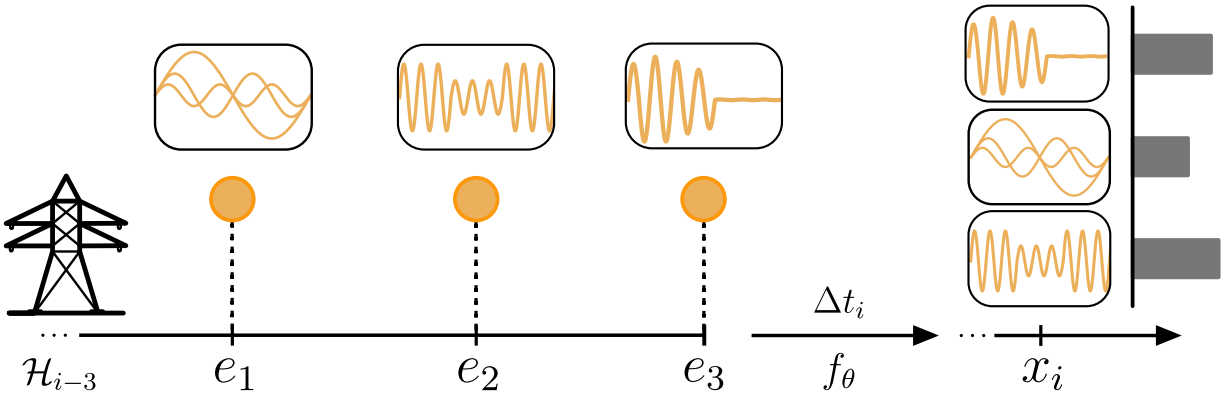
CHALMERS
UNIVERSITY OF TECHNOLOGY



Towards Event Sequence Foundation Models

Exploring Temporal Point Process Transformers for Power Grid Fault Prediction

Master's thesis in Complex Adaptive Systems

MARCUS ANDERSSON
FILIP OLSSON

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Towards Event Sequence Foundation Models

Exploring Temporal Point Process Transformers for
Power Grid Fault Prediction

MARCUS ANDERSSON
FILIP OLSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2025

Towards Event Sequence Foundation Models
Exploring Temporal Point Process Transformers for Power Grid Fault Prediction
MARCUS ANDERSSON, FILIP OLSSON

Supervisors: Viktor Olsson, Eneryield
Jakob Lindqvist, Eneryield
Examiner: Mohsen Mirkhalaf, Department of Physics

Master's Thesis 2025
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of an event sequence of historical disturbances in a power grid, and a simplified model prediction of the next event.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Towards Event Sequence Foundation Models
Exploring Temporal Point Process Transformers for Power Grid Fault Prediction
MARCUS ANDERSSON
FILIP OLSSON
Department of Physics
Chalmers University of Technology

Abstract

Interruptions in electric power systems cause significant disruptions and economic losses. The growing availability of fault data from electric power grids, combined with advances in machine learning, particularly deep learning, is opening up new opportunities for fault prediction. Transformer architectures are especially well-suited to model sequential data, like the temporal sequence of historical disturbances in the power grid. This thesis proposes a foundation model approach to fault prediction, in which a transformer-based model is pretrained on large-scale event sequence datasets from diverse domains. We then fine-tune the model on downstream fault prediction tasks in a continual learning manner, using historical fault data segmented into 3-day sequences. While an equivalent specialized fault model outperforms this smaller foundation model by achieving 21.8 % higher average precision, fine-tuning on multiple proxy tasks - common in foundation model training - significantly improved fault prediction performance of both models. These results suggest that while foundation models for event sequences are still emerging, the idea of proxy task fine-tuning is already beneficial to existing models. Notably, the foundation model exhibited less forgetting and higher forward transfer than the specialized model, indicating superior retention of knowledge and ability to adapt to new tasks. Scaling up foundation models remains a promising path towards reliable and scalable fault prediction systems for the power grid.

Keywords: machine learning, foundation models, temporal point process, transformers, event sequence, transfer learning, power grid fault prediction.

Acknowledgements

We would like to express our gratitude to Eneryield for allowing us to join their inspiring and welcoming team for this spring of thesis work. Thank you Kalle, Johan, Ebrahim, Jesse, Fredrik, Ellen, Fllanza, Linnea and Max. Your company culture and the highlight of the week: the friday lunch wheel spins, made our time here very enjoyable.

Thank you Albin for your help and ideas when we were stuck deep with coding bugs and in need of a mood boost. And of course, a special thanks to our main supervisors Viktor Olsson and Jakob Lindqvist, for your continued guidance and feedback.

Finally, a big thank you to our friends and family for your support during our five years at Chalmers.

Marcus & Filip, Gothenburg, May 2025

Nomenclature

This section summarizes the nomenclature and notation conventions used throughout the thesis. Uppercase letters, such as X, denote matrices, while lowercase bold letters, such as $\mathbf{x}$, denote vectors.

Sets

$\mathcal{E}$	Event sequence
$\mathcal{H}_t$	History of events up until time step t
$\mathcal{M}$	Set of all marks
$\mathcal{T}$	Task
$\mathcal{D}$	Domain
$\mathcal{S}$	Dataset

Variables & Parameters

t_i	Time of event i
e_i	Mark of event i
τ	Inter-event times
α	Learning rate
β	Exponential decay rate
γ	Weight decay factor
ξ	Tuning threshold
$\mathbf{h}$	History encoding
$f_{\boldsymbol{\theta}}$	Transformer encoder with parameters $\boldsymbol{\theta}$
$h_{\boldsymbol{\phi}}^{(i)}$	Task-specific heads with parameters $\boldsymbol{\phi}^{(i)}$

Contents

Nomenclature **ix**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Problem formulation . . . 1
1.2 Related work . . . 3
1.3 Survey of event sequence datasets . . . 4

2 Theory **7**
2.1 Temporal point processes . . . 7
2.1.1 Neural temporal point processes . . . 8
2.1.2 Likelihood estimation . . . 9
2.2 Transformers . . . 11
2.2.1 Attention mechanism . . . 12
2.2.2 Positional embeddings . . . 13
2.3 Foundation models . . . 13
2.3.1 Pretraining methods . . . 14
2.3.2 Fine-tuning pretrained models . . . 14
2.3.3 Continual learning and catastrophic forgetting . . . 15
2.3.4 Activation functions . . . 17

3 Methodology **19**
3.1 Data pre-processing . . . 19
3.1.1 Creating event sequences . . . 21
3.1.2 Merging datasets . . . 21
3.2 Model architecture . . . 22
3.2.1 Output heads for pretraining . . . 23
3.2.2 Task-specific heads . . . 23
3.3 Training . . . 24
3.3.1 Pretraining . . . 24
3.3.2 Fine-tuning . . . 25
3.3.3 Proxy tasks . . . 25
3.3.4 Optimizing the fine-tuning strategy . . . 27
3.3.5 Training algorithm . . . 29

4 Results **31**
4.1 Baseline task performance 31
4.2 Shortening the prediction window 35
4.3 Fine-tuning strategy 36
4.4 Catastrophic forgetting 39

5 Conclusion **43**

Bibliography **45**

A Appendix **I**
A.1 Derivation of event sequence likelihood I
A.2 Evaluation metrics III
A.3 Extensive list of hyperparameters IV

List of Figures

1.1 Comparison of a standard time series and an event sequence. 2

2.1 Illustration of a neural TPP model with three output heads, each parameterizing a distribution. One distribution is over inter-event times τ, one over feature values and the final is a categorical mark distribution. The figure is modified with written permission from authors [16]. 8
2.2 Illustration of the transformer architecture by dvgodoy/CC BY [23]. . 11

3.1 Overall model architecture of the *Conditionally Independent Point Process Transformer* used in this work. Here $\{\mathbf{x_1}...\mathbf{x_n}\}$ is the output of the data embedding layer and $\{\mathbf{y_1}...\mathbf{y_n}\}$ is the latent space representation of the input event sequence. 22
3.2 Illustration of the learning objective for the *Class distribution* task. The model is trained to minimize the error between predicted and actual distribution of event types in the input sequence. 26
3.3 Illustration of the learning objective for interruption prediction tasks. Context window refers to the length of the input sequence. 27

4.1 Average precision over baseline and precision over baseline at ξ for different prediction window lengths. Here a prediction window of zero days represents the proxy task *Interruption in sequence*. Each task used the threshold ξ presented in Table 4.2a. Evaluation was done on the held-out set. 35
4.2 Train loss across all training phases in the best performing fine-tuning strategies. CD is shorthand for *Class distribution*, EL is *Event label*, I0 is *Interruption in sequence*, I3 is *Interruption 3-day*, and so on. . . 37
4.3 Precision, recall and F1-score as a function of the threshold ξ. Values were evaluated on *Interruption 7-day* on the held-out set. Note that these plots were visualized after final evaluation, and therefore not used to tune thresholds, ensuring no data leakage occurred. 38
4.4 Evaluation of catastrophic forgetting for specialized model. Performance of *Class distribution* on held-out set, after continued fine-tuning on more tasks. 39

List of Tables

1.1 Summary of included event-based time series datasets. The size of a dataset is specified by the number of events, n, and the number of features, m. 5

3.1 General structure of the raw datasets before any pre-processing. . . . 19
3.2 The structure of the raw data in the dataset "eCommerce 1". 20
3.3 Structure of the *subject* data frame for dataset "eCommerce 1". 20
3.4 Structure of the *event* data frame for dataset "eCommerce 1". 20
3.5 Example of a *task* data frame on a synthetic dataset. This example has binary labels, but regression tasks can have floating point label values as well. 25

4.1 Important hyperparameters for all tasks across the two models. Values were found with a Bayesian Weights & Biases hyperparameter sweep. All tasks were trained until the tuning loss started to show overfitting. The learning rate is α and the weight decay factor is denoted γ. . . . 32
4.2 Baseline task performance of both approaches on the held-out set containing 100 sequences. Evaluation was done directly after pretraining on their respective pretraining datasets and fine-tuning on only that specific task. The threshold ξ represents the decision boundary at which probabilities are mapped to binary predictions. 33
4.3 Optimal fine-tuning strategy with task ordering for both approaches. 36
4.4 Performance on *Interruption 7-day* following the optimal fine-tuning strategy for both approaches. Values in parentheses present the change from their respective baseline performance on *Interruption 7-day*. The threshold ξ was tuned on the train and tuning set. Final evaluation was done on the held-out set. 38
4.5 Forgetting measure (FM) for both models, evaluated following each model's optimal fine-tuning strategy, starting from the different tasks, on the held-out set. Performance was measured using average precision. 40
4.6 Backward transfer (BWT) and forward transfer (FWT) for both models, evaluated on the held-out set. Performance was measured using average precision. 41

A.1 Confusion matrix for a binary classification task. tp is the true positive count, fp is the false positive count, fn is the false negative count, and tn is the true negative count. III

A.2 Extensive list of hyperparameters used for pretraining and fine-tuning the specialized fault model and the foundation model. SM is shorthand for specialized model, FM is foundation model, CD is class distribution, EL is event label, IX is the collection of all interruption classification tasks. IV

1
Introduction

As of 2022 the use of electricity in Sweden amounted to 134 terra watt hours and that number is expected to increase in the coming years [1]. This rise puts ever increasing demands on the power grid and the organizations that operate it, resulting in a higher frequency of faults and interruption in the power grid [2]. Faults and interruptions can cause significant disruptions and economic losses, hence minimizing the occurrences of these events is a high priority to increase grid reliability. Such faults can stem from a variety of different root causes, but often include short circuits, open circuits, device failure or overloads [3]. Being able to predict the timing of future faults at an early stage, based on historical events, is therefore crucial. This capability enables grid operators to address potential problems before they become critical, thereby increasing the reliability of the power grid.

Machine learning methods are well suited for the fault prediction task because of their ability to handle high-dimensional signals and learn complicated temporal dependencies [4]. Deep learning models such as transformers are especially good at modeling sequential data, like the temporal sequence of historical disturbances and interruptions in this problem. Power grid events can also be mathematically modeled as a temporal point process with stochastic arrival times. In practice, however, fault data suffers from label scarcity and varying sampling rates, making generalization capabilities important for a machine learning model to be used to this end.

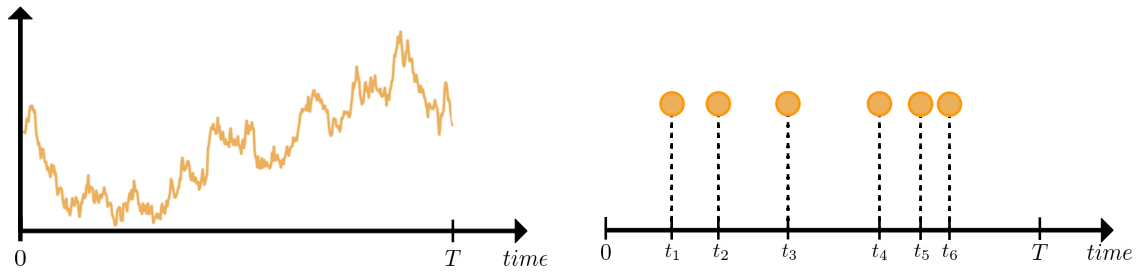
This thesis has societal benefits by improving the reliability and efficiency of power grids, reducing economic losses and the inconvenience caused by outages. Enhancing fault prediction can also support the integration of renewable energy, contributing to a more sustainable energy future. Furthermore, the development of large general deep learning models tailored for event sequence data could benefit a wide range of applications beyond power systems that also use data of the same modality.

1.1 Problem formulation

Recently, foundation models for time series have emerged as a promising solution to previously stated issues with fault data. A foundation model is a large deep learning model with general knowledge, that can be applied to new downstream tasks with little or no fine-tuning of its parameters [5]. This is achieved by training the model on diverse proxy tasks on huge datasets. Most existing work on foundation models

for time series data focuses on inputting a standard time series and forecasting future values. However, our prediction problem and data do not directly fit into this standard formulation. The fault prediction problem differs from most forecasting tasks in that the input is a collection of discrete events with stochastic arrival times—known as an *event sequence*—rather than a regularly sampled time series. Figure 1.1 below illustrates this difference in data modality.

Definition 1. *An **event sequence** is a temporally ordered set of events that describe the progression of occurrences within a specific context [6]. It can be formalized as* $\mathcal{E} = \{(t_1, e_1), (t_2, e_2), ..., (t_N, e_N)\}$*, where* e_i *is an event described by some structure that captures the nature of the occurrence, and* t_i *denotes the timestamp when* e_i *occurs.*



(a) Standard time series **(b)** Event sequence

Figure 1.1: Comparison of a standard time series and an event sequence.

The fault prediction task is to estimate the next event and its timing based on historical occurrences, making it a temporal point process task. In our case, each event consists of a short recording of voltage and current waveforms, typically around one second long. These waveforms are recorded if there is a detected disturbance or interruption. Foundation models for this modality have not been extensively explored, however, recent research includes a few promising models, mostly focusing on medical applications [7].

The purpose of this thesis is to investigate the potential of foundation models for fault prediction in electric power systems. This will be done by training a large model on several fault prediction tasks and benchmarking it against a specialized fault model. This research is both interesting and important for two main reasons. First, a large-scale general model has the advantage of being able to utilize vast amounts of unlabeled grid data, allowing it to learn complicated failure patters. Secondly, adaptability to new grid operators is crucial for real-world deployment, making generalization capabilities important. There are however several challenges posed by the complex nature of time series data that need to be addressed to achieve reliable fault prediction. Given this problem formulation, the following three research questions are investigated in this thesis.

How does the performance of a specialized model compare to that of a foundation model when applied to the fault prediction task?
This question aims to evaluate the differences between training a model specifically for a given task and leveraging multiple datasets with a foundation model.

What is the impact of fine-tuning strategies on the fault prediction task, how does it differ between a foundation model and a specialized model?
This question investigates whether incorporating proxy tasks — intermediate tasks related to the final task — can enhance the model's ability to transfer knowledge and learn effectively from limited task-specific data.

