Unsupervised Deep Learning Algorithms for Early Bushfire Detection

A thesis submitted in partial fulfilment of the degree of
Bachelor of Engineering (Honours)

by
Jasper Japp
U73155348

Supervisor: Xiangyun Zhou
Examiner: Salman Durrani

Australian National University

College of Engineering and Computer Science
The Australian National University

November 2025

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university. To the best of the author's knowledge, it contains no material previously published or written by another person, except where due reference is made in the text.

Jasper Japp
13 November 2025

Acknowledgements

I'd like to first thank my supervisor Xiangyun Zhou for being incredibly supportive throughout the project and encouraging me to learn as much as possible. I'd also like to thank my family in Perth, who have been immensely supportive throughout my time at university. I'm very grateful to have the support of my girlfriend, Scarlett, who has given a lot of feedback, advice, and proofreading. Finally, I'd like to thank my friends and, in particular, housemates who have heard too much about this project and given their fair share of advice in various aspects.

Abstract

Bushfires (also known as wildfires or forest fires) are increasing in frequency and severity, making early detection critical. One method of bushfire detection is the use of Internet of Things (IoT) sensor networks, which provide line-of-sight-independent coverage. The performance of IoT sensor networks hinges on algorithms that can run on microcontroller-class nodes. This project developed two unsupervised deep-learning detectors, a GRU Gaussian-Mixture VAE (GGM-VAE) and a Hybrid-VAE that fuses a Gated Recurrent Unit (GRU) time branch with a short-time Fourier transform convolutional branch (STFT-CNN). The models used carbon dioxide and total volatile organic compound data from background collections to train without labelled data. The models were evaluated across eight ignition events using F1-score, false-positive rate, and time-to-detect. The best model, the GGM-VAE, achieves an F1-score of 0.57, a false-positive rate of 0.038%, and detects fires in approximately 4 minutes on average, outperforming both a more complex time-frequency domain model (Hybrid-VAE) and a threshold baseline. The GGM-VAE was then deployed to a microcontroller node based on an RAK3172, where it fits in device memory and runs inference in approximately 150 ms per window, showing feasibility at the edge. The work provides a practical accuracy-versus-complexity trade-off and a reproducible path from training to firmware.

Contents

List of figures

List of tables

Nomenclature

Abbreviations

ANU	Australian National University
AE	Autoencoder
CNN	Convolutional Neural Network
ELBO	Evidence Lower BOund
FLOPs	Floating-Point Operations
FP	False Positive
FN	False Negative
TP	True Positive
FPR	False-Positive Rate
TPR	True-Positive Rate
FFT	Fast Fourier Transform
STFT	Short-Time Fourier Transform
GGM-VAE	GRU Gaussian-Mixture Variational Autoencoder
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
IoT	Internet of Things
KL	Kullback–Leibler divergence
LoRa	Long Range radio
LoRaWAN	LoRa Wide Area Network
MCU	Microcontroller Unit
NLL	Negative Log-Likelihood
RH	Relative Humidity
SNR	Signal-to-Noise Ratio
TVOC	Total Volatile Organic Compounds
UART	Universal Asynchronous Receiver-Transmitter
VAE	Variational Autoencoder

Chapter 1 Introduction

1.1 Introduction

The frequency and severity of bushfires (also known as wildfires or forest fires) in Australia and globally have increased in recent years as a result of climate change [1], [2]. This has led to record-breaking fires becoming standard year after year [3]. Australia has been particularly prone to this trend where the 2019-2020 bushfire season has been termed the "Black Summer", burning 19 million hectares, destroying over 3,000 homes and claiming 33 lives directly [4]. Further studies have attempted to look at the prolonged impact, estimating an additional 417 deaths and 3,154 hospitalisations for cardiovascular or respiratory problems [5]. World Wildlife Fund estimated that almost 3 billion animals were killed, harmed, or displaced by these fires [6]. These figures underscore that bushfires pose a serious threat to human life, ecosystems, and critical infrastructure.

Fires spread and grow quickly, meaning small ignitions can explode into large uncontrolled fires. Accurate bushfire detection within the early stages is crucial to limiting their spread and impact, as early intervention greatly increases the likelihood of successful containment and extinguishment [7].

1.2 Background

Bushfire detection methods encompass a diverse range of scales and technologies, all of which have various strengths and limitations. To successfully detect fires quickly and effectively it is likely that a combination of several methods will be used to mitigate blind spots or oversights in each approach. Common methods include fire towers, satellites, vision-based, and environmental sensors [8]. These systems detect various emissions released from a fire, such as light, smoke, flames, and gas compounds, such as CO, CO_2, and volatile organic compounds (VOCs).

A traditional approach, such as fire watchtowers, involve large structures that are erected and manned where people spot smoke or the fire itself [8] which can be fast and reliable. However, these approaches are limited by line-of-sight blockers such as hills or trees. As such automated fire detection methods have been emerging.

Satellite-based fire detection uses earth-observing satellites to spot fires over wide areas. They have proven invaluable in remote areas [8] where their scale and incredibly expansive coverage have been their biggest asset. However, they are limited by spatial and temporal resolution. A satellite may only pass over areas a few times a day, leading to longer detection time and allowing fires to spread and grow undetected. The use of geostationary satellites has been explored, but most systems require direct line of sight with fire indicators such as smoke or thermal signals [8]. This can be obstructed by natural products such as cloud or smoke cover.

In this case the fire must be significantly large to detect whilst in orbit [8]. So, while satellites offer broad coverage they can miss fires in the early stages.

Vision-based fire detection methods have addressed some of these gaps and can take a variety of forms, whether mounted to existing watchtowers, trees or attached to aerial drones. Watchtower or airborne solutions often use optical or infrared imaging leveraging computer vision approaches using indicators such as smoke or fire. These use proven algorithms and can work at high resolution to provide accurate and fast detection. Yet as with satellites, they are prone to environmental effects, such as fog or smoke and low visibility at night [8]. Systems involving large deployment of small ground-based Internet of Things (IoT) camera nodes also are in development. They face additional limitations such as power draw and local processing constraints on inexpensive hardware [8]. They operate using reduced resolution and frame rate to save power and decrease costs. These can again be combined with IR cameras, but this increases cost and complexity. Overall, vision-based detection methods provide a lot of information and can be highly accurate in ideal conditions, however, are limited by local coverage, and environmental conditions such as darkness, smoke, or fog.

In this project the focus will be on IoT-based gas/environmental sensor networks that can mix a variety of detectors and be deployed in fire-prone areas. They can monitor parameters such as temperature, humidity, CO_2 content, and other compounds. Through using measured changes and relative concentrations of these parameters, the goal is to predict smoke or other fire gases indicating the start of a fire. This may allow detection before visible indicators show and unlike vision-based or satellite-based methods they don't require direct line-of-sight. These networks can be made up of a series of edge nodes or sensors that communicate using local transmission or satellite-based communication. They can incorporate local temporal-detection on the edge nodes and spatial-detection using the network. These nodes tend to be lower cost, smaller, and more efficient than ground-based vision nodes allowing for denser deployment.

Large-scale deployment with IoT sensor networks is still a challenge. To achieve wide and reliable coverage, many nodes with careful placement may be required. This can massively increase deployment costs and time. Maintenance and reliability in harsh conditions is an additional concern. Sensors may become damaged, drift, or increase in noise over time. Despite these limitations, their flexibility and relative scalability, mean that they are well suited to deployment in high-risk areas, where early detection is critical [8].

The success of IoT-based environmental sensor networks depends largely on the performance of the detection algorithms used [9]. As previously mentioned, these networks can make predictions based on the individual predictions of multiple nodes, thus taking into consideration spatial trends [10]. This spatial detection algorithm can be treated separately from the time-based detection that occurs at individual nodes. Spatial detection has shown its ability to heavily reduce the number of false alarms when compared to a purely temporal approach [10]. Spatial detection can occur at more powerful hubs that don't have the limitation of being deployed in the environments.

It is very power and bandwidth intensive to transmit all raw sensor data to these hubs, thus it is important to move the first time-based detection to the edge nodes of a network. These edge nodes integrate the gas sensors themselves, a microcontroller, and a transmitter. For time-based detection it is common to use anomaly detection, where it classifies inputs as "normal" or "anomalous". In bushfire detection, anomalies correspond to fire presence. This approach has been applied in fields like cybersecurity, indoor fire detection, and computer vision [11]. Two major methods dominate in this domain, thresholding and neural network-based approaches [8].

Thresholding uses fixed or statistical limits (e.g., z-score, Local Outlier Factor) to flag anomalies [8]. While very computationally efficient and easy to implement, thresholding lacks adaptability, especially in multi-sensor scenarios.

Neural networks, including deep learning (DL) models, offer greater adaptability and can model complex relationships between variables. They have demonstrated high fire detection accuracy in both indoor and outdoor settings. For instance, indoor detection models using deep learning have achieved accuracies above 99% [12]. These anomaly detection models are a supervised approach, where they are trained using data labelled as containing fire or not containing fire and are rewarded for correctly differentiating these labels.

However, applying these methods outdoors is more challenging due to variable environmental factors such as wind dispersion and inconsistent sensor exposure, leading to ambiguity in labelling ground truth events [10], [13] . Essentially, it is difficult to be sure that the data contains indicators of fire like gases or smoke. To address this, unsupervised models have gained interest. These train only on "normal" data, that is data that does not contain fire. They are trained to recognise usual patterns and trends within this data, and detect deviations, that might indicate fire. This is particularly valuable in outdoor contexts, where acquiring clean and consistent labelled data is difficult. However, unsupervised models often require complex architectures and more computational resources, presenting challenges for deployment on low-power devices [8].

1.3 Scope and Contribution

This project focuses on the development and optimisation of edge-node temporal detection for early bushfire detection. It explores unsupervised deep learning to achieve low false-alarm rates, consistent detection, and fast response on multivariate environmental time series.

For training we used unlabelled multivariate environmental data from the ANU Bushfire Initiative. Models were evaluated on existing ANU IoT hardware under realistic deployment constraints (compute, memory, and energy) to characterise on-device feasibility. Spatial detection/aggregation, hardware design, and inter-node communication protocols are out of scope for this study.

The overall project aims to contribute to two key areas. First, it seeks to further research and development into unsupervised deep learning-based anomaly detection algorithms for early bushfire detection, using time series environmental sensor data to develop and evaluate two separate models. Second, it aims to assess the feasibility of deploying these deep learning models on real IoT hardware, considering computational and power constraints. The goal is to determine whether, and how, deep learning-based approaches can be effectively implemented in this resource-constrained environment. This project will also provide recommendations for how to design algorithms for the hardware constraints.

1.4 Research Questions

The project will address the following research questions, presented hierarchically:

1. *What unsupervised deep learning anomaly detection algorithms can be developed and optimised to improve early bushfire detection accuracy using real-time environmental data from IoT sensor nodes?*

2. *Can unsupervised deep learning algorithms be implemented on IoT sensor hardware effectively?*

1.5 Thesis Organisation

This thesis is organised into 5 chapters:

- **Chapter 1 Introduction –** This chapter introduces the problem, the background, the scope, the aims and the research questions for the project.
- **Chapter 2 Literature Review –** This chapter surveys and reviews the current relevant literature, including detection approaches and hardware implementation.
- **Chapter 3 Model Development** – This chapter discusses and presents the methodology, architecture and results of the two unsupervised deep learning approaches for bushfire detection.
- **Chapter 4 Hardware Implementation –** This chapter evaluates the feasibility of deploying the previously developed model on real MCU-class hardware.
- **Chapter 5 Conclusions and Future Work** – This chapter provides a summary for the complete body of work present and provides recommendations for future work.

Chapter 2 Literature Review

This chapter summarises key machine learning approaches to bushfire detection on environmental IoT sensors, and methods in adjacent and relevant domains. It then discusses the literature surrounding the implementation of these detection models on hardware. The aim is to provide further motivation and identify gaps within the literature that are explored within this project.

2.1 Machine learning anomaly detection approaches

Machine learning, in which algorithms learn patterns from data, has emerged as one of the most powerful and popular detection methods within the bushfire detection domain. They allow for more complex modelling of multivariate signals. Building on this, deep learning techniques, which are a subset of machine learning based on multi-layer neural networks, can model long- to short-term temporal dependencies. These are important improvements over simpler techniques such as thresholding due to the ambiguous and varying nature of this domain.

2.1.1 Clustering

Clustering is primarily an unsupervised learning approach where data is organised in similar groups known as clusters. These clusters each contain data that is similar in at least one way but still distinct from those in other clusters [14]. Anomalies are detected based on how far a point lies from these clusters. In bushfire-related anomaly detection this is often done either directly, such as k-means followed by a distance-to-centroid rule [15], or via "clustering-adjacent" one-class/neighbour methods that rely on local density or decision boundaries rather than explicit clusters (e.g., LOF, Isolation Forest, one-class SVM) [14]. These latter methods are commonly used on features learned by an encoder to reduce dimensionality while preserving structure.

K-means remains attractive for edge-constrained deployments because it is simple and fast. It partitions data into k groups by minimising squared distances within clusters. An anomaly score is the point's distance to its nearest cluster centre. Andrade et al. [16] developed a two-step lightweight unsupervised k-means clustering approach for wireless sensor networks. First it clusters sensor nodes with k-means, then applies a compact weighted averaging statistic inside each cluster to suppress outliers. They showed the ability for k-means clustering to successfully cluster information in the presence of outliers and predict anomalies. Within the bushfire domain, Dampage et al. [17] also used k-means clustering to partition their dataset into fire and non-fire clusters. Then they trained a multiple linear regression detector that achieved an accuracy of 81% of environmental data composed of temperature, light intensity level, CO level, and humidity.

Üstek et al. [18], presented a comparison of three well-established clustering techniques: isolation forest, local outlier factor (LOF), and one-class support vector machine (SVM). LOF scores a point by how sparse its local neighbourhood is compared with its neighbours, the lower the local density, the higher the outlier score. Isolation Forest builds many random decision trees where rare points get isolated in fewer splits. One-class SVM learns a boundary enclosing the “normal” class in feature space and points outside are flagged as anomalies. They use eight features from unlabelled historical weather and vegetation index data to predict the presence of bushfires. They demonstrated that while all models performed similarly the LOF outperformed the isolation forest and one-class SVM, achieving a maximum classification accuracy of 65%. Notably, they showed that these clustering techniques are sensitive to specific parameters and were outperformed by recurrent neural network (RNN) techniques and autoencoders.

2.1.2 Multi-layer Perceptron (MLP)

A Multi-layer Perceptron (MLP) is feed-forward neural networks composed of fully connected layers with non-linear activations. They have been widely used in anomaly detection inside and outside the bushfire domain. They learn to map inputs to a target output, which in the case of anomaly detection, are typically configured either as binary classifiers (fire or non-fire) or reconstructions to predict sensor readings. Their appeal in IoT-based detection is that they are lightweight to deploy on resource-constrained nodes [8].

Putzu et al. [19] implemented a MLP for real-time forest fire detection, in a LoRa-based wireless sensor network. They demonstrate effective early alerts while minimising computational overheads. Pokhrel and Soliman [20], fused temperature, light and sound signals to develop a small semi-supervised MLP. They incorporated both spatial and temporal detection for a wireless sensor network (WSN) made for outdoor bushfire detection, which achieved 89% fire detection accuracy. Illustrating that even basic neural networks can serve as anomaly detectors and offer a step up in integrating multiple sensors compared to clustering methods.

MLPs are limited by both their lack of memory and their inability to model temporal dependencies which is inherent in this domain. They process each input independently from the last. In practice this can be improved through feature engineering, as Pokhrel and Soliman [20], created “supervectors” with consecutive readings so their MLP could learn short-term time-dependencies.

2.1.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a form of Artificial Neural Network (ANN) and are a deep learning approach. They are primarily distinguished by an internal feedback loop. They can be thought to be "unrolled" as the network operates like a chain of N-many identical networks, where N is the sequence input length. This means that they have two inputs, the current data input, and the processed information of all data previously. This allows RNNs to more easily model time features when compared to other ANNs [21]. This feature has made

Recurrent Neural Networks (RNNs), especially long short-term memory (LSTMs) and gated recurrent units (GRUs), well suited for time series anomaly detection. Both GRUs and LSTMs use additional memory states over the traditional RNN, with the goal of modelling long-term dependencies [21] [22]. GRUs, with fewer parameters than LSTMs, are computationally efficient and competitive in accuracy [22].

Numerous studies have explored RNNs for fire detection. Kim et al. [12] applied both LSTM and GRU networks in a supervised indoor fire detection context using multi-sensor data. Their results showed that even relatively simple architectures outperformed more complex models like Transformers and InceptionTime, with the LSTM achieving a near-perfect F1-score of 0.9999. While the GRU underperformed marginally in terms of metrics, its simplicity and smaller size made it a favourable candidate for constrained hardware.

