Representation Learning (MRL) - Nomic version 1.5 incorporates MRL, which trains the model to produce nested embeddings where shorter prefixes of the full embedding vector still form meaningful representations [39, 58]. While the full 768-dimension embedding was used for this project, with MRL creating a structured

embedding space, it opens up the possibility of detecting patterns that aid fault categorisation more easily.

- Modern Architecture Choices - Nomic also incorporates other modern design choices adapted from BERT, such as removing the next sentence prediction pre-training objective, which simplifies pre-training and allowing for longer contiguous text processing, and using 0% dropout, reflecting findings on training with very large datasets [57, 46, 92]. These contribute to its overall efficiency and performance.

- Benchmark Performance - Nomic version 1.5 demonstrates strong performance on the MTEB leaderboard, particularly for its size, ranking well on clustering tasks and providing empirical validation for its selection [54, 59].

For further technical details on the Nomic architecture, pre-training and MRL, readers are referred to the relevant publications [57, 58, 39].

3.4.4 Section Summary

MPNet and Nomic satisfied the criteria set out initially, including the hardware requirements and dataset features identified previously. They are therefore carried forward through the remaining portion of the pipeline.

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' [85], see Section 2.3 for more details). This section explores and justifies the applicability of two techniques for this purpose: Principal Component Analysis (PCA) [62, 30], a linear method, and Uniform Manifold Approximation and Projection (UMAP) [49], 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 [49]. 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 3.5. 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 3.5, 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 [49]. To account for this variability, Figure 3.4 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 3.5 and the consistently lower (although variable) normalised variance scores, indicative of the better preservation of manifold structure, in Figure 3.4, 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

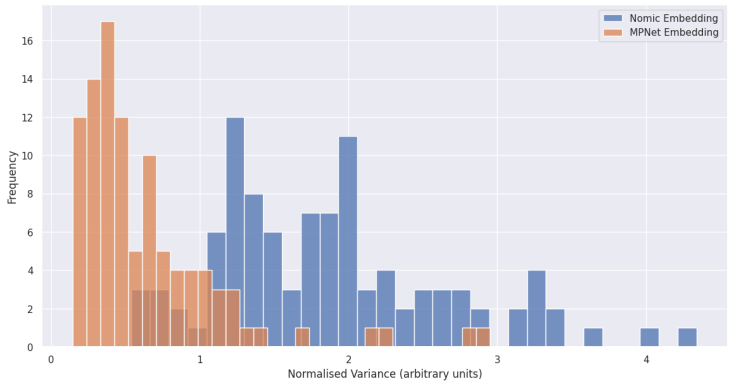


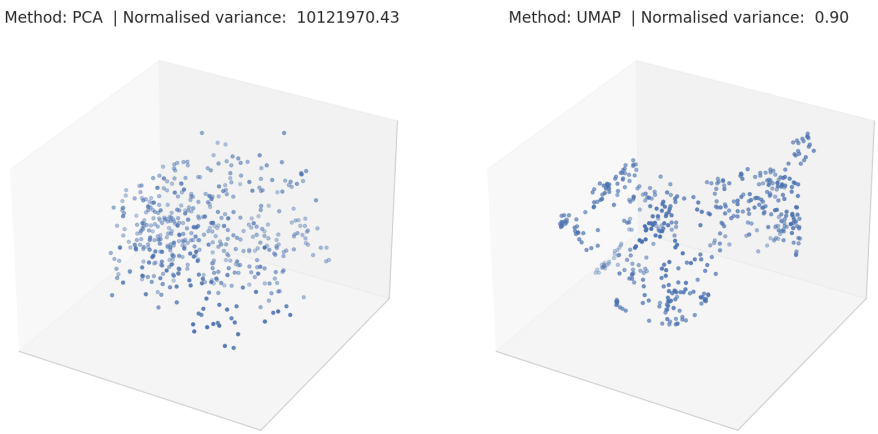
Figure 3.4: Histogram plot of 100 UMAP runs' normalised variance (Equation 3.1) is shown for both MPNet and Nomic embedding spaces of the Operalog `FaultDescription` entries. UMAP parameters: `min_dist` = 0.1, `n_neighbors` = 15, `metric` = cosine.

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.2. 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.