To what extent does fine-tuning sequentially lead to catastrophic forgetting?
This question examines the problem of catastrophic forgetting, where a model loses performance on its prior tasks after being fine-tuned [8].

1.2 Related work

Current literature divides time series foundation models according to the modality of data they process. The taxonomy presented in Liang *et al.* [6] recognizes three principal branches of time series foundation models: *standard* time series, *spatial* time series, and other, under which the event sequence definition falls. Given their prominence, adapting a standard time series foundation model to our prediction problem was the initial consideration. However, this approach presents several complications.

Firstly, in order for the event sequence data to fit the modality of a standard time series foundation model, the sequence of discrete event has to be adapted to a standard time series format. Since the discrete events of the fault dataset consist of short waveform recordings, this is possible. However, if the events consist of data of another modality, such as categorical or text data, conversion to a standard time series would be less feasible. Even still, the needed aggregation to make our data fit the modality of a standard time series could potentially lead to information loss, especially regarding inter-event dependencies, such as the time between events.

Another challenge is the input length of the sequences that the foundation model has been trained on. Open-source time series foundation models, such as UniTS and TTM, are often pretrained with an input sequence length on the order of 10^3 [9], [10]. Without pre-processing, sequences in our dataset span up to 10^5 timesteps, far exceeding the length of the input sequences used during pretraining of UniTS and TTM. Hence, to fit the context window of a pretrained standard timeseries foundation model, significant down-sampling would be required. This increases the risk of substantial loss of information. Due to stated complications, standard time series foundation models will not be considered in this study.

To the best of our knowledge, there are no open-source pretrained event sequence foundation models at the time of writing this thesis. This poses a significant challenge, since training a large foundation model from scratch requires huge amounts of data and computing power [11]. Nonetheless, insights from a smaller scale model that follow the same pretraining then fine-tuning paradigm may still generalize to larger models. As a result, the chosen model architecture will be trained from scratch.

One of the few architectures explicitly designed for event sequences is the model EventStreamGPT proposed by McDermott *et al.* [7]. The authors also introduce a data processing pipeline that transforms raw event sequences into representations suitable for deep learning, along with two generative model architectures for time-to-event prediction. The first architecture, the *Conditionally Independent Point Process Transformer*, predicts the timing and content of future events independently. In contrast, the *Nested Attention Point Process Transformer* incorporates intra-event dependencies by using nested attention mechanisms to account for both temporal and content dependencies. Both architectures aim to model the joint probability distribution

$$p(t_i, \mathbf{x}_i \mid (t_1, \mathbf{x}_1), \ldots, (t_{i-1}, \mathbf{x}_{i-1})),$$

where $\mathbf{x}_i$ denotes the event content at timestamp t_i. This formulation allows the model to generate both the time until the next event and the content of future events by sampling from the learned distribution.

1.3 Survey of event sequence datasets

The main proprietary fault dataset consists of 78 000 recorded disturbances and interruptions in an electric power grid during a five year period. Each of these events have a timestamp, event label, and voltage and current waveforms of each of the three phases plus the neutral phase. From the waveforms, 370 representative scalar values have been calculated, of which a predefined subset of 27 scalar values will be used as features for the machine learning models.

A foundation model with general knowledge about event sequences requires event sequence data from a wide set of domains, which is why multiple publicly available datasets are included in this work. This has benefits of increasing the amount of training data for better generalization and capturing general temporal patterns and suitable feature embeddings. Table 1.1 summarizes the included datasets.

Table 1.1: Summary of included event-based time series datasets. The size of a dataset is specified by the number of events, n, and the number of features, m.

Dataset	**Domain**	**Size** (n, m)	**Features**
Fault dataset	Electricity grid	78k, 370	Three-phase U/I, Label
eCommerce 1 [12]	Online purchases	845k, 9	Price, View/Cart/Purchase
eCommerce 2 [13]	Online purchases	1 654k, 9	Price, View/Cart/Purchase
Pred. Maintenance [14]	Device failure	124k, 12	Unspecified metrics

These datasets were chosen because their data fit the event sequence definition well and because they come from different domains. A common feature of all datasets is that they include a timestamp and corresponding event content, in line with the definition of event sequences.

eCommerce 1 is a large publicly available dataset of online purchase activity from an electronics store. It contains the event types *view*: a user has viewed a specific product online, *cart*: user put product in cart and *puchase*: user has purchased a product. Each of these event types have additional features with product information such as price, category and brand, and a user id. **eCommerce 2** is a similar dataset with identical event types and features, but from an online cosmetics store.

Predictive Maintenance is a publicly available dataset of daily sensor readings from a fleet of devices. This dataset was collected for the purpose of predicting device failures proactively. It contains seven metrics and a binary label column specifying failures.

MIMIC-IV-ED is an event-based healthcare dataset. A subset of this dataset, called MIMIC-IV-ED demo, is publicly available and contains 100 subjects from the original dataset. It contains a patient tracking table, edstays, as well as five data tables: diagnosis, medrecon, pyxis, triage and vitalsign. The tracking table edstays contains each patient's subject id, admission and discharge times and other basic information related to the patient. Each data table is linked through subject id. This dataset was excluded because of its split structure differing from the other datasets. It could have been incorporated in two ways: using separate tables where each dataset having its own embedding or combining all tables to a combined dataset, none of which are optimal.

2
Theory

The following sections will present relevant theory for the methodology of this thesis work. It will cover essential background on temporal point processes, the transformer architecture and an overview of foundation models and continual learning.

2.1 Temporal point processes

A Temporal Point Process (TPP) is a stochastic process first studied in the early 20th century. One early application was in 1948, when it was used to model telephone traffic in the work of Brockmeyer *et al.* [15]. Since then, these types of models have been used in a wide range of fields, such as seismology, neuroscience and financial market modeling.

In such applications, TPPs are commonly used to model event sequences of discrete events observed in some continuous time interval [16]. The content of an event is often referred to as its *mark*. A *marked* TPP has a realization of an event sequence, in accordance to definition 1 given in the introduction, with $\mathcal{E} = \{(t_1, e_1), (t_2, e_2), ..., (t_N, e_N)\}$. In this formulation, $t_i \in \mathbb{R}^+$ is the arrival time of event $i \in \mathbb{N}^+$, and $e_i \in \mathcal{M}$ is the corresponding mark. Depending on the application, the set of all marks can be continuous $\mathcal{M} = \mathbb{R}^d$, or categorical with $\mathcal{M} = \{1, \ldots, K\}$. Note that the number of events in a sequence, N, is itself a random variable.

TPPs are characterized by an intensity function $\lambda_k(t \mid \mathcal{H}_t)$, which models the instantaneous rate at which events with mark k are arriving. An intensity function for events with categorical mark k can be defined as

$$\lambda_k(t \mid \mathcal{H}_t) = \lim_{\Delta t \to 0^+} \frac{\Pr\{\text{event of type } k \text{ in } [t, t + \Delta t) \mid \mathcal{H}_t\}}{\Delta t}, \quad \forall k \in \mathcal{M} \quad [16]. \tag{2.1}$$

From this, the probability density function of event times can be calculated as

$$f(t \mid \mathcal{H}_t) = \lambda(t \mid \mathcal{H}_t) \exp\left(-\int_{t_{i-1}}^{t} \lambda(u \mid \mathcal{H}_t) du\right) \quad [17]. \tag{2.2}$$

A derivation of this probability density function, derived from the intensity function, is given in Appendix A.1. Traditional TPPs can be extended by using neural networks to parameterize the conditional probability density function $f(t \mid \mathcal{H}_t)$. This class of machine learning models is called Neural TPPs, which can learn complex temporal dependencies and outperform traditional methods.

2.1.1 Neural temporal point processes

Neural TPPs are autoregressive models, as introduced by Du *et al.* [18], that sequentially predict arrival times and marks. The prediction is achieved through three steps: 1) representing the events as feature vectors, 2) encoding history into a fixed-dimensional vector $\mathbf{h}_i \in \mathbb{R}^{d_{hidden}}$, and 3) parameterizing the next event's conditional distributions using the history encoding. An illustration of a neural TPP model is shown in figure 2.1.

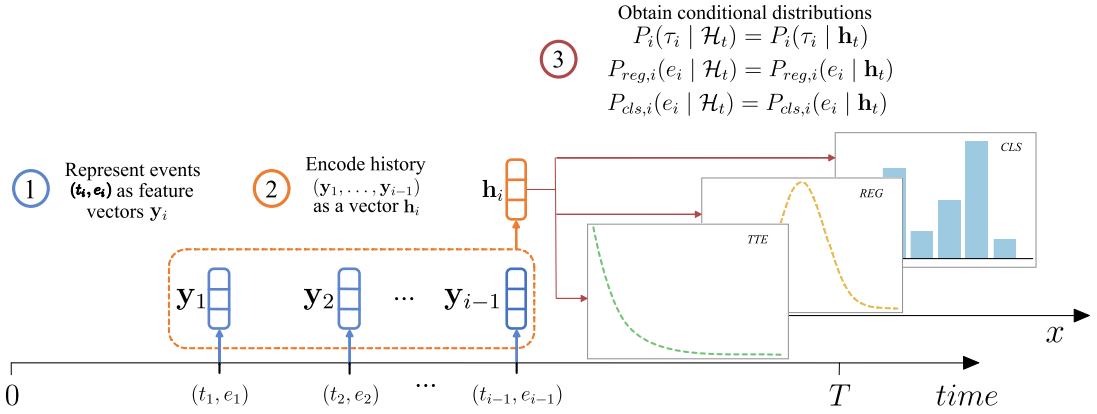


Figure 2.1: Illustration of a neural TPP model with three output heads, each parameterizing a distribution. One distribution is over inter-event times τ, one over feature values and the final is a categorical mark distribution. The figure is modified with written permission from authors [16].

Predicting future events is often difficult, and can be simplified through suitable assumptions. One such assumption is that arrival times and marks are conditionally independent of each other, meaning the joint probability distribution $P_i(\tau_i, e_i \mid \mathcal{H}_t)$ can be factorized as

$$P_i(\tau_i, e_i \mid \mathcal{H}_t) = P_i(\tau_i \mid \mathcal{H}_t) \cdot P_i(e_i \mid \mathcal{H}_t), \tag{2.3}$$

where $\tau_i = t_i - t_{i-1}$ represents the inter-event times between consecutive events. This factorization allows for a separate estimation of the temporal and mark distributions. The temporal distribution can be represented by a range of functions, such as: the probability density function $f_i(\tau_i \mid \mathcal{H}_t)$, the cumulative distribution function $F_i(\tau_i \mid \mathcal{H}_t)$, the survival function $S_i(\tau_i \mid \mathcal{H}_t)$, the hazard function $\phi_i(\tau_i \mid \mathcal{H}_t)$, or the cumulative hazard function $\Phi_i(\tau_i \mid \mathcal{H}_t)$ [16]. The mark distribution can be modeled as a categorical distribution with probability mass function $P_i(e_i = k \mid \mathcal{H}_t)$. For example, if we use the hazard function to model the temporal distribution, the conditional intensity can be calculated as

$$\lambda_k(t \mid \mathcal{H}_t) = p_i(e_i = k \mid \mathcal{H}_t) \cdot \phi_i(t - t_{i-1} \mid \mathcal{H}_t), \tag{2.4}$$

where t_{i-1} is the time of the most recent event prior to t.

2.1.2 Likelihood estimation

The aim of neural TPPs is to learn a model that best estimates the conditional intensity function $\lambda(t \mid \mathcal{H}_t)$. This can be done by parameterizing any of the previously described functions for the temporal distribution. If we assume conditional independence of arrival times and marks, both $P_i(\tau_i \mid \mathcal{H}_t)$ and $P_i(e_i \mid \mathcal{H}_t)$ are parameterized using the history encoding $\mathbf{h}_i$. Define

$$P(e_i = k \mid \mathcal{H}_t; \boldsymbol{\theta}) = g_{\boldsymbol{\theta}}(\mathbf{h}_i) \tag{2.5}$$

$$\phi(t - t_{i-1} \mid \mathcal{H}_t; \boldsymbol{\theta}) = h_{\boldsymbol{\theta}}(\mathbf{h}_i, t - t_{i-1}) \tag{2.6}$$

where $g_{\boldsymbol{\theta}}$ and $h_{\boldsymbol{\theta}}$ are neural network functions that model the mark probability and temporal hazard function respectively, and $\boldsymbol{\theta}$ represents all learnable parameters. With this parameterization, the conditional intensity for mark k becomes

$$\lambda_k(t \mid \mathcal{H}_t; \boldsymbol{\theta}) = g_{\boldsymbol{\theta}}(\mathbf{h}_i) \cdot h_{\boldsymbol{\theta}}(\mathbf{h}_i, t - t_{i-1}) \tag{2.7}$$

Using this along with Equation 2.2, we can write the likelihood of observing an event sequence $\mathcal{E} = \{(t_i, e_i)\}_{i=1}^N$ with K categorical marks as

$$p(\mathcal{E}; \boldsymbol{\theta}) = \prod_{i=1}^{N} \sum_{k=1}^{K} \mathbb{1}(e_i = k) \lambda_k(t_i \mid \mathcal{H}_t; \boldsymbol{\theta}) \exp\left(- \sum_{k=1}^{K} \int_0^T \lambda_k(u \mid \mathcal{H}_t; \boldsymbol{\theta}) du \right), \tag{2.8}$$

where T is the final time in the studied time interval [16], [17]. A full derivation of this likelihood is given in Appendix A.1. The first factor accounts for the probability of observing the event sequence $\mathcal{E}$, while the second factor accounts for the probability of not observing any events in the rest of the observation interval.

When dealing with a dataset containing multiple event sequences, the total likelihood is simply the product over each individual sequence likelihood. Assuming we have a dataset $\mathcal{S} = \{\mathcal{E}^{(m)}\}_{m=1}^M$, where each $\mathcal{E}^{(m)} = \{(t_i^{(m)}, e_i^{(m)})\}_{i=1}^{N_m}$, we can write the total likelihood as

$$p(\mathcal{S}; \boldsymbol{\theta}) = \prod_{m=1}^{M} p(\mathcal{E}^{(m)}; \boldsymbol{\theta}). \tag{2.9}$$

It is common to use the negative log-likelihood as loss function

$$\mathcal{L}(\boldsymbol{\theta}) = - \sum_{m=1}^{M} \log p(\mathcal{E}^{(m)}; \boldsymbol{\theta}), \tag{2.10}$$

which can be used for training a neural network model. The optimal model parameters $\boldsymbol{\theta}_{\text{MLE}}$ are found by maximizing the likelihood of the observed data, or equivalently minimizing the negative log-likelihood

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\min \mathcal{L}(\boldsymbol{\theta}), \tag{2.11}$$

which corresponds to maximum likelihood estimation. This is asymptotically equivalent to minimizing the Kullback-Leibler divergence between the data and model distributions [19]. In practice however, minimizing this loss can be challenging due

to the integral in Equation 2.8 often being intractable [16]. An alternative approach is therefore to use a sampling-based loss function on the form

$$\mathcal{L}_{\text{sample}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{E}\sim p(\mathcal{E};\boldsymbol{\theta})}\left[\text{score}(\mathcal{E})\right], \tag{2.12}$$

using a scoring function to evaluate the similarity of event sequences in the training set and generated event sequences [16]. Some examples of scoring functions include Wasserstein distance or adversarial losses. The EventStreamGPT framework instead directly models the inter-event times τ by assuming they follow a parameterized distribution [7]. EventStreamGPT models τ with either an exponential distribution

$$p(\tau) = \lambda \exp\left(-\lambda\tau\right), \tag{2.13}$$

or a log-normal mixture distribution with L components as

$$p(\tau) = \sum_{l=1}^{L} w_l \cdot \frac{1}{\tau\sigma_l\sqrt{2\pi}} \exp\left(-\frac{\log(\tau)-\mu_l}{2\sigma_l^2}\right), \tag{2.14}$$

where w_l is a learnable mixture weight for each log-normal distribution. The model is then trained on minimizing the negative log-likelihood of observed inter-event times

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \log p(\tau_i;\boldsymbol{\theta}). \tag{2.15}$$

Gradients are computed with backpropagation, and parameters optimized by some gradient-based optimization method, such as stochastic gradient descent with the Adam Optimizer [20]. The Adam algorithm first calculates the gradients at time t as

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_{t-1}), \tag{2.16}$$

where $\boldsymbol{\theta}_t \in \mathbb{R}^n$ are the model parameters at timestep t. Then, first and second moment estimations, $\mathbf{m}_t, \mathbf{v}_t \in \mathbb{R}^n$ are updated as

$$\mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1-\beta_1)\cdot \mathbf{g}_t \tag{2.17}$$

$$\mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1-\beta_2)\cdot \mathbf{g}_t^2, \tag{2.18}$$

where β_1, $\beta_2 \in [0,1]$ are exponential decay rates. The Adam Optimizer also has a bias-correcting step, where the moments are updated as

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \quad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}. \tag{2.19}$$

The model parameters are then iteratively updated as

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t}+\varepsilon}, \tag{2.20}$$

where α is the learning rate and ε is a small constant to prevent division by zero. In order to prevent overfitting, the EventStreamGPT framework also uses decoupled weight decay as a regularization technique, with PyTorch's AdamW optimizer. Including decoupled weight decay, the update rule can be written as

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha\left(\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t}+\varepsilon} + \gamma\boldsymbol{\theta}_{t-1}\right), \tag{2.21}$$

where γ is the weight decay factor [21]. Model parameters are updated until convergence of the tuning (validation) loss.