Benzekri et al. [23] evaluated and compared basic RNN, LSTM, and GRU architectures for wildfire detection using supervised learning. Their models incorporated data from multiple gas sensors along with a derived fire risk index. Among the tested architectures, the GRU achieved the highest classification accuracy of 99.89%. However, despite the strong performance, the study lacks transparency regarding the dataset used. There is no information provided on data source, labelling methodology, or experimental setup. This omission limits the interpretability and reproducibility of the results and makes it difficult to assess the model's performance.

In the unsupervised domain, Xu et al. [24] proposed an LSTM-based variational autoencoder (LSTM-VAE) that combined sequence modelling with probabilistic latent space representation. This approach achieved perfect detection performance on simulated and experimental indoor datasets. Connell [13] adapted this model for outdoor wildfire scenarios and achieved an F1-score of 0.9, outperforming several supervised baselines and demonstrating robustness to noisy inputs.

Other works, such as Üstek et al. [18], compared LSTM-based models with fully connected autoencoders and clustering techniques. Although their results favoured simpler fully connected architectures for some datasets, their LSTM configurations suffered from inadequate hyperparameter tuning and premature stopping, suggesting underutilised potential.

Guo et al. [25] introduced the GRU-based Gaussian Mixture Variational Autoencoder (GGM-VAE), which uses a probabilistic mixture model to capture complex distributions in multivariate time series data. Evaluated on benchmark datasets from Intel and Yahoo, the GGM-VAE achieved up to 7.2% higher F1-score compared to GRU-VAE and GRU-AE variants. Its ability to model multiple latent distributions may prove advantageous in bushfire detection, where environmental data distributions vary due to time of day, season, and sensor deployment conditions.

2.1.4 Transformers

Transformers have recently emerged as a powerful architecture for sequence modelling, particularly in natural language processing tasks such as those used in large language models (LLMs) for AI chatbots. By leveraging self-attention mechanisms, Transformers can model long-range dependencies without recursion [26]. Unlike RNNs, Transformers can process entire sequences in parallel, which reduces training time and improves scalability [26]. Their flexibility has led to success in language modelling, speech recognition, and increasingly in anomaly detection.

Xu et al. [27] introduced the Anomaly Transformer, a model specifically designed for unsupervised anomaly detection. It outperformed RNN-based models such as LSTM-VAE and Omni-Anomaly across multiple public benchmarks. Zia et al. [28] proposed a Transformer-based anomaly detection model for IoT data from the Mars Science Lab. It achieved superior performance with reduced complexity relative to other transformer models.

Despite their performance advantages, Transformers present challenges for embedded deployment. Their memory footprint, reliance on matrix operations, and difficulty in scaling down for low-power hardware limit their practicality for current IoT sensors. Additionally, the parallelisation benefits of Transformers may not translate well to real-time streaming scenarios common in sensor networks.

2.2 Pre-processing Approaches

Performance of these algorithms can be improved by using various preprocessing techniques, this may involve transforming the input into another domain or engineering another feature.

2.2.1 First-order Difference

A common approach for bushfire detection is to engineer the first-order difference of a feature, which is the difference between the current time point and the previous one. Introducing the first-order difference as a feature can allow the model to more easily learn short-term temporal patterns. Zhuang et al. [10] developed a thresholding detector using only the first-order differences of eCO_2 levels to great effect, although with the use of spatial-detection. This is used after analysis of the data determined fluctuations, rather than absolute level to be more indicative of fire. Liu et al. [29] developed an environmental fusion LSTM and GRU and uses first-order differences of multiple variables to improve detection, again supporting this approach.

2.2.2 Frequency Domain

Frequency domain approaches have proven valuable for detecting anomalies in time series data by revealing patterns and relations that do not present in the time domain. Signal processing

techniques such as Fast Fourier Transform (FFT) and Short-Time Fourier Transform (STFT) enable the translation and subsequent analysis of signals in the frequency or time-frequency domains respectively. This can help identify patterns or spectral signatures associated with anomalies [30].

In recent years, several studies have explored the combination of frequency domain representations with deep neural networks to better anomaly detection performance. Lu et al. [31] introduced F-SE-LSTM, which constructs a frequency domain feature matrix using FFT applied over a time series window. They employ Squeeze-and-Excitation network (SENet), a convolutional neural network structure, with an LSTM to extract frequency-features across time periods. This effectively aims to capture periodic patterns in the frequency domain, indicative of anomalies. Using this, the F-SE-LSTM was able to detect subtle anomalies that purely time domain methods would have missed, outperforming several state-of-the-art deep learning detectors on public benchmark datasets. Similarly, Tu et al. [32] developed the STFT-TCAN architecture, where time series windows and STFT generate a spectrogram-like frequency matrix that combines time and frequency components. A temporal Convolution Network combined with a transformer-based attention module form the TCAN that learns features from this matrix. They achieved an improvement in anomaly detection over state-of-the-art in an industrial domain on multiple datasets. These highlight the benefit of leveraging the frequency domain. Backhus et al. [30] also support this trend with an implementation of a CNN-LSTM for ultrasonic vibration signals to detect structural damage in pipeline.

2.2.3 Wavelet Domain

Aside from the frequency domain approaches, several studies have transformed input data to the wavelet-domain for improved performance. The Wavelet Transform provides a multi-scale time-frequency representation, where it can capture temporal and frequency information across different scales [30]. This allows the model to localise transient events that may be missed on larger signal scale. Baek et al. [33] utilise wavelet transforms to predict fires on an indoor dataset. They implement a nearest neighbour algorithm to cluster data by various types: flaming, heating, and soldering. This non-deep learning approach outperforms many state-of-the-art models such as LSTM and thresholding. That said, wavelet pipelines tend to introduce additional hyperparameters and compute/memory overhead, which can be problematic on compute-limited nodes.

2.3 Design for IoT Hardware

To be effective in the field, detection algorithms must run on resource-constrained IoT devices. Several metrics, including number of parameters, memory usage, power consumption, and Floating-Point Operations (FLOPs), are used to evaluate model deployability.

2.3.1 Model Compression Techniques

Given the constraints of IoT hardware, recent work has explored various model compression and optimisation techniques. We will discuss two particularly relevant techniques: Knowledge Distillation and Quantisation.

Knowledge distillation is a promising technique for reducing model complexity while retaining performance. In this method, a smaller student model is trained to mimic the behaviour of a larger teacher model. Cheng et al. [34] applied this technique in the context of wildfire image classification building a generative adversarial network (GAN) based knowledge distillation framework in smart IoT networks. First, they train a heavy GAN to learn normal sensor data patterns, then progressively distil it to a lightweight student model. The distilled model achieved massive compression, up to 48:1, and with significantly fewer FLOPs with minimal loss in accuracy. Indicating that small models can approach the behaviour of large models on IoT sensor data, making for easier deployment on hardware.

Another useful strategy is quantisation, which involves reducing the numerical precision of model parameters and operations. This could be converting, for example, 32-bit floating points to 8-bit integers, dramatically reducing model size by almost four times. Nafis et al. [35], implemented a CNN on IoT-based hardware for flame recognition in indoor fire detection. They achieved an almost 4× reduction in size to 1.8 MB with a significant reduction in accuracy from 96% to 88%. They also monitored power consumption and inference time, both of which showed showing minimal change.

2.3.2 Implementation

Some studies have successfully implemented fire detection algorithms on IoT microcontrollers. For example, Papaioannou et al. [36] used a lightweight MLP model on an STM32 microcontroller integrated with environmental sensors. By heavily optimising code and using duty cycling, they achieved a very low power consumption and reliable fire detection. Notably using a 5100 mAh battery, they could run for over 37 days. This demonstrated the feasibility for lightweight model deployment. Additionally, this model consumed 60 mW in active mode which is still within limits of their solar power generation.

Vorwerk et al. [37] used transfer learning to optimise an image-based model for indoor fire detection. This was then implemented on a Raspberry-Pi with the sensors located on an ESP32 microcontroller, achieving moderate success. This highlights that with further optimisation an ESP32-class device can potentially host the model itself.

Putzu et al. [19] design a LoRa-based WSN with Arduino-class nodes measuring environmental variables and a small MLP for on-node detection. The network emphasises low-power, solar-powered operation and shows that very compact neural nets can fuse multivariate signals and trigger timely alerts in the field. Dampage et al. [17], similarly implement a WSN but use a k-

means clustering and regression approach, showing strong performance and detailing some practical implementation related features. This includes node design, power using solar and batteries, and placement height for a reliable system.

2.4 Literature Summary and Gaps in Current Research

Across clustering, MLPs, RNNs, and Transformers, and additionally pre-processing techniques, the literature shows a clear progression from lightweight models towards sequence-aware deep models. Clustering methods remain an attractive and functional baseline for simplicity and label-efficiency, but performance is highly sensitive to hyperparameters and environmental drift. Additionally, they have a reported plateau in accuracy achieving only moderate performance in outdoor datasets [16], [17], [18]. MLPs improve from clustering, using non-linear fusion of multivariate sensors and remain deployment-friendly on MCU-class hardware [19], [20]. Yet, they are memoryless and rely on feature engineering to incorporate any temporal dependencies which is still limited to short-term trends. RNNs address this limitation directly, demonstrated that in both supervised and unsupervised approaches they consistently outperform MLPs, and clustering methods in time series anomaly detection [12], [13], [23], [24], [25]. Transformers are growing to be a popular option in recent years, and have delivered strong benchmark results, [27], [28]. However, their quadratic attention cost, memory footprint, and reliance on large matrix operations pose a challenge for MCU deployment without significant model compression. Parallel to these methods, frequency domain and wavelet domain modelling has shown to uncover spectral and transient features missed in purely time domain modelling [31], [32], [33]. This could improve performance although it does present additional complexity, design choices and optimisations that could impact edge-node deployments.

From an implementation perspective, there is encouraging evidence that on-device deployment is feasible on MCU-class hardware. Many studies involve optimisations to model size and duty-cycling to reduce power consumption [37]. Compression methods such as quantisation and knowledge distillation have proven to be capable of massive size compression [34], [35]. Despite this progress, several gaps remain within the bushfire detection domain:

1. **Unsupervised deep learning models on device.** There is limited research on tailoring unsupervised deep learning models for environmental time series data and deploying them on embedded systems. Many studies either focus solely on algorithmic performance or ignore real-world hardware constraints.
2. **Time-frequency Integration.** Within the environmental bushfire detection domain, there is limited exploration of the frequency domain.
3. **Ground truth ambiguity.** While growing in popularity, development of unsupervised approaches for bushfire detection is largely lacking. Many methods such as GGM-VAE are unexplored in this domain.

This project aims to address and contribute these gaps by developing, evaluating, and optimising unsupervised deep learning models for IoT-based bushfire detection, considering both detection accuracy and hardware feasibility.

Chapter 3 Model Development

This section outlines the engineering approach used to develop two unsupervised anomaly-detection models for early fire detection:

(i) a GRU Gaussian Mixture Variational Autoencoder (GGM-VAE)

(ii) a Hybrid-VAE, which combines a GRU encoder with a time-frequency pathway based on a Short-Time Fourier Transform (STFT) and a light-weight Convolutional Neural Network (CNN).

It then compares the results to a simple thresholding baseline. The methodology below is designed to be reproducible and to support later deployment on constrained hardware.

3.1 Methodology

The development of the two models took a sequential approach, where the GGM-VAE was first developed using Guo et al. [25] and the Hybrid-VAE followed using the learnings of the previous model. Both models attempt to explore elements within the domain, which will be discussed. The models were implemented in PyTorch [38] and trained using the Adam optimiser [39]. A third pure thresholding baseline model is also developed for comparison.

3.1.1 Performance Metrics

To mitigate ground truth ambiguity, fire detection is scored per ignition. Any fire detection within 20 minutes post-ignition is counted as one true positive (TP) for that event. Multiple detections during this period are not double counted. If no detection occurs within that window this is a false negative (FN).

Alongside this we also report a window-level false alarm on non-burn data. The primary performance indicators are:

- **True Positive Rate (TPR):** Proportion of fire experiments with at least one detection.
- **False Positive Rate (FPR):** Proportion of windows in non-burn data incorrectly flagged as fire.
- **Average time-to-detect:** Time (in seconds) from ignition to first detection averaged across all experiments.

This evaluation protocol balances sensitivity and false alarms while being robust to sensor delay and ambient noise. This does lead to the imbalance of possible true positives (total number of ignitions) and possible false positives (total number of non-burn data windows in evaluation dataset), traditional metrics such as F1-score are heavily penalised for any false positives (FP). F1-score still presents a useful metric that balances TP, FP and FN, and as such is included to combine these as seen in eq. (3.1).

$$F1 - Score = \frac{TP}{TP + 0.5(FP + FN)} \tag{3.1}$$

Additionally, metrics relating to computational complexity were included to attempt to predict implications for final deployment. It is important to make a distinction between training complexity and deployment complexity. Training complexity relates to the process in training and computing backpropagated gradients from loss functions. Whereas deployment complexity has fixed weight and bias matrices that determine the model prediction given an input. The focus is on deployment complexity where the constrained IoT hardware will be the bottleneck. As such the computational deployment complexity metrics are as follows:

- **Floating-Point Operations (FLOPs):** A measure of the number of floating-point computations for a single forward pass.
- **Total Parameters:** The sum of every entry in weight and bias matrices.
- **Model Size:** The stored size in kilobytes (KB) of the model's weight and bias matrices. Eq. (3.2) shows the model size. The tensors can have different numeric types and as such we group them by type, t (e.g. Float32, int8, int4, etc.) Let N_t be the number of parameters of a given type and $bytes(t)$ the size of that type.

$$Model\ Size_{KB} = \frac{1}{1000}\sum_{t} N_t bytes(t) \tag{3.2}$$

  A KB = 1000 bytes.

Here FLOPs measure the complexity of a given inference. This will relate to the final deployment's inference time and power usage. Whereas total parameters and model size aim to add a comparative storage size for the model.

3.1.2 Threshold Calibration

All models evaluated will require a threshold value to be set. The sensitivity of the model is dependent on this threshold, thus for comparability amongst models, a loop was implemented to find and set the threshold to achieve the minimum FPR given every fire is detected. At this 'detect-all-fires' operating point, such as in comparisons, the TPR can be assumed to be 100%. This is to say that we are trying to compare the model's ability to differentiate non-burn from burn data.

In deployment there is a compromise between detection time and sensitivity, which would affect real world performance. The final deployment would additionally involve using spatial correlation and detection from multiple sensors to improve FPR. As this is outside the scope it is difficult to comment on what is an appropriate local FPR.

3.1.3 Hyperparameter Selection

Hyperparameter optimisation is an important step in the evaluation and comparison of machine learning model architectures. Hyperparameters are the definable parameters set before training that control things such as dimensionality of various layers, learning rate, window size, and loss weights amongst other things. Hyperparameters can significantly influence a model's performance, and the optimal choice often varies depending on the task domain or pre-processing methods used. One common approach is manual search, where hyperparameters are iteratively adjusted and evaluated by training the model multiple times and comparing key performance metrics. This method allows developers to leverage theoretical understanding of the model and its behaviour, but it is time-consuming and relies heavily on prior experience and domain expertise [40].

In contrast to manual search, automated hyperparameter optimisation aims to deliver consistent and reproducible results with minimal manual intervention. Techniques such as grid search and random search are simple and widely used, but they can be inefficient. This is especially relevant in high-dimensional search spaces [41]. More advanced approaches, such as Bayesian optimisation, typically achieve better performance with fewer evaluations.

Bayesian optimisation models the relationship between hyperparameters and the objective function using a surrogate model (e.g., a Gaussian process). It selects hyperparameter configurations based on both predicted performance and uncertainty, favouring regions that are likely to yield improvement. This targeted exploration enables more efficient optimisation than random or exhaustive methods. Bayesian optimisation is also scalable and capable of handling large hyperparameter spaces and has been shown to match or even outperform expert-tuned configurations [40].

For this project Bayesian optimisation will be used via the Optuna library. Optuna constructs a surrogate model of the objective function (e.g., validation loss or F1-score) and iteratively proposes parameter combinations to evaluate [41]. In this project, we will optimise for an F1-Score, as while severely punished, it does represent the optimisation of FPR, and TPR. The optimisation history is logged and visualised to inform trade-offs between complexity and performance.

For all trials, each model was limited to a maximum of 600 epochs (iterations of batches) with a batch size of 128. This was found to balance training time, convergence, and model performance in testing. Additionally, an early stopping constraint was set that if performance hasn’t improved in 20 epochs on the validation dataset, then the model will stop training and reset to the most performant model. This is to limit overfitting on the training dataset. Further discussion on these datasets will occur in Section 3.2.7.

3.2 Dataset and Preprocessing

This section describes the source data, selection criteria, preprocessing pipeline, and the partitions used for training, validation, and evaluation. The goal is to provide a reproducible path from raw sensor data to model-ready inputs.