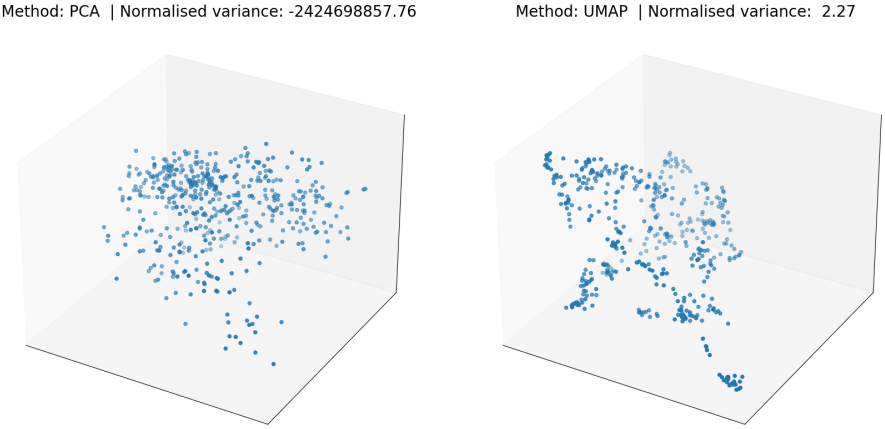
Table 3.5: UMAP hyperparameter values explored in naive grid search

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 3.5: 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 2.5), will serve as a proxy for how well the UMAP projection preserved the relevant structure for categorization. The next section introduces the clustering algorithms and metrics chosen for this purpose

3.6 Unsupervised categorisation through clustering

The low-dimensional embedding space, projected by UMAP, is now ready for clustering. Clustering is the unsupervised machine learning task of grouping similar points together (see Section 2.4). Intuitively, each cluster produced by a clustering algorithm can be mapped uniquely to an Operalog fault category. However, as we are inducing fault categories 'unsupervised', it is not immediate how to evaluate any one clustering. Three clustering algorithms were chosen, to compare against each other: k-Medoids [36], a partitioning-based clustering algorithm; DBSCAN [24], a density-based clustering algorithm; and HDBSCAN [15], a hybrid density and hierarchical clustering algorithm which is based on DBSCAN. In addition to this, k-Means [48], another partition-based clustering algorithm was considered but not chosen. This is due to the well-known disadvantages of k-Means; it is highly sensitive to the initialised configuration and it has issues with instability [**?**]. This section outlines the clustering algorithms used and motivates the clustering heuristics used for clustering evaluation.

Maybe list these clustering heuristics explicitly here?

3.6.1 Clustering Candidates

The k-Medoids clustering algorithm has one key hyperparameter (see Section 2.4.1), k, the number of initial clusters. Due to this, to select the optimal configuration, various values of k must be tested against clustering heuristics. Additionally, as mentioned in the detailed description of the k-Medoids algorithm, the runtime complexity of the algorithm

The DBSCAN algorithm has two key hyperparameters (see Section **??**), `Eps` and `MinPts`.

The HDBSCAN algorithm only has one key hyperparameter (see Section **??**), m_{pts}.

Heavily fix the wording here, discuss benefits and drawbacks of each for our operalog and motivate why we need to test all three.

3.6.2 Clustering heuristics

Motivated by the necessity to evaluate this open-ended problem, four clustering heuristics are utilised: inertia, ...

list the others here

:

- mention k-medoids sklearn implementation uses PAM algorithm. Touch on the fact it only has one parameter, which is significant for our usecase
- motivate the clustering - DONE ISH
- talk about clustering algorithms we did not consider, with research - DONE ISh
- explain the 3 clustering methods we have chosen to use and the hyperparameters in a table
- show test clustering
- this section is not really quantitative, highlight how we have to introduce clustering heuristics

3.7 Hyperparameter optimisation

3.8 Label generation

3.9 CLI Application

:

- Okay so we have done hyper parameter tuning etc but the real benefit of this tool is that it gives you a qualitatively assessable result in an easy to run format.

: Describe the methods and procedures used. The things in this section will include.

- Explaining data format and data visualisation: wordcloud.
- Data cleaning steps, including removing key words such as Ion Source.
- Text preprocessing steps (cleaning) and computational challenges (tensorflow).
- Choosing the best sentence embedding transformer: MPNET, NOMIC.
- Data visualisation (before and after sentence embedding): similarity visualisation, explain unique sentences, token length distribution.
- Motivate why clustering in higher dimensions performs worse
- UMAP, PCA, t-SNE comparison. Motivate using UMAP.
- UMAP hyperparameter optimisation.
- Performing clustering with kmedoids, dbscan, hdbscan.
- Using optuna.
- Evaluation of results and choosing the best model (and arguing why hdbscan is the best by looking at the variance of dbscan and inflexibility of kmedoids)
- Touch on the production of a CLI application that allows you to mix and match various parts of the pipeline. Motivate the need for command line tool.

Chapter 4

Results and Discussion

: Describe the results and analyse the results

- Analyse the word cloud.
- Analyse the sentence embedding results.
- Analyse UMAP vs. PCA vs. t-SNE qualitatively and later quantitatively (compared to the clustering).
- 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] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

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

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

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

[39] 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.

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

[41] 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.

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

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

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

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

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

[52] 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.

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

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

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

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

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

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

[65] 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.