2.2 Transformers

The transformer first introduced in the paper *Attention is all you need* in 2017 is an encoder-decoder based deep learning model architecture used for sequence modeling [4]. Transformers have been shown to be superior encoders for the history in neural TPPs [22]. As opposed to other sequence modeling architectures, the transformer solely relies on the notion of *attention* to capture global dependencies. The encoder takes an input sequence and transforms it into a sequence of tokens in some continuous representation space. The task of the decoder is then to, given the encoded sequence, autoregressively generate a sequence of new tokens one at a time.

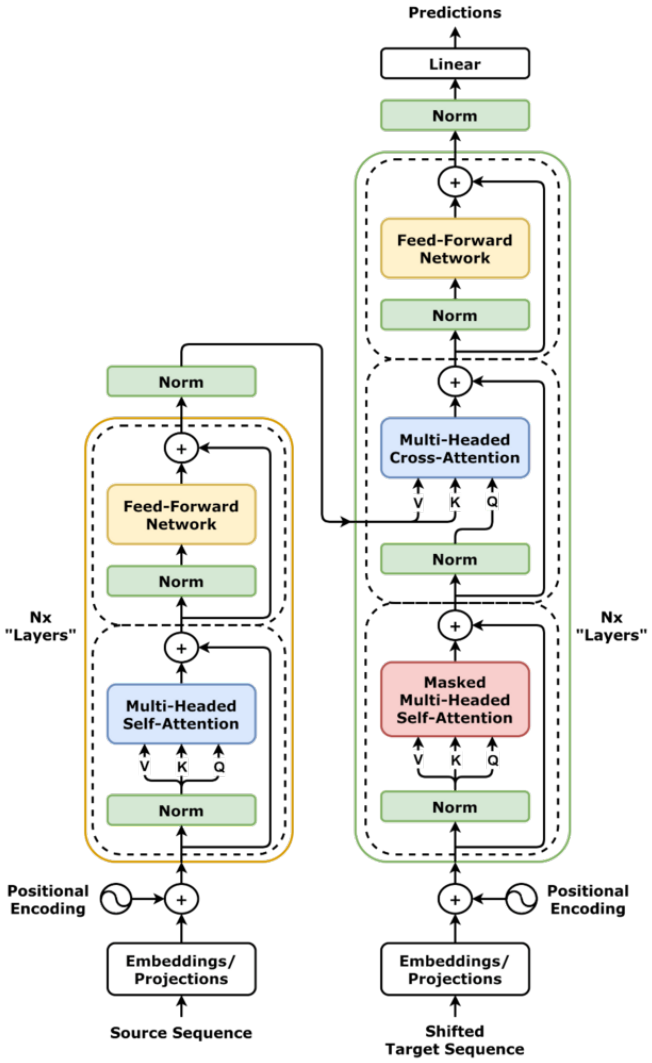


Figure 2.2: Illustration of the transformer architecture by dvgodoy/CC BY [23].

The encoder consists of N stacked blocks, each consisting of a multi-head self-attention layer and a position-wise fully connected feed forward network. The output from each output layer is fed through a normalization layer that also takes a residual input from before each sub-layer.

The decoder has a similar structure to the encoder as it also consists of N stacked blocks of multi-head self-attention layers and fully connected feed forward networks with residual connections around each sub-layer. In addition to these layers, the decoder has a second multi-head attention layer that takes as input the output of the first attention layer as well as the output from the encoder stack. Furthermore, to ensure that the output at position i only can depend on information at positions less than i the outputs are shifted by one position to the right before being fed to the decoder. The first attention layer is also modified to make it impossible to attend to subsequent positions by masking them out.

2.2.1 Attention mechanism

Attention allows the model to dynamically weigh the relevance of tokens in the input sequence, regardless of their actual distance. Two particular attention mechanisms are commonly used: *additive attention* and *multiplicative attention* [4]. Multiplicative attention, also called dot product attention, is the type of attention utilized in the transformer architecture with the addition of a scaling factor $\sqrt{d_k}$, where d_k is the dimension of the keys and queries. The scaling is introduced to counteract the dot products growing large in magnitude, which can push the softmax outputs into regions of vanishing gradients, especially when d_k is large. The attention is defined as follows

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.22}$$

where Q, K, and V are the query, key, and value matrices. These are computed as $Q = HW_Q$, $K = HW_K$, and $V = HW_V$, where H is a matrix containing stacked token embeddings, and W_Q, W_K, and W_V are learned projection matrices [24].

To capture richer relationships and dependencies within the data, the transformer architecture employs multi-head attention [4]. Multi-head attention allows the model to attend to information from different subspace representations at different positions in parallel. This is done by linearly projecting the queries, keys, and values into multiple learned subspaces, and applying the attention mechanism to each projected set in parallel. After the attention is done, the output of each head is concatenated into a vector of dimension $n \cdot d_v$, where n is the number of heads and d_v is the dimension of the value vectors. This output is then once again linearly projected into a d_{hidden}-dimensional vector to produce the final value.

2.2.2 Positional embeddings

Embeddings serve two main purposes: allowing the model to handle input sequences of different lengths, and giving the model positional information of each token [4]. The embedding layer of the transformer uses learned embeddings to convert each token into a d_{hidden}-dimensional vector.

The positional encoding is added to the input sequence after the embedding layer [4]. This is done by summing the positional encoding vector and the input embedding vector, hence the positional encoding has to match the dimensionality of the input embedding. There are several types of positional encoding schemes that can be used: learned, fixed or a relative encoding [25]. The fixed positional encoding scheme, initially proposed by Vaswani *et al.* [4], utilizes sine and cosine functions of different frequencies to embed positional information into the tokens. Specifically:

$$\mathrm{PE}_{(pos,2i)} = \sin(pos/10000^{2i/d_{hidden}}), \tag{2.23}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{hidden}}), \tag{2.24}$$

where pos denotes the position and i the dimension.

2.3 Foundation models

Foundation models represent a general class of machine learning models trained on diverse datasets that can be further adapted to a wide range of downstream tasks with little or no fine-tuning of its parameters [5]. Some prominent examples of foundation models include GPT-4 and BERT [26], [27]. These models are examples foundation models for natural language processing (NLP), which together with computer vision (CV) and graph learning are the three dominant applications of foundation models [28]. Developing foundation models for other modalities like event sequences comes with some added complexities due to the heterogeneity of the data that make up an event sequence. More specifically, datasets that contain different modalities and complex internal dependencies [7].

Foundation models are generally based on standard deep learning architectures and transfer learning techniques, most commonly the transformer architecture [28]. These are not new concepts, however the sheer scale of foundation models result in new emergent capabilities. These large-scale models are often trained used self-supervised learning techniques, which allow them to utilize vast amounts of unlabeled data from diverse domains. Larger models can be motivated by the empirical scaling laws introduced in 2020 by Kaplan *et al.* [29]. Their work shows that increasing model size, dataset size and compute power leads to predictable performance improvements.

2.3.1 Pretraining methods

Foundation models are often trained in two stages: pretraining and fine-tuning [5]. During pretraining, the models learn general-purpose representations of large-scale datasets using self-supervised training on pretext tasks. Training in a self-supervised manner on pretext tasks is useful because of its scalability. Depending on the general application of the foundation model, such as NLP or CV, different pretext tasks are used [28]. Some common tasks used for NLP foundation models include masked language modeling (MLM), replaced token detection (RTD) and next sentence prediction (NSP).

Masked language modeling, used in BERT, is the process of masking one word in a sentence and having the model predict the masked word based solely on the context of the remaining words [27]. A variant of MLM is autoregressive modeling, used in GPT-4, which concerns iteratively predicting the next token given the context of previous tokens [26].

RTD and NSP are both contrastive learning tasks [28]. Contrastive learning, common in computer vision tasks, is a self-supervised technique where the model is trained on comparing training examples to each other, grouping similar ones closely in the feature space. Specifically, RTD is a discriminant task where the model's target is to decern if the current token has been replaced or not [28]. NSP on the other hand aims to give the model sentence level knowledge by taking two sentences as input and comparing the order of the them.

Fine-tuning involves further tuning the parameters of a pretrained model on a specific downstream task of interest, often with a smaller task-specific dataset [5]. This is enabled by transfer learning, a machine learning technique where the knowledge gained from one task is applied to another related task [30]. This process utilizes the vast amount of pretraining data and reduces the need for large annotated datasets.

2.3.2 Fine-tuning pretrained models

Fine-tuning foundation models involves adapting the pretrained model to a new target task that can be widely different from the pretext tasks [31]. This can be achieved in various different ways, such as through supervised fine-tuning or reinforcement learning [32].

Definition 2. *Let $\mathcal{T}$ be a **task**. Then $\mathcal{T}$ is the combination of the label space $\mathcal{Y}$ and a decision function h, hence $\mathcal{T} = \{\mathcal{Y}, h\}$ [28]. The function h is not explicitly given but rather intended to be learned from data.*

Supervised fine-tuning utilizes supervised learning where each input has a corresponding label that the model learns to predict by adjusting its weights [32]. Two fine-tuning techniques that use supervised learning are Transfer learning and Multi-task learning.

Definition 3. *Let $\mathcal{D}$ be a **domain**. Then $\mathcal{D}$ consists of a feature space $\mathcal{X}$ and a marginal distribution $P(X)$, where X is defined as $X = \{\mathbf{x}|\mathbf{x_i} \in \mathcal{X},\ i = 1, \ldots, n\}$ [28].*

With the above definition of a domain, transfer learning can be formalized as follows:

Definition 4. *Given observations from $m_S \in \mathbb{N}^+$ source domains and tasks and observations from $m_T \in \mathbb{N}^+$ target domains and tasks, where m is the number of domains. Then **transfer learning** aims to utilize the knowledge contained in the source domains to enhance the performance of the decision functions h_{T_i} on the target domains [28].*

With multitask learning, the model is fine-tuned on several proxy tasks $\mathcal{T}_i$ [31], [32]. Such tasks may cover the information needed in the target task, they can therefore serve as a intermediate step between pretraining and fine-tuning on the target task. Multi task learning can then be formalized as:

Definition 5. *Suppose there are M proxy tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_M\}$. Given the pre-trained model $f_{\boldsymbol{\theta}}$, then fine-tuning the model is achieved by the following objective:*

$$\min_{f_{\boldsymbol{\theta}}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_i(\mathcal{T}_i, f_{\boldsymbol{\theta}}), \tag{2.25}$$

where $\mathcal{L}_i$ is the loss function for each task $\mathcal{T}_i$ [31].

2.3.3 Continual learning and catastrophic forgetting

Continual learning refers to the concept of learning a sequence of tasks or domains one after the other and subsequently behave as if they were learned simultaneously [33]. Fayek *et al.* [34] further defines continual learning as:

> *"The system has performed K tasks. When faced with the $(K+1)$th task, it uses the knowledge gained form the K tasks to help the $(K+1)$th task."*

Continual learning can be performed in many different ways and Wang *et al.* [33] introduce several common scenarios, of which a few are described below.

Instance-Incremental Learning refers to the model being trained on a single task where the training data comes in batches. All samples share the same objective throughout training. *Domain-Incremental Learning* by contrast, involves multiple tasks that share a common label space but have different input distributions and the model is not informed about which task it is currently addressing. Finally, in *Task-Incremental Learning* each task has a distinct label space and the model is informed on which task is currently at hand, both during training and evaluation.

A significant challenge often encountered in continual learning is the notion of catastrophic forgetting, where the model's performance on task K decreases after it has been trained on task $(K+1)$ [33], [34]. This problem is a result of the stability-plasticity dilemma [35]. The stability-plasticity dilemma represents the trade-off between retaining information of prior tasks and the ability to learn new information when presented with unseen tasks [36]. The model has to have sufficient plasticity to be able to acquire knowledge from learning on new tasks, however, too much plasticity leads to forgetting of previously learned tasks. This is also true for the opposite.

The stability of a model, i.e. how prone it is to catastrophic forgetting can be measured by a *forgetting measure* (FM), which Wang *et al.* [33] defines as the average forgetting across all tasks. The forgetting of a task is defined as the difference between the maximum performance and the current performance on a task.

$$f_{j,k} = \max_{i \in \{1,\, ...,\, k-1\}} (p_{i,j} \; - \; p_{k,j}), \quad \forall j < k \tag{2.26}$$

where $p_{k,j}$ is the performance of j-th task after training on the k-th task ($j \leq k$) in the continual learning setting, evaluated on the held-out test set. The forgetting measure is then calculated as

$$\mathrm{FM}_k = \frac{1}{k-1} \sum_{j=1}^{k-1} f_{j,k}. \tag{2.27}$$

Wang *et al.* [33] also introduces *backward transfer* (BWT), which measures how much learning a new task influences the old tasks on average. *Backward transfer* is defined as:

$$\mathrm{BWT}_k = \frac{1}{1-k} \sum_{j=1}^{k-1} (p_{k,j} \; - \; p_{j,j}). \tag{2.28}$$

Plasticity can, on the other hand, be measured by *forward transfer* (FWT) [33]. Let $\tilde{p}_j$ be the performance of a randomly initialized model, trained only on the j-th task. Then *forward transfer* is defined as the average influence that previous tasks has on the current task

$$\mathrm{FWT}_k = \frac{1}{k-1} \sum_{j=2}^{k} (p_{j,j} \; - \; \tilde{p}_j). \tag{2.29}$$

2.3.4 Activation functions

Activation functions are fundamental to neural networks, introducing nonlinearity and allowing the networks to learn complex patterns. Initial artificial neurons employed a binary threshold [37]. Later on, this threshold was smoothed using a sigmoid function in order to allow gradients to be calculated, which in turn enabled backpropagation, the important algorithm introduced in the work of Hopfield in 1982 [38]. With the rise of deep neural networks, selecting the activation function has now become a standard design choice. Introduced in 2016, the Gaussian Error Linear Unit (GELU) has been shown to match or exceed models using popular activation functions such as ReLU or ELU across several tasks [39]. The GELU function is used for the neural network in this thesis work and is defined by

$$\mathrm{GELU}(x) = x\Phi(x) = x\frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right], \tag{2.30}$$

where $\Phi(x)$ is the standard Gaussian cumulative distribution function [39]. The error function is defined as $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

3
Methodology

To model event sequences, and subsequently predict the time until the next grid interruption, we utilize and build on the work presented in McDermott *et al.* [7]. Their framework EventStreamGPT (ESGPT) offers several utilities that fit both the problem formulation, as well as the data described in Section 1.3. For example, handling diverse datasets in a unified manner and converting data to a deep-learning friendly format. The framework contains two model architectures and scripts for pretraining, fine-tuning and hyperparameter sweeps through Weights & Biases.