3.2.1 ANU Bushfire Initiative

The dataset used in this project is provided by the ANU Bushfire Initiative and includes various fire-related experiments: six experimental burns, two smoke machine tests, eight prescribed burns, and several background data collections. Sensor readings vary across experiments but commonly include equivalent CO_2 (eCO_2), total volatile organic compounds (TVOC), temperature, pressure, humidity, and, in some cases, raw MQ2 sensor outputs.

The eight prescribed burns and two smoke-machine tests were excluded from use. The prescribed burns usually use chemical starters and are likely to produce smoke with different chemical composition and dynamics than bushfires. Smoke-machine trials likewise differ in particulate profile. Including these may shift the data distribution and thus are excluded to maintain a consistent domain.

The Experimental setup varies in sensor distance to the fire, and number of sensors. This information can be seen in Table 3.1. Experimental burn 6 lacks reliable ignition start time and is excluded. Sampling rates differ slightly, with most sensors operating at 1 Hz and some at 0.2 Hz. To preserve short-term temporal features, this project uses only the 1 Hz data. Down sampling was avoided to prevent potential loss of fire-related information. Thus, using this criterion, Experimental burn 3-5 were used totalling 8 ignitions. The burn for each ignition lasts for 20 minutes for every experiment, as such, for each ignition we include the 20-minute post-ignition period. Thus for 8 ignitions we have a total of 160 minutes of 1 Hz data.

Table 3.1, ANU Bushfire Initiative Burn Datasets

Experimental Burn	*Ignitions*	*Number of Sensors*	*Sampling Frequency (Hz)*	*Sensor distance (m)*
1	3	9	0.2	10
2	2	10	0.2	30
3	3	8	1	30 to 40
4	4	8	1	30 to 50
5	1	6	1	30 to 50
6	1	10	1	30 to 52

Background recordings include multiple sites and can be seen in Table 3.2. To match the 1 Hz criterion, the Blue Range recording forms the bulk of the training data. This is then supplemented with pre-ignition segments from other experimental burns.

Table 3.2, ANU Bushfire Initiative Background Datasets

Background dataset	*Number of Sensors*	*Sampling Frequency (Hz)*	*Length (Hours)*
Backyard	3	0.2	15
Blue Range	2	1	26
Stefan's Backyard	2	0.01	254

20 minutes of initial setup was excluded from the data, as this is dominated by human effects on the sensors including setup and sensor stabilisation artifacts. In some experiments this remains for longer than 20 minutes but including it provides a good false-positive case for the model to distinguish. Any more than this may affect model training performance.

3.2.2 Windowing

For RNNs a sequence of time series data is required. For this we will construct a window of time data of length N (seconds at 1 Hz). N is an important hyperparameter to vary and gives insight into the length of time meaningful for detection.

$$x = \begin{pmatrix} x_1(t-N+1) & x_2(t-N+1) & \cdots & x_k(t-N+1) \\ x_1(t-N+2) & x_2(t-N+2) & \cdots & x_k(t-N+2) \\ \vdots & \cdots & \ddots & \vdots \\ x_1(t) & x_2(t) & \cdots & x_k(t) \end{pmatrix} \tag{3.3}$$

The multi-sensor time series window at time t, can be seen in eq. (3.3). Where N is the window length, k is the number of features, $x_i(\cdot)$ is the i-th feature at a given time. These windows are prepared so that they only use data from one node and have a hop of 1 (e.g. our first window is from index $0,1,\ldots,N$ then the second is $1,2\ldots,N+1$ and so on) to maximise the amount of data.

3.2.3 Feature Selection

eCO_2 and TVOC were selected as the only sensor features for the models due to their demonstrated correlation with fire occurrence in previous studies [24] [10]. Other variables such as temperature and humidity may influence false positives, and while their inclusion could enhance model generalisability, it may also increase complexity. Expanding the features within the model provides an avenue for future work.

3.2.4 First-order Differences

To further capture local signal dynamics, two additional features were engineered: the first-order differences of each variable (i.e., ΔeCO_2 and ΔTVOC). These can be seen in eq. (3.4) and **(3.5)**.

$$\Delta eCO_2(t) = eCO_2(t) - eCO_2(t-1) \tag{3.4}$$

$$\Delta TVOC(t) = TVOC(t) - TVOC(t-1) \quad (3.5)$$

These were used within the GGM-VAE. While possible for the model to learn to extract these first-order features, it is common in time series anomaly detection to engineer them and has been shown to improve detection and convergence [29].

3.2.5 Time-Frequency Domain

First-order differences often improve detection performance in time series models because they emphasise rapid changes. However, they also attenuate slow trends and low-frequency structure. To capture information that may be lost by differencing and to further explore this bushfire domain, the time-frequency domain was analysed.

Using Short-Time Fourier Transforms (STFTs), spectrograms, which are time-frequency domain representations, can be produced as can be seen in Figure 3.1. These spectrograms and magnitude plots were produced by averaging STFTs, computed using a 15-second Hann window, as the data is not continuous time series (it contains multiple sensors and multiple experiments).

Comparing the non-burn to burn spectrograms shows significant variance, indicating different frequency response from the burn to non-burn datasets.

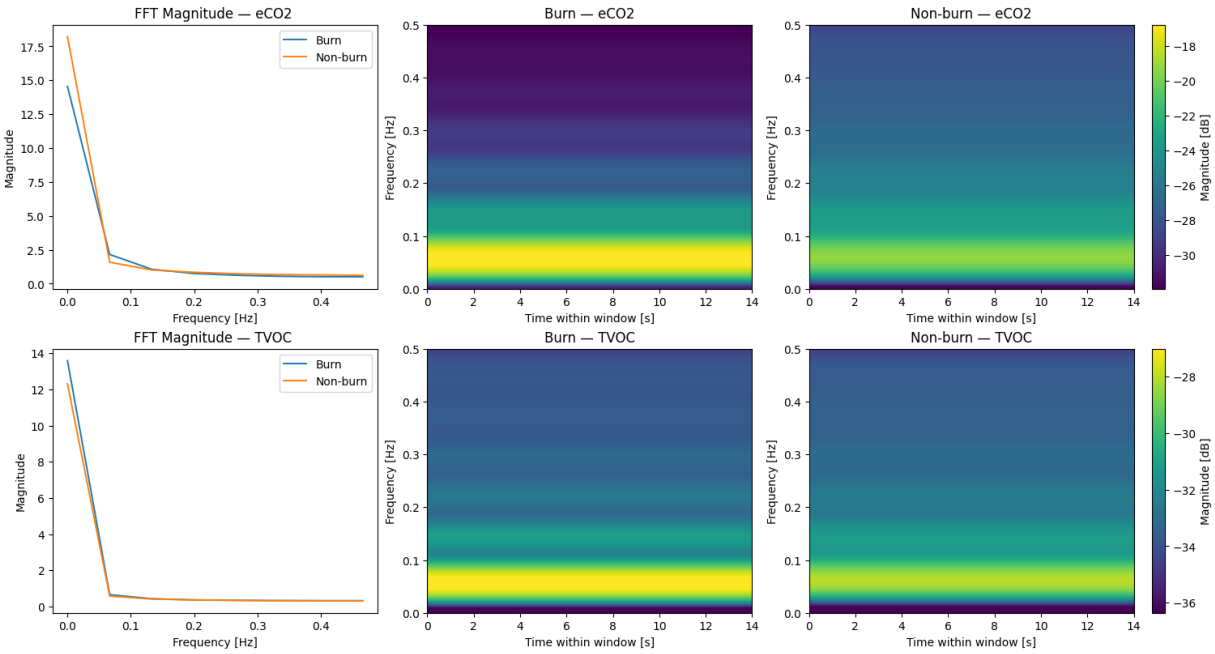

Figure 3.1, Spectrogram of burn and non-burn data

All spectrograms contain significant magnitude components in low frequencies indicating slow, longer-term variance. It is important to note, that the intensity of the burn magnitude is noticeably higher than that of the non-burn data among both TVOC and eCO_2. This is supported by the FFT magnitude plots. It is also clear that the variance between the non-burn to burn for

eCO_2, is much greater than the TVOC, with much more magnitude components clustered around lower frequencies in burn data than non-burn.

This difference in lower frequency components is likely missed by first-order differences and may allow the introduction of a frequency model to benefit. Other time-frequency components may also be present, but as the data consists of multiple sensors in multiple experiments it is difficult to visualise and analyse.

This insight informed the Hybrid-VAE, where STFTs are accepted as an additional input. For each already-windowed time domain data $x[\cdot]$, a time-frequency representation is computed per input channel. In our case this is one for eCO_2 and TVOC. Let $w[n]$ denote a Hann analysis window of length N_s the STFT window size seen in eq. (3.6). The STFT at time index t and frequency bins $\lambda_k = 2\pi k/N_s$ can be seen in eq. (3.7).

$$w[m] = \frac{1}{2}\left(1 - \cos\frac{2\pi m}{N_s - 1}\right), m = 0, \dots, N_s - 1 \tag{3.6}$$

$$STFT_{N_s}(x) = \sum_{\mathrm{m}=0}^{N_s-1} \mathrm{x[n+m]w[m]}e^{-j\left(\frac{2\pi k}{N_s}\right)m} \tag{3.7}$$

This is the computed local spectrum, which is a discrete Fourier Transform on this windowed section. For real-valued signals we retain the one-sided spectrum with $F = N_s/2 + 1$ frequency bins.

For each sensor channel, in our case eCO_2 and TVOC, we compute two independent STFT tensors. The model uses the magnitudes of the STFT and ignores phase as it can be difficult to handle with 2π-wrapping discontinuities and high sensitivity to time shifts [42]. This is treated as a separate input to the time series windows.

3.2.6 Scaling

Robust scaling is applied to each window independently post feature engineering (first-order differences or STFT), using the window's median (M) and interquartile range (IQR) per feature using eq. (3.8) [43].

$$x_{scaled} = \frac{x - M_{window}}{IQR_{window}} \tag{3.8}$$

This per-window scaling preserves the structure of typical background variability while minimising the impact of outliers. Without scaling, the larger-scale features dominate the loss and back-propagated gradients, this forces the optimiser to take very small steps or overfit to a

single channel [44]. This scaling allows for the loss terms to be contributed more evenly and thus decreases convergence time.

Per-window scaling can also be useful for non-stationary sensors. Baselines can shift depending on environmental conditions like temperature, humidity, or device placement, known as sensor drift. This scaling focuses on shape and short-term dynamics making it more consistent across a range of deployment environments.

The model performance was also tested with a global scaling based on the background dataset only, but this proved to be less effective likely as it assumes a stationary distribution across sensors and days.

For the time-frequency representation, the same idea is applied per channel and per frequency bin after a log-magnitude transform to compress dynamic range. This prevents high-energy bands from swamping weaker but informative bands.

3.2.7 Dataset Split

The non-burn windows were split into training, validation and testing datasets with a 70%, 15%, and 15% partition respectively. The training dataset was used for the training of the model's themselves and to set the thresholds. The validation dataset is to limit overfitting and as such, an early stopping constraint of 20 is included for the developed models. The testing set was reserved only for gathering final comparison metrics. The testing dataset was also supplemented with the fire data from the experimental burns, leading to eight possible ignitions. Resulting in 244,640 non-burn data points and 64,671 possible burn data points.

3.3 Pure Thresholding Model

The thresholding model serves as a non-machine-learning baseline against which more sophisticated detectors are compared. Following the local-first, network-later pattern in Zhuang et al. [10], where node-level triggers are subsequently filtered by spatial corroboration to reduce false alarms. We implement a single-variable, local detector using the first-order difference of eCO_2.

Let x_t denote the eCO_2 measurement (ppm) at time t and Δx_t its first-order difference (as defined in section 3.2.4.) The detector raises a local alert when the instantaneous slope exceeds a preset threshold as seen in eq. (3.9).

$$\Delta x_t > \tau \Rightarrow \text{anomaly} \tag{3.9}$$

Using Δx_t (rather than the raw level) remains more robust to sensor drift while remaining sensitive to abrupt rises consistent with smoke/combustion plumes. The threshold is calibrated as previously mentioned.

This method poses some large limitations, pure slope thresholds are sensitive to sensor noise and non-fire transients (human/wildlife proximity, ventilation changes). They also ignore multivariate context and temporal shape beyond immediate slope. Consequently, while this baseline is useful and easy to tune for a target false-alarm rate, we expect it to be outperformed by sequence-aware, multi-sensor models. In a deployed system it is best used as a first-stage trigger, with spatial fusion (as in [10]) or a heavier model verifying the event upstream.

3.4 GGM-VAE Model

The first chosen architecture is a Gated Recurrent Unit - Gaussian Mixture Variational Autoencoder (GGM-VAE), inspired by Guo et al. [25]. It combines a GRU encoder with a Gaussian Mixture Model (GMM) latent space and a decoder network. This section will outline its underlying components, the GRU, VAE and GM-VAE before discussing how these are combined to form the GGM-VAE architecture.

3.4.1 Gated Recurrent Unit (GRU)

A GRU is a form of RNN, with the goal of modelling long-term dependencies by mitigating the vanishing/exploding gradient problem. Compared with LSTMs, GRUs typically have fewer parameters and comparable accuracy, which makes them attractive under compute and data constraints. They also typically require less data to generalise [22]. This makes them more suited when there is a strict computational limit or a small dataset size.

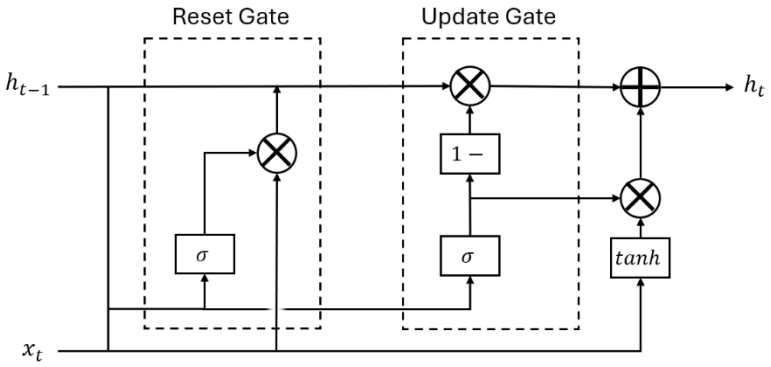

Figure 3.2, GRU Diagram

Figure 3.2 show the flow of information through a GRU. It uses only one state, the hidden state, $h_t \in \mathbb{R}^H$(where H is the hidden state size hyperparameter). The GRU hidden state is responsible for both the long-term and short-term dependencies. Within a GRU cell, there are two gates, the update, and the reset gate [22].

The update gate controls the extent to which past inputs should be preserved in the hidden state. It uses the weighted summation of the current input, x_t, and the previous hidden state, h_{t-1}, then feeds this through a sigmoid activation function, σ, producing values between 0 and 1 as seen in eq. (3.10) [25]. The result, z_t, controls the balance between old and new information. If z_t is close to one it mostly retains old information, conversely if it is close to zero it prioritises new information.

$$z_t = \sigma(W_z \times [h(t-1), x(t)] + b_z) \tag{3.10}$$

The reset gate determines how much of the past information to reset from the hidden state, h_{t-1}. To do this, it also uses a weighted summation of the current input, x_t, and the previous hidden state, h_{t-1}, it then is passed through a sigmoid function as seen in eq. (3.11) [25]. The output of the reset gate, r_t, forgets the hidden state value when it is close to zero. When it is near one this state is retained.

$$r_t = \sigma(W_r \times [\mathrm{h}(t-1), x(t)] + b_r) \tag{3.11}$$

The GRU then generates a candidate hidden state, $\tilde{\mathrm{h}}_t$, this combines the current input, x_t, and the previous hidden state after it is filtered through the reset gate seen in eq. (3.12) [25]. Finally, h_t, is calculated using the candidate hidden state combined with the previous hidden state and z_t from the update gate, eq. (3.13).

$$\tilde{\mathrm{h}}(t) = tan\mathrm{h}(W_{\mathrm{h}} \times [r(t) \odot \mathrm{h}(t-1), x(t)] + b_{\mathrm{h}}) \tag{3.12}$$

$$\mathrm{h}(t) = \big(1 - z(t)\big) \odot \mathrm{h}(t-1) + z(t) \odot \tilde{\mathrm{h}}(t) \tag{3.13}$$

Like a typical RNN, the GRU cell is unrolled to fit the input sequence length which is a windowed sequence of time series data.

3.4.2 Variational Autoencoder (VAE)

Variational Autoencoders (VAEs) are an unsupervised deep learning approach that consist of an encoder and decoder structure [45]. The encoder aims to reduce the dimensionality of the input data. It does that by translating the input data into an L-dimensional latent space (where L is a key hyperparameter), resulting in a latent vector, z. This latent space acts as a bottleneck for the encoder. The latent vector representation is the distilled input data to its most important features. This is then fed into the decoder which converts the latent space representation back to the original size with the aim to perfectly reconstruct the input.