[66] 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.

[67] 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.

[68] 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.

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

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

[71] 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.

[72] 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.

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

[74] 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.

[75] 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.

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

[77] 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.

[78] 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.

[79] 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.

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

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

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

[87] 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.

[88] 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.

[89] 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.

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

[91] 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.

[92] Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling llm under token-crisis. *Advances in Neural Information Processing Systems*, 36:59304–59322, 2023.

[93] 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.

[94] 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.

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

[96] 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

Further Discussion

A.1 MPNet Mathematical Formulations

This appendix provides supplementary mathematical details for the MPNet model discussed in Section 3.4.3.

A.1.1 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 [72]:

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

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

A.1.2 MPNet Input Structuring Example

To implement the objective (Equation A.1), 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 [72]:

$$\langle x_{z_{\leq c}}; M_{z>c}; x_{z_{>c}} \rangle = \langle x_3, x_5, x_2; [M], [M], [M]; x_1, x_4, x_6 \rangle \tag{A.2}$$

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

A.2 UMAP Hyperparameter Search

Figures A.1 through A.6 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.

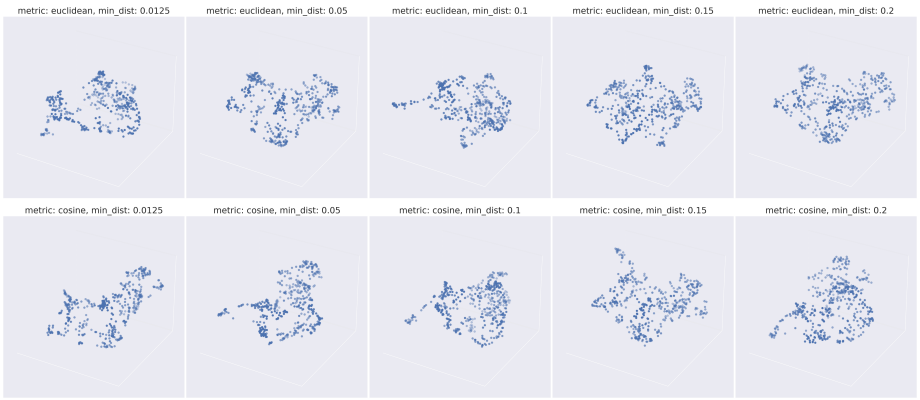


Figure A.1: MPNet, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `metric` and `min_dist`

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

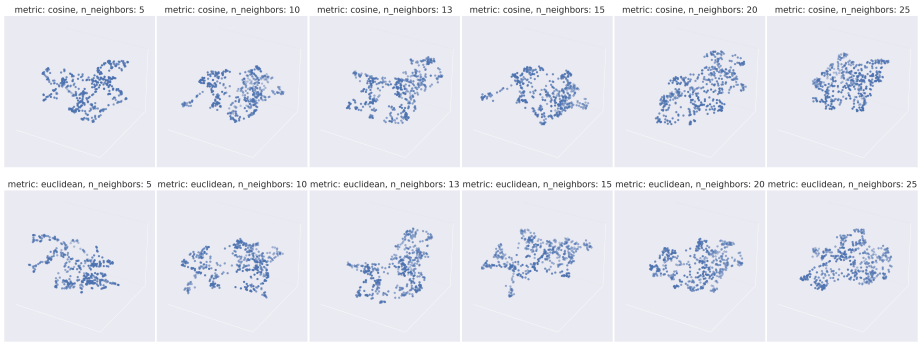


Figure A.3: MPNet, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `metric` and `n_neighbors`

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

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

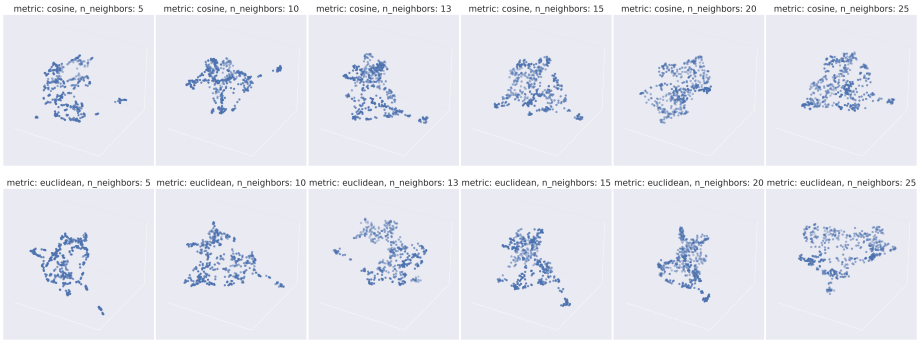


Figure A.6: Nomic, grid search for UMAP hyperparameter (see Table 3.5) over the parameters `metric` and `n_neighbors`