We denote the shared transformer encoder as f_θ and task-specific heads as $h_\phi^{(i)}$. We will also follow the naming convention of the ESGPT framework with train, tuning (validation) and held-out (test) data splits.

3.1 Data pre-processing

To be able to utilize all the pre-processing functionality that ESGPT provides, some manual pre-processing needs to be done on the raw data. The datasets are stored in CSV-files, with a general structure depicted in Table 3.1.

Table 3.1: General structure of the raw datasets before any pre-processing.

`timestamp`	`event`	`feature 1`	`feature 2`	⋯
YYYY-MM-DD hh:mm:ss	event 1	xxx	xxx	⋯
YYYY-MM-DD hh:mm:ss	event 2	xxx	xxx	⋯
YYYY-MM-DD hh:mm:ss	event 2	xxx	xxx	⋯
⋮	⋮	⋮	⋮	⋱

The pre-processing pipeline within ESGPT needs two data frames in order to be able to process the data correctly, namely a *subject* data frame and an *event* data frame. The subject data frame contains the `subject_id`:s that uniquely identifies each event sequence and any static measures that is tied to each respective subject. The event data frame on the other hand contains all dynamic features, i.e. information that can vary over time, as well as the timestamps of each event. To get a better notion of what a *subject* and *event* data frame may contain for a specific dataset, an example is provided below.

Example 1. *Let's consider the dataset "eCommerce 1". Before any pre-processing the data would be formatted in the following structure.*

Table 3.2: The structure of the raw data in the dataset "eCommerce 1".

event_time	event_type	user_id	price	⋯
2020-09-24 11:57:06	view	1515915625519388267	31.90	⋯
2020-09-24 11:57:26	view	1515915625519380411	17.16	⋯
2020-09-24 12:02:53	cart	1515915625519390468	217.57	⋯
2020-09-24 12:04:10	purchase	1515915625519390468	217.57	⋯
2020-09-24 12:09:33	view	1515915625356203891	21.43	⋯
⋮	⋮	⋮	⋮	⋱

After the initial pre-processing we would instead have the following two data frames given in Table 3.3 and Table 3.4:

Table 3.3: Structure of the *subject* data frame for dataset "eCommerce 1".

subject_id	static_measure_1
0	1
1	1
2	1
3	1
⋮	⋮

Table 3.4: Structure of the *event* data frame for dataset "eCommerce 1".

subject_id	event_time	event_type	price
0	2020-09-24 11:57:06	view	31.90
1	2020-09-24 11:57:26	view	17.16
2	2020-09-24 12:02:53	cart	217.57
2	2020-09-24 12:04:10	purchase	217.57
3	2020-09-24 12:09:33	view	21.43
⋮	⋮	⋮	⋮

We now have the two data frames required by ESGPT. As can be observed in table 3.3 and 3.4, the coulmn ***user_id*** *is now replaced by the column* ***subject_id****, which essentially contains the same information but the unique* ***user_id****:s in table 3.2 have been mapped to values in the range* $[1, n_{unique}]$*, where* n_{unique} *is the number of unique* ***user_id****:s in the dataset.*

The renaming from `user_id` *to* `subject_id` *and the remapping of the values in that column is due to the fact that when combining multiple datasets, it is only possible to have a single subject data frame for all datasets. The column name* `subject_id` *in the subject data frame also has to be the same in the event data frame for each dataset.*

3.1.1 Creating event sequences

In the given example, each event sequence corresponds to a unique `subject_id`, which need to be assigned manually. Event sequences for the fault dataset were created by selecting all events within a 3-day window and assigning the same `subject_id` to these events. The window was then shifted by three days, and the new set of events were assigned a new `subject_id`. This approach ensures that the event sequences are non-overlapping, which helps minimize redundancy in the dataset and reduces risk of data leakage. With this setup, the fault data was split into 445 non-overlapping event sequences, of which 100 were reserved for the held-out set. Because of the limited number of sequences available, a relatively large proportion was set aside for the held-out set to ensure reliable evaluation. Out of the remaining 345 sequences, 90 % were used for the training set and the remainder in the tuning set.

Several configurations of the sequence generation pipeline were evaluated, varying both sequence length and overlap. The chosen setup was eventually settled on because it maximized performance on the proxy tasks. The trade-off between increasing the number of training examples by using overlap, and the resulting correlation and redundancy between sequences, was not obvious to the performance.

The other datasets used for the foundation model approach had some intrinsic notion of subjects, such as the column `user_id` described in Example 1. These were directly used as subjects, meaning no manual creation of event sequences was necessary other than for the fault dataset.

3.1.2 Merging datasets

In order to pretrain the foundation model on several general datasets in parallel, we need to combine all datasets into one. This combined dataset is only used in the pretraining stage of the foundation model and will be referred to as $\mathcal{S}_{\text{pre}}$. Because of the sheer size of datasets *eCommerce 1* and *eCommerce 2*, only the first 100 000 subjects were selected from these. The `subject_id`:s of the first dataset were mapped to integers starting from zero. Each subsequent dataset had its subjects mapped starting from the last used integer plus one, ensuring that all id:s remain unique. The fault dataset was added last in the combined dataset to simplify the later splitting of datasets into train, tuning and held-out sets. Lastly, a combined subject data frame was created containing all `subject_id`:s. The dataset $\mathcal{S}_{\text{pre}}$ contains approximately 202 500 subjects and 5.9 million events.

After the initial manual pre-processing, the ESGPT data pre-processing pipeline was used. This pipeline performs several operations on the input data, such as split into train, validation and test sets, fit outlier detector, normalizers and vocabularies etc. [7]. Furthermore, the pipeline then converts the data into a deep learning friendly representation that is stored in PyTorch datasets and dataloaders.

3.2 Model architecture

The model consists of two main components, the transformer block and task-specific output heads. The transformer used in the model is called *Conditionally Independent Point Process Transformer* (CIPPT), which is a decoder-only transformer with an architecture identical to that of a standard GPT Neo-X transformer, as stated in the supplementary material of McDermott *et al.* [7]. An overview of the model architecture is provided in Figure 3.1.

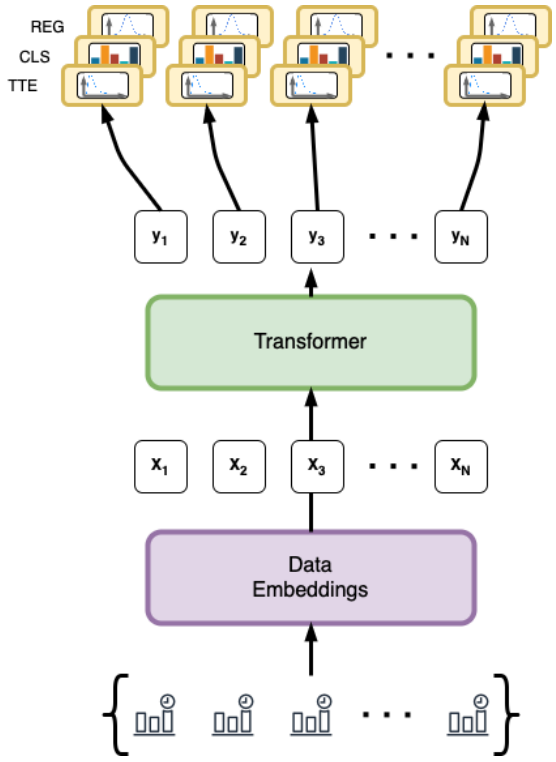


Figure 3.1: Overall model architecture of the *Conditionally Independent Point Process Transformer* used in this work. Here $\{\mathbf{x_1}...\mathbf{x_n}\}$ is the output of the data embedding layer and $\{\mathbf{y_1}...\mathbf{y_n}\}$ is the latent space representation of the input event sequence.

The input data is initially fed through an embedding layer before being fed through the transformer backbone to produce the latent representation $\{\mathbf{y_1} \dots \mathbf{y_n}\}$. Finally, each $\mathbf{y}_i$ is then passed to the three output heads, which performs the final downstream task. The base heads in the ESGPT framework perform classification, regression and time-to-event. When fine-tuning the model, the $\mathbf{y_i}$:s are also fed through the task-specific head.

The task-specific output heads, as the name suggests, have different properties depending on the specific task the model is being fine-tuned on. To summarize, the model has six different output heads in total that each handle different types of training tasks. Three of these output heads are provided within the ESGPT library, namely a pretrain classification head, pretrain regression head and a time-to-event head. The remaining three output heads were our own additions, added with the purpose to fit specific fine-tuning tasks. The task-specific heads *Event head*, *Sequence head* and *Prediction head* are discussed in more detail in Section 3.2.2

3.2.1 Output heads for pretraining

The ESGPT framework features three output heads: time-to-event head, and regression and classification heads. The time-to-event and regression output heads are responsible for modeling and subsequently predicting the inter-event time between sequential events and any numerical features connected to each event [7], [40]. This is achieved by parameterizing a probabilistic distribution over the inter-event times and each feature for each event, conditioned on the event encoded in the latent representation produced by the transformer.

Let $\mathbf{h} \in \mathbb{R}^{B \times L \times d_{hidden}}$ denote the latent representation produced by the transformer, where B is the batch size, L the sequence length and d_{hidden} is the hidden dimension. Each d_{hidden}-dimensional vector $\mathbf{y}_i$ in $\mathbf{h}$ is passed through a linear projection layer that maps the latent space vector into a vector containing the parameters of the distribution that we want to parametrize. As mentioned in Section 2.1.2, the time-to-event output head has two distributions already implemented to choose from, namely an exponential distribution or a log-normal mixture distribution. The pretrain regression output head on the other hand parametrizes a Gaussian distribution by default to regress feature values. During generation (prediction), we sample from the specific distribution with the parameters obtained in each head respectively and thus obtaining numerical predictions for both time-to-event and numerical features.

The pretrain classification output head on the other hand handles all classification tasks, i.e. event type and any categorical features connected to each event. This is done by feeding the latent space representation of each event through a linear layer with a output size corresponding to the vocabulary size.

3.2.2 Task-specific heads

Three output heads were added to the ESGPT model architecture in order to achieve outputs for different types of proxy tasks. The first head processes the transformer latent representation into one output for each individual event, this head is referred to as the *Event head*. It is a linear layer with input size matching the output of the transformer backbone $\mathbf{h}$, and one output neuron. Let $h_\phi^{(1)}$ denote the *Event head*, defined as a function

$$h_\phi^{(1)} : \mathbb{R}^{B \times L \times d_{hidden}} \to \mathbb{R}^{B \times L}.$$

The output of this head has shape $B \times L$, where each element corresponds to a single logit per event. These logits are then passed element-wise through a sigmoid function to map them to the range (0,1). The outputs can therefore be interpreted as some task-specific probability and compared to event labels. This head will be used for tasks dealing with classifications on event-level, which will be discussed in further detail in Section 3.3.3.

The second head, referred to as the *Sequence head*, was added in order to deal with tasks that have labels on the sequence-level. It is similar to the *Event head*, but instead maps the entire sequence of events to a single logit. This is done with a two-layer fully connected neural network, which has the same input dimension, $B \times L \times d_{hidden}$, a second layer with L neurons and a GELU activation function, and a final single output neuron. Let $h_\phi^{(2)}$ denote the *Sequence head* as

$$h_\phi^{(2)} : \mathbb{R}^{B \times L \times d_{hidden}} \to \mathbb{R}^B.$$

This output head will be used to fine-tune on proxy tasks that have sequence-level labels.

The third head, *Prediction head*, has the same architecture and purpose as the *Sequence head*. There are multiple sequence-level tasks, and this head was added in order to have a separate head specifically for tasks dealing with fault prediction. Let $h_\phi^{(3)}$ denote the *Prediction head* with

$$h_\phi^{(3)} : \mathbb{R}^{B \times L \times d_{hidden}} \to \mathbb{R}^B,$$

which is used to fine-tune the model on prediction tasks.

3.3 Training

This section will describe how the machine learning model is trained in the ESGPT framework, as well as with our own additions. The optimal hyperparameters are found via a separate Weights & Biases hyperparameter sweep for pretraining and each fine-tuning task. This sweep searches for hyperparameters in specified ranges that result in the lowest tuning loss.

3.3.1 Pretraining

Large deep learning models often need huge amounts of training data, and foundation models often utilize a training paradigm called self-supervised learning [5]. Pretraining with this technique is often preferred, since it enables training on large amounts of unlabeled data. Large-scale self-supervised training allows the model to gain general knowledge about the data it is being trained on. This is done by designing *pretext tasks* where the model is trained to predict some aspect of the input data itself [41].

Specifically, CIPPT is pretrained on *next token prediction*, where each token is the latent space representation of an event. The actual prediction is then performed on three sub-tasks in parallel. These are time-to-event, feature value regression and event label classification. At each training step, a single event in the input sequence is masked, and the self attention mechanism attends to all preceding events, thus giving the model context for predicting the subsequent token [41]. The predicted values for each sub-task are compared to the actual values of the masked event and the loss for that event and sub-task can be calculated. The total loss function is then calculated as the sum of the losses from each sub-task. This is repeated for every event sequence in the training data.

3.3.2 Fine-tuning

Fine-tuning is done on specific *proxy tasks*, these are training tasks that are related to the goal task but can be simpler in nature or use a task-specific dataset. The purpose of fine-tuning the model weights on proxy tasks is to gradually train the model on tasks of increasing complexity and improve its general understanding of event sequence data. In order to fine-tune on specific tasks, we first need to create a *task* data frame to be used in the ESGPT framwork. This data frame contains one label for each `subject_id`. The label can be an integer for classification tasks or a float value for regression tasks.

Table 3.5: Example of a *task* data frame on a synthetic dataset. This example has binary labels, but regression tasks can have floating point label values as well.

subject_id	start_time	end_time	label
0	2015-01-01 00:00:00	2015-01-08 00:00:00	0
2	2015-01-08 00:00:00	2015-01-15 00:00:00	0
1	2015-01-15 00:00:00	2015-01-22 00:00:00	0
3	2015-01-22 00:00:00	2015-01-29 00:00:00	1
⋮	⋮	⋮	⋮
9986	2206-07-24 00:00:00	2206-07-31 00:00:00	1
9997	2206-07-31 00:00:00	2206-08-07 00:00:00	0
9998	2206-08-07 00:00:00	2206-08-14 00:00:00	1
9999	2206-08-14 00:00:00	2206-08-21 00:00:00	0

3.3.3 Proxy tasks

The set of proxy tasks used in this work is described below. Ideally, proxy tasks should be orthogonal, meaning they capture different properties of event sequences and each contribute unique information to the model. All proxy tasks are derived from the fault dataset and will be used for both the specialized model and the foundation model approach.

Event label (EL) is a simple proxy task where the model is trained on classifying which events are interruptions. Interruption events get positive labels and all other classes get negative labels for this task, which is optimized by minimizing a binary cross-entropy loss. Since this proxy task has event-level labels, it uses the *Event head*.

Class distribution (CD) has the purpose of further strengthening the model's ability to tell event types apart. Given an event sequence, the task is to predict the distribution of classes in this sequence. This is a regression task and is trained on a mean square error loss using the time-to-event and regression pretraining heads, and is therefore only training the embedding and transformer weights.