VAEs encode the input data to a probability distribution (usually Gaussian) in the latent space rather than a point. VAEs generate two latent vectors, one for the mean, $\mu \in \mathbb{R}^L$, and one for

the standard deviation, $\sigma \in \mathbb{R}^L$. This allows the position in latent space to have meaning. This means vectors close together in latent space will result in outputs that are similar. This approach can allow VAEs to generate new data by applying small changes to a features latent vector. This makes VAEs less susceptible to noise where small deviations in inputted data due to noise in VAEs can typically still be approximated by the probabilistic distribution within latent space.

An inherent issue with modelling latent vectors using a Gaussian distribution is that it cannot be optimised using backpropagation. VAEs solve this using the reparameterisation trick as seen in eq. (3.14) and (3.15) [45].

$$z = \mu + \sigma \odot \epsilon \tag{3.14}$$

$$\epsilon \sim \mathrm{N}(0, \mathrm{I}_L) \tag{3.15}$$

The latent vector, z, can be represented using the vector of means, μ, and the vector of standard deviation, σ. This trick introduces a new variable, ϵ, which is a random variable following a Gaussian distribution. This allows the mean and standard deviation to be optimised using backpropagation while the random variable is ignored.

For VAEs, the negative log-likelihood loss is used in conjunction with the Kullback-Leibler (KL) divergence, D_{KL}, resulting in the combined loss seen in eq. (3.16) [45]. Where x is the original input data, $\hat{x}$ is the output, $q_\theta(z|x)$ is the encoder distribution, $p_\phi(x|z)$ is the decoder distribution and $p(z)$ is the prior distribution, which in our case is Gaussian. The KL divergence measures the difference between two distributions, in this case the divergence of the latent variable distribution and a Gaussian distribution. Minimising the function encourages the VAE to distribute the latent variables normally, thus leading to the latent structure that characterises VAEs.

$$\mathcal{L}(\mathrm{x}, \hat{x}) = -\mathbb{E}_{q_\theta(z|x)}\big[log p_\phi(x|z)\big] + \mathrm{D}_{KL}\big(q_\theta(z|x)||p(z)\big) \tag{3.16}$$

In the context of anomaly detection, VAEs provide a probabilistic output, rather than just an error. This value can be interpreted more intuitively and thus the methods to separate an anomaly from normal data can be lower complexity.

The structure of the encoder and decoder within a VAE can vary. For example, to incorporate time dependencies and features can be augmented with GRU or LSTM cells. These form architectures that allow for unsupervised time series anomaly detection problems. In this model we augmented the encoder and decoder with a GRU.

3.4.3 Gaussian Mixture Variational Autoencoder (GM-VAE)

Gaussian Mixture Variational Autoencoder (GM-VAE), is an unsupervised anomaly detection model. It extends the standard Variational Autoencoder (VAE), which assumes the latent space follows a single normal distribution. GM-VAE instead models the latent space as a mixture of multiple Gaussians, allowing it to capture multiple modes or clusters [25]. The decoder outputs probabilistic parameters (mean and variance) for reconstructing the input data, enabling probabilistic rather than deterministic reconstruction.

This approach adds complexity to the model but has potential to lower the false positive rate if the input data follows multiple "modes" or clusters as is common in real-world datasets. For example, within the bushfire domain, normal data may follow a different distribution at night than it does during the day, or when humans or animals are nearby.

$$\mathrm{p(z, k)} = \mathrm{p(z|k)} \times p(k) \tag{3.17}$$

The model assumes data comes from eq. (3.17) [25]. Where k is the index of Gaussians out of a total of K Gaussian (where K is a hyperparameter). $p(k)$ is the probability of picking the Gaussian k, and $\mathrm{p(z|k)}$ is the Gaussian for component k with the latent space.

To select the specific Gaussian component from the Gaussian mixture latent space, the categorical distribution (Cat) is used. The categorical operation effectively chooses one of the discrete components, k, based on their respective probabilities $p(k)$. During training, this is done by sampling from the categorical probability distribution determined by the encoder's learned probabilities. This enables the GM-VAE to flexibly model data belonging to different modes or clusters within the latent space [25].

The goal is to reconstruct the input while ensuring the latent space matches the Gaussian mixture structure. This is done by maximising the Evidence Lower Bound (ELBO). ELBO balances the reconstruction quality (how well the output matches the input) and the regularisation (KL Divergence). As such the loss is a sum of the reconstruction loss and KL divergence.

ELBO gives the lower bound of the true log-likelihood (objective to maximise) seen in eq. (3.18) [25]where the entire expression after "≥" is the ELBO. The first term on the right-hand side is the reconstruction term, and the second term is the KL divergence. Where the $\mathbb{E}$ term is the expected value of the average over possible values. In practice it means that we sample the latent variables during training to estimate the expected value.

$$\log \mathrm{p(x)} \;\geq\; \mathbb{E}_{q_\theta(Z|X)}[log\ p(x|z,k)] - \left(q(z,k|x)||p(z,k)\right) \tag{3.18}$$

$q(z, k \mid x)$ is the model's belief about which z and k are likely, given the data x. In practice, since you cannot sum over all possibilities, you sample z and k from $q(z, k \mid x)$ and average their results.

3.4.4 GGM-VAE Architecture

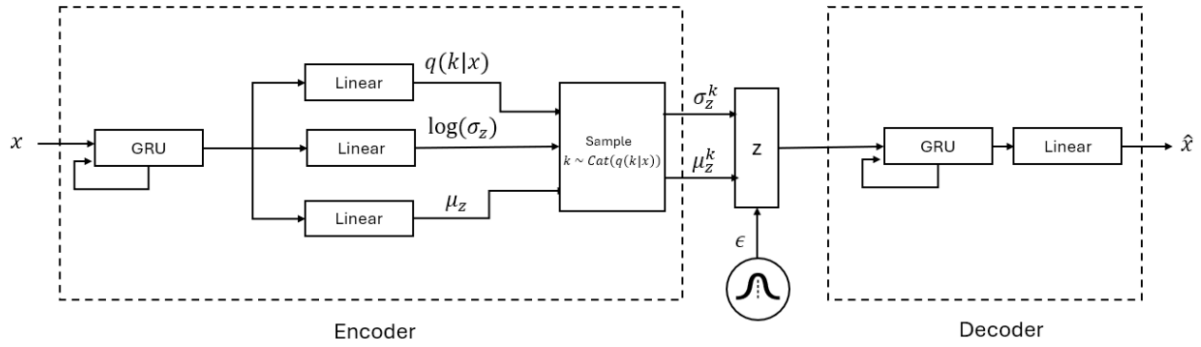

Figure 3.3, GGM-VAE Pipeline

Figure 3.3, illustrates the full GGM-VAE architecture, from pre-processed time series input, $x \in \mathbb{R}^{N \times F}$ (where N is the window size hyperparameter and F is the fixed feature dimension), to reconstruction probability, $\hat{x} \in \mathbb{R}^{N \times F}$. We can see the combination of elements previously presented, the GRU encoder with a Gaussian-mixture latent prior and a GRU decoder. This is consistent with the model developed by Guo et al. [25].

3.4.4.1 Encoder

First, a GRU ingests the window, reads $x_{1:N}$ and outputs the final hidden state, aiming to summarise the temporal patterns. Three linear heads map this hidden state, $h \in \mathbb{R}^H$, to parameters of the mixture posterior:

- **Component Posterior:** $q(k|x) = softmax(W_\pi h + b_\pi) \in \mathbb{R}^K$
- **Per-component mean:** $\mu_z^k(x) \in \mathbb{R}^L$
- **Per-component log-variance**: $log\ \sigma_z^{2,k}(x) \in \mathbb{R}^L$

These are produced as tensors with shape (B, K, L) for μ and $\log \sigma^2$ and (B, K) for $q(k|x)$, where B is the batch size used in training. We then sample a hard component as seen in eq. (3.19).

$$w \sim q(k \mid x) \tag{3.19}$$

Given the selected component w, we sample each latent dimension independently from a 1-D Gaussian with its own mean and variance as seen in eq. (3.20).

$$z_i \sim \mathcal{N}\left(\mu_{z,i}^{(k)}(x), (\sigma_{z,i}^{(k)}(x))^2\right), i = 1, \dots, L. \tag{3.20}$$

Using the reparametrisation trick we get the reparametrised form as previously seen in eq. **(3.15)**.

3.4.4.2 Decoder

To reconstruct the window, z is tiled across time ($z \mapsto [z, \dots, z] \in \mathbb{R}^{N \times L}$), passed through a GRU and a linear layer, yielding $\hat{x} \in \mathbb{R}^{N \times F}$. This matches the "GRU→Linear" block in Figure 3.3.

3.4.4.3 Loss

The loss or training objective is to minimise to the negative ELBO. The positive ELBO is seen in eq. **(3.18)**.

3.4.4.4 Anomaly decision and deployment

We train on non-burn windows from the training dataset. At inference we use a latent-only score seen in eq. (3.21).

$$s(x) = D_{KL}\big(\mathrm{q}(k \mid x) \parallel \mathrm{p}(k)\big) + D_{KL}\big(\mathrm{q}(z \mid x, k) \parallel \mathcal{N}(0, I_L)\big) \tag{3.21}$$

This is calibrated to a chosen percentile on training non-burn data. Thresholding $s(x)$ yields an anomaly flag which we will call a positive detection. Because $s(\cdot)$ depends only on the encoder heads and the learned prior $p(k)$, the decoder can be removed on the edge device to reduce compute and memory.

3.5 Hybrid-VAE Model

Building on the GGM-VAE presented above and attempting to explore varying areas of the bushfire detection domain, the Hybrid-VAE incorporates frequency domain modelling in addition to the time domain modelling. This section will present any additional foundations needed for this model, namely a CNN and then discusses how the complete architecture is structured.

3.5.1 Convolutional Neural Network (CNN)

Convolutional neural networks (CNNs) are a form of artificial neural network designed to learn spatial features from input data. The core of their operation is the convolutional layer, in which learnable filters, known as kernels, scan across the data to produce a feature map from the convolution of the kernel and the data subset [46]. CNNs have proved to be well-suited for structured 2D data types such as images, and spectrograms such as in our case.

Each kernel can learn specific patterns such as edges, textures, or key frequency components. The deeper the CNN layer typically the more abstract the representation. In this work, the spectrogram is arranged as frequency by time. A 2D convolution captures short time-frequency features. This may be narrowband burst, or harmonics. More abstract representations from lower layers may prove useful at lowering the false positive rate.

Another key component of CNNs are pooling layers, in which the spatial dimension of the feature map is reduced to summarise the presence of features [47]. Max pooling is most used as it preserves the most prominent feature by taking the maximum value in a small region. This aims to make the data more robust to the exact location of the signal and thus more robust to noise. Average pooling is also commonly used for its ability to be more robust to noise and is often used as a pooling layer towards the end of a CNN [47].

3.5.2 Hybrid-VAE Architecture

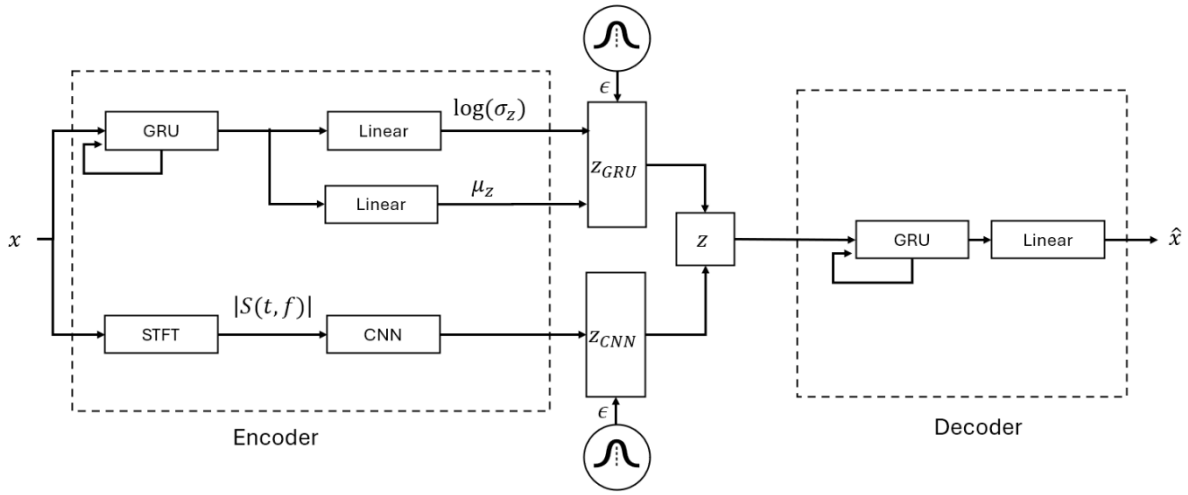

Figure 3.4, Hybrid-VAE Pipeline

The Hybrid-VAE model combines a time domain GRU-VAE and a frequency domain CNN that operates using STFT magnitudes. This essentially forms a STFT-CNN-GRU-VAE. The input is a windowed multivariate time series $x \in R^{N \times F}$. In parallel we compute $|S(t,f)|$, the STFT magnitude spectrogram for the same window (Section 3.2.5). The full model pipeline can be seen in Figure 3.4. The same hyperparameters as are in the GGM-VAE are used in the Hybrid-VAE, with some additions for the CNN and without the number of Gaussians, K.

3.5.2.1 Encoder

There are two encoder branches:

1. GRU branch. A GRU processes x and produces z_{GRU} using the reparameterisation trick as covered in Section 3.4.2. This branch is responsible for the time domain processing.
2. CNN Branch. A small CNN uses $|S(t,f)|$ as its input. We pool over time, apply global average pooling over (f,t), and then use linear layers to produce the parameters for a

second probabilistic latent vector, $z_{CNN} \in \mathbb{R}^L$. This branch also uses the reparameterisation trick as seen in eq. (3.22) with its mean μ_{CNN} and log-variance ($\log \sigma^2_{CNN}$).

$$z_{CNN} = \mu_{CNN} + \sigma_{CNN} \odot \varepsilon \tag{3.22}$$

Table 3.3, CNN Layers

Layer	*Type*	*Kernel Size*	*Stride*
1	Conv2d	3x3	
2	ReLU	-	-
3	MaxPool2d	1x2	1x2
4	Conv2d	3x3	
5	ReLU	-	-
6	AvgPool2d	1x1	
7	Flatten	-	-
8	Linear → μ_{CNN}	-	-
9	Linear→ $\log \sigma^2_{CNN}$	-	-

The construction of the CNN can be seen in Table 3.3. This design keeps the network small, which limits overfitting and keeps compute small for hardware implementation, while still allowing the convolutions to learn local spectral motifs. Time-only pooling preserves frequency resolution when F is small, and the global average pooling produces a fixed-size descriptor regardless of the STFT parameters. Additionally, the output channels of both convolutional channels are hyperparameters, that is C_1 and C_2.

The two latent samples are fused by a linear layer so, the decoder sees a single L-dimensional code. This is seen in eq. (3.23), where $W[\cdot]$ is the trainable linear weight matrix.

$$z = W[z_{GRU}; z_{CNN}] + b \tag{3.23}$$

3.5.2.2 Decoder

The decoder repeats z across time, feeds it through a GRU and a linear layer, and outputs a reconstruction $\hat{x} \in R^{N \times F}$.

3.5.2.3 Loss

The training for this model is again unsupervised and follows the VAE objective. Eq. (3.24) gives the time reconstruction loss, $\mathcal{L}_{NLL}(x, \hat{x})$.

$$\mathcal{L}_{NLL}(x,\hat{x}) = -\frac{1}{\mathrm{NF}}\sum_{n,f}\log\left(\mathrm{p}(x_{n,f}|\hat{x}_{n,f},\sigma^2)\right) \tag{3.24}$$

For the STFT-CNN a new loss term is introduced, a spectral STFT loss, $\mathcal{L}_{MR-STFT}(x,\hat{x})$, much like how the reconstruction loss operates we take the reconstructed signal $\hat{x}$, and perform an STFT to compare to the original [48]. Eq. (3.25) shows the single-resolution STFT loss where $\mathrm{S} = \left|STFT_{N_s}(x)\right|$ and $\hat{S} = \left|STFT_{N_s}(\hat{x})\right|$. The first term in the loss function is spectral convergence and the second is log-mag L_1.

$$\mathcal{L}_{STFT}(x,\hat{x}) = \frac{\left\|S - \hat{S}\right\|_F}{\|S\|_F} + \left\|\log(S) - \log\left(\hat{S}\right)\right\|_1 \tag{3.25}$$

The total loss function, $\mathcal{L}(x,\hat{x})$, is given in eq. (3.26). This combines the NLL loss, the STFT loss, and the KL divergence terms from both encoder branches. The weights β and α_{spec} balance latent regularization and spectral fidelity.

$$\begin{aligned}\mathcal{L}(x,\hat{x}) = \ & \mathcal{L}_{NLL}(x,\hat{x}) \\ & + \beta\Big(D_{KL}\big(q_{gru}(z|x)\|\mathcal{N}(0,I)\big) \\ & + D_{KL}(q_{CNN}(z||S|)\|\mathcal{N}(0,I))\Big) + \alpha_{spec}\mathcal{L}_{STFT}(x,\hat{x})\end{aligned} \tag{3.26}$$

We compute KL for each branch's posterior against $\mathcal{N}(0, I_L)$ and sum them. In implementation, we normalise this sum by $L \times 2$ (latent size times number of branches) before multiplying by β. This keeps the magnitude comparable when changing latent size.

3.5.2.4 Anomaly decision and deployment

Anomaly detection for the Hybrid-VAE works similarly to the GGM-VAE outlined in Section 3.4.4.4. The update anomaly score equation can be seen in eq.(3.27).

$$s(x) = D_{KL}(\mathrm{q}_{GRU}(\,z \mid x\,) \parallel \mathcal{N}(0, I_L)\,) + D_{KL}(\mathrm{q}_{CNN}(\,z \mid s\,) \parallel \mathcal{N}(0, I_L)\,) \tag{3.27}$$

3.5.2.5 Alternative placement of the CNN

An alternative integration of the CNN to the GRU-VAE is to compute features from $|S(t,f)|$ and concatenate them into the original input x, then feed the result into a single GRU-VAE, providing an early fusion of models. We keep the parallel-branch design because the STFT already captures short temporal context inside each time-frequency patch, and the CNN can learn frequency-selective patterns independently of the recurrent encoder. If the frequency representation had no time axis, such as a plain FFT, early fusion before the GRU might be more appropriate, such as in literature [31].

3.6 Model Results and Analysis

In this section the performance of the unsupervised deep learning approaches, the GGM-VAE and the Hybrid-VAE, will be compared to the simple thresholding baseline model. This will be compared using the metric developed for fair comparison. Additionally, implications for deployment will be discussed. All performance metrics are gathered using the testing dataset, with the threshold and the model's training using the training dataset.

3.6.1 Parameter Optimisation

To optimise the hyperparameters for the GGM-VAE and the Hybrid-VAE, 90 trials were run for each. The hyperparameter range can be seen in Table 3.4 for GGM-VAE. The model converged on only one Gaussian component, meaning that the model is effectively running as a GRU-VAE without any Gaussian mixture modelling. This indicates that the data likely has no significant variant case/false-positive case or that it is not present in enough data points. Additionally, a 15-second window is selected, which is relatively short. This was selected over a shorter 10 second window indicating that it is extracting more useful information from this period. This indicates that useful information for bushfire detection occurs roughly in these 15 seconds, or the model has difficulty extracting key information from longer sequence sizes.

Table 3.4, GGM-VAE Hyperparameter Range and Optimal Value

Hyperparameter	*Range*	*Optimal*
Window Size	10,15,20,30,60	15
Number of Gaussian Components	1 - 8	1
Hidden Size	16,32,64,128	16
Latent Dimension	1,4,8,16,32,64	1
KL-weight	0.01 - 10	2
Learning Rate	0.0001 - 0.001	0.0003
Sigma	0.1 - 3	0.91

For the GGM-VAE the evolution of the optimiser can be seen in Figure 3.5. As can be seen, the F1-score peaks at 0.57. Since there were multiple hyperparameters that result in the same F1-score, the model with smallest parameter size while hitting this maximum F1-score was selected and is presented throughout.

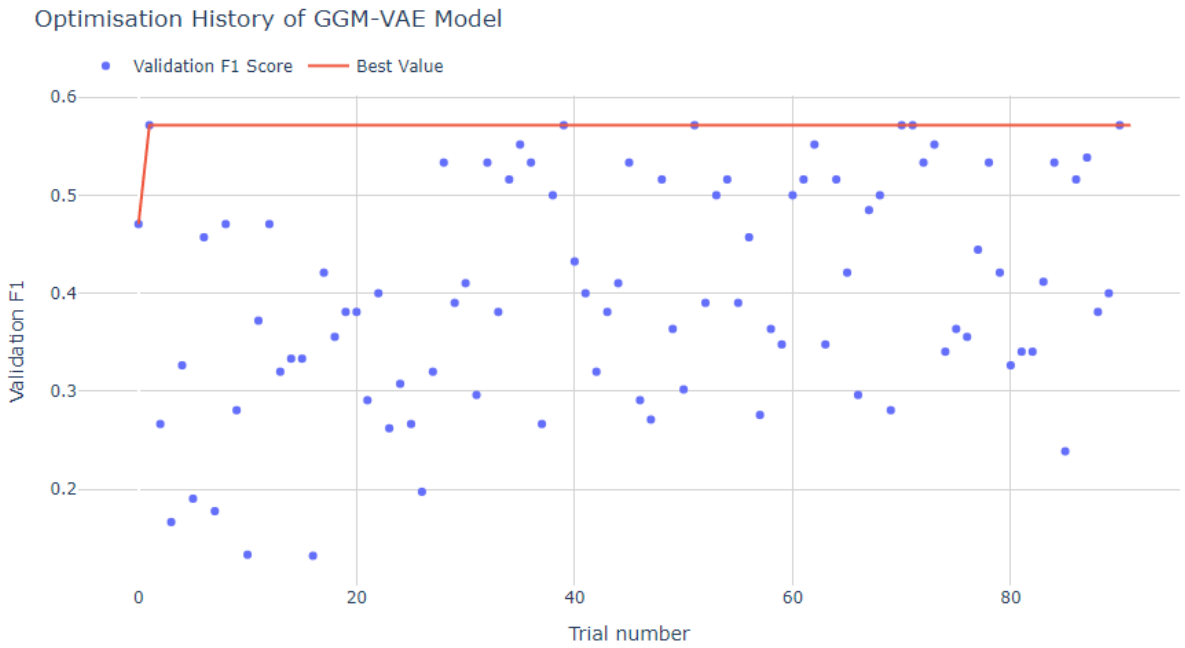

Figure 3.5, GGM-VAE Optuna Trial F1-score Evolution

A maximum F1-score of 0.57 was reached very early on (Trial number = 2) in the optimisation process. This supports the notion that the model is relatively robust to hyperparameter selection. There are a few key hyperparameters that have emerged from this optimisation, with window size being most impactful on performance. The optimisation never exceeded a maximum value F1-score of 0.57. This will have been influenced by the resolution of the script that determines threshold, as well as the resolution of the data itself. As all tests were evaluated with the same random seed, to ensure repeatability, it is possible that certain outliers were left in the testing script that could not be resolved. Only very limited outlier removal was done, namely the first 20 minutes of recording being removed. This may have left large spikes in the data whether legitimate, or a result of factors such as sensor calibration. Nevertheless, this still represents real data that was collected over extended periods and is useful to separate model performance more.

The hyperparameter range and optimal values for the Hybrid-VAE can be seen in Table 3.5. There are many more hyperparameters and more scope for this model to increase in size when compared to the GGM-VAE. Interestingly the model converges on small values for window size and thus subsequently the STFT transform window will be of size 1. This likely indicates the ineffectiveness of the STFT addition. This differs from the STFT analysis conducted earlier and may be due to the architecture implementation, where it may be poorly incorporating the STFT. This may have resulted in the model struggling to resolve the STFT information and adding nothing but noise. It also may have largely increased training difficulty, which would require additional data to better generalise.

Table 3.5, Hybrid-VAE Hyperparameter Range and Optimal Value

Hyperparameter	*Range*	*Optimal*
Window Size	1,4,6,8,15,20,30,60	2
STFT Ratio	0.1,0.25,0.5,0.75	0.5
Hidden Size	1,8,16,32,64,80,96,128	64
Latent Dimension	1,4,8,16,32,64	32
CNN 1 size	1,4,8,16,24	1
CNN 2 size	1,8,16,24,32	1
KL-weight	0.1-5	1.39
Learning Rate	0.0001 - 0.001	0.000186
Sigma	0.1 - 10	4.78

The increase in the Hybrid-VAE size not only increases individual model training times but also increases the Optuna optimisation time as can be seen in Figure 3.6. This history graph shows a much steadier increase in F1-score. This indicates that the model's performance is significantly more sensitive to hyperparameters that the GGM-VAE. We don't observe a score as high as the GGM-VAE, and consequently, we don't see the same plateau in the F1-score, which was likely due to the data itself.

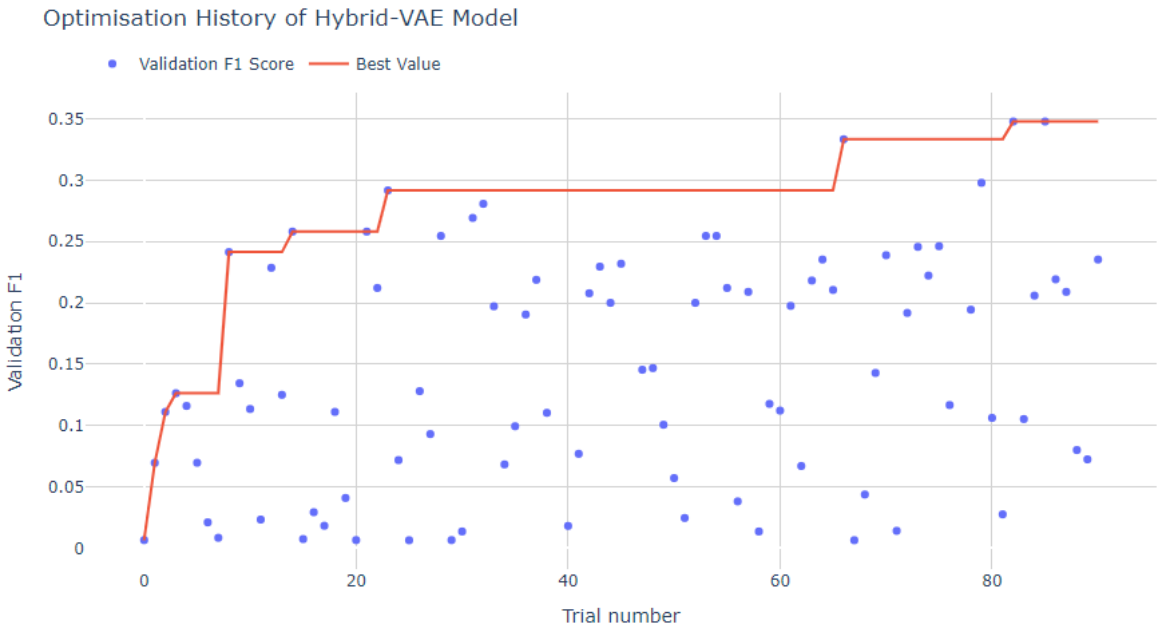

Figure 3.6, Hybrid-VAE Optuna Trial F1-score Evolution

In Figure 3.6,a peak F1-score of 0.35 can be seen with two hyperparameter combinations reaching this. Similarly, the model with the smallest parameter size was for further analysis.

3.6.2 F1-score and True and False Positive Rates

F1-score, false-positive rate (FPR), and true-positive rate (TPR) for the thresholding, GGM-VAE and Hybrid-VAE can be seen in Table 3.6.

Table 3.6, Models F1-score and true and false positive rates

Model	*F1-Score*	*FPR*	*TPR*
Threshold	0.1468	0.2535%	100%
GGM-VAE	0.5714	0.0382%	100%
Hybrid-VAE	0.3478	0.0763%	100%

As the threshold for all models is set to detect all fires the TPR is always 100%. As was previously discussed the F1-score is highest for the GGM-VAE, then Hybrid-VAE then the thresholding method. This metric gives some insight to how well the model is distinguishing burn from non-burn data. Whilst the Hybrid-VAE is significantly outperformed by GGM-VAE, it still outperforms the thresholding-based model by a large margin. These trends are all reflected in the FPR. We see approximately a halved FPR in the GGM-VAE compared to the Hybrid-VAE and an almost 8× reduction compared to the thresholding model.

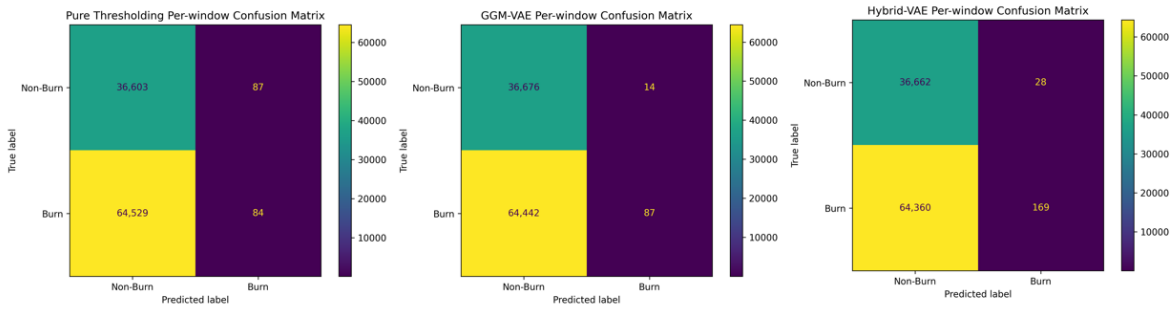

Figure 3.7, Confusion Matrix For GGM-VAE and Hybrid-VAE

Ultimately the goal is to differentiate burn from non-burn data correctly and whilst the evaluation metrics used have given insight and allowed for easy and fair comparison we also should look at how they differentiate this data. Figure 3.7, shows the confusion matrix for both the GGM-VAE and the Hybrid-VAE. This shows the number of windows predicted as a Burn or Non-burn compared to that actually containing a burn or non-burn. It is important to consider the ground truth ambiguity so "True label" is partially misleading, and the burn label refers to potentially containing burn data. Taking that into account, looking at the burn predicted label on the GGM-VAE we can clearly see a significant rate difference between these windows. This indicates that the model is correctly predicting a higher proportion of burns in the burn data, about 4× more than the non-burn data. This also happens in the Hybrid-VAE, however to a lesser extent. Generally, the GGM-VAE threshold is a lot less sensitive where it detects 50% fewer fires and non-fires compared to the Hybrid-VAE. Conversely, the thresholding model has a greater rate of false positives to true positives on a per-window basis. The difference between these rates is very small, and likely indicates this model is operating almost as a random predictor.

Furthermore, is that the GGM-VAE only has 14 false-positives. With a window size of 15 seconds this could likely be due to 1 or 2 anomalous data points. The 87 correct burn predictions compared to the 64 thousand data points from the GGM-VAE highlights the ground-truth

ambiguity issue. Only a very small subset of this data contains information to distinguish a burn, this further supports taking unsupervised learning approaches.

3.6.3 Detection Times

With the goal of detecting fires faster than traditional methods the average detection time for each model is presented in Table 3.7.

Table 3.7, Models average detection time

Model	*Average Detection Time*
Threshold	473.5 s
GGM-VAE	235.5 s
Hybrid-VAE	230.4 s

We see a slightly different trend, with the Hybrid-VAE outperforming the GGM-VAE. This is likely due to the more than double number of true-positive predictions in the Hybrid-VAE over the GGM-VAE. Both models still outperform the thresholding-based model by more than 200 seconds.

One thing to note, is that the models detect fire in an average of about 4 minutes, this is not representative of a deployment detection time. This is due to how the threshold for these models were set and will be discussed further in Section 3.6.5.

3.6.4 Model Complexity

The model complexity metrics are presented in Table 3.8. Unfortunately, the thresholding method cannot be evaluated as it lacks floating-point operations and parameters, aside from the threshold itself. This does mean that the thresholding method, while not having any metrics, is by far the least computationally complex. We can compare the GGM-VAE and the Hybrid-VAE and see a massive difference.

Table 3.8, Model Complexity - Full Models

Model	*FLOPs*	*Total Parameters*	*Model Size (KB)*
Threshold	-	-	-
GGM-VAE	67,780	2,087	8.17
Hybrid-VAE	1,721,000	29,229	128.90

For FLOPs, we can see about a 22× increase from the Hybrid-VAE over the GGM-VAE. This will make the Hybrid-VAE significantly slower when making inferences on the same hardware which poses a threat to MCU-class devices. This is supported by the total number of parameters and the model sizes. Which both take around a 15× increase. This may make it difficult to package the Hybrid-VAE on hardware without optimisations.

It is important to note, that as an inference is made within the latent dimension, the decoders for both models are not needed for deployment. Table 3.9, shows the complexity metrics with only the encoders.

Table 3.9, Model Complexity - Encoder only.

Model	*FLOPs*	*Total Parameters*	*Model Size (KB)*
Threshold	-	-	-
GGM-VAE	35,136	1,107	4.08
Hybrid-VAE	860,592	14,739	64.25

Here we can observe similar trends with the values being about halved. The variance from an exact half comes from the differences in encoder and decoder structures. This difference can vary to much greater extents depending on the hyperparameters.