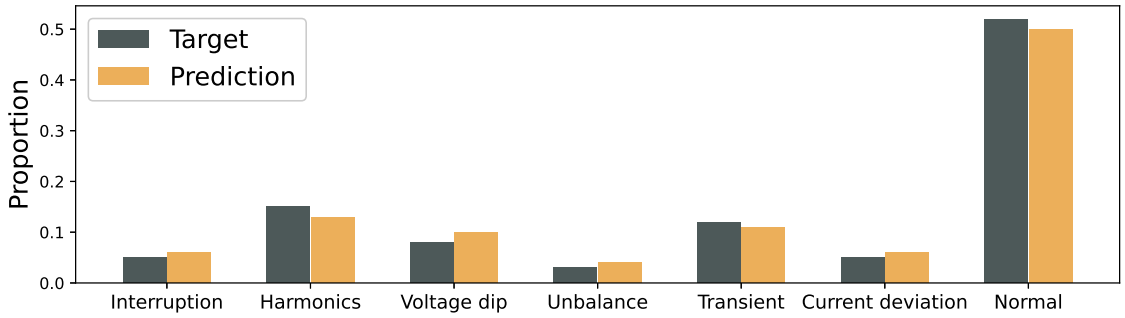


Figure 3.2: Illustration of the learning objective for the *Class distribution* task. The model is trained to minimize the error between predicted and actual distribution of event types in the input sequence.

Interruption in sequence, which will be referred to as I0 in subsequent figures, is a binary classification proxy task with the purpose of enhancing the model's ability to discern interruptions from other disturbances. Given a sequence of events, the task is to predict label 1 if at least one of the events in the sequence is an interruption, and 0 otherwise. This is therefore a sequence-level proxy task, and uses the *Sequence head*.

Interruption 3-day concerns predicting if an interruption will happen in the next three days. The `end_time` column in the task data frame is used as the time of prediction for each input sequence. All proxy tasks concerning prediction of future interruptions use the *Prediction head*. Variants of this proxy task were also implemented with increasing prediction window lengths, for example 5-day and 7-day, where prediction with a window length of 7 days is the goal downstream task. In subsequent figures, I3, I5 and I7 will be used to reference the proxy tasks for interruption in 3, 5 and 7 days respectively.

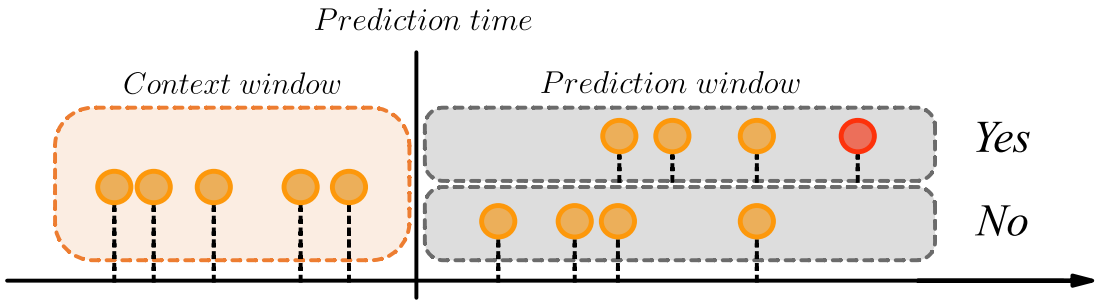


Figure 3.3: Illustration of the learning objective for interruption prediction tasks. Context window refers to the length of the input sequence.

Time-to-interruption was initially considered to be used as a proxy task. This regression task deals with trying to predict the time until the next fault will occur, based on the history of events. After careful consideration and testing, this proxy task was discarded due to it requiring its own output head for regression tasks, and importantly, being even more challenging than the goal task *Interruption 7-day*.

During fine-tuning, the model trains on the pretraining tasks in parallel to one of the fine-tuning proxy tasks. This is achieved by constructing a combined loss function on the form

$$\mathcal{L} = \mathcal{L}_{TTE} + \mathcal{L}_{PC} + \mathcal{L}_{PR} + w_{task}\mathcal{L}_{task}, \tag{3.1}$$

where w_{task} is a weighting factor for the task loss, TTE represents time-to-event, PC pretrain classification and PR is pretrain regression. The task loss $\mathcal{L}_{task}$ is either a binary cross-entropy or mean square error loss depending on the current training task. By giving each fine-tuning task a weight factor for its loss part, we can tune its influence on the overall training.

3.3.4 Optimizing the fine-tuning strategy

Predicting if an interruption will occur in the following seven days is a challenging task. By following a structured finetuning strategy, we can allow the model to gradually work its way up to this complex end task. A suitable strategy could be to start pretraining from the simplest and most general tasks, and later moving towards the end prediction task. Pretraining datasets aim to give the model a better starting point to learn from. Each proxy task requires its own task data frame containing one label per subject. Because of this limitation, fine-tuning on several proxy tasks needs to be done sequentially, rather than in parallel.

Training sequentially can be problematic for two main reasons. First, each dataset has a different vocabulary size, which is the input dimensionality for the embedding layer. It can therefore be challenging to reuse the same embedding weights. This is however not an issue in our case, since pretraining is self-supervised and done in parallel, and fine-tuning is only done on the fault dataset. The second potential

problem with training proxy tasks sequentially is the risk of *catastrophic forgetting*, where the knowledge from early tasks are overwritten and forgotten [8].

The end task is *Interruption 7-day*, and it makes sense to train on this task last. The interruption prediction tasks with shorter prediction windows are seen as sub-tasks for the seven day prediction, and are trained on right before training on *Interruption 7-day*. The optimal ordering of the remaining proxy tasks is however ambiguous, especially considering that it is difficult to rank them in terms of complexity. Consequently, all possible permutations of the remaining proxy tasks were tested, followed by training on the interruption tasks in order of increasing prediction window length. Evaluation was done on performance of the end task using the held-out fault dataset.

3.3.5 Training algorithm

This subsection will summarize the training process of pretraining and fine-tuning with pseudo-code in Algorithm 1. Hyperparameters for each task are found from a hyperparameter sweep for each task. Ordering of the fine-tuning tasks were found as described in Section 3.3.4. The pretraining dataset $\mathcal{S}_{\text{pre}}$ is unlabeled, which is why the pretraining stage uses self-supervised training with masking. Each fine-tuning dataset $\mathcal{S}_{\text{fine}}^{(i)}$ has labels from its corresponding task data frame. In the specialized fault model, all datasets are derived from the proprietary fault dataset, only differing in the task labels. In the foundation model approach, the pretraining dataset contains multiple datasets.

Algorithm 1 Entire training procedure for pretraining and sequential fine-tuning of the transformer model with added output heads.

Input: Pretrain data $\mathcal{S}_{\text{pre}}$, Finetuning datasets $\{\mathcal{S}_{\text{fine}}^{(i)}\}$ for tasks $i = 1, \ldots, N$
Input: Transformer f_θ, Task-specific output heads $\{h_\phi^{(i)}\}$
Input: Epochs E_{pre}, $\{E_{\text{fine}}^{(i)}\}$, Learning rates α_{pre}, $\{\alpha_{\text{fine}}^{(i)}\}$, Weight decays γ_{pre}, $\{\gamma_{\text{fine}}^{(i)}\}$
Initialize transformer parameters $\boldsymbol{\theta}$ and head parameters $\{\boldsymbol{\phi}^{(i)}\}$
for epoch = 1 to E_{pre} **do**
  for each batch B in $\mathcal{S}_{\text{pre}}$ **do**
    Apply masking to obtain $\tilde{B}$
    $Z \leftarrow f_\theta(\tilde{B})$
    Compute loss $\mathcal{L}_{\text{pre}}(Z, B)$
    Update $\boldsymbol{\theta}$ using eq. 2.21 with learning rate α_{pre}, weight decay γ_{pre}
  end for
end for
for task $i = 1$ to N **do**
  for epoch = 1 to $E_{\text{fine}}^{(i)}$ **do**
    for each batch (B_i, y_i) in $\mathcal{S}_{\text{fine}}^{(i)}$ **do**
      $Z \leftarrow f_\theta(B_i)$
      $\hat{y}_i \leftarrow h_\phi^{(i)}(Z)$
      Compute loss $\mathcal{L}_{\text{fine}} = \mathcal{L}_{\text{pre}}(Z, B_i) + \mathcal{L}(\hat{y}_i, y_i)$
      Update $\boldsymbol{\theta}$, $\boldsymbol{\phi}^{(i)}$ using eq. 2.21 with learning rate $\alpha_{\text{fine}}^{(i)}$, weight decay $\gamma_{\text{fine}}^{(i)}$
    end for
  end for
end for
Return: Fine-tuned model $(f_\theta, \{h_\phi^{(i)}\})$

4
Results

This chapter will present and discuss the results achieved in this thesis work. Results will be presented in four sections: first, we present baseline task performance, second we show how shortening the prediction window effects the prediction performance. Third, which fine-tuning strategy was found to be optimal and finally how prone are the models are to catastrophic forgetting. Each section will contain comparisons between the specialized model, which has only been trained on fault data from electric grids, and the foundation model approach which has been trained on data from several diverse domains.

The two models were trained on a server equipped with two Nvidia GeForce RTX 3080 GPUs. Given these hardware constraints, some hyperparameters regarding the model size were set beforehand, and was kept fixed for all experiments. Since foundation models typically are very large neural networks, these parameters were found through trial and error in order to achieve the maximum possible model size. The following backbone parameters were used: `head_dim = 18`, `hidden_size = 180`, `intermediate_size = 128`, `num_attention_heads = 10` and `num_hidden_layers = 8`. These parameters result in models with approximately 1.6 million trainable parameters. An extensive list of used hyperparameters is found in Appendix A.3.

Furthermore, the task loss weighting factor introduced in Equation 3.1, was found through trail and error for each task individually, and kept constant for all experiments. The following weights were used: *Class distribution*: 1e4, *Event label*: 10, and all interruption classification tasks had a task loss weighting of 2. These weights assure that the task loss is a large contributing factor in the overall train loss.

4.1 Baseline task performance

This section will present results on the baseline performance on all proxy tasks, without any fine-tuning strategy. After pretraining on their respective pretraining datasets, the models were fine-tuned on only one proxy task, and then evaluated on the held-out set. While the pretraining datasets differ between the specialized model and the foundation model, all fine-tuning will be done using the proprietary fault dataset. This dataset contains 445 number of training examples, where one training example here is an entire event sequence. Note that the class imbalances in the task data frames differ substantially depending on the proxy task. *Event label* only

has 0.44 % positive samples, but *Interruption in sequence* has 13.0 %, *Interruption 3-day* has 10.0 %, *Interruption 5-day* has 16.0 % and *Interruption 7-day* has 21.0 % positive labels. The class imbalances were calculated on the held-out set containing 100 sequences, since these values have an impact on the evaluation metrics that are also calculated on the held-out set. The class imbalances over all data splits was very similar, differing at most about one percentage point, also confirming that the held-out set is representative. *Event label* has very imbalanced classes, and to counteract this, a positive weight of 600 was used when training this task. Since the end task is fault prediction, correctly identifying interruptions while minimizing the amount of false alarms is critical. The focus is therefore on achieving a sufficiently high precision.

Hyperparameter sweeps were run on all tasks and pretraining individually to find optimal hyperparameters, a subset of which is presented in Table 4.1. The best hyperparameters for each task was chosen from a combination of high average precision and low tuning loss. Overfitting was then controlled by the tuning loss, which resulted in training for n_{epochs} epochs.

Table 4.1: Important hyperparameters for all tasks across the two models. Values were found with a Bayesian Weights & Biases hyperparameter sweep. All tasks were trained until the tuning loss started to show overfitting. The learning rate is α and the weight decay factor is denoted γ.

(a) Specialized fault model.

Task	$\boldsymbol{\alpha}_{\text{init}}$	$\boldsymbol{\alpha}_{\text{end}}$	$\boldsymbol{\gamma}$	$\boldsymbol{n}_{\text{epochs}}$
Pretraining Specialized	8.00e-4	3.20e-7	3.00e-4	115
Class distribution	9.42e-4	1.99e-7	3.92e-3	16
Event label	6.10e-2	7.27e-5	5.69e-4	23
Interruption in sequence	7.60e-3	1.34e-5	1.56e-1	14
Interruption 3-day	1.25e-1	1.26e-5	2.15e-1	15
Interruption 5-day	7.86e-2	1.10e-2	3.66e-1	30
Interruption 7-day	7.93e-2	2.00e-2	3.13e-1	50

(b) Foundation model.

Task	$\boldsymbol{\alpha}_{\text{init}}$	$\boldsymbol{\alpha}_{\text{end}}$	$\boldsymbol{\gamma}$	$\boldsymbol{n}_{\text{epochs}}$
Pretraining Foundation	5.11e-6	3.85e-6	4.14e-1	87
Class distribution	7.20e-4	6.88e-7	7.79e-2	50
Event label	4.64e-4	2.13e-4	6.15e-2	33
Interruption in sequence	6.60e-4	1.22e-4	2.17e-1	16
Interruption 3-day	2.06e-4	3.21e-5	8.11e-2	50
Interruption 5-day	5.81e-5	1.36e-8	7.77e-2	27
Interruption 7-day	5.20e-5	7.01e-9	8.58e-2	100

What can be noticed here is the value of the weight decay γ, which controls how strong the regularization is, and is used to prevent overfitting. For the specialized model it is largest for tasks concerning predicting future faults, which makes sense under the hypothesis that these are the most challenging tasks to learn, and prone

to overfitting. The weight decay for pretraining foundation model is three orders of magnitude larger than the value found for the specialized pretraining.

With solid hyperparameters, a baseline task performance on all tasks was inferred after pretraining on the respective pretraining datasets, and fine-tuning on only the specific task at hand. The regression task *Class distribution* used mean square error (MSE) as its evaluation metric, while the remaining classification tasks used a combination of precision, recall and average precision (AP). Average precision summarizes the precision-recall curve by computing the mean of precision across different recall values. Precision is a metric of the proportion of predicted positives that were actual positives. Recall instead measures the proportion of all positive labels that were correctly predicted by the model. The ranking metrics precision, recall and AP were calculated at a global level on the entire held-out set. Further discussion and definitions of used evaluation metrics are found in Appendix A.2.

Table 4.2: Baseline task performance of both approaches on the held-out set containing 100 sequences. Evaluation was done directly after pretraining on their respective pretraining datasets and fine-tuning on only that specific task. The threshold ξ represents the decision boundary at which probabilities are mapped to binary predictions.

(a) Specialized fault model baseline task performance on held-out set.

Task	$\boldsymbol{\xi}$	**Precision**	**Recall**	**AP**	**AP/Baseline**	**MSE**
Class distribution	-	-	-	-	-	0.0045
*Event label**	0.53	0.0392	0.0435	0.0150	3.47	-
Interruption in seq.	0.44	0.3077	0.3333	0.3116	2.40	-
Interruption 3-day	0.25	0.1099	1.0000	0.1036	1.04	-
Interruption 5-day	0.32	0.1852	1.0000	0.2116	1.79	-
Interruption 7-day	0.35	0.2198	1.0000	0.3031	1.44	-

(b) Foundation model baseline task performance on held-out set.

Task	$\boldsymbol{\xi}$	**Precision**	**Recall**	**AP**	**AP/Baseline**	**MSE**
Class distribution	-	-	-	-	-	0.0037
*Event label**	0.65	0.0353	0.6957	0.0303	6.95	-
Interruption in seq.	0.45	0.2353	0.3333	0.2499	1.92	-
Interruption 3-day	0.40	0.1014	0.7000	0.1424	1.42	-
Interruption 5-day	0.49	0.1923	0.3333	0.1850	1.16	-
Interruption 7-day	0.50	0.2045	0.4500	0.1954	0.93	-

Average precision does not give the full picture of the classification performance alone. Like many evaluation metrics, the values need to be put into perspective by the class imbalance. A random classifier achieves an average precision of $P/(P+N)$, which was the value used as *baseline* [42]. Here, P is the number of positive samples in the held-out set, and N is the number of negative samples. In Table 4.2, the

*Extreme class imbalance: *Event label* has 0.44 % positive labels.

column AP/baseline therefore shows how many times better than random guessing the classifier is, which is a more interpretable performance measure.

The threshold ξ was tuned on the train and tuning set because of the limited size of the tuning set. Due to lack of time it was not possible to retrain the models on a different split with a larger proportion allocated for the tuning set. This might have some impact on generalization capabilities and there is a risk of overfitting on the training data. In fact, some overfitting was observed as all the performance metrics were substantially higher when evaluated on the train set.

The same procedure to optimize the baseline task performance was followed for both the specialized model and the foundation model, and even though they use different hyperparameters, the performance can therefore still be compared. Table 4.2 paints a fairly clear picture as to which of the two model that has the best performance. The specialized model outperforms the foundation model on the majority of the tasks and metrics. Most notably, it achieves a very high recall while still retaining a precision that is as good or better than that of the foundation model. However, there is one task where the foundation model clearly outperforms the specialized model, namely *Event label*. For this task it achieves better metrics across the board except for precision which is more or less equal to that of the specialized model.

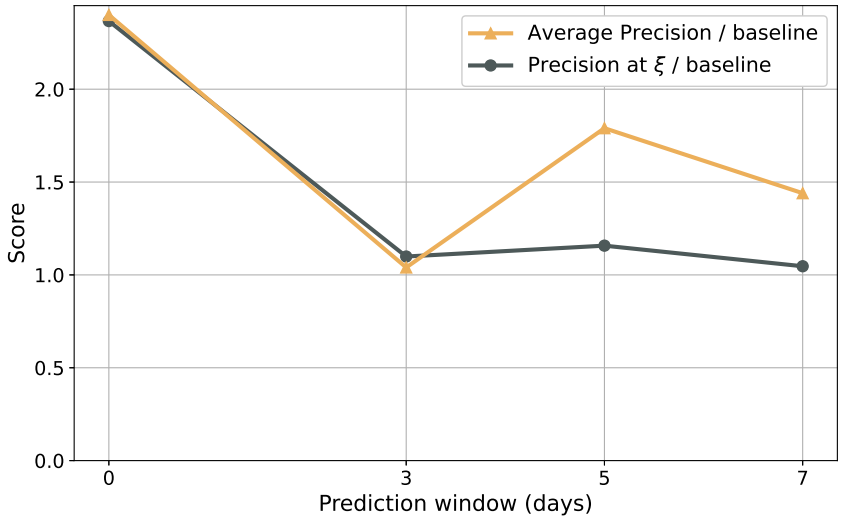
Ideally, both precision and recall should be high, however in practice it is rarely the case. The threshold value ξ was tuned to favor high precision over high recall, ensuring that the number of false positives are minimized. This is the probability above which the prediction is classified as positive. The threshold controls the trade-off between precision and recall, essentially tuning how sensitive the model is to faults. The optimal threshold value therefore depends on the application and users of the model. As seen in Table 4.2, even though the threshold was tuned to maximize the precision, the precision is still low with relatively high recall on all tasks. This suggests that the model may be inherently over sensitive. One contributing factor could be the imbalanced dataset, which in some cases are extreme. This implies that falsely predicting a couple samples as positive can have a big impact on the precision since the number of actual positives are small.

If we take the use-case perspective, it makes sense to tune the model to maximize precision rather than recall. If we achieve a precision of 1, that means that all the fault alarms from the model were true positives. Even though the model misses the majority of faults, it is still better than capturing all faults but be very uncertain if a predicted fault is a false alarm or not. A high recall value of course also desirable, but given that the starting point without a model is that no faults are predicted, having a model with high precision is only a net benefit, with trustworthy predictions.

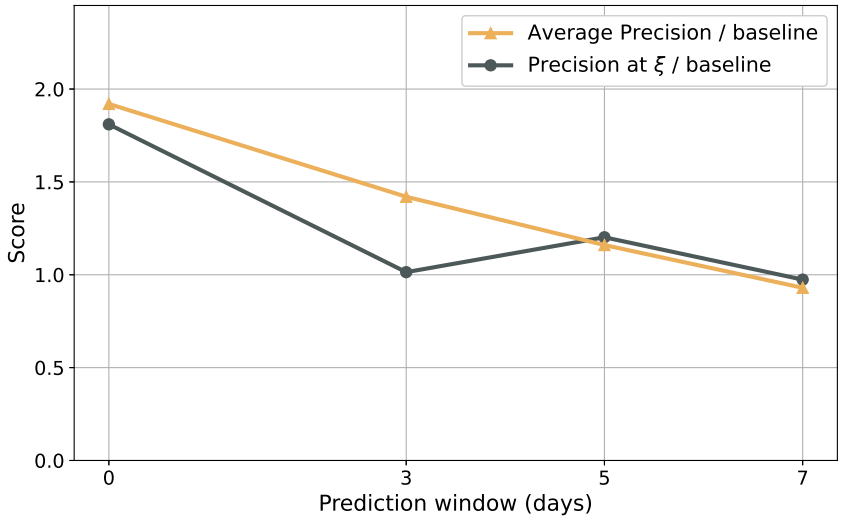
It is also worth noting that both models are very prone to overfitting, especially the foundation model, which on average achieved twice as high average precision on the training data compared to the held out set.

4.2 Shortening the prediction window

The end task is *Interruption 7-day*, and in theory, it should be easier to predict faults in shorter prediction windows. Such tasks can therefore be used as proxy tasks before training on the end task. Shortening the prediction window does however increase the imbalance between positive and negative labels. This is due to the fact that an interruption is less likely to occur within a shorter period of time as opposed to a longer period of time. The change in class balance could also potentially impact the learning difficulty of the task. Figure 4.1 presents average precision and precision at ξ for different prediction window lengths.



(a) Specialized fault model baseline.



(b) Foundation model baseline.

Figure 4.1: Average precision over baseline and precision over baseline at ξ for different prediction window lengths. Here a prediction window of zero days represents the proxy task *Interruption in sequence*. Each task used the threshold ξ presented in Table 4.2a. Evaluation was done on the held-out set.

The values in Figure 4.1 are divided by each tasks corresponding baseline value. The purpose of this is to eliminate the impact of varying class imbalance between tasks to ensure precisions are comparable. The specialized fault model (Figure 4.1a) performs best at the *Interruption in seq.* task. Precision drops for the prediction tasks, but is unintuitively lowest when using a prediction window of three days.

The foundation model approach (Figure 4.1b) has a linear decreasing average precision when increasing the prediction window length. For this model, predicting interruptions closer in time are easier, consistent with the prior belief. The thresholds for *Interruption 5-day* and *Interruption 7-day* result in higher than average precision on the held-out set.

The average precision metric is independent of the sensitivity tuning of the model, while precision at ξ is not. By comparing these metrics, we can get an estimate of how well-tuned the threshold is to maximize precision on the held-out set. In the specialized model, *Interruption in seq.* and *Interruption 3-day* have thresholds that result in about average precision. However, prediction windows of five and seven days have thresholds resulting in worse than average precision values. Due to the limited dataset size, there are distributional differences across data splits, meaning the threshold maximizing precision on the train set may not be the same for the held-out set.

4.3 Fine-tuning strategy

Fine-tuning is common when training foundation models. It involves training a pretrained model on proxy tasks to improve performance on the end task. We define the optimal fine-tuning strategy as the task ordering that maximizes performance on *Interruption 7-day*. In order to find the optimal strategy, different permutations of task ordering were trained and evaluated. The only constraint was that the final tasks were *Interruption 3-day*, *Interruption 5-day* and *Interruption 7-day* last, to ensure no catastrophic forgetting has occurred. Tasks were trained sequentially, meaning they continue updating model parameters where the previous task left off. The optimal fine-tuning strategies for both approaches are presented in table 4.3.

Table 4.3: Optimal fine-tuning strategy with task ordering for both approaches.

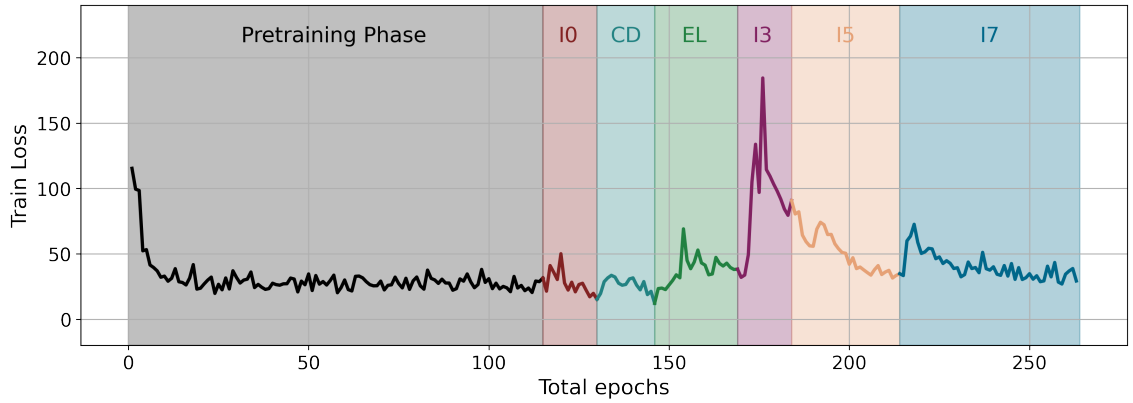
(a) Specialized fault model

Task	$\mathbf{n}_{\text{epochs}}$
1. *Interruption in seq.*	14
2. *Class distribution*	16
3. *Event label*	23
4. *Interruption 3-day*	15
5. *Interruption 5-day*	30
6. *Interruption 7-day*	50

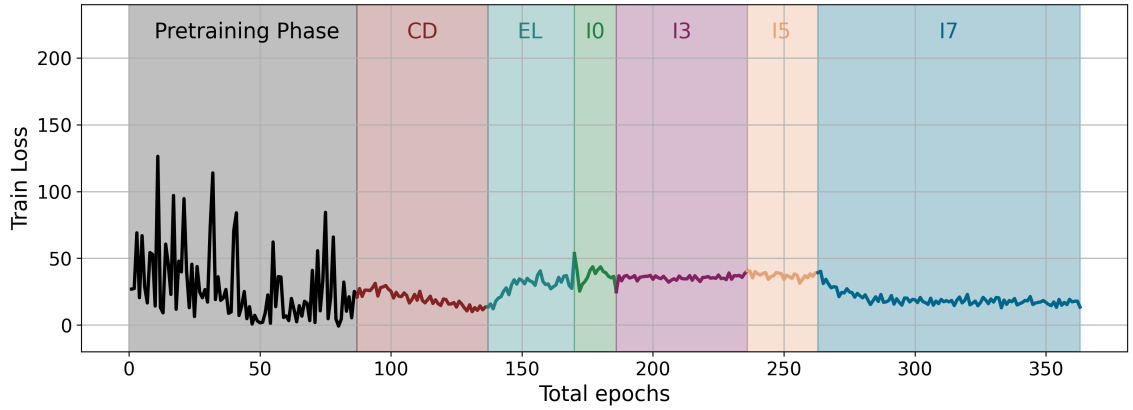
(b) Foundation model

Task	$\mathbf{n}_{\text{epochs}}$
1. *Class distribution*	16
2. *Event label*	23
3. *Interruption in seq.*	14
4. *Interruption 3-day*	15
5. *Interruption 5-day*	30
6. *Interruption 7-day*	100

As evident, the optimal fine-tuning strategy differs between the two approaches. To visualize part of the training process across the whole fine-tuning strategy, a learning curve of the train loss is shown in Figure 4.2. Note that only the *train loss* is shown, not the total loss, which also includes *task loss*. This is because the task loss differs depending on task, while the train loss is part of all training. As a reminder, the *train loss* i defined as $\mathcal{L} = \mathcal{L}_{TTE} + \mathcal{L}_{PC} + \mathcal{L}_{PR}$.



(a) Specialized fault model following its optimal fine-tuning strategy.



(b) Foundation model following its optimal fine-tuning strategy.

Figure 4.2: Train loss across all training phases in the best performing fine-tuning strategies. CD is shorthand for *Class distribution*, EL is *Event label*, I0 is *Interruption in sequence*, I3 is *Interruption 3-day*, and so on.

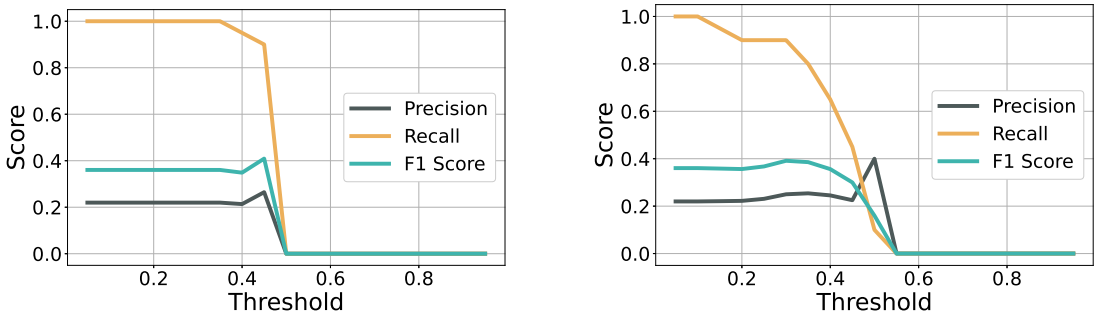
While pretraining the foundation model is significantly more volatile than the specialized model, the reverse is true when it comes to fine-tuning. This likely stems from the fact that the foundation model pretraining dataset is very diverse. Since Figure 4.2 only shows the pretraining loss term, no conclusions of individual task learning can be drawn. Following these fine-tuning strategies, final performance on *Interruption 7-day* on the held-out set and comparison to baseline task performance is shown in Table 4.4.

Table 4.4: Performance on *Interruption 7-day* following the optimal fine-tuning strategy for both approaches. Values in parentheses present the change from their respective baseline performance on *Interruption 7-day*. The threshold ξ was tuned on the train and tuning set. Final evaluation was done on the held-out set.

Model	ξ	Precision	Recall	AP
Specialized fine-tuned	0.40	0.2714 (+23 %)	0.9500 (−5.0 %)	0.3397 (+12 %)
Foundation fine-tuned	0.50	0.4000 (+96 %)	0.1000 (−78 %)	0.2788 (+43 %)

Once again the threshold ξ was tuned to maximize precision on the train and tuning set. While recall has taken a hit, both precision and average precision are improved in the fine-tuned models. As stated, we value precision the highest, meaning the fine-tuning strategies have improved the results for both model approaches. The foundation model had the largest improvements in precision and average precision over its baseline, but also the biggest drop in recall. The fine-tuned foundation model only catches 10 % of all interruptions, and of its alarms 40 % were true positives. Contrast this with the specialized model, which catches 95 % of interruptions, but out of its alarms 27 % turned out to be actual interruptions. When put into words, it becomes evident that the specialized model is the better choice in most cases.

The tuning has a major impact on model performance, and while a high average precision usually indicates a better model, a specific threshold value still has to be chosen. Figure 4.3 shows the resulting precision, recall and the combined metric F1-score for different threshold values.



(a) Specialized model fine-tuned **(b)** Foundation model fine-tuned

Figure 4.3: Precision, recall and F1-score as a function of the threshold ξ. Values were evaluated on *Interruption 7-day* on the held-out set. Note that these plots were visualized after final evaluation, and therefore not used to tune thresholds, ensuring no data leakage occurred.

In the specialized model (Figure 4.3a), precision and recall quickly drops to zero above a threshold of 0.5. This indicated that most of the predicted probabilities are around 0.5, and that the model is very uncertain. This is often undesirable, and various attempts to address this was taken during this thesis work. The foundation model however (Figure 4.3b) has probabilities more spread out.

4.4 Catastrophic forgetting

The main drawback of training on proxy tasks sequentially is the risk of forgetting early tasks and overwriting the knowledge, a phenomenon called *catastrophic forgetting* [8]. In order to quantify the extent of forgetting, an experiment was set up as to evaluate the regression task *Class distribution*. Normal pretraining was run, followed by fine-tuning on *Class distribution*, *Event label*, *Interruption in sequence*, *Interruption 3-day*, *Interruption 5-day* and *Interruption 7-day*. Between each fine-tuning task, the performance on *Class distribution* was evaluated on the held-out set. Figure 4.4 shows this performance on *Class distribution*, when continuing fine-tuning on more and more tasks.