3.6.5 Effect of the Threshold

As has been mentioned previously the threshold set for all the models for comparison above has been done to achieve the lowest FPR given all fires are detected. This does mean the model is on the cusp of being just sensitive enough to detect all the fires. This is almost certainly not desirable for deployment. As we only have 8 ignitions there is a high likelihood that real fires will be missed. While the FPR achieved is good, it is likely to be further reduced with the introduction of spatial-relation modelling as demonstrated by Zhuang et al. [10]. It is also beneficial to increase the sensitivity to achieve faster detection. To investigate the effects of the threshold on detection time and performance, the threshold was varied to be more sensitive from the value used in previous sections.

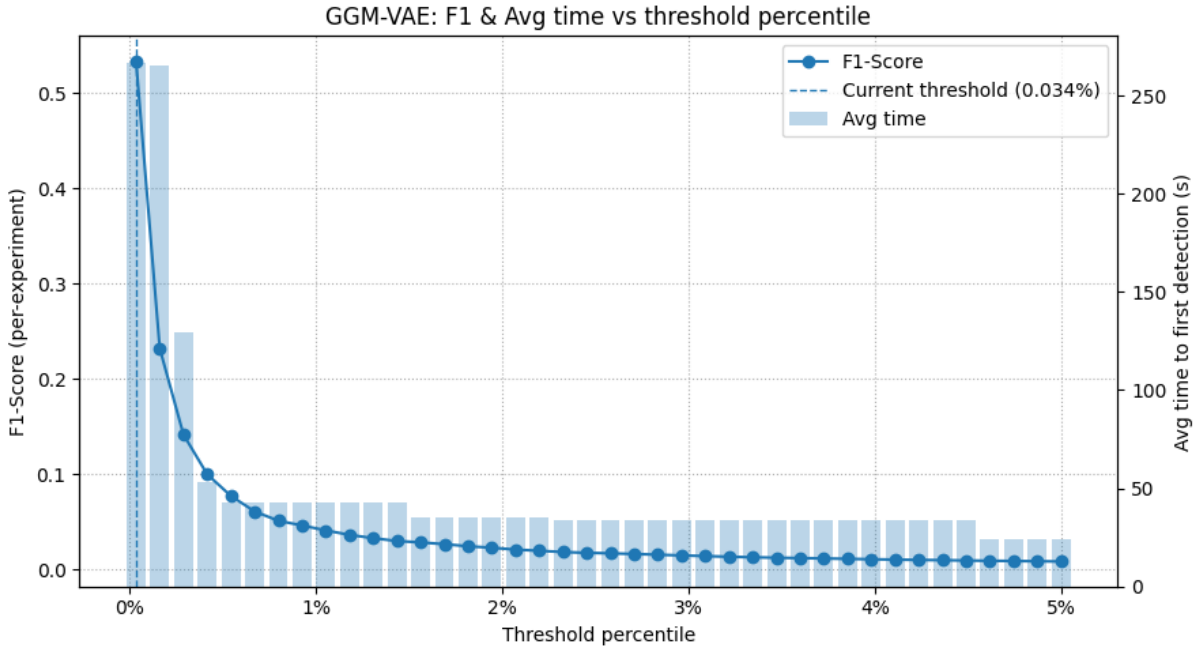

Figure 3.8, GGM-VAE: F1-score vs average detection time vs threshold percentile

In Figure 3.8, we can visualise the relationship for the GGM-VAE between the percentile threshold and the average detection time and F1-score. We can clearly see a strong correlation between F1-score and detection time, and exponential decay for the percentile threshold. This relationship can be used in conjunction with further research to determine what a suitable local threshold for real-world deployment as it is dependent on the target metrics for the system as a whole.

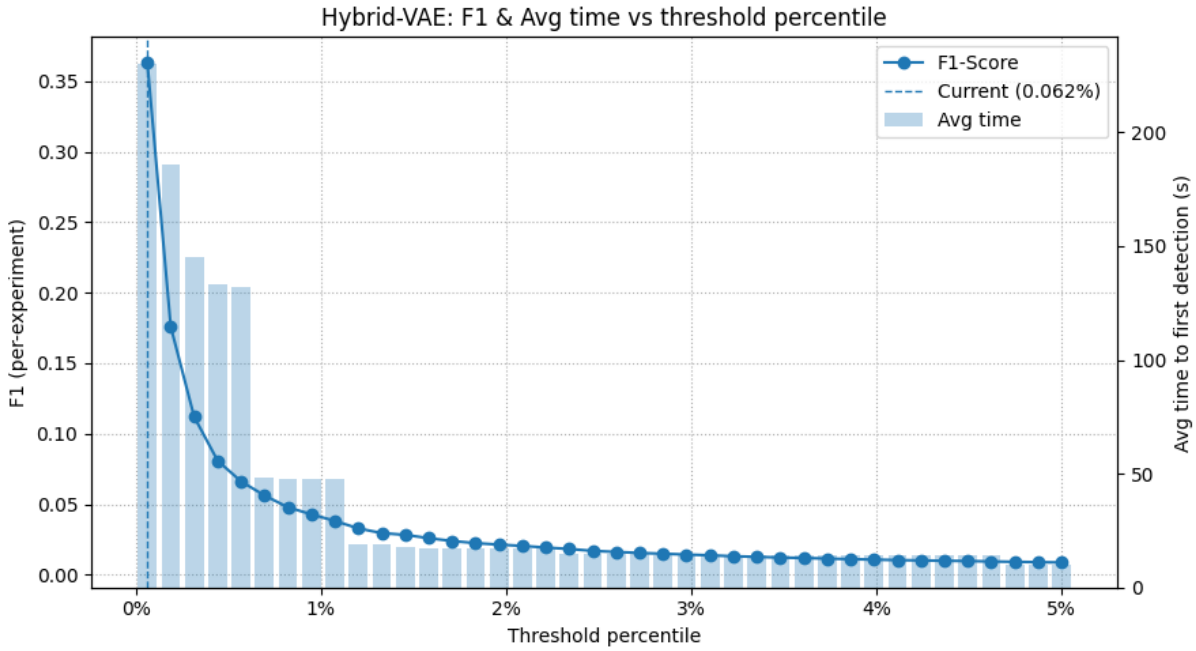

Figure 3.9, Hybrid-VAE: F1-score vs average detection time vs threshold percentile

The trend continues in Figure 3.9 for the Hybrid-VAE. Although we see a much less steep decline in F1-score with a similar reduction in detection time. Direct comparison of both models show that the GGM-VAE can achieve lower detection times given the same F1-score. Highlighting once again, the GGM-VAE's superior ability to differentiate burn from non-burn data.

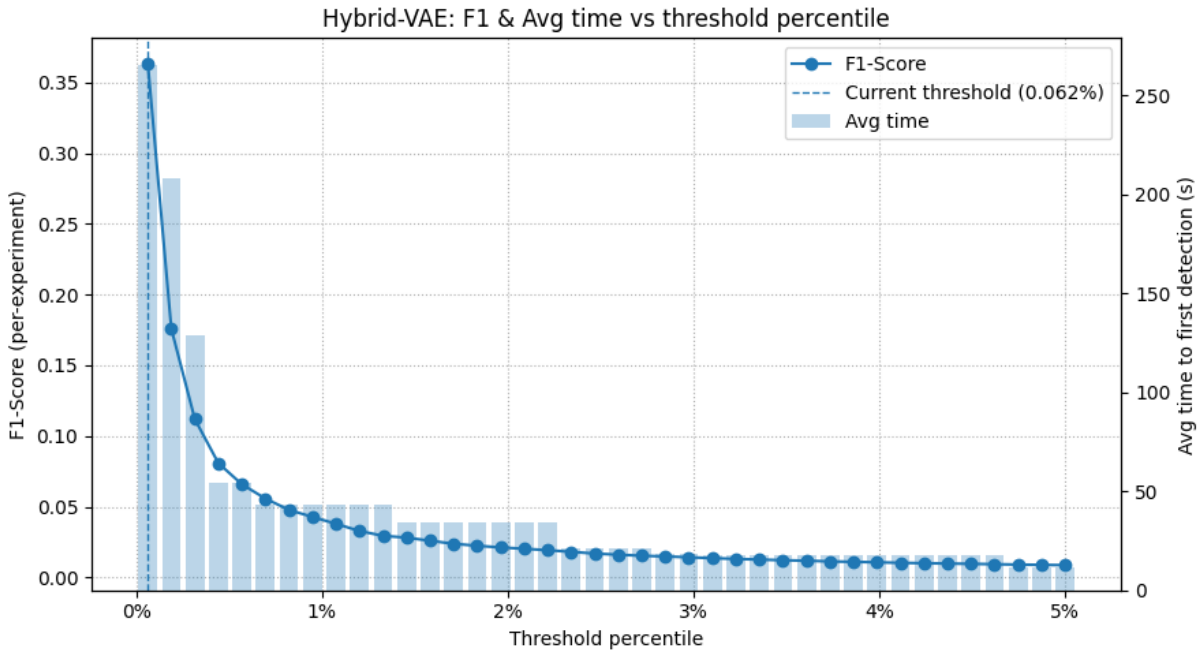

Figure 3.10, Thresholding: F1-score vs average detection time vs threshold percentile

Figure 3.10 shows the relationship for the thresholding method. We see an expected trend and overall lower F1-Scores and higher detection times than the other two models.

3.6.6 Summary of Results

Overall, the GGM-VAE delivered the best results in almost all evaluated metrics. With the 'detect all fires' operating point it achieved the highest F1-score (0.5714), and the lowest false-positive rate (0.0382%), outperforming both the Hybrid-VAE (F1-score 0.3478, FPR 0.0763%) and the thresholding baseline (F1-score 0.1468, FPR 0.2535%). In terms of average time-to-first detection, the Hybrid-VAE was fastest (230.4s), followed by GGM-VAE (235.5 s), with the threshold method slowest (473.5s) showing that the deep models activate significantly earlier than simple thresholds. Whilst the Hybrid-VAE outperforms in detection time than the GGM-VAE in this comparison we also saw that if F1-scores were matched the GGM-VAE would likely match or outperform.

Hyperparameter optimisation reinforces these trends. For the GGM-VAE, Optuna quickly plateaued at the best F1-score (≈0.57) and converged to K = 1 mixture component, tiny latent (1-D) and small hidden size (16) with a 15-second window, effectively a GRU-VAE. This suggests the available "normal" data are largely unimodal at the chosen window scale, or that there is insufficient coverage of distinct regimes to justify a multi-component prior. In contrast, the Hybrid-VAE's trials improved more gradually and were sensitive to hyperparameters, converging to very short windows and unit-size CNN kernels, indicating that the STFT path contributed little under current settings and data resolution. This differs from the STFT analysis conducted earlier and likely suggests either poor implementation or more benefits resulting from the first-order differences.

Complexity measurements underline the edge feasibility of the GGM-VAE. It requires ≈35.1 k FLOPs and ≈1.1 k parameters, whereas the Hybrid-VAE is ≈24× heavier in FLOPs (860.6 k) and ≈14× larger in parameters (≈14.7 k). Given limited MCU-class compute, the GGM-VAE is clearly the more practical candidate.

Error characteristics align with the aggregate metrics. The GGM-VAE produced 12 false positives across the evaluation set (15 s windows), while correctly concentrating "burn" predictions within burn segments by roughly 4× compared to non-burn segments. Hybrid-VAE shows weaker separation consistent with its higher FPR. These patterns should be interpreted alongside ground-truth ambiguity, reinforcing the case for unsupervised temporal modelling.

Finally, varying the operating threshold shows the expected trade-off between detection speed and false-alarm control: lower percentiles yield earlier detection but degrade F1-score via rising FPR. The GGM-VAE's curve is smooth and monotone, enabling principled calibration of an implementation and site-specific operation.

3.7 Limitations of Approach

This chapter developed and evaluated two unsupervised detectors (GGM-VAE and Hybrid-VAE) against a thresholding baseline. Despite promising results, limitations and threats to validity necessitate further hardware implementation and testing.

3.7.1 Data Coverage and Ground Truth

The dataset comprises a small number of experimental burns (eight ignitions) and background recordings from a limited set of sites. This produced two constraints. First, event scarcity limits statistical power: metrics may be sensitive to a few events. Second, label ambiguity in outdoor sensing introduces noise into the "true" class, which can inflate false positives or mask early detections. Excluding prescribed burns and smoke-machine trials improved domain consistency but further narrows distributional coverage (e.g., different fuels, seasons, microclimates). As a result, generalisation to other environments is not guaranteed.

3.7.2 Model Assumptions and Behaviour

The GGM-VAE consistently collapsed to K = 1 component, effectively behaving as a GRU-VAE. This may reflect truly unimodal non-burn behaviour at the 15 s scale or simply insufficient coverage of distinct regimes (e.g., day/night). Either way, the current configuration does not yet validate the hypothesised benefit of a multi-modal prior for field data. For the Hybrid-VAE, the optimiser favoured very short windows and unit-size kernels, implying the STFT branch contributed little signal under the chosen transforms and training budget. This could be due to over-compressed spectrograms, or limited data to learn stable time-frequency filters.

3.7.3 Optimisation and Evaluation Protocol

Hyperparameters were explored with 90 trials per model and a single random seed for repeatability. This budget is modest for the Hybrid-VAE's larger search space and thus conclusions about its relative performance may change with broader search. This was primarily due to the massive compute requirement for training the Hybrid-VAE, with a full Optuna trial often taking a week on available hardware. Additionally, the operating point for all models was chosen to detect all fires (TPR = 100%), which while helpful for fair comparison, does not reflect the final trade-off operators may choose.

3.7.4 External Validity and Spatial Fusion

This chapter evaluated temporal, single-node detectors. In practice, spatial corroboration across nodes can dramatically reduce false alarms and stabilise operating points, but this was outside the scope of this modelling. The absence of network-level logic means that some seemingly high FPRs may be acceptable once spatial fusion is applied, and conversely some early detections may be delayed by network protocols in deployment.

3.8 Chapter Summary

This chapter developed two unsupervised detectors for early fire detection, first a GRU Gaussian-Mixture VAE (GGM-VAE) and then a Hybrid-VAE with an STFT-CNN branch. These were benchmarked against a simple thresholding baseline. At the 'detect-all-fires' operating point, the GGM-VAE achieved the strongest discrimination (highest F1-score, lowest FPR), while the Hybrid-VAE triggered earliest on average, albeit with more false alarms. Encoder-only, latent-space scoring makes the GGM-VAE particularly attractive for MCU-class deployment, given its substantially lower FLOPs and parameter count.

The study also highlighted important constraints: limited event counts and label ambiguity, mixture collapse to $K = 1$ (reducing the GGM-VAE to a GRU-VAE in practice), and a weak contribution from the current STFT branch under the chosen settings. These factors shape how operating thresholds should be selected and highlight the need for spatial corroboration in a full system.

The next chapter turns to hardware implementation, quantifying latency, memory, and energy on MCU-class targets, and exploring compression to close the gap between algorithmic feasibility and field deployment.

Chapter 4 Hardware Implementation

This chapter describes the methodology and results from implementing the two deep-learning models from Chapter 3 on existing MCU-class hardware. The aim is to assess feasibility on constrained IoT edge devices and to identify the constraints these devices impose on model choice and configuration.

4.1 Methodology

The hardware used in this project is the current ANU bushfire sensor node which is a custom PCB built around a RAK3172 LoRa module and SGP30 and BME680 previously designed by Pulsford [49]. Figure 4.1 shows a labelled image of the hardware used.

Figure 4.1, ANU IoT Sensor Hardware

The RAK3172 is based on the STM32WLE5CC MCU and supports long range (LoRa) wireless transmission using the LoRaWAN protocol [50]. It also supports operation and configuration over UART from an external device.

The SGP30 is a gas sensor originally made for indoor air quality. It provides measurements for TVOC and eCO_2. It works by heating a micro hotplate and using a metal-oxide gas sensor to make H_2 and ethanol readings which are then converted to the final metrics. This provides low sensor drift and long-term stability [51]. The BME680 provides measurements for barometric pressure, temperature, relative humidity (RH), and air quality [52].

This allows for translation and implementation of the developed models onto the hardware. We will translate the GGM-VAE, Hybrid-VAE, and pure thresholding method from Python used for development to C++ used for the hardware. For this project many of the features of the board are outside the scope, namely the LoRa communication. The LoRaWAN libraries are still linked so that the final binary size reflects a realistic deployment footprint. Additionally, we only use TVOC and eCO_2 in the models developed, although the BME680 does still play a role

in periodically calibrating the SGP30 RH to improve accuracy. Thus, both sensor libraries are included and necessary.

4.1.1 Constraints

The MCU, the sensors and the complete package have several constraints that need to be managed.

4.1.1.1 ROM Memory

Read only memory (ROM) is non-volatile flash memory that contains the whole application. This includes the startup code, drivers, LoRa stack, model code and its weights. The STM32WLE5CC has 256 KB of ROM [53], although the size of the ROM that can be written to for the program is only 200 KB.

4.1.1.2 Power

The hardware is designed to be powered by a 3.7V Lithium Polymer (LiPo) battery with eventual support for a solar panel. While no specific power capacity, it is important to consider usage as low-cost battery size is limited, and inexpensive solar panels are unlikely to provide significant power. The inference must not materially increase power draw.

4.1.1.3 Sampling Frequency

The limit on sample frequency will be derived either from the inference time, that is the time that it takes to make a prediction, or from the limits from the sensors. The SGP30 has a maximum gas sensing frequency of 40 Hz but more relevant for our use is the maximum air quality sampling frequency of 1 Hz [51]. This corresponds to the frequency the models were trained for.

4.1.2 Performance Metrics

To assess the feasibility of implementing deep learning models on this hardware, a few metrics will be measured, to check against maximum values and to compare to one another.

4.1.2.1 Memory Usage (KB, % of Flash)

The MCU is very memory limited, so assessing usage once all libraries and model weights have been installed is important to limit the number of model parameters. Thus, we will track the ROM memory usage both in kilobytes (KB) and as a percentage of maximum capacity (200 KB). This is tracked by the Arduino IDE after it has compiled the code.

4.1.2.2 Inference Time (ms)

We will measure inference time, that is, the time it takes the model to make a prediction. This will indicate whether the MCU can compute the model without losing information. Essentially, we want to determine whether the MCU can make an inference within its sample period and thus maintains the sampling frequency.

4.1.2.3 Power Consumption (mW)

To measure power consumption, we will measure the maximum current at an input voltage of 3.7V during steady operation with radio off, measured with an inline meter. This will be converted to mW. The goal is to determine whether the models affect the power draw of the system.

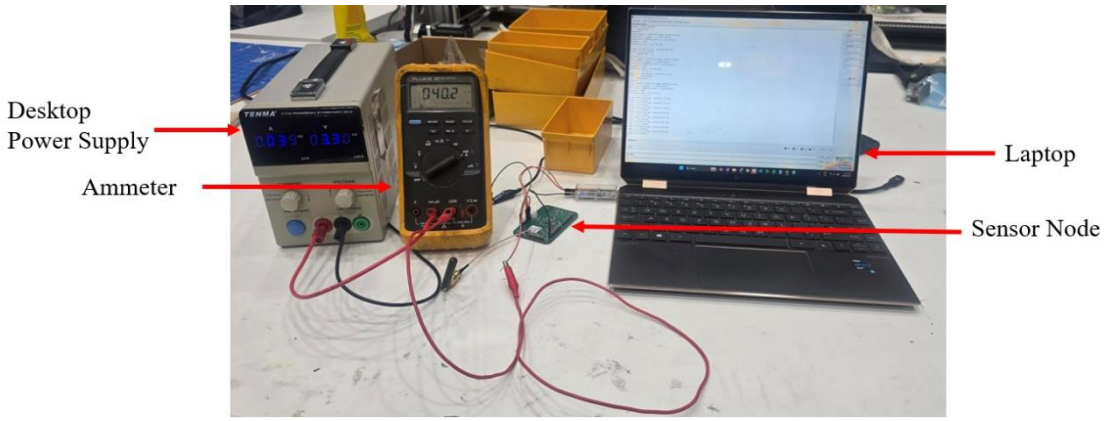

Figure 4.2, Power measurement experimental setup

Figure 4.2, shows the experimental setup to measure power. The setup consists of:

- A DC desktop power supply used to emulate the 3.7V battery
- A multimeter configured as an Ammeter in series between the power supply and the sensor node.
- The Sensor node PCB, containing the MCU and both sensors.
- A laptop, only connected to receive UART messages to ensure correct running of the program.

4.1.3 Quantisation

To try to allow larger models to be deployed on hardware the reduction of model parameter size using the quantisation from 32-bit floating points to 8-bit integers was used (float32 →int8.) The quantisation that we will use in this project is W8A32 (8-bit weights and 32-bit activations) and follows a similar approach to Jacob et al. [54]. The bulk of the size comes from the models' weights, so activations remain 32-bit to conserve accuracy.

We use a symmetric, per-output-channel quantisation for every weight matrix (linear, GRU gates) and every convolution kernel. This is performed offline before deployment on the trained float32 weights. For a 2-D weight matrix $W \in \mathbb{R}^{M\times T}$, where M is output size and T is the input

size, we use eq. (4.1) to define one scale per row. ε is very small but larger than zero to avoid a division-by-zero error (we will use 10^{-12}) in eq. (4.2).

$$s_i = max\left(\frac{max_j|W_{i,j}|}{127}, \varepsilon\right) \tag{4.1}$$

Using eq. (4.2) the quantised weights, $W_{i,j}^q$, are found [54].

$$W_{i,j}^q = round\left(\frac{W_{i,j}}{s_i}\right) \in [-127,127] \tag{4.2}$$

We can then store int8 weights W^q (int8), and per-row float scales s_i (float32), float biases b_i. GRU parameters are exported per gate. Input weights W_z, W_r, W_n and recurrent weights U_z, U_r, U_n are each quantised per output row. Biases are fused using eq. (4.3) and kept in float32.

$$b_{\text{gate}} = b_{\text{gate}}^{(\text{ih})} + b_{\text{gate}}^{(\text{hh})} \tag{4.3}$$

Once saved we must convert the int8 to float32 on device. We do this using eq. (4.4).

$$y_i = \left(s_i \sum\nolimits_j W_{i,j}^q x_j\right) + b_i \tag{4.4}$$

This process introduces a level of error between the original values and the compressed values. To assess how large this error is and if the error is meaningful, the parameter error will be evaluated and given both in a signal-to-noise ratio (SNR) in dB (eq. (4.5)) and the absolute error (eq. (4.6)) with $\widehat{W} = S \odot W^q$. Additionally, the model's FPR will be re-tested with this change. This will all occur outside of the hardware.

$$\text{SNR}_{\text{dB}} = 20\log_{10}\left(\frac{\| W \|_2}{\| W - \widehat{W} \|_2}\right) \tag{4.5}$$

$$\max|err| = \max\nolimits_i | W_i - \widehat{W}_i | \tag{4.6}$$

For a layer with $M \times T$ weights, float32 storage is $4MT\ (weights) + 4M(bias)$ bytes. The used scheme stores $1MT(weights) + 4M(scales) + 4M(bias)$ bytes, i.e., $\approx 4\times$ reduction on the dominant term with small $O(M)$ overhead.

4.1.4 On-device Translation Scope and Pipeline

The Python reference implementations were translated to C++ for the STM32WLE5-based node, with the goal of reproducing the inference-time data path on device. Each 1 Hz sensor stream (eCO_2, TVOC) is windowed to length N with hop 1 s, first-order differences (ΔeCO_2, ΔTVOC) are computed per window, and robust scaling (median/IQR) is applied per feature using eq. **(3.8)**. These steps are executed on the MCU exactly as in Chapter 3 so that encoder inputs match the training distribution.

For the GGM-VAE, only the encoder and latent scoring are deployed. A GRU ingests the window, and linear heads produce the mixture logits and per-component parameters. The anomaly score is computed in latent space (eq. (**3**.**21**)) against the learned prior and compared to a stored percentile-based threshold. The decoder is omitted in firmware, as reconstruction is not required for the decision rule.

For the Hybrid-VAE, again only the encoder was implemented. A Hann STFT is computed on device using the study's hop and window length. Magnitude information feeds a two-layer Conv2D branch (Table 3.3) in parallel with a GRU branch. Then a fusion linear layer maps to the latent code used for scoring.

Firmware is built with the Arduino IDE and flashed over UART. At runtime the node prints the latent score, averaged inference time and the anomaly decision for auditability. To check operation, a simple sanity check was conducted where a candle was placed and wafted near the sensors to verify that a sharp VOC rise produces an elevated score and a trigger.

4.2 Results

The results show the implementation of both the developed and optimised models on the sensor node hardware, with a comparison to the benchmark pure thresholding model.

4.2.1 Quantisation Error

We report weight quantisation error per layer as SNR (dB) and max |err| on weights. Higher SNR is better and as a rule of thumb, 40 dB ≈ 1% RMS error, 46 dB ≈ 0.5%, 50 dB ≈ 0.32%.

Table 4.1, GGM-VAE Quantisation Error

Layer / Param	*SNR (dB)*	$max\|err\|$
Latent log-variance head (log σ²)	47.5	4.71e-03
GRU encoder - recurrent weights	47.9	8.23e-03
Latent mean head (μ)	48.6	3.78e-03
GRU encoder - input weights	49.8	5.34e-03

Table 4.2, Hybrid-VAE Quantisation Error

Layer / Param	*SNR (dB)*	$max\|err\|$
Fusion layer ($\mu_{GRU} \oplus \mu_{CNN} \rightarrow z$)	40.3	3.20e-03
Log-variance head ($\log \sigma^2$)	42.8	4.45e-03
Latent mean head (μ)	44.2	3.63e-03
GRU encoder - recurrent weights	46.1	1.12e-02
CNN encoder - Conv2D	48.5	8.72e-04

Table 4.1 and Table 4.2 show the quantisation error for the GGM-VAE and the Hybrid-VAE respectively. We can observe very minimal value error across all encoder weights. We also need to determine if this difference has a meaningful impact on performance.

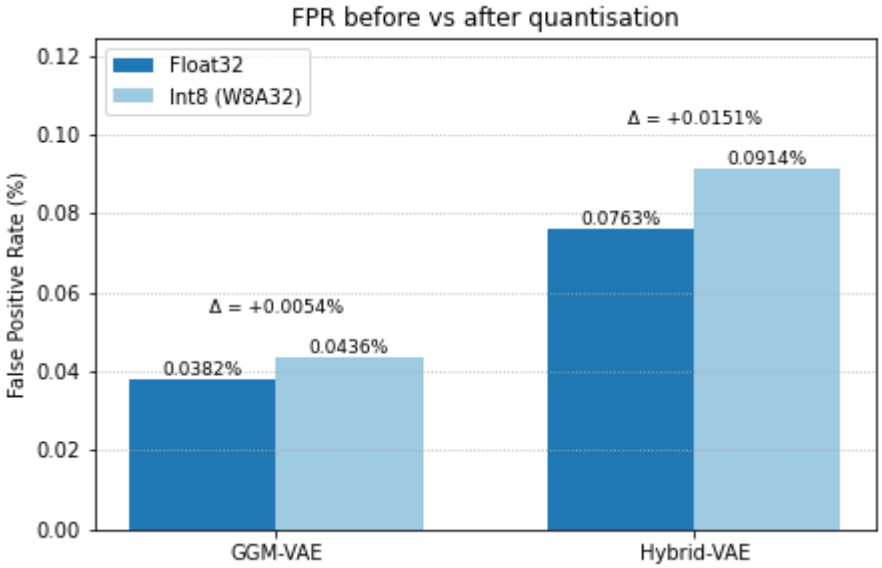

Figure 4.3, Quantisation effect on FPR

Figure 4.3 shows the effect of quantisation on the false-positive rate (FPR). There are noticeable impacts of quantisation, although both FPRs remain small. This loss in performance needs to be considered when choosing a model for deployment, especially if there are models that are close in performance to start with initial complexity trade-offs.

4.2.2 Memory Usage

The first key finding was that the bulk of the ROM size, ≈170 KB, is occupied by the startup code, drivers and supporting libraries. This leaves only ≈30 KB to store the model and its logic. Table 4.3, presents the used ROM.

Table 4.3, Memory usage for models

Model	*ROM size (KB)*	*ROM usage (%)*
Threshold	170.1	84%
GGM-VAE	174.2	86%
GGM-VAE Quantised	173.5	86%
Hybrid-VAE	241.9	120%
Hybrid-VAE Quantised	204.6	102%

We can see a very strong correlation between implemented ROM size and the encoder only model size evaluated earlier and presented in Table 3.9. We have only a small deviation for the GGM-VAE model and about a 5 KB increase in deployed size for the Hybrid-VAE. Both results are to be expected as there is additional logic and code that was not included in the model size (as its parameters only). What this means is that this metric provides a usable check before the model is converted for hardware implementation.

Another key finding was that the Hybrid-VAE massively exceeds the ROM limit, this is even true for the quantised model. This is as the CNN weight matrices are extremely large in size. For this model to fit constraints would need to be put on this size during development to limit the model size. This will worsen the model's performance, considering that small sizes were included in the optimisation process but were not selected and directly comparing smaller hidden sizes, and latent dimensions result in significant reductions in performance. Also important to note is that as the Hybrid-VAE did not fit on ROM, we are unable to gather any of the following metrics.

The Quantisation had varying impact where on large models such as the Hybrid-VAE the effect was massive reducing the size by about 37 KB. This is contrasted by the smaller GGM-VAE where we saved about 0.7 KB. This is due to the extra on device processing required, this occupies some ROM space and if the model is already small this may outweigh the reduction in size in some cases.

4.2.3 Inference Time

The average inference time was measured on hardware after reaching steady operation. The results in Table 4.4, show a massive difference between the thresholding method and the GGM-VAE. However, the GGM-VAE still operates within the 1 Hz sampling frequency and thus inference time has no real effect on model operation.

Table 4.4, Average inference time

Model	*Average Inference Time (ms)*
Threshold	0.000354
GGM-VAE	128.9
GGM-VAE Quantised	167.9
Hybrid-VAE	-
Hybrid-VAE Quantised	-

Through comparison of the quantised performance to the non-quantised performance, we observe an increase in inference time. This is caused by the additional processing required on the hardware side to convert the int8 back to float32. The quantised inference time still remains in the sampling period so is not a concern for this project.

Unfortunately, as previously mentioned no data was gathered for the Hybrid-VAE but based on differences in FLOPs it is safe to assume a similarly large increase in inference time. Whether this time now limits model performance is difficult to say.

A limitation of this methodology is that we fail to assess the required transmission time as it is outside the scope, resulting in additional time if a transmission is made after each or even after positive inferences.

4.2.4 Power Consumption

Power consumption shown in Table 4.5 all fall in similar ranges. A slight increase is shown in max power draw following the inference time trends. This correlation is expected as longer inference requires the MCU to draw more power. However, it does seem that the bulk of the power draw is from the steady-state operation of the sensors and their hotplates and the MCU itself.

Table 4.5, Power Consumption

Model	*Max Current Draw (mA)*	*Max Power Draw (mW)*
Threshold	34.87	129.02
GGM-VAE	34.94	129.28
GGM-VAE Quantised	34.96	129.36
Hybrid-VAE	-	-
Hybrid-VAE Quantised	-	-

The presented max power draw limits analysis. A better metric would be to measure average power draw to incorporate inference time better. However, the requisite equipment for this was unavailable.

4.2.5 Summary of Results

Overall, the GGM-VAE is the only deep model that cleanly fits and runs on the STM32WLE5 platform. It compiles to ≈174.2 KB (≈173.5 KB with W8A32), stays below the 200 KB application flash limit, and meets the 1 Hz sensing cadence with an inference time of 128.9 ms rising to 167.9 ms with W8A32. The Hybrid-VAE exceeds flash even after quantisation (241.9 KB → 204.6 KB), so it could not be executed on-device. The threshold baseline is trivially the smallest (≈170.1 KB) and fastest (≈0.35 µs).

Quantisation behaved as expected. Encoder weight errors were small (≈40–50 dB SNR across layers), but the added on-device dequantisation introduced modest latency and a small off-device FPR increase (e.g., GGM-VAE: 0.0382% → 0.0436%; Hybrid-VAE: 0.0763% → 0.0914%), trading a slight accuracy loss for substantial flash savings on larger networks. Power draw was essentially unchanged and dominated by sensors/MCU: peak ≈129.0 → 129.4 mW from baseline to quantised GGM-VAE.

Considered together, these results underline edge feasibility. On this MCU, the GGM-VAE offers the best balance of memory footprint, timing, and accuracy. Quantisation is useful when flash is the bottleneck, but for an already-compact GRU encoder it yields little memory benefit and slightly higher latency.

4.3 Limitations of approach

This chapter ported the deep models to an STM32WLE5-based node and measured flash usage, latency, quantisation error and power. While the results are promising, several limitations qualify the findings and point to future work.

4.3.1 Platform and Measurement Constraints

Inference latency was measured at 1 Hz with radio off and minimal interrupt activity. Jitter under concurrent sensor I/O, timers and LoRaWAN MAC activity was not characterised. Additionally maintaining the sample rate was not assessed and the effect of variance from 1 Hz was also not assessed.

As mentioned, power was measured as peak current at 3.7 V with a bench supply and digital multimeter. Average energy per sample cycle, sleep/idle fractions, and radio TX/RX costs were not profiled. Results therefore reflect a best-case, radio-off regime. Additionally, battery and temperature effects were not explored. It would be valuable to explore the real-world run-time implications on the battery and to assess solar power feasibility.

4.3.2 Quantisation Methodology

We used post-training, symmetric per-output int8 for weights with float32 activations (W8A32). This simplifies deployment but forgoes potential gains from activation quantisation. The use of quantisation-aware training could also be explored during model development to reduce error on final deployment.

4.4 Chapter Summary

This chapter translated the two deep-learning models to an STM32WLE5-based sensor node and measured feasibility on MCU-class hardware. The GGM-VAE, with latent-space scoring and W8A32 post-training quantisation, fit within the practical flash budget (≈174 KB total image with libraries) and met the 1 Hz sampling target, with mean inference time ≈129 ms (≈168 ms quantised). Quantisation slightly increased false alarms (GGM-VAE FPR 0.0382% → 0.0436%) but preserved behaviour. Power draw during steady operation was dominated by sensors, with only a small rise between models (≈129 mW). By contrast, the Hybrid-VAE exceeded flash limits even after quantisation, confirming that it is not presently deployable on this platform without further downsizing.

These results show that GGM-VAE is the practical candidate for the current node, while Hybrid-VAE requires architectural simplification or more aggressive compression. Measurements also reflect controlled conditions: radio off during timing and peak-current tests, average energy per cycle not profiled, and per-window FP32 normalisation assumed on device.

The following chapter concludes the thesis, drawing together algorithmic and hardware findings, stating the recommended deployment path, and outlining priorities for future work.

Chapter 5 Conclusions and Future Work

5.1 Conclusions

This project set out to answer two questions:

1. *What unsupervised deep-learning anomaly-detection algorithms can be developed and optimised to improve early bushfire detection accuracy using real-time environmental data from IoT sensor nodes?*

2. *Can unsupervised deep-learning algorithms be implemented on IoT sensor hardware effectively?*

On the first research question, two sequence models were engineered and evaluated against a simple thresholding baseline: a GRU Gaussian-Mixture VAE (GGM-VAE) and a Hybrid-VAE with a parallel STFT-CNN branch. Using the 'detect-all-fires' operating point (TPR = 100%) for fair comparison, the GGM-VAE achieved the strongest discrimination with F1-score = 0.5714 and FPR = 0.0382%, outperforming both the Hybrid-VAE (F1-score = 0.3478, FPR = 0.0763%) and the threshold baseline (F1-score = 0.1468, FPR = 0.2535%). Average time-to-first-detection was fastest for the Hybrid-VAE (230.4 s) followed by the GGM-VAE (235.5 s) and the thresholding method (473.5 s), indicating that both VAEs trigger materially earlier than the pure threshold and that the Hybrid model's extra positives also translate to slightly earlier alarms.

Bayesian optimisation stabilised quickly for the GGM-VAE and consistently selected K = 1 mixture component, a 1-D latent, hidden size 16, and a 15 s window, essentially the model behaved as an efficient GRU-VAE at this temporal scale. This suggests the available "normal" regime in the dataset is largely unimodal over 15 s windows, or that distinct modes (e.g., day/night) were insufficiently represented. The Hybrid-VAE's improvement curve was slower and more sensitive to hyperparameters. Trials converged to very short windows and unit-size CNN kernels, implying that under current data resolution the STFT branch contributed limited additional signal beyond the first-order differences.

Complexity measurements underline practicality for edge deployment. Encoder-only inference for the GGM-VAE requires ≈35.1k FLOPs with ≈1.1k parameters, while the Hybrid-VAE encoder is ≈860.6 k FLOPs with ≈14.7k parameters. Without the latent-space scoring design these metrics almost double. This design is therefore a key enabler for MCU-class targets.

On the second research question, the GGM-VAE encoder-only implementation fit within flash on the STM32WLE5-based node (≈174 KB total image, with the board support, sensor, and LoRaWAN libraries consuming ≈170 KB) and met the 1 Hz sampling constraint with measured inference around 129-168 ms (quantised slightly slower due to dequantisation). Power remained dominated by steady-state sensor draw (≈129 mW), with negligible differences

between models. By contrast, the Hybrid-VAE encoder exceeded the 200 KB practical flash limit even after quantisation (≈205-242 KB), so timing and power could not be recorded on device. This demonstrates that unsupervised deep models can run effectively on this class of hardware if the design uses an encoder-only, lightweight recurrent pathway. Heavier time-frequency branches require either more aggressive compression or a larger MCU.

Overall, the work demonstrates that, unsupervised, sequence-aware models yield large improvements in false-alarm control over a simple threshold while keeping detection times competitive. In addition, careful architectural choices (latent scoring and small GRU) and compression (W8A32) enable real-time operation on constrained nodes.

5.2 Future Work

This section aims to summarise and present the key recommendations for future work.

5.2.1 Broader data

The current evaluation is limited by event scarcity (eight ignitions) and outdoor label ambiguity. Extending the dataset to more fuels, seasons, microclimates, and non-fire “nuisance” activities (human/animal proximity, vehicle exhaust, rain events) would improve statistical power and stress-test false-positive behaviour.

5.2.2 State-of-the-art Comparisons

To identify the most suitable model for future work an evaluation against stronger sequence baselines such as LSTM-VAE and compact attention mechanisms would be beneficial.

5.2.3 Spatial Correlation

This thesis focused on single-node temporal detectors. In deployment, spatial fusion across nodes is expected to reduce FPR substantially. Designing and validating a lightweight, LoRaWAN-friendly fusion layer is a natural next step. Developing this may also provide insight into what a target FPR and detection time is for a local detector.

5.2.4 Field Experiments

It would be beneficial to conduct field deployments with telemetry to characterise real-world TPR/FPR, threshold stability, and mean time-to-detect under environmental variability. Measure average power over duty-cycled operation and validate battery-plus-solar energy budgets across seasons.

References

[1] N. Wilson and M. Yebra, “The Role of Climate in Ignition Frequency,” 2023.

[2] J. G. Canadell, C. P. (. Meyer, G. D. Cook, A. Dowdy, P. R. Briggs, J. Knauer, A. Pepler and V. Haverd, “Multi-decadal increase of forest burned area in Australia is linked to climate change,” 2021.

[3] J. MacCarthy, J. Richter, S. Tyukavina and N. Harris, “The Latest Data Confirms: Forest Fires Are Getting Worse,” 21 July 2025. [Online]. Available: https://www.wri.org/insights/global-trends-forest-fires#:~:text=This%20increase%20in%20fire%20activity,forest%20fires%20occurring%20since%202020.

[4] Natural Hazards Research Australia, “Understanding the Black Summer bushfires through research: a summary of key findings from the Bushfire and Natural Hazards CRC,” 2023.

[5] N. B. Arriagada, A. J. Palmer, D. M. Bowman, G. G. Morgan and B. B. Jalaludin, “Unprecedented smoke-related health burden associated with the 2019–20 bushfires in eastern Australia,” 2020.

[6] WWF, “AUSTRALIA’S 2019-2020 BUSHFIRES: THE WILDLIFE TOLL,” 2020.

[7] M. Rodrigues, F. Alcasena and C. Vega-García, “Modeling initial attack success of wildfire suppression in Catalonia, Spain,” 2019.

[8] C. C. Chan, S. A. Alvi, X. Zhou, S. Durrani, N. Wilson and M. Yebra, “A Survey on IoT Ground Sensing Systems for Early Wildfire Detection: Technologies, Challenges and Opportunities,” 2024.

[9] N. Andelic, S. B. Segota, I. Lorencin and Z. Car, “The Development of Symbolic Expressions for Fire Detection with Symbolic Classifier Using Sensor Fusion Data,” 2024.

[10] Q. Zhuang, C. C. Chan, X. Zhou and S. Durrani, “Early Detection of Wildfire Using IoT Sensors without Labelled Fire Data,” 2024.

[11] G. Pang, C. Shen, L. Cao and A. v. d. Hengel, “Deep Learning for Anomaly Detection: A Review,” 2021.

[12] Y. Kim, Y. Heo, B. Jin and Y. Bae, “Real-Time Fire Classification Model Base on Deep Learning for Building an Intelligent Multi-Sensor System,” 2024.

[13] J. Connell, “Early Bushfire Detection using Long Short-term Memory Networks,” 2024.

[14] H. Yin, A. Aryani, S. Petrie, A. Nambissan, A. Astudillo and S. Cao, “A Rapid Review of Clustering Algorithms,” 2024.

[15] A. M. Ikotun, A. Ezugwu, L. Abualigah, B. Abuhaija and J. Heming, “K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data,” 2023.

[16] A. Andrade, C. Montez, R. Moraes, A. Pinto, F. Vasques and G. da Sliva, “Outlier Detection Using k-means Clustering and Lightweight Methods for Wireless Sensor Networks,” 2016.

[17] U. Dampage, L. Bandaranayake, R. Wanasinghe, K. Kottahachchi and B. Jayasanka, “Forest fire detection system using wireless sensor networks and machine learning,” 2022.

[18] I. Üsek, M. Arana-Catania, A. Farr and I. Petrunin, “Deep Autoencoder for Unsupervised Anomaly Detection in Wildfire Prediction,” 2024.

[19] M. Putzu, D. Loru, F. Carta, A. Ledda, A. Chirigu, M. Sole, M. Anedda and D. Giusto, “Enhancing Wildfire Risk Management Through Sensor-Based AI Integration in Social IoT Frameworks,” 2024.

[20] P. Pokrel and H. Soliman, “Advancing Early Forest Fire Detection Utilizing Smart Wireless Sensor Networks,” 2018.

[21] S. Hochreiter and Schmidhuber, “Long Short-Term Memory,” 1997.

[22] J. Chung, C. Gulchre, K. Cho and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” 2014.

[23] W. Benzekri, A. El Moussati, O. Moussaoui and M. Berrajaa, “Early Forest Fire Detection System using Wireless Sensor Network and Deep Learning”.

[24] Z. Xu, Y. Guo and J. Homer Saleh, “Advances Toward the Next Generation Fire Detection: Deep LSTM Variational Autoencoder for Improved Sensitivity and Reliability,” 2021.

[25] Y. Guo, W. Liao, Q. Wang, L. Y. T. Ji and P. Li, “Multidimensional Time Series Anomaly Detection: A GRU-based Gaussian Mixture Variational Autoencoder Approach,” 2018.

[26] V. Ashish, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Popsukhin, “Attention Is All You Need,” 2017.

[27] J. Xu, H. Wu, J. Wang and M. Long, “Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy,” 2022.

[28] S. Zia and N. Bibi, “Enhanced Anomaly Detection in IoT: A Transformer based Approach for Multivariate Time Series Data,” 2024.

[29] L. Pinshan, P. Xiang and D. Lu, “ A new multi-sensor fire detection method based on LSTM networks with environmental information fusion,” 2022.

[30] J. Backhus, A. Rajendra Rao, C. Venkatraman and C. Gupta, “Time Series Anomaly Detection Using Signal Processing and Deep learning,” 2025.

[31] Y.-X. Lu, X.-B. Jin, J. Chen, D.-J. Liu and G.-G. Gen, “F-SE-LSTM: A Time Series Anomaly Detection Method with Frequency Domain Information,” 2024.

[32] F. F. Tu, D. J. Liu, Z. W. Yan, X. B. Jin and G. G. Geng, “STFT-TCAN:A TCN-attention based multivariate time series anomaly detection architecture with time-frequency analysis for cyber-industrial systems,” 2024.

[33] J. Baek, T. J. Alhindi, Y.-S. Jeong, M. K. Jeong, S. Seo, J. Kang, W. Shim and Y. Heo, “A wavelet-based real-time fire detection algorithm with multi-modeling framework”.

[34] W. Cheng, Y. Li and T. Ma, “S-KDGAN: Series-Knowledge Distillation With GANs for Anomaly Detection of Sensor Time-Series Data in Smart IoT,” 2024.

[35] F. A. Nafis, A. Nahiduzzaman, G. Saha, S. S. Alvi, I. Ahammed and A. J. Anonna, “An Efficient IoT-based Fire Detection System Using Quantized Deep Learning Model on Resource-Constrained Devices,” 2024.

[36] A. Papaioannou, P. Verikios, C. Kouzinopoulos, D. Ioannidis and D. Tzovaras, “A low-power embedded system for fire monitoring and detection using a multilayer perceptron,” 2020.

[37] P. Vorwerk, S. Muller, J. Kelleter and U. Krause, “Classification in Early Fire Detection Using Multi-Sensor Nodes - A Transfer Learning Approach,” 2024.

[38] PyTorch, “PyTorch,” 2025. [Online]. Available: https://pytorch.org/.

[39] D. P. Kingma and J. L. Ba, “Adam: A Method for Stochastic Optimization,” 2015.

[40] J. Bergstra, D. Yamins and D. Cox, “Making a Science of Model Search: Hyperparameter Optimisation in Hundreds of Dimensions for Vision Architecture,” 2013.

[41] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” 2019.

[42] S.-W. Fu, T.-y. Hu, Y. Tsao and X. Lu, “Complex spectrogram enhancement by convolutional neural network with multi-metrics learning,” 2017.

[43] L. Amorim, G. Cavalcanti and R. Cruz, “The choice of scaling technique matters for classification performance,” 2024.

[44] K. M. Sujon, R. B. Hassan, Z. T. Towshi, M. A. Othman, A. Samad and K. Choi, “When to Use Standardization and Normalization: Empirical Evidence From Machine Learning Models and XAI,” 2024.

[45] D. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” 2013.

[46] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” 2014.

[47] A. Zafar, M. Aamir, N. M. Nawi, A. Arshad, S. Riaz, A. Alruban, A. K. Dutta and S. Almotairi, “A Comparison of Pooling Methods for Convolutional Neural Networks,” 2022.

[48] R. Yamamoto, E. Song and J.-m. Kim, “Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram,” 2019.

[49] F. Pulsford, “PCB Design and Satellite-IoT Connectivity for the ANU Bushfire Sensor Node,” 2024.

[50] RAK, “RAK3172 WisDuo LoRaWAN Module Datasheet,” 2025. [Online]. Available: https://docs.rakwireless.com/product-categories/wisduo/rak3172-module/datasheet/. [Accessed 28 October 2025].

[51] Sensirion, “Datasheet SGP30,” 2020. [Online]. Available: https://sensirion.com/media/documents/984E0DD5/61644B8B/Sensirion_Gas_Sensors_Datasheet_SGP30.pdf.

[52] Bosch Sensortec, “Gas sensor BME680,” 2025. [Online]. Available: https://www.bosch-sensortec.com/products/environmental-sensors/gas-sensors/bme680/#technical.

[53] ST, “STM32WLE5CC,” 2025. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32wle5cc.html.

[54] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko, “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” 2017.

Appendix A: Example Training Curve

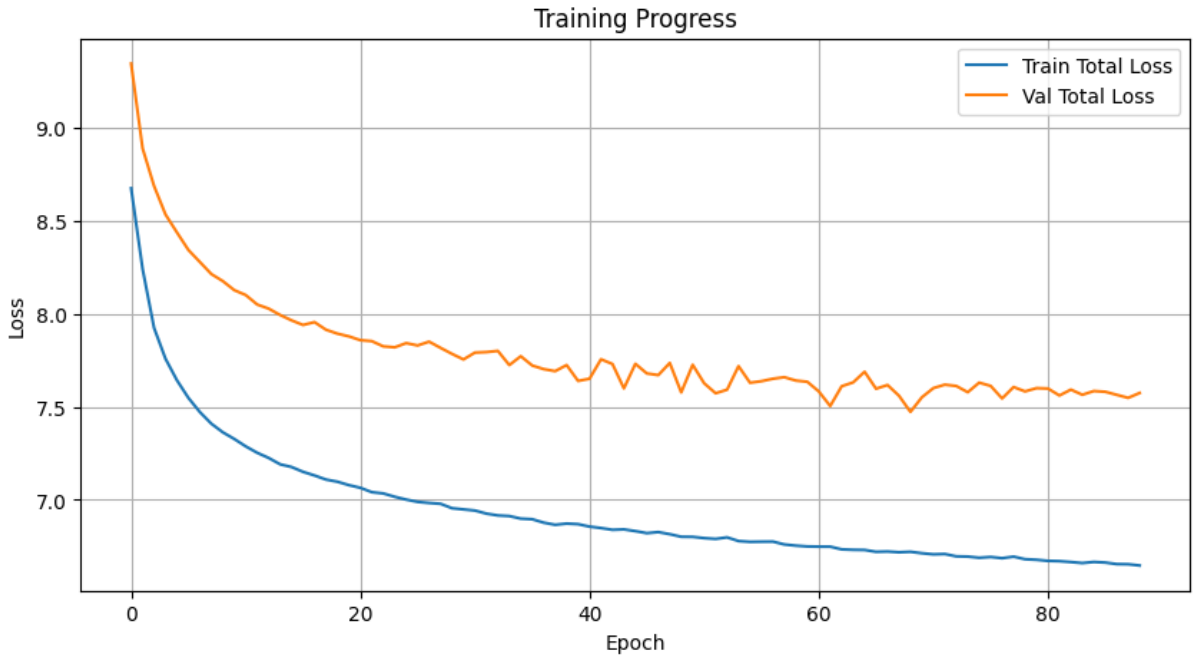

Figure A.1, GGM-VAE Training Curve