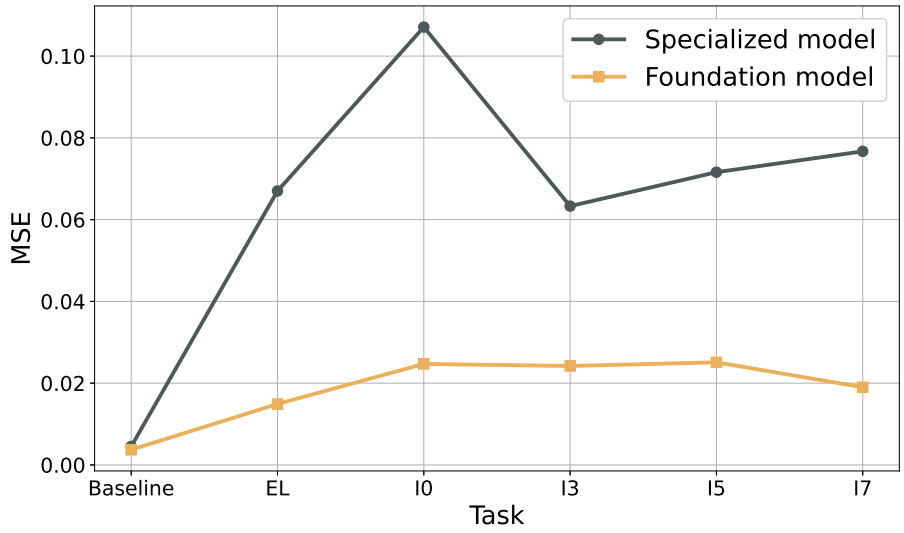


Figure 4.4: Evaluation of catastrophic forgetting for specialized model. Performance of *Class distribution* on held-out set, after continued fine-tuning on more tasks.

The graph in Figure 4.4 clearly shows that forgetting is present. There are however substantial differences between the specialized model and the foundation model. The specialized model seems to be more prone to forgetting since the MSE increases to a maximum of 0.1071 from the baseline of 0.0045. Since the target is a distribution in the interval $[0, 1]$ an absolute MSE increase of 0.1026 is a substantial performance drop. The foundation model also seems to exhibit some forgetting, though not to the same degree as the specialized model. In this case the maximum absolute difference in MSE compared to the baseline is 0.0214 for the foundation model, significantly less compared to 0.1026 for the specialized model.

The difference in performance most likely lies in the different amount of training data that each model has been pretrained on. Since the specialized only has been pretrained on a smaller homogeneous dataset, it may not be able to generalize across different tasks. The foundation model on the other hand has a pretraining dataset that is 500 times larger and more diverse. This is also the only real difference between the two different models, which further reinforces the notion that the difference in performance is due to the difference in training data.

To get a full picture of the influence of training on the classification proxy tasks, the metrics introduced in Section 2.3.3 were evaluated. Refer to this section for the mathematical definitions. The metrics are *forgetting measure*, *backward transfer* and *forward transfer*. These metrics measures differences in performance, and can be calculated using a variety of performance metrics. In this case average precision was chosen as it is independent of the threshold tuning. Since the *Class distribution* task performance cannot be measured in average precision, it was excluded in this part of the evaluation of catastrophic forgetting. Other than that, each model followed its optimal fine-tuning strategy, which was presented in Table 4.3. The forgetting measure is presented in Table 4.5. Here, the average forgetting was evaluated for the models optimal fine-tuning strategies from different starting points.

Table 4.5: Forgetting measure (FM) for both models, evaluated following each model's optimal fine-tuning strategy, starting from the different tasks, on the held-out set. Performance was measured using average precision.

(a) Specialized fault model

Optimal strategy from task	**FM**
Interruption in seq.	0.1332 (43 %)
Event label	0.1077 (61 %)
Interruption 3-day	0.0834 (38 %)
Interruption 5-day	0.0205 (8.7 %)
Interruption 7-day	N/A
Average	**0.0862**

(b) Foundation model

Optimal strategy from task	**FM**
Event label	0.0025 (7.3 %)
Interruption in seq.	0.1383 (39 %)
Interruption 3-day	0.0000 (0.0 %)
Interruption 5-day	0.0008 (0.4 %)
Interruption 7-day	N/A
Average	**0.0354**

The forgetting measure quantifies the performance loss on the evaluated task as a result of learning subsequent tasks. To gain intuition, this metric reflects how much the model forgets about each task after learning new ones. A lower forgetting measure indicates that the model is more *stable*, and retains more of the knowledge of that task when learning future tasks. Overall, the foundation model shows better retention of its knowledge by achieving a lower average forgetting measure (0.0354), compared to the specialized model (0.0862). This is most evident when evaluating starting from tasks *Interruption 3-day* and *Interruption 5-day*, where the foundation model shows little to no forgetting at all.

Backward transfer measures the average performance loss on previously learned tasks, as a result of learning the current task. Think of it as indicating whether learning new tasks helped or hurt what the model already knew. In contrast, backward transfer measures how much *average* performance was lost on *all* prior tasks, when learning the current task. A positive backward transfer indicates that learning that task improved performance on previous tasks, whereas a negative value indicates performance loss. The transfer metrics were evaluated for both models, and the results are presented in Table 4.6.

Table 4.6: Backward transfer (BWT) and forward transfer (FWT) for both models, evaluated on the held-out set. Performance was measured using average precision.

(a) Specialized fault model

Task	**BWT**	**FWT**
Interruption in seq.	N/A	N/A
Event label	−0.1370	0.0682
Interruption 3-day	−0.1030	0.0911
Interruption 5-day	−0.0100	0.0689
Interruption 7-day	−0.0918	0.0466
Average	**−0.0855**	**0.0687**

(b) Foundation model

Task	**BWT**	**FWT**
Event label	N/A	N/A
Interruption in seq.	−0.0076	0.3789
Interruption 3-day	−0.1800	0.1805
Interruption 5-day	−0.1312	0.1189
Interruption 7-day	−0.0964	0.0961
Average	**−0.1038**	**0.1936**

The foundation model shows slightly more negative backward transfer (−0.1038) than the specialized model (−0.0855). This indicates that the foundation model has a greater tendency to interfere with previous tasks.

Forward transfer measures how much learning the previous tasks improved the performance on the current task, in comparison to baseline performance, essentially measuring the effectiveness of the fine-tuning strategy. A positive value indicates positive knowledge transfer to future tasks, and can therefore be seen as an indication of the model's *plasticity*. The foundation model achieves significantly higher average forward transfer (0.1936), compared to the specialized model (0.0687). These results indicate that the previous tasks had a positive influence on performance of the final task for both models, but even more so for the foundation model. This is also evident in Table 4.4 showing the performance difference before and after following a fine-tuning strategy. There, the foundation model improved average precision on the final task by 43 %, compared to the specialized model's 12 % gain.

Based on these results, it is evident that the foundation model exhibits both better stability and plasticity by achieving lower forgetting and higher forward transfer than the specialized model.

5
Conclusion

This chapter summarizes the findings of this thesis, based on the results, and addresses the research questions stated in the introduction. Finally, it will discuss potential directions of future work. The central aim of this thesis was to investigate a foundation model approach to event sequence data, specifically fault event data in electric power grids. To do this, two models were trained: one specialized fault model trained solely on fault data, and one foundation model pretrained on diverse event sequence datasets.

Experimental results show that the specialized model outperforms the foundation model on the fault prediction task after fine-tuning, by achieving a test precision of 0.27 at a test recall of 0.95. In comparison, the foundation model achieved a test precision of 0.4, at a recall of 0.1. High precision is essential to minimize the number of false alarms, but at such a low recall, it is of little use to actual fault prediction as the vast majority of faults are missed. The optimal model achieves a high precision, so that it is certain of its alarms, at a reasonable level of recall. Both models also struggle with overfitting, as the average precision on the train set is about twice that of the held-out test set, despite the use of regularization techniques such as weight decay and dropout. The issue with overfitting is not unexpected, due to the complexity of the prediction task combined with the limited size of the fault dataset.

A key finding in this work is that proxy task fine-tuning significantly improved fault prediction performance, even for the specialized model. Training on six proxy tasks sequentially improved test precision for the specialized model by 23 %, at a cost of only 5 % lower recall, over baseline values. The foundation model also saw benefits from this training strategy, increasing its test precision by 96 %, however at 78 % lower recall. This suggests that proxy task fine-tuning can be an advantageous training strategy, improving model generalization, even when not working with foundation models.

This study used a selected subset containing 27 scalar values as features, out of the available 370. Feature selection can have a significant impact on the performance of the model, but since the other subsets were not investigated here, it is impossible to say how this would have affected the results. Some other data pre-processing choices regard whether or not to include events labeled as "normal", and choices of context window length and overlap. As discussed previously, there is a trade-off between more training examples, and increased redundancy, when using overlapping event sequences.

Referring to the results section and Table 4.4 we can show that the specialized model outperforms the foundation model on the fault prediction task, both in baseline and after fine-tuning in a continual learning manner. The foundation model does however perform better at identifying interruption events in sequences and estimating class distribution in sequences. The fine-tuned foundation model did however achieve the highest test precision, but at the cost very poor recall, which is why the specialized model was deemed to be the overall better model. We can further conclude that following fine-tuning on different proxy tasks is very beneficial for both models. Resulting in an increase of average precision by 12 % for the specialized model, and by 43 % for the foundation model approach. By evaluating all orderings of proxy tasks, we found that the optimal strategy, hyperparameters, and number of epochs differ between the investigated models.

Finally, catastrophic forgetting and the models' stability and plasticity were evaluated when following their respective optimal fine-tuning strategy. The regression task *Class distribution* was evaluated separately by running inference between each subsequent proxy task in the training protocol. Results show that *Class distribution* performance quickly drops after the following two tasks, but after that remains, or event starts to improve again when training on the fault prediction tasks. Interestingly, the foundation model was not as affected by catastrophic forgetting on this task as the specialized model was. The classification tasks were evaluated on forgetting, backward transfer and forward transfer. The most notable findings were that the foundation model achieves lower average forgetting, and higher average forward transfer. This suggests that the foundation model performs better on both stability and plasticity, compared to the specialized model.

The limiting factor for the maximum possible model size was the GPU memory. Since this thesis investigates foundation models - inherently large neural networks - it is likely that much of their benefit is unseen in this work. Larger model sizes, trained on bigger event sequence datasets are therefore the main research directions needed to improve event sequence foundation models. This thesis investigated the foundation model approach on a smaller scale with a model containing 1.6 million trainable parameters. Many state-of-the-art foundation models are however substantially larger, with models containing billions of parameters.

In conclusion, this thesis demonstrates the potential of a foundation model approach to event sequence modeling on a smaller scale. While the specialized model proved to be more effective at the fault prediction task, the foundation model showcased greater generalization capabilities, stability and plasticity in continual learning settings. Concepts from foundation model training, namely proxy task fine-tuning, was effective in improving fault prediction performance of both models. Future work should aim to utilize larger model sizes and datasets to fully realize the benefits of foundation models of this modality.

Bibliography

[1] "Kortsiktsprogons vinter 2024", Energimyndigheten, Tech. Rep. [Online]. Available: https://energimyndigheten.a-w2m.se/System/TemplateView.aspx?p=Arkitektkopia&id=fdc950ae8cc14627afe8b7392ed3f30f&q=vinter%202024&lstqty=1 (visited on 05/14/2025).

[2] E. Balouji, K. Bäckström, V. Olsson, P. Hovila, H. Niveri, A. Kulmala, and A. Salo, *Distribution Network Fault Prediction Utilising Protection Relay Disturbance Recordings And Machine Learning*, arXiv:2306.12724 [eess], Jun. 2023. [Online]. Available: http://arxiv.org/abs/2306.12724 (visited on 05/07/2025).

[3] H. Culley, "Fault Intelligence: Distribution Grid Fault Detection and Classification", en, Sep. 2017.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention Is All You Need*, arXiv:1706.03762 [cs], Aug. 2023. [Online]. Available: http://arxiv.org/abs/1706.03762 (visited on 02/06/2025).

[5] R. Bommasani *et al.*, *On the Opportunities and Risks of Foundation Models*, arXiv:2108.07258 [cs], Jul. 2022. [Online]. Available: http://arxiv.org/abs/2108.07258 (visited on 03/11/2025).

[6] Y. Liang, H. Wen, Y. Nie, Y. Jiang, M. Jin, D. Song, S. Pan, and Q. Wen, "Foundation Models for Time Series Analysis: A Tutorial and Survey", en, in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Barcelona Spain: ACM, Aug. 2024, pp. 6555–6565, ISBN: 979-8-4007-0490-1. [Online]. Available: https://dl.acm.org/doi/10.1145/3637528.3671451 (visited on 01/22/2025).

[7] M. B. A. McDermott, B. Nestor, P. Argaw, Y. Jin, and I. Kohane, "Event Stream GPT: A Data Pre-processing and Modeling Library for Generative, Pre-trained Transformers over Continuous-time Sequences of Complex Events", en, Jun. 2023. [Online]. Available: https://arxiv.org/abs/2306.11547 (visited on 02/04/2025).

[8] J. Kirkpatrick, "Overcoming catastrophic forgetting in neural networks", en, Feb. 2017. [Online]. Available: https://www.pnas.org/doi/epdf/10.1073/pnas.1611835114 (visited on 04/10/2025).

[9] S. Gao, T. Koker, O. Queen, T. Hartvigsen, T. Tsiligkaridis, and M. Zitnik, *UniTS: Building a Unified Time Series Model*, en, arXiv:2403.00131 [cs], Feb. 2024. [Online]. Available: http://arxiv.org/abs/2403.00131 (visited on 01/20/2025).

[10] V. Ekambaram, A. Jati, P. Dayama, S. Mukherjee, N. H. Nguyen, W. M. Gifford, C. Reddy, and J. Kalagnanam, *Tiny Time Mixers (TTMs): Fast Pre-trained Models for Enhanced Zero/Few-Shot Forecasting of Multivariate Time Series*, arXiv:2401.03955 [cs], Nov. 2024. [Online]. Available: http://arxiv.org/abs/2401.03955 (visited on 01/21/2025).

[11] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, *Language Models are Few-Shot Learners*, arXiv:2005.14165 [cs], Jul. 2020. [Online]. Available: http://arxiv.org/abs/2005.14165 (visited on 05/12/2025).

[12] *eCommerce Events History in Electronics Store*, en. [Online]. Available: https://www.kaggle.com/datasets/mkechinov/ecommerce-events-history-in-electronics-store (visited on 03/24/2025).

[13] *eCommerce Events History in Cosmetics Shop*, en. [Online]. Available: https://www.kaggle.com/datasets/mkechinov/ecommerce-events-history-in-cosmetics-shop (visited on 03/24/2025).

[14] *Predictive Maintenance Dataset*, en. [Online]. Available: https://www.kaggle.com/datasets/hiimanshuagarwal/predictive-maintenance-dataset (visited on 03/24/2025).

[15] E. Brockmeyer, H. Halstrøm, A. Erlang, and A. Jensen, *The Life and Works of A.K. Erlang* (Academy of Technical sciences). Copenhagen Telephone Comp., 1948. [Online]. Available: https://books.google.se/books?id=JQ7vAAAAMAAJ.

[16] O. Shchur, A. C. Türkmen, T. Januschowski, and S. Günnemann, "Neural Temporal Point Processes: A Review", en, ISSN: 1045-0823, vol. 5, Aug. 2021, pp. 4585–4593. [Online]. Available: https://www.ijcai.org/proceedings/2021/623 (visited on 02/07/2024).

[17] H. Lin, L. Wu, G. Zhao, P. Liu, and S. Z. Li, *Exploring Generative Neural Temporal Point Process*, en, arXiv:2208.01874 [cs], Aug. 2022. [Online]. Available: http://arxiv.org/abs/2208.01874 (visited on 03/26/2025).

[18] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent Marked Temporal Point Processes: Embedding Event History to Vector", en, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California USA: ACM, Aug. 2016, pp. 1555–1564, ISBN: 978-1-4503-4232-2. [Online]. Available: https://dl.acm.org/doi/10.1145/2939672.2939875 (visited on 03/11/2025).

[19] S. Xiao, M. Farajtabar, X. Ye, J. Yan, L. Song, and H. Zha, "Wasserstein Learning of Deep Generative Point Process Models", in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/hash/f45a1078feb35de77d26b3f7a52ef502-Abstract.html (visited on 03/25/2025).

[20] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 [cs], Apr. 2015. [Online]. Available: http://arxiv.org/abs/1412.6980 (visited on 03/24/2025).

[21] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, arXiv:1711.05101 [cs], Jan. 2019. [Online]. Available: http://arxiv.org/abs/1711.05101 (visited on 05/09/2025).

[22] Q. Zhang, A. Lipani, O. Kirnap, and E. Yilmaz, "Self-Attentive Hawkes Process", en, in *Proceedings of the 37th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Nov. 2020, pp. 11 183–11 193. [Online]. Available: https://proceedings.mlr.press/v119/zhang20q.html (visited on 03/26/2025).

[23] *Dvgodoy/dl-visuals: Over 200 figures and diagrams of the most popular deep learning architectures and layers FREE TO USE in your blog posts, slides, presentations, or papers.* [Online]. Available: https://github.com/dvgodoy/dl-visuals?tab=CC-BY-4.0-1-ov-file (visited on 05/12/2025).

[24] C. C. Aggarwal, "Attention Mechanisms and Transformers", en, in *Machine Learning for Text*, C. C. Aggarwal, Ed., Cham: Springer International Publishing, 2022, pp. 369–391, ISBN: 978-3-030-96623-2. [Online]. Available: https://doi.org/10.1007/978-3-030-96623-2_11 (visited on 04/09/2025).

[25] H. Irani and V. Metsis, *Positional Encoding in Transformer-Based Time Series Models: A Survey*, arXiv:2502.12370 [cs], Feb. 2025. [Online]. Available: http://arxiv.org/abs/2502.12370 (visited on 03/13/2025).

[26] OpenAI *et al.*, *GPT-4 Technical Report*, arXiv:2303.08774 [cs], Mar. 2024. [Online]. Available: http://arxiv.org/abs/2303.08774 (visited on 03/11/2025).

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv:1810.04805 [cs], May 2019. [Online]. Available: http://arxiv.org/abs/1810.04805 (visited on 03/11/2025).

[28] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, H. Peng, J. Li, J. Wu, Z. Liu, P. Xie, C. Xiong, J. Pei, P. S. Yu, and L. Sun, "A comprehensive survey on pretrained foundation models: A history from BERT to ChatGPT", en, *International Journal of Machine Learning and Cybernetics*, Nov. 2024, ISSN: 1868-808X. [Online]. Available: https://doi.org/10.1007/s13042-024-02443-6 (visited on 04/25/2025).

[29] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, *Scaling Laws for Neural Language Models*, arXiv:2001.08361 [cs], Jan. 2020. [Online]. Available: http://arxiv.org/abs/2001.08361 (visited on 03/13/2025).

[30] S. Thrun and L. Pratt, Eds., *Learning to Learn*, en. Boston, MA: Springer US, 1998, ISBN: 978-1-4613-7527-2 978-1-4615-5529-2. [Online]. Available: https://link.springer.com/10.1007/978-1-4615-5529-2 (visited on 03/13/2025).

[31] Z. Xu, Z. Shi, J. Wei, F. Mu, Y. Li, and Y. Liang, *Towards Few-Shot Adaptation of Foundation Models via Multitask Finetuning*, arXiv:2402.15017 [cs], Feb. 2024. [Online]. Available: http://arxiv.org/abs/2402.15017 (visited on 04/28/2025).

[32] D. M. Anisuzzaman, J. G. Malins, P. A. Friedman, and Z. I. Attia, "Fine-Tuning Large Language Models for Specialized Use Cases", *Mayo Clinic Proceedings: Digital Health*, vol. 3, no. 1, p. 100 184, Mar. 2025, ISSN: 2949-7612. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2949761224001147 (visited on 04/28/2025).

[33] L. Wang, X. Zhang, H. Su, and J. Zhu, *A Comprehensive Survey of Continual Learning: Theory, Method and Application*, arXiv:2302.00487 [cs], Feb. 2024. [Online]. Available: http://arxiv.org/abs/2302.00487 (visited on 05/13/2025).

[34] H. M. Fayek, L. Cavedon, and H. R. Wu, "Progressive learning: A deep learning framework for continual learning", *Neural Networks*, vol. 128, pp. 345–357, Aug. 2020, ISSN: 0893-6080. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608020301817 (visited on 04/10/2025).

[35] W. C. Abraham and A. Robins, "Memory retention–the synaptic stability versus plasticity dilemma", eng, *Trends in Neurosciences*, vol. 28, no. 2, pp. 73–78, Feb. 2005, ISSN: 0166-2236.

[36] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring Catastrophic Forgetting in Neural Networks", en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018, Number: 1, ISSN: 2374-3468. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11651 (visited on 05/13/2025).

[37] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", en, *Bulletin of Mathematical Biology*, vol. 5, pp. 115–133, 1943.

[38] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.", *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554.

[39] D. Hendrycks and K. Gimpel, *Gaussian Error Linear Units (GELUs)*, arXiv:1606.08415 [cs], Jun. 2023. [Online]. Available: http://arxiv.org/abs/1606.08415 (visited on 03/11/2025).

[40] M. McDermott, *Mmcdermott/EventStreamGPT*, original-date: 2023-02-27T17:54:09Z, Feb. 2025. [Online]. Available: https://github.com/mmcdermott/EventStreamGPT (visited on 03/19/2025).

[41] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha, *AMMUS : A Survey of Transformer-based Pretrained Models in Natural Language Processing*, arXiv:2108.05542 [cs], Aug. 2021. [Online]. Available: http://arxiv.org/abs/2108.05542 (visited on 04/16/2025).

[42] T. Saito and M. Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets", en, *PLOS ONE*, vol. 10, no. 3, G. Brock, Ed., e0118432, Mar. 2015, ISSN: 1932-6203. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0118432 (visited on 04/15/2025).

[43] Z. C. Lipton, C. Elkan, and B. Narayanaswamy, *Thresholding Classifiers to Maximize F1 Score*, arXiv:1402.1892 [stat], May 2014. [Online]. Available: http://arxiv.org/abs/1402.1892 (visited on 03/27/2025).

[44] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves", en, in *Proceedings of the 23rd international conference on Machine learning - ICML '06*, Pittsburgh, Pennsylvania: ACM Press, 2006, pp. 233–240, ISBN: 978-1-59593-383-6. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1143844.1143874 (visited on 04/11/2025).

A
Appendix

A.1 Derivation of event sequence likelihood

In this section, we derive the likelihood function of an event sequence, following the formulation introduced in Section 2.1. Given a marked event sequence of N events, $\mathcal{E} = \{(t_i, e_i)\}_{i=1}^{N}$, our goal is to express the likelihood of observing this sequence under a temporal point process model.

Start by considering the likelihood of observing a single event (t_i, e_i), which is composed of a temporal component and a mark component. By assuming conditional independence, we can write

$$p(t_i, e_i \mid \mathcal{H}_{t_i}) = P(e_i = k \mid \mathcal{H}_{t_i}) \cdot f(t_i \mid \mathcal{H}_{t_i}), \tag{A.1}$$

where f is the conditional probability density function of the arrival time, and $P(e_i = k \mid \mathcal{H}_{t_i})$ is the probability of the mark.

The intensity function describes the instantaneous rate at which events occur [16]. Using this, we can derive an expression for the probability density function by following the approach given by Lin *et al.* [17]. First we write

$$\lambda(t \mid \mathcal{H}_t)dt = \mathbb{E}\left[N(t + dt) - N(t) \mid \mathcal{H}_t\right] \tag{A.2}$$
$$= \mathbb{P}(t_i \in [t, t + dt) \mid \mathcal{H}_t), \tag{A.3}$$

where $N(t)$ represents a counting process. Here, $\mathbb{P}$ is the notation used for a probability measure. With this, we can derive the relationship between the intensity function and the conditional probability density as

$$\begin{aligned}
\lambda(t \mid \mathcal{H}_t)dt &= \mathbb{P}(t_i \in [t, t + dt) \mid \mathcal{H}(t)) \\
&= \mathbb{P}(t_i \in [t, t + dt) \mid t_i \notin [t_{i-1}, t), \mathcal{H}(t_{i-1})) \\
&= \frac{\mathbb{P}(t_i \in [t, t + dt), t_i \notin [t_{i-1}, t) \mid \mathcal{H}(t_{i-1}))}{\mathbb{P}(t_i \notin [t_{i-1}, t) \mid \mathcal{H}(t_{i-1}))} \\
&= \frac{\mathbb{P}(t_i \in [t, t + dt) \mid \mathcal{H}(t_{i-1}))}{\mathbb{P}(t_i \notin [t_{i-1}, t) \mid \mathcal{H}(t_{i-1}))} \\
&= \frac{f(t \mid \mathcal{H}(t_{i-1}))}{1 - F(t \mid \mathcal{H}(t_{i-1}))},
\end{aligned}$$

where $F(t \mid \mathcal{H}_t) = \int_{t_{i-1}}^{t} f(u \mid \mathcal{H}_t)du$ is the cumulative distribution function. Solving for f, we obtain

$$f(t \mid \mathcal{H}_t) = \lambda(t \mid \mathcal{H}_t) \exp\left(-\int_{t_{i-1}}^{t} \lambda(u \mid \mathcal{H}_t)\, du\right) \quad [17]. \tag{A.4}$$

To derive the full likelihood of the observed event sequence $\mathcal{E}$, we assume that events are conditionally independent given the history. The likelihood then becomes a product over the individual event likelihoods

$$p(\mathcal{E}) = \prod_{i=1}^{N} p(t_i, e_i \mid \mathcal{H}_{t_i}) = \prod_{i=1}^{N} P(e_i = k \mid \mathcal{H}_{t_i}) \cdot f(t_i \mid \mathcal{H}_{t_i}). \tag{A.5}$$

We now adopt a formulation inspired by Shchur *et al.* [16], where the likelihood is written as a sum over all possible marks, using an indicator function to select the observed mark. Since each mark has its own intensity function $\lambda_k(t)$, the total intensity across all marks is used in the survival term. This gives the likelihood of an event sequence as

$$p(\mathcal{E}) = \prod_{i=1}^{N} \sum_{k=1}^{K} \mathbb{1}(e_i = k)\lambda_k(t_i \mid \mathcal{H}_{t_i}) \cdot \exp\left(-\sum_{k=1}^{K} \int_{0}^{T} \lambda_k(u \mid \mathcal{H}_u)\, du\right), \tag{A.6}$$

where T is the end time of the observation window. The integral accounts for the probability of no events occurring outside the observed sequence, ensuring proper normalization.

A.2 Evaluation metrics

This section outlines the classification metrics used in this work. We begin by presenting the general confusion matrix for a binary classification task, in Table A.1 below.

Table A.1: Confusion matrix for a binary classification task. tp is the true positive count, fp is the false positive count, fn is the false negative count, and tn is the true negative count.

	Actual Positive	Actual Negative
Predicted Positive	tp	fp
Predicted Negative	fn	tn

Using the counts in Table A.1, the *F1-score* is calculated as:

$$F1 = \frac{2tp}{2tp + fp + fn}. \tag{A.7}$$

The F1-score is the harmonic mean of precision and recall, offering a balanced metric compared to alternatives [43]. If false positive and false negative predictions are equally important, the F1-score serves as an effective summary metric.

For a metric suitable for cases with imbalanced classed, we use *Average Precision (AP)* as a classification metric, which is derived from the Precision-Recall curve [44]. Average precision is the harmonic mean of precision across different recall levels, offering a more robust evaluation in imbalanced settings. Using the confusion matrix counts from Table A.1, precision and recall are defined as:

$$\text{precision} = \frac{tp}{tp + fp} \tag{A.8}$$

$$\text{recall} = \frac{tp}{tp + fn} \tag{A.9}$$

A higher average precision indicates a better classifier, with a perfect score of 1.0 and random guessing achieving an average precision in line with the fraction of positive labels. Precision is essentially showing how many of the predicted positives are actual positives. Recall instead shows what percentage of all positive labels we catch. Both of these metrics are important in the application of fault prediction.

A.3 Extensive list of hyperparameters

Table A.2: Extensive list of hyperparameters used for pretraining and fine-tuning the specialized fault model and the foundation model. SM is shorthand for specialized model, FM is foundation model, CD is class distribution, EL is event label, IX is the collection of all interruption classification tasks.

Parameter	Value	Parameter	Value
num_epochs_pretrain_SM	115	num_epochs_pretrain_FM	60
num_epochs_SM_CD	55	num_epochs_FM_CD	50
num_epochs_SM_EL	23	num_epochs_FM_EL	33
num_epochs_SM_I0	14	num_epochs_FM_I0	16
num_epochs_SM_I3	15	num_epochs_FM_I3	50
num_epochs_SM_I5	30	num_epochs_FM_I5	27
num_epochs_SM_I7	50	num_epochs_FM_I7	100
lr_init_pretrain_SM	8.00e-4	lr_init_pretrain_FM	5.11e-6
lr_init_SM_CD	9.42e-4	lr_init_FM_CD	7.20e-4
lr_init_SM_EL	6.10e-2	lr_init_FM_EL	4.64e-4
lr_init_SM_I0	7.60e-3	lr_init_FM_I0	6.60e-4
lr_init_SM_I3	1.25e-1	lr_init_FM_I3	2.06e-4
lr_init_SM_I5	7.86e-2	lr_init_FM_I5	5.81e-5
lr_init_SM_I7	7.93e-2	lr_init_FM_I7	5.20e-5
lr_end_pretrain_SM	3.20e-7	lr_end_pretrain_FM	3.85e-6
lr_end_SM_CD	1.99e-7	lr_end_FM_CD	6.88e-7
lr_end_SM_EL	7.27e-5	lr_end_FM_EL	2.23e-4
lr_end_SM_I0	1.34e-5	lr_end_FM_I0	1.22e-4
lr_end_SM_I3	1.26e-5	lr_end_FM_I3	3.21e-5
lr_end_SM_I5	1.10e-2	lr_end_FM_I5	1.36e-8
lr_end_SM_I7	2.00e-2	lr_end_FM_I7	7.01e-9
weight_decay_pretrain_SM	3.00e-4	weight_decay_pretrain_FM	0.3822
weight_decay_SM_CD	3.92e-3	weight_decay_FM_CD	7.79e-2
weight_decay_SM_EL	5.69e-4	weight_decay_FM_EL	6.15e-2
weight_decay_SM_I0	1.56e-1	weight_decay_FM_I0	2.17e-1
weight_decay_SM_I3	2.15e-1	weight_decay_FM_I3	8.11e-2
weight_decay_SM_I5	3.66e-1	weight_decay_FM_I5	7.77e-2
weight_decay_SM_I7	3.13e-1	weight_decay_FM_I7	8.58e-2
task_loss_weight_CD	1.00e4	task_loss_weight_EL	10
task_loss_weight_IX	2	positive_weight_EL	600
num_attention_heads	10	num_hiddden_layers	8
head_dim	18	hidden_size	180
intermediate_size	128	max_seq_len	256
cat_embedding_dim_SM	82	cat_embedding_dim_FM	90
cat_embedding_weight_SM	0.91	cat_embedding_weight_FM	0.01
seq_attention_types	global	padding_side	left

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY