StormMind
AI based Web-Service for Storm Damage Prediction

Bachelor Thesis

ZHAW School of Engineering
Institute of Computer Science

Submitted on:
06.06.2025

Authors:
Gämperli Nils
Ueltschi Damian

Supervisor:
Andreas Meier

Study Program:
Computer Science

Declaration of Authorship

We, Damian UELTSCHI, Nils GÄMPERLI, declare that this thesis titled, “StormMind - AI based Web-Service for Storm Damage Prediction” and the work presented in it are our own. We confirm that:

- All external help, aids and the adoption of texts, illustrations and programs from other sources are properly referenced in the work.
- This is also a binding declaration that the present work does not contain any plagiarism, i.e. no parts that have been taken over in part or in full from another’s text or work under pretence of one’s own authorship or without reference to the source.
- In the event of misconduct of any kind, Sections 39 and 40 (Dishonesty and Procedure in the Event of Dishonesty) of the ZHAW Examination Regulations and the provisions of the Disciplinary Measures of the University Regulations shall come into force.

I have no limitations.
— Thomas Shelby

To our parents…

Abstract

Extreme weather events cause considerable damage to infrastructure, the economy and the environment worldwide. Predicting such damage can help to optimize preventive measures and reduce costs. In this project, a neural network is being developed that analyzes historical weather data and damage reports to predict potential storm damage in Switzerland based on new weather forecasts. Using modern machine learning techniques, relevant patterns are recognized in order to train a predictive model. The model is tested for accuracy and optimized to enable reliable predictions.

Key words: Machine Learning, Neural Network, Storm Damages

Acknowledgements

We would like to thank Andreas Meier for his valuable support and guidance throughout this project. Additionally, we would like to thank Dr. Katharina Liechti, who provided us with valuable insights into the science of weather and storm damage. We also extend our gratitude to the Swiss Federal Institute for Forest, Snow and Landscape Research (WSL) for collecting and providing the data that made this project possible.

Preface

TODO

Key words: Machine Learning, Timeseries Forecasting, Software Engineering

Contents

List of Abbreviations

FNN	Feedforward Neural Network
LSTM	Long Short Term Memory Neural Network
NN	Neural Network
RNN	Recurrent Neural Network
VGP	Vanishing Gradient Problem
WANDB	Weights & Biases
WSL	Swiss Federal Institute for Forest Snow and Landscape Research WSL

1 Introduction

1.1 Motivation

In a rapidly changing world where ecological awareness is more crucial than ever, the aim is to leverage the latest technologies to contribute positively to the environment. While many people use generative AI for entertainment or convenience—such as creating humorous images or drafting emails—thus indirectly accelerating climate change through increased energy consumption, this thesis seeks to utilize artificial intelligence to serve a more meaningful purpose: to detect and warn about emerging threats and damages caused by environmental and climatic changes.

1.2 Work Outline

The objective of this project is to establish a baseline and explore initial steps toward predicting or forecasting environmental damage. As outlined throughout this thesis, numerous geological processes and a range of influencing factors would typically require in-depth expertise in geology—areas that were deliberately excluded to keep the scope at a foundational level. Instead, the focus was placed on weather data as a key influencing factor, while the primary data source for recorded damages consisted of reports published in regional and local newspapers. Various deep learning approaches, differing in complexity and methodology, were applied to assess how weather data may offer initial insights and help identify which predictive strategies warrant further exploration in future research.

2 Theoretical Background

This chapter presents the theoretical foundations required to understand the methods and models used in this work. The first part provides an overview of the physical and environmental aspects relevant to the prediction of storm-induced damage, including the meteorological mechanisms behind severe weather events and their typical impact patterns. This domain knowledge is essential for identifying meaningful input features and understanding the context of the prediction task.

The second part introduces the machine learning and deep learning concepts that underpin the modeling approach. It starts with basic **(?? FNN ??)** and progresses toward more advanced architectures tailored for sequential data, such as Recurrent Neural Network (**RNN**), Long Short Term Memory Neural Network (**LSTM**), and Transformer models. These architectures form the core of the predictive models developed in this thesis.

By combining insights from both atmospheric science and data-driven modeling, this chapter establishes the conceptual framework for the design and implementation of the storm damage forecasting pipeline.

2.1 Weather Research

The causes mentioned in this section are neither exhaustive nor account for all damages listed in [1]. In a conversation with K. Liechti[1], several contributing factors were discussed, of which only a subset were considered in the model. The influencing factors discussed were:

Topological gradients: The question addressed was whether known threshold values of slope (in percent or degrees) exist to differentiate between low, intermediate, and high landslide risk. Unfortunately, the answer was negative. Given the limited timeframe of the

[1] Dr. Käthi Liechti, Wissenschaftliche Mitarbeiterin Gebirgshydrologie und Massenbewegung Hydrologische Vorhersagen, Eidg. Forschungsanstalt WSL

thesis project, further research — e.g., literature reviews — could not provide any conclusive information.

Forest/Deforestation: According to Wikipedia ([2]), deforestation contributes to landslides by eliminating trees, thereby destroying the stabilizing effect of their roots. Liechti confirmed this statement but also pointed out that forests themselves possess considerable weight, which may also contribute to slope instability. Applying this insight would require additional data and investigation, and was therefore deprioritized due to time limitations.

Soil condition: This factor is discussed in detail within each relevant subsection.

Ground frost: Frozen soil exhibits increased cohesive strength, which helps prevent landslides.

Rainfall: Closely linked to soil conditions and therefore specifically addressed in each subsection.

Snowfall: Snowfall itself does not directly cause the damages under investigation but serves as a source for later snowmelt.

Snowmelt: Significantly contributes to the total volume of water input and has effects comparable to rainfall.

Small animals and soil organisms: This topic lies outside the scope of K. Liechti's expertise and was excluded from further consideration due to limited time.

2.1.1 Reasons for Flooding

Flooding is primarily mitigated by the soil's capacity to absorb water. The composition of the topsoil is the most relevant factor: a non-permeable surface—such as rock or compacted clay—prevents infiltration, resulting in all incoming water contributing to surface runoff. In contrast, permeable materials such as sand or loose soil can absorb substantial amounts of water, depending on the depth of the soil layer (as elaborated in Section 2.1.2). Even permeable ground can temporarily become impermeable during early spring when frozen. Conversely, drought has a similar effect: extended periods of high temperature and absent precipitation dry out the surface layer of the soil, thereby reducing or even eliminating its ability to absorb water. The behavior of completely dry soil is comparable to that of any other non-permeable material. All of these mechanisms are governed by the total water input, which originates either from rainfall, melting snow, or temperature conditions. [3]

Prolonged precipitation continuously supplies water to the soil. If the ground is permeable and allows both infiltration and subsurface drainage, the overall impact remains limited. However, when the absorbed water cannot drain away, saturation occurs. Once the soil reaches full capacity, it effectively becomes impermeable—regardless of its natural permeability—and behaves like rock or clay, leading to increased surface runoff. In addition, sudden and sustained temperature increases—often indicative of significant snowmelt—can further augment the water load on the soil surface alongside rainfall. [3]

2.1.2 Reasons for Landslide

Of the potential causes for landslides, three were selected for consideration, as they were highlighted by Liechti [3] as the most relevant and easily identifiable, without requiring in-depth geological analysis. The first—loose gravel or rock—is not directly influenced by current weather conditions and was therefore excluded. The second and relevant factor is the water absorption capacity of loose soil (e.g., dirt). As water is absorbed, the soil mass increases in weight—from dry ($0.83kg/dm^3$ [4]) to wet ($1.6 - 1.76kg/dm^3$ [5]). This gain in mass reduces the soil's ability to remain stable. Third, subsurface clay layers limit water infiltration on the one hand, but on the other hand, they promote landslides by forming smooth and slippery interfaces between different soil types and by preventing subsurface drainage. In contrast, a jagged and solid rock surface also impedes subsurface drainage but may reduce landslide risk by mechanically anchoring the overlying soil layer. [3]

Of the potential causes for landslides, three were selected for consideration, as they were highlighted by Liechti [3] as the most relevant and easily identifiable, without requiring in-depth geological analysis:

1. Loose gravel or rock, is not directly influenced by current weather conditions and was therefore excluded.
2. Water absorption capacity of loose soil (e.g., dirt). As water is absorbed, the soil mass increases in weight, from dry ($0.83kg/dm^3$ [4]) to wet ($1.6 - 1.76kg/dm^3$ [5]). This gain in mass reduces the soil's ability to remain stable.
3. Subsurface clay layers limit water infiltration on the one hand, but on the other hand, they promote landslides by forming smooth and slippery interfaces between different soil types and by preventing subsurface drainage. In contrast, a jagged and solid rock surface also impedes subsurface drainage but may reduce landslide risk by mechanically anchoring the overlying soil layer. [3]

2.2 Deep Learning

Deep Learning has gained increasing popularity in recent years, particularly through advancements in Neural Networks (**NNs**). These developments have significantly expanded the capabilities of automated data-driven modeling across various domains. In this chapter, we focus primarily on **NNs** architectures, as they form the core modeling approach used in this project.

2.2.1 Feedforward Neural Networks

(?? FNN ??)s are a class of machine learning models inspired by the structure and function of the human brain. In biological systems, neurons are interconnected through synapses, and their strengths change in response to external stimuli—a process that underlies learning. **FNNs** mimic this behavior by using computational units, also called neurons, connected by weighted links. These weights are adjusted during training to improve the model's predictions, analogous to synaptic strength adjustments in the brain.

Each artificial neuron receives inputs, scales them using learned weights, applies a non-linear activation function, and forwards the result to subsequent neurons. Through this architecture, an **FNN** models complex functions by propagating signals from input to output layers. Learning in **FNNs** occurs via exposure to training data consisting of input–output pairs. The network adjusts its weights to reduce the difference between its predictions and the target outputs, thereby minimizing the prediction error.

While the biological analogy is imperfect, it has historically guided the development of neural architectures. More formally, **FNNs** can also be viewed as compositions of simple mathematical units—such as logistic or linear regressors—structured into a computational graph. Their expressive power arises from stacking these units into deeper networks, enabling them to approximate highly non-linear relationships in data. This capacity to learn from examples and generalize to unseen inputs makes **FNNs** a powerful tool in modern machine learning.

Architecture

An **FNN** trained with backpropagation, which is discussed in Section 2.2.2, can be illustrated as a directed acyclic Graph with inter-connections. It contains a set of neurons distributed in different layers.

- Each neuron has a activation function.
- The first layer, shown on the left side in Figure 2.1, is called the input layer and has no predacessors in the inter-connection graph. Additionally, is their input value the same as their output value.
- The last layer, shown on the right side in Figure 2.1, is called the output layer and have no successors in the inter-connection graph. Their value represents the output of the Network
- All other neurons are grouped in the so called hidden layers. In Figure 2.1 this is represented by the layer in the middle. A neural network can have an arbitrary amount of hidden layers.
- The edges in the inter-connection graph, are so called weights, which represent an arbitrary number in $\mathbb{R}$. These weights are updated during the trianing process.

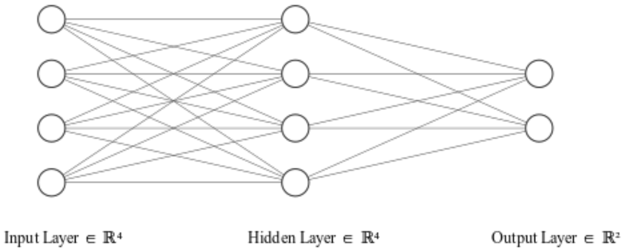

Figure 2.1 – Illustration of a Neural Network with 3 layers. Illustrated with [6]

Computation of the Output

The calculation of the output of the **FNN** is also called a forward pass. To do so, the value of each neuron needs to be calculated. This is done by summing all the inputs and then put this value into a given activation function. Mathematically, this process can be represented as: $y = f\left(\sum_i^n \xi_i\right)$ wher f represents the activation function of the neuron and ξ_i the input or also called the potential of a neuron. To compute a full forward pass, this is done for each neuron from the input layer towards the output layer. When a value is passed throughtouh a weight to a successor neuron, the value is multiplied by the value of the weight. This process can then be summarized to:

$$\begin{aligned} &y_1 = f\left(\sum_i^n w_{ij} x_i\right) \text{ [input to hidden layer]} \\ &y_{j+1} = f\left(\sum_i^n w_{ij} y_j\right) \forall j \in \{1...k-1\} \text{[hidden to hidden layer]} \\ &o = f\left(w_{ij+1} y_j\right) \text{ [hidden to output layer]} \end{aligned} \tag{2.1}$$

[7] where j denotes the layer, ascending from input layer to output layer, f activation function, w_{ij} weight at index i and layer j, x as input at index i. The state of the neuron in the output layer o can then by denoted as the output vector.

2.2.2 The Backpropagation Training Algorithm

The backpropagation training algorithm is used to train all deep learning models described in this section.

Objective: To identify a set of weights that guarantees that for every input vector, the output vector generated by the network is identical to (or sufficiently close to) the desired output vector.

Note: The actual or desired output values of the hidden neurons are not explicitly specified by the task.

For a fixed and finite training set: The objective function represents the total error between the desired and actual outputs of all the output neurons for all the training patterns.

Error Function

$$E = \frac{1}{2} \sum_{p}^{P} \sum_{i}^{N} \left(y_{ip} - d_{ip}\right)^2 \tag{2.2}$$

[8]

where P is the number of training patterns, N the number of output neurons, d_{ip} is the desired output for pattern p, y_{ip} the actual output of the neuron i and output neuron i.

Procedure

1. Compute the actual output for the presented pattern
2. Compare the actual output with the desired output
3. Adjustment of weights and thresholds against the gradient of the error function (Equation (2.2)) for each layer from the output layer towards the input layer

Figure 2.2 – Training Procedure of the backpropagation algorithm

Adjustment Rules

$$\begin{aligned} w_{ij}(t+1) &= w_{ij} + \Delta_E w_{ij}(t) \\ \Delta_E w_{ij} &= -\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial w_{ij}} \end{aligned} \tag{2.3}$$

[8]

where $\Delta_E w_{ij}$ denotes the change of the Error Function with respect to w_{ij}, E the Error Function, y_j the output of the output neuron j, ξ_j the potential of the neuron j and w_{ij} the weight with index i at layer j.

2.2.3 Recurrent Neural Networks

The feedforward **FNNs** discussed in Section 2.2.1 are inherently limited to fixed-size, unordered input representations. This makes them unsuitable for sequential data, where both the order and length of the input can vary. To address this limitation, we introduce a class of models specifically designed to process variable-length sequences: **RNNs**

Architecture

A **RNN** consits of the following components:

- Input signal: The external data which is fed into the network at a timestep n and represent the current information which the network is processing.
- State signal: Also known as the hidden state, represents the memory of the **RNN** for a given neuron. It contains information about the past inputs in the sequence and is updated at each time step based on the current input and the previous state. The hidden state is updated with the following formula: $h_t = f(h_{t-1}, x_t)$. After the update, the hidden state of neuron i serves as input into the neuron $i + 1$
- Weights: The weights of the **RNN** neurons are shared among all different states.
- Output: Each neuron has a output, which is denoted as y_1 - y_4 in Figure 2.4. This output can serve as the output for the current state or as input into the next neuron.

[9], [7]

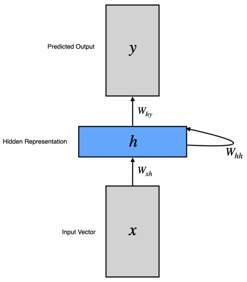

Figure 2.3 – **RNN**

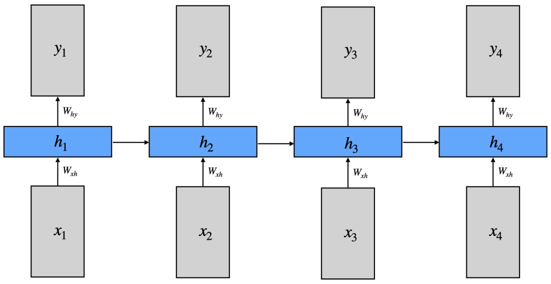

Figure 2.4 – 4 times unrolled **RNN**

Vanishing Gradient Problem

The Vanishing Gradient Problem(**VGP**) is a challenge encountered during the training of of **RNNs**, particullary dealing with deep **RNNs** and long input sequences. It arieses from the way how gradients are updated during the backpropagation algorithm (discussed in Section 2.2.2), which updates the network's paramertes / weights by propagating gradients backward through each time step. In backpropagation, gradients are computed via the chain rule, resulting in repeated multiplication of weight matrices and derivatives of activation functions across time steps. When these values are consistently smaller than one, the gradients exponentially decrease as they traverse earlier layers or time steps. Consequently, the gradients become vanishingly small, leading to negligible updates for earlier parameters and impairing the network's ability to learn long-range dependencies. The same principle aries, when the gradients become too large. In this case, the problem is called the exploding gradient problem.

[7]

2.2.4 Long Short Term Memory Neural Networks

Long Short Term Memory Neural Networks (**LSTMs**) are a special form of **RNNs** designed to address the problem of Vanishing Gradients while having a more fine-grained control over the previous input data and were introduced for the first time by Sepp Hochreiter in 1997 [10]. **LSTMs** are an enhanement of **RNNs**, because the recurrence conditaions of how the hidden state h_t is processed. To achieve this aim, we introduce a new hidden state of the same dimesion as h_t, which is called the cell state and is denoted as c_t. The key innovation of the **LSTM** lies in its ability to control the flow of information using a set of gating mechanisms. These gates regulate how information is added to, removed from, or exposed from the cell state. Each gate is implemented as a sigmoid-activated neural layer and serves a distinct role in the update process.

Architecture

The internal structure of an **LSTM** cell is shown in Figure 2.5. The figure illustrates how, at each time step t, the cell takes in the input vector x_t, the previous hidden state h_{t-1}, and the previous cell state c_{t-1}, and uses them to compute updated values for the current cell state c_t and hidden state h_t.

Figure 2.5 shows an illustration of an **LSTM** Cell.

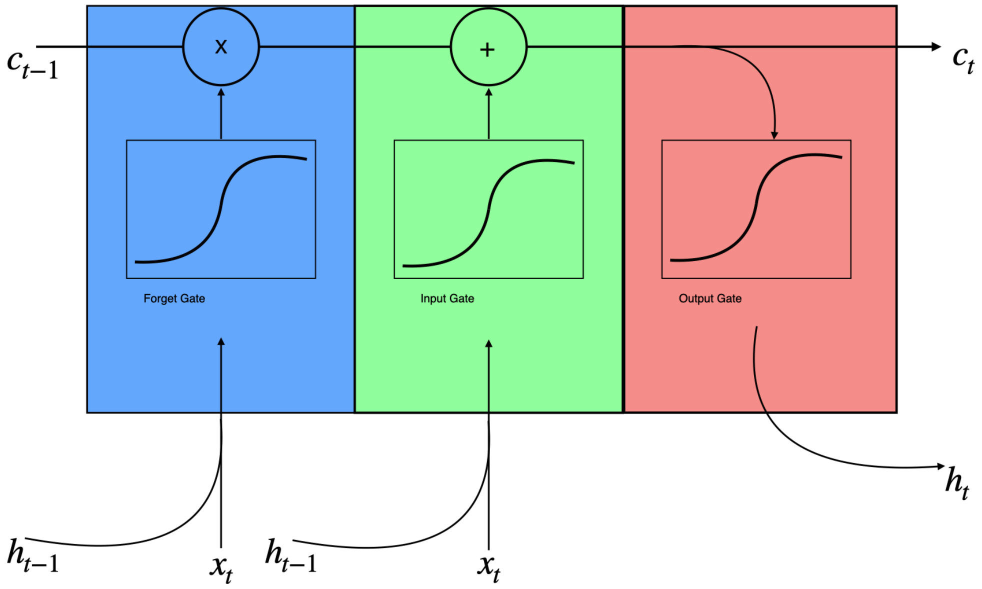

Figure 2.5 – Schematic illustration of an **LSTM** cell highlighting the internal gating structure. The colored blocks represent the three core gates—Forget (blue), Input (green), and Output (red)—and show how they interact with the cell and hidden states to regulate information flow.

At each time step t with a given input vector x_t, previous hidden state h_{t-1} and previous cell state c_{t-1}, the **LSTM** performs the following computations:

- Forget Gate (shown in the blue part of Figure 2.5): This gate decides which parts of the previous cell state should be forgotten. The value of the forget gate is calculated as:
  - $f_t = \sigma\big(w_f[h_{t-1}, x_t]\big) + b_f)$
- Input Gate (shown in the green part of Figure 2.5): Decides which new information will be added to the cell state and is calculated as:
  - $i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$
- Output Gate (shown as the red part in Figure 2.5): Determines which part of the cell state influences the hidden state and therefore the output. It is computed with:
  - $o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$
- Candidate Cell State: Computes possible candidates $\tilde{c}_t$ which can be added to the cell state, computed as:
  - $\tilde{c}_t = \tanh(w_c[h_{t-1}, x_t] + b_c)$
- Cell state update: Given the candidates, the cell state can be updated as:
  - $c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$
- Hidden state update: The final hidden state is computed by applying the output gate to the activated cell state. It is computed with:

- $h_t = o_t \cdot \tanh(c_t)$

Note: w_x represents a complete weight matrix for each gate, b_x denotes the bias for the corresponding gates, and σ denotes the sigmoid function.

[11], [12]

2.2.5 Transformer

With the advent of Large Language Models and influential works such as Attention Is All You Need [13], Transformer architectures have gained significant traction in the field of machine learning. Originally developed for natural language processing tasks, Transformers have since been successfully adapted to a variety of domains, such as time series forcasting as shown by Q. Wen et. al. in "Transformers in Time Series: A Survey" [14] due to their ability to model long-range dependencies.

In the following section, the core components and mechanisms of the Transformer architecture are outlined. Furthermore, special emphasis is placed on its applicability to time series forecasting—a setting in which capturing temporal patterns and complex dependencies is crucial.

Architecture

An important concept in the Transformer architecture is Attention. It allows the model to capture dependencies between elements in the input sequence. An attention function can be viewed as a mapping from a query and a set of key–value pairs to an output. The output is a weighted sum of the values, where the weights are determined by a compatibility function between the query and the keys. This mechanism is illustrated in Figure 2.6, where the input sequence is linearly projected into query, key, and value matrices to compute attention scores and generate contextualized representations. This procedure can be calculated with:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.4}$$

[13]

When the dot product QK^T yields large values, the resulting attention scores can produce extremely sharp probability distributions after applying the softmax function. This can lead to vanishing gradients during training, making optimization unstable. T To mitigate this effect, the attention scores are scaled by a factor of $\frac{1}{\sqrt{d_k}}$, where d_k is the dimensionality of the key vector and described as D in Figure 2.6.[13]

In tasks involving sequential data, such as language modeling or time series forecasting, the model should not have access to future positions when making a prediction. To enforce this constraint, the Transformer uses a technique called masked attention, in which the attention weights for all positions beyond the current one are set to zero. This ensures that, when computing the representation for position x_n, the model can only attend to $x_{<n}$ through x_n, but not to any $x_{>n}$.

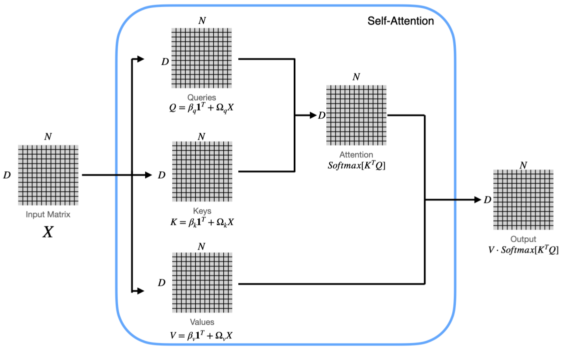
Self-Attention
N
D
Input Matrix
X
N
D
Queries
$Q = \beta_q \mathbf{1}^T + \Omega_q X$
N
D
Keys
$K = \beta_k \mathbf{1}^T + \Omega_k X$
N
D
Values
$V = \beta_v \mathbf{1}^T + \Omega_v X$
N
D
Attention
$Softmax[K^T Q]$
N
D
Output
$V \cdot Softmax[K^T Q]$

Figure 2.6 – Self-attention mechanism illustrated with matrices. All matrices have shape $D \cdot N$, where D is the sequence length and N is the feature dimension. The input matrix is projected into three separate matrices: Queries (Q), Keys (K), and Values (V). The attention weights are computed by multiplying Q with the transpose of K, followed by the Softmax function. The result is then used to weight the V matrix, producing the final output as $\mathrm{Softmax}(QK^T) \cdot V$. [15]

While basic self-attention allows a model to compute contextual relationships between sequence elements, it operates in a single projection space, potentially limiting the diversity of information captured. To address this, Transformers employ Multi-Head Attention, a mechanism that enables the model to attend to information from multiple representation subspaces simultaneously, which was firstly described in "Attention is All You Need" [13] Instead of computing attention just once, the input sequence is projected into multiple sets of Queries, Keys, and Values using learned linear transformations—typically with smaller dimensionality $q < p$, where p is the original embedding size. Each set of projections corresponds to a separate attention head, allowing the model to focus on different semantic or temporal aspects ("meta-concepts") of the sequence. These k parallel attention heads independently compute attention outputs, which are then concatenated into a single vector of dimension $k \cdot q$. Since this dimensionality may differ from the original embedding size, the concatenated output is passed through a final linear projection layer (denoted W^{text}) to produce the final attention output. This structure is illustrated conceptually in Figure 2.7, and it significantly enhances the expressiveness and robustness of the attention mechanism. This process can also be caluclated with:

$$\begin{aligned} &\mathrm{MultiHead}(Q, K, V) = \mathrm{Conact}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O \\ &\text{where}\ \ \mathrm{head}_i = \mathrm{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2.5}$$

[13]

Multi-head attention not only improves model performance but also enables parallel computation of attention heads, which leads to efficient training, especially on modern hardware. Despite its complexity, studies have shown that only a few attention heads may be essential, and redundant heads can be pruned without degrading model accuracy—improving interpretability and computational efficiency. [7]

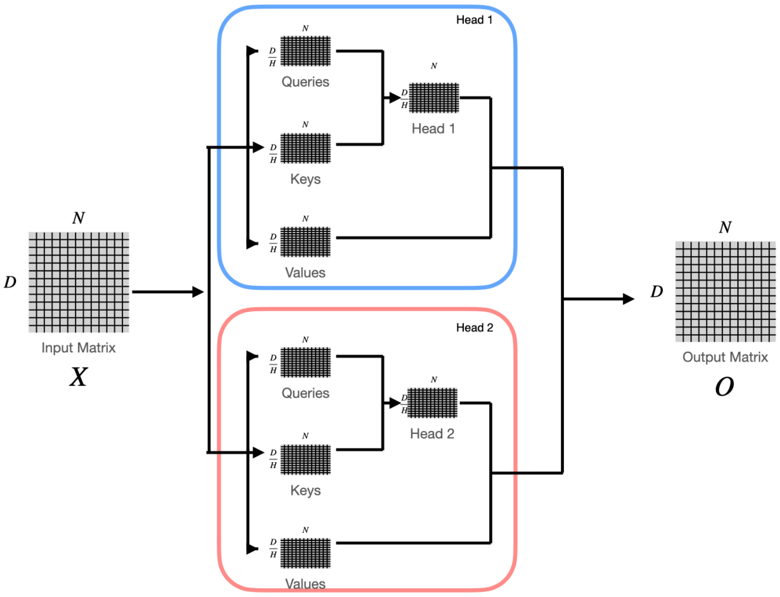

Figure 2.7 – Multi-head attention mechanism. The input matrix X of shape $D \times N$, where D is the feature dimension and N is the sequence length, is linearly projected into multiple sets of Queries, Keys, and Values. Each set defines an individual attention head (e.g., Head 1, Head 2), which independently computes scaled dot-product attention. The outputs from all H heads, each of size $\frac{D}{H} \times N$, are then concatenated and projected through a final linear layer to produce the output matrix O of shape $D \times N$. [15]

Embedding

In tasks such as machine translation, input sequences composed of discrete tokens (e.g., words) must first be mapped to continuous vector representations through an embedding layer, which captures semantic information about each token. In contrast, time series data is inherently numerical and already exists in a continuous vector space. Nevertheless, to align the input dimensionality with the model's internal representation size (denoted as d_model in the Transformer architecture), we apply a linear transformation to project the raw input features into the desired embedding space.

Although this operation is technically a linear projection, it is commonly referred to as an "embedding" in the literature, including in the context of non-textual data, such as in the Vision Transformer (ViT) by Dosovitskiy et al. [16]. Following this convention, we refer to this input transformation layer as an embedding in our architecture as well.

Positional Encoding

While the self-attention mechanism is effective at capturing relationships between elements in a sequence, it lacks an inherent notion of order. Specifically, self-attention is permutation-equivariant, which means it produces the same output, regardless of the input sequences order. However, order is important when the input is a time series or a sentence. To address this limitation, positional encodings are introduced to inject information about the position of each element in the input sequence. A common approach is to define a positional encoding matrix P_i and add it to the original input matrix X. Each column of P_i encodes a unique absolute position within the sequence, allowing the model to distinguish between inputs based not only on content but also on their order.

This encoding matrix P_i can either be fixed e.g. using sinus function as it was done in "Attention is All You Need" [13] or learned during training. By learning absolute position information in this way, the Transformer model gains the ability to capture the sequential structure of the data, which is essential for tasks like damage forecasting or language understanding.

Encoder Decoder

Transformers follow an encoder–decoder architecture, as illustrated in Figure 2.8. In this framework, the encoder processes the full input sequence and produces a contextual representation, called the Encoder Vector as shown in Figure 2.8 in grey. The decoder uses the Encoder Vector to generate the output sequence token by token. While both encoder and decoder are composed of multiple stacked layers and share a similar modular structure, including Multi Head Attention Layers, feedforward sub-layers, skip connections, and normalization steps. The decoder includes additional mechanisms to ensure autoregressive generation, which refers to the process of generating output tokens one at a time and each token is generated based on all previous generated tokens.

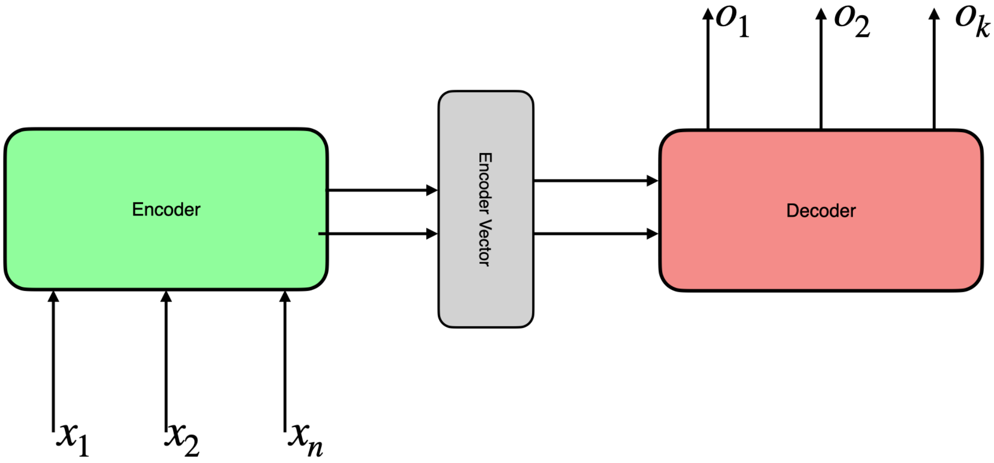

Figure 2.8 – Abstracted illustration of the encoder–decoder architecture. The encoder receives an input sequence $x_1, ..., x_n$ and transforms it into a sequence of contextualized representations, here called the Encoder Vector, which is symbolically represented by two arrows to emphasize its role in guiding the decoding process. The encoder vector are passed to the decoder, which generates an output sequence $y_1, ..., y_k$, where the output length k may differ from the input length n.[7]

Combining the encoder–decoder structure with the attention mechanism results in the full Transformer model. In this architecture, self-attention is used within both the encoder and decoder to enable each position in a sequence to access contextual information from all other positions.

During training, the encoder receives the full observed input sequence, such as past weather patterns over several weeks in the domain of weather forcasting. The decoder is provided with the leftmost portion of the target sequence, which is, the known values from the beginning of the forecast window. For example, if the goal is to predict the temperature over the next 10 days, the decoder might initially receive only a start-of-sequence token or the first known value and must predict the next value in the sequence. To prevent the decoder from accessing future values during training, a masking strategy is applied in the self-attention, layers. This ensures that each prediction depends only on earlier positions in the output sequence, simulating real-world forecasting conditions. The model is trained by comparing each predicted value to the actual value using a suitable loss function such as cross-entropy or mean squared error, depending on the output type. This approach enables the model to learn autoregressive generation, where each future value is predicted step by step, conditioned on both the encoder input and previously predicted outputs. [7]

Although the original Transformer combines encoder and decoder modules, simplified variants such as BERT and GPT-3 omit either the decoder or encoder component. BERT uses an encoder-only architecture suited for classification and representation tasks, while GPT-3 is built on a decoder-only architecture optimized for generative tasks.

[13], [17], [15]

2.3 Software Engineering

3 Methodology

TODO write introduction

3.1 Data

The underlying data was provided by the Swiss Federal Institute for Forest Snow and Landscape Research WSL (**WSL**), with Dr. K. Liechti serving as the primary contact. Her contributions offered valuable insights not only into the dataset itself but also into the relevant geographical and meteorological processes.

In accordance with the legal restrictions outlined in Appendix A.1—specifically, the rounding of damage values and the aggregation of location data—the use of **WSLs** data in this thesis was subject to certain limitations. Interestingly, these constraints ultimately proved advantageous for the modeling process, as evidenced by the experimental results.

The sources for the recorded incidents were local and regional Swiss newspapers. As a result, the accuracy of the incident locations cannot be guaranteed, and the (financial) extent of the damages is only an approximation. In some cases, the location could not be precisely determined; thus, only the region or canton was recorded. Due to these uncertainties, the financial extent had to be rounded, and the damages grouped by canton or region prior to publication.

As outlined in Appendix A.2, the scope of the dataset was extensive. The features selected for this thesis were limited to the following: “Gemeindenamen”, “Datum”, “Hauptprozess”, and “Schadensausmass”, which were identified as the most relevant variables related to damage.

Based on the inputs of K. Liechti, the relevant meteorological variables were identified as sunshine duration, ground temperature, snowfall, and rainfall.
The rationale for this selection is briefly summarized below; detailed explanations can be found in Section 2.1:
Sunshine hours influence ground temperature, which in turn can cause snowmelt or thaw

ground frost.
Ground temperature was not available via open-meteo[18]; therefore, the temperature at 2 meters above ground was used as a proxy.
Snowfall can contribute to snowmelt processes later in the seasonal cycle.
Rainfall directly contributes to the potential for flooding and can also indirectly increase ground temperature.

In a subsequent experiment, snowfall was found to have no significant impact and was therefore completely removed from the dataset.

3.1.1 Data Cleaning

The damage data referenced in Section 3.1 required several processing steps before it could be used in the modeling phase. As noted in Appendix A.1, the municipality names correspond to the administrative boundaries of 1996 and are thus not up to date. To identify outdated names, GPS coordinates were retrieved using the Geocoding API [19]. For approximately 300 out of 2759 municipalities, no coordinates could be retrieved. Manual analysis of these cases revealed recurring issues.

For some incidents, as described in Section 3.1, **WSL** could not determine the exact location and had to assign them to a canton (30 of 28,515 cases), region (3), or district (10). Due to their low frequency, these entries were excluded from the dataset.

Common abbreviations used in the **WSLs** dataset—such as “a.A.” for “am Albis” or “St.” for “Sankt”—were standardized. Additionally, some municipalities had been merged into others since 1996. These cases were manually updated with their current names.

The weather data required less preprocessing. In eight municipalities, occasional values were set to ‘null’ (no data available); these were replaced by 0.

3.1.2 Availability of Sources and Data Collection

The data currently in use was collected with relatively little difficulty. The damage data was kindly provided by K. Liechti from the **WSL** following a formal request via email [20].

For the collection of weather data, the initial approach was to use official government data provided by MeteoSwiss [21]. However, due to the structure of the website and the raw nature of the station-based measurement data, this approach was ultimately abandoned. During further research, the open-meteo API [18] was discovered. To avoid excessive costs, a free academic access key was requested and kindly provided [22].

To obtain information on soil conditions, the first resource consulted was the Swiss federal geoportal map.geo.admin [23]. However, the format of the data was mostly incompatible with the tools available for this thesis. An alternative considered was the GIS Browser [24], which is the cantonal equivalent of map.geo.admin [23]. Unfortunately, it posed the same limitations as the federal source.

Given that new buildings are constantly being constructed in Switzerland and that the Swiss Confederation is actively researching locations for a nuclear waste repository [25], it was assumed that public institutions must maintain relevant geotechnical data.

First, the building construction office of Affoltern am Albis was contacted [26]. They referred the inquiry to the cantonal building construction office, which also denied possession of such data and redirected the request to the Office for Spatial Development [27].

The contact person from the Office for Spatial Development [28] was likewise unable to provide relevant data or further contacts. Their suggestion was to consult the GIS Browser [24] or map.geo.admin [23].

After these repeated unsuccessful attempts, the GIS Helpdesk was contacted [29]. The proposed solution [30] was again to use the GIS Browser or map.geo.admin, which had already proven inadequate. Due to time constraints, this approach was ultimately abandoned.

3.1.3 Data Preparation

After collecting all relevant datasets, a series of preprocessing steps were applied to construct a complete spatio-temporal dataset suitable for storm damage forecasting.

Adding Non-damage Data:

The original dataset, discussed in Section 3.1 provided by **WSL** contained only records of storm damage events, each described by the attributes: Date, Municipality, Main Process, and Extent of Damage. However, to train a forecasting model, it was necessary to include days and locations with no reported damage. Therefore, the dataset was extended by computing the Cartesian product of:

$$\text{Dates} \times \text{Municiaplities} \tag{3.6}$$

Let D denote the set of all the dates from 1972 to 2023 and M the set of all Swiss municiaplities based on the Swiss official commune register [31] published in 2013. We constructed:

$$X = \{(d, m)\} \mid d \in D, m \in M \tag{3.7}$$

This set was then left-joined with the original storm damage records. For entries where no damage was raported, the fields Extent of Damage and Main Process were inputed with zeros. Furthermore, due to political changes over the decades (e.g., municipal mergers), all historical municipality names were mapped to their most recent equivalent, based on the Swiss official commune register [31]. As a result, the final base dataset consited of 52,399,36 rows of which:

- 52′372′088 represented non-damage instances
- 24′613 corresponded to small damage events
- 1′800 were classified as medium damage
- 859 indicated large-scale damages

Spatial Clustering:

To address the extreme class imbalance and to comply with **WSLs** data usage disclaimer (Appendix A.1), we aggregated municipalities into k spatial clusters using k-means clustering on geographic coordinates (latitude and longitude). Let $x_i = (\lambda_i, \varphi_i)$ be the coordinates for municipality i. The clustering objective was to minimize:

$$\sum_{i=1}^{N} \min_{j \in \{1 \ldots k\}} \left(\left\| x_i - \mu_j \right\| \right)^2 \tag{3.8}$$

[32] where μ_j denotes the centroid of cluster j. This was implemented using the KMeans algorithim from SciKitLearn [33].

To ensure deterministic behavior of the KMeans algorithm from SciKitLearn [33], we specified both the random_state parameter and a fixed number of initializations. In particular, we set: random_state= 42 and n_init = 10. This guarantees that, for a given number of clusters k, the clustering results are identical across repeated runs. The random_state controls the random number generation used for centroid initialization, and setting it ensures reproducibility of the clustering outcome. [33] Figure 3.9 presents an illustrative example of the spatial clustering of all municipalities into $k = 6$ clusters.

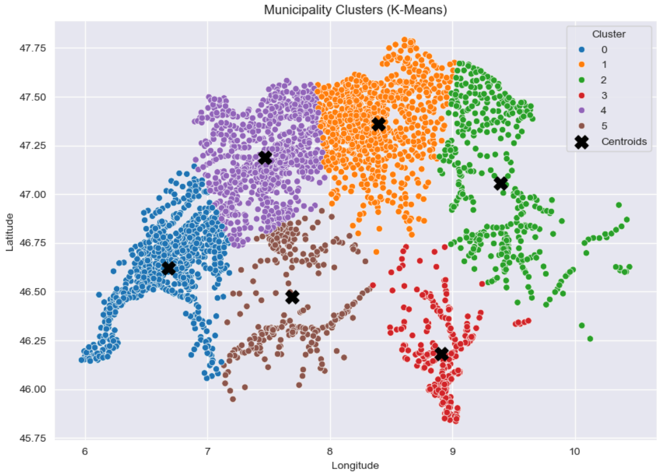

Figure 3.9 – Example clustering of all Swiss municiaplities with $k = 6$. The black crosses indicate the centroids of the respective clusters.

Determining the optimal number of clusters proved challenging, as no clear "elbow point" could be identified in the curve shown in Figure 3.10. Instead of relying on a single fixed value, we opted to use a set of cluster counts with $k = 3$ and $k = 6$. This range was chosen based on the observation that the within-cluster sum of squares decreases most noticeably in this interval, indicating a diminishing return in compactness beyond six clusters.

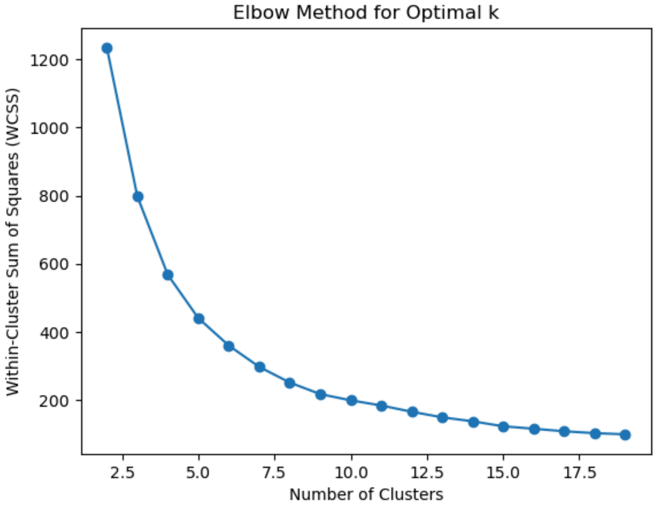

Figure 3.10 – Elbow plot showing the number of clusters on the x-axis and the corresponding within-cluster sum of squares (WCSS) on the y-axis

Each damage entry was then aggregated per cluster center and normalized by a weighted sum reflecting the severity of the damage class (small, medium, large). This yielded a dataset with k time series, one for each cluster.

Temporal Grouping:

The data were then aggregated at weekly intervals. For each cluster and week, the total storm damage was computed by summing the mean monetary value assigned to each damage class. Specifically, each daily damage event was replaced by the average monetary damage associated with its class (as derived from the original dataset). Then, the total weekly damage was calculated as:

$$\mathrm{Damage}_{\mathrm{week}} = \sum_{\mathrm{day} \, \in \, \mathrm{week}} \mathrm{MeanDamage}_{\mathrm{class(day)}} \tag{3.9}$$

$\mathrm{MeanDamage}_{\mathrm{class(day)}}$ is the average damage in CHF for the class of the damage event on that day. The averages were provided by K. Liechti (**WSL**):

- Class 1 (small): 0.06 Mio CHF
- Class 2 (medium): 0.8 Mio CHF
- Class 3 (large): 11.3 Mio CHF

The final dataset consists of entries with the following attributes per time window (week) and cluster center:

- end_date: last day of the week
- center_municipality: name of the cluster centroid
- cluster_center_latitude, cluster_center_longitude: Geographical coordinates of the cluster center
- damage_grouped: aggregated and binned damage label (0-3)

To convert the continuous aggregated damage values into categorical classes, we defined a binning procedure based on quantiles of the non-zero damage distribution.

Let $D = \{d_1, d_2, ..., d_n\}$ be the set of non-zero aggregated damae values and q_1, q_2, q_3 be the proportions of the damage classes where $q_1 = 0.9005, q_2 = 0.0667, q_3 = 0.0328$. The bin thresholds T_{lowe} and T_{mid} were computed as:

$$\begin{aligned} T_{\mathrm{low}} &= \mathrm{percentile}(D, 100 * q_1) \\ T_{\mathrm{mid}} &= \mathrm{percentile}(D, 100 * (q_1 + q_2)) \end{aligned} \tag{3.10}$$

They also depend on the number of spatial clusters k, which determines how many data points contribute to the distribution of damages per region. Then, the aggregated damage values were classified into four ordinal classes based on the following thresholds:

- Class 0: $(d = 0)$
- Class 1: $(0, T_{\mathrm{low}}]$

- Class 2: $(T_{\mathrm{low}}, T_{\mathrm{mid}}]$
- Class 3: $(T_{\mathrm{mid}}, \infty)$

3.2 Deep Learning Experiments

The goal of the experiments was to identify the most suitable deep learning architecture for predicting storm damage events based on weather-related input features. We evaluated different types of neural networks, beginning with a baseline Feedforward Neural Network (**FNN**), and compared their performance on a held-out test set. The result of the different models are discussed in Section 4.1.

Datasets

As shown in Table 3.1, the dataset was split temporally into a training set and a hold-out test set in order to simulate realistic forecasting scenarios and to prevent information leakage. The training set spans the years 1972–2013, while the test set covers the period from 2013 to 2023.

To further support model selection and hyperparameter tuning, the training set was divided using 5-fold time-aware cross-validation. Each fold was split into a training and a validation subset. This approach allowed us to assess model generalization performance over temporally consistent data splits, where earlier data was used for training and later data for validation.

All features were normalized using Z-score normalization, defined as $Z = \frac{X-\mu}{\sigma}$ [34]. The mean μ and standard deviation σ were computed from the training set only, and these values were reused to normalize the test set. This ensures that no information from the test set leaks into the training or validation stages.

Table 3.1 – Summary of dataset splits used for training and evaluation.

Nr of Clusters	Set	Number of Patterns	Years	Damages	No Damages
3	Train	6′573	1971–2013	2′242	4′331
3	Test	1′566	2013–2023	697	859
6	Train	13′146	1971–2013	2′872	10′274
6	Test	3′132	2013–2023	910	2′222

Training Pipeline

The complete training workflow is illustrated in Figure 3.12. Model development was carried out using PyTorch, and experiment tracking including metric logging, configuration management, and model evaluation was conducted using Weights & Biases (**WANDB**).

We first initialized the environment by detecting the available compute device (CPU or GPU) and configuring the training run. All the models were trained on a NVIDIDA L4 Tensor Core GPU.

The dataset was initially split into training and test sets as shown in Table 3.1, followed by 5-fold cross-validation on the training portion using a custom splitter based on SciKit-Learn's BaseCrossValidator [33]. This custom method, ClusteredTimeSeriesSplit, is shown in Figure 3.11. It ensures chronological consistency by keeping validation data strictly later in time than training data within each geographic cluster.

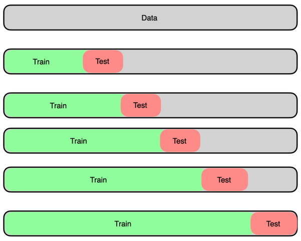

Figure 3.11 – Chronological 5-fold cross-validation. Each fold validates on a later time window, preserving the time series structure.

Within each fold, the model was trained over multiple epochs, which is shown in Figure 3.12 on the 4th line. We used the Adam [35] optimizer, as it provides adaptive learning rate updates and has been shown to work well in practice for deep learning tasks involving sparse gradients. To further improve training stability and avoid overfitting, we employed a learning rate scheduler (ReduceLROnPlateau), which reduces the learning rate by a factor of 0.5 if the validation loss does not improve for 5 consecutive epochs. To address class imbalance in the storm damage classes, class-specific weights were computed from the training set and used in the respective loss function. At the end of each epoch, the models performance was evaluated on the validation fold using accuracy, precision, recall, and F1 score, which were logged via **WANDB**.

After all folds were completed, the model with the highest average F1 score across validation folds was selected. As shown in steps 5 and 6 of Figure 3.12, this model was retrained on the entire training set without validation and subsequently evaluated on the held-out test set. The final performance metrics were recorded in the experiment summary for comparison between architectures.

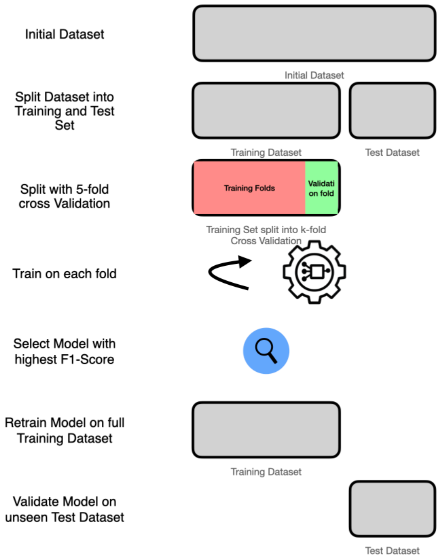
Initial Dataset
Initial Dataset
Split Dataset into Training and Test Set
Training Dataset
Test Dataset
Split with 5-fold cross Validation
Training Folds
Validati on fold
Training Set split into k-fold Cross Validation
Train on each fold
Select Model with highest F1-Score
Retrain Model on full Training Dataset
Training Dataset
Validate Model on unseen Test Dataset
Test Dataset

Figure 3.12 – End-to-end training pipeline, from dataset preparation through cross-validation and final testing.

3.2.1 Initial Findings and Design Decisions

In the early stages of model development, we conducted exploratory experiments using a prototype model to predict the exact extent of storm damage, framed either as a multi-class classification or regression task. These experiments included variations in the number of clusters k used for spatial grouping. However, initial results revealed that the model consistently converged toward predicting only the majority class or mean value, regardless of the input sequence. This behavior led to poor discriminative power in practice.

As shown in Section 3.1, the dataset is highly imbalanced, with the vast majority of events corresponding to class 0 (no damage) or low average damage values. This imbalance caused the model to exploit the loss function by minimizing risk through constant prediction of the dominant class. Consequently, it failed to capture meaningful distinctions between damage levels.

Given these outcomes and the underlying class distribution, we reframed the problem as a binary classification task: predicting whether any storm damage will occur (damage/no-damage). This formulation reduces modeling complexity and mitigates the effects of class imbalance, while still offering practical value for early-warning systems and resource allocation.

To address the problem of class imbalance, all the models were trained to minimize the loss function, using class weights computed via sklearn's [33] compute_class_weight function. This approach assigns higher loss penalties to underrepresented classes. The computed class weights were passed during initailitation to the PyTorch [11] CrossEntropyLoss function, allowing the model to pay more attention to minority class predictions.

3.2.2 FNN based forecasting model

Our first experiment employed a baseline Feedforward Neural Network (**FNN**), whose architecture is illustrated in Figure 3.13. The **FNN** was used as a baseline model, as they are simple to create and light in computation time. The network consists of 10 fully connected layers with ReLU activation functions. This depth was chosen to for a sufficient level of nonlinearity to capture complex feature interactions, while keeping the model small enough to avoid overfitting. The model was trained using the Adam optimizer and Cross Entropy Loss Function.

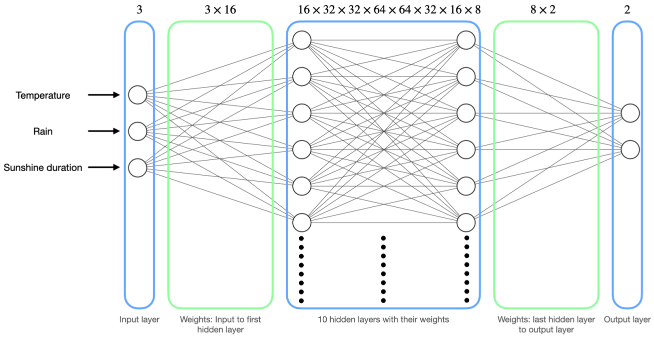
3
3 × 16
16 × 32 × 32 × 64 × 64 × 32 × 16 × 8
8 × 2
2
Temperature
Rain
Sunshine duration
Input layer
Weights: Input to first hidden layer
10 hidden layers with their weights
Weights: last hidden layer to output layer
Output layer

Figure 3.13 – Illustration of the **FNN**: 3 Input Neurons, 2 Output Neurons. 8 hidden layers with Neurons variing between 8 and 64 as shown in the illustration respectively.

Input features

The input to the **FNN** consisted of three weather-related variables: mean temperature at 2 meters, total rainfall, and sunshine duration. All features were normalized to zero mean and unit variance, as described in Section 3.1, and were provided on a weekly basis for each cluster in the dataset.

3.2.3 LSTMs based Forecasting Model

To model temporal dependencies in the weather-related input features, we implemented a sequence model based on (**LSTM**) units. The architecture is illustrated in Figure 3.14 and consists of a stack of **LSTM** layers followed by a fully connected output layer.

The LSTM block, shown in the middle of Figure 3.14, receives as input a multivariate time series of weather features, such as temperature, rainfall, and temperature, over a fixed sequence window. To increase the model's representational capacity and capture higher-level temporal abstractions, multiple .[LSTM] layers were stacked, as shown in Figure 2.5. This resulted in a additional hyperparameter k, for the number of stacks.

The output of the final LSTM layer is a hidden state for each time step in the input sequence. To reduce this sequence to a single prediction, we extract the hidden state from the last time step. This strategy assumes that the final time step contains the most relevant information for predicting the next event, which aligns with common practices in time series classification and forecasting as shown in the official PyTorch documentation [11].

The last layer of the model is a fully connected linear layer that maps the **LSTM** output to a 2-dimensional output space, corresponding to a binary classification task. This is shown on the right side in Figure 3.14. This architecture was chosen based on the findings of Steven Elsworht et. al in "Time Series Forecasting Using LSTM Networks: A Symbolic Approach" [36]. Additionally, this architecture had the advantages of having a balance between compactness and computational efficiency, which made it suitable for forecasting storm damages over a long historicaltime span.

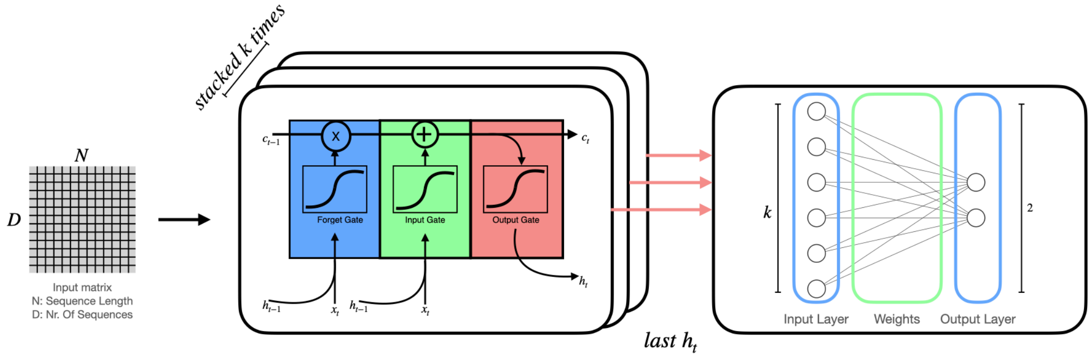

Figure 3.14 – Architecture of the LSTM-based forecasting model. The model consists of stacked LSTM layers followed by a fully connected output layer.

3.2.4 Transformer-Based Forecasting Model

In addition to recurrent architectures, we implemented a Transformer-based model to evaluate whether attention mechanisms could better capture long-range temporal dependencies in the weather time series data. The architecture is shown in Figure 3.15 and is composed of a sequence embedding layer, positional encoding, stacked Transformer encoder layers, and a feedforward output layer.

The input to the model is a multivariate sequence of weather observations, same as in Section 3.2.3. Each input vector at a time step is first passed through a linear embedding layer that projects it to a fixed-dimensional representation (d_model). Since Transformers do not have a built-in notion of sequence order, a trainable positional encoding is added to each embedded input vector. This enables the model to learn relative and absolute temporal positions within the input sequence. [13]

TODO: Embedding

TODO: Position Encoding using [37]

The core of the architecture is a Transformer encoder block consisting of num_layers stacked layers of multi-head self-attention and feedforward sub-layers. Here we used the built in Transformer module of the PyTorch Library [11]. These layers allow each time step to attend to all others in the sequence, enabling the model to capture both short- and long-term dependencies without recurrence. The encoder is applied in a symmetric fashion (x is passed as both source and target), as no autoregressive decoder is needed for our forecasting task.

The Transformer output is then passed through an additional feedforward layer with ReLU activation, followed by a final linear layer that maps the representation to the target space of 2. As in the LSTM model, only the representation of the final time step is used for prediction, assuming the most recent observations are most relevant for forecasting the next event.

This architecture was selected to leverage the Transformer's strength in modeling global dependencies and to benefit from parallelizable training, which is particularly valuable for long weather sequences.

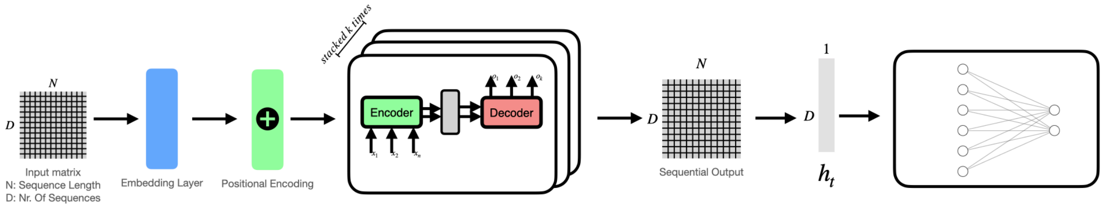

Figure 3.15 – Architecture of the Transformer-based forecasting model. The model includes input embeddings, positional encoding, stacked self-attention layers, and a feedforward output module.

3.2.5 Model Comparison Setup

The three evaluated architectures **FNN**, **LSTM**, and the Transformer offer distinct capabilities for modeling storm damage based on weather sequences. The Feedforward Neural Network (FNN) serves as a non-sequential baseline and is limited to using only the current week as input. The LSTM, in contrast, can model short- and medium-range temporal dependencies through its memory mechanism. The Transformer introduces a more powerful attention-based framework, enabling the modeling of long-range dependencies across input sequences. This architectural progression allowed us to systematically assess the relationship between model complexity and forecasting performance in a controlled setting.

To ensure a fair comparison, all models were trained under the same experimental conditions. Hyperparameter optimization was performed using the Sweep API provided by **WANDB**, with a maximum of 20 trials per architecture. The hyperparameters were optimized via Bayesian search, which balances exploration and exploitation to efficiently converge to high-performing configurations. [38]

Table 3.2 – Hyperparameter search space used during model sweeps.

Hyperparameter	**Search Space**
Batch Size	[32, 64, 128]
Learning Rate	0.00001–0.01
Epochs	10–100
Sequence Length	[0, 2, 4, 12, 52]

In addition, to assess the impact of spatial aggregation on model performance, each architecture was trained using different cluster counts (k in {3, 4, 5, 6}), which determine the number of geographic regions derived from the spatial clustering process (see Section 3.1.3). By evaluating model performance across different levels of spatial granularity, we aimed to determine whether finer or coarser regional segmentation improves generalization and damage detection performance.

3.3 Software Engineering

3.3.1 Backend

Technologies

BEcause of its reliability and years of improvement as the base of the backend, java was chosen. To imporve the startup time and quickly create functioning code, springboot was the chosen framework. its dependency on new weather data provided through an API arrose the need for a database to relieve the API. ostgreSQL was chosen for its reliability and strong compatibility with standard SQL.

Architecture

TODO

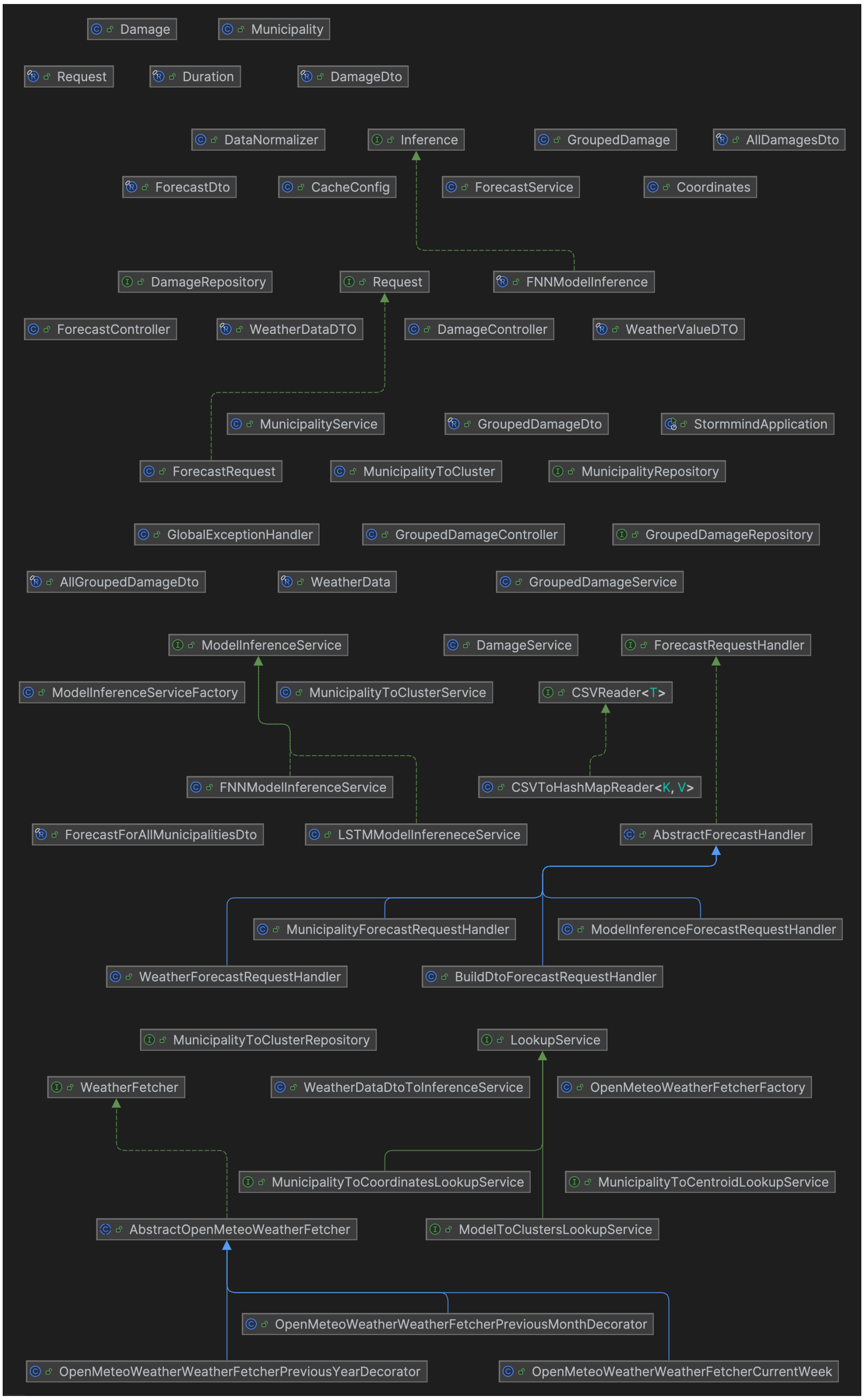

Figure 3.16 – Class Diagramm: generated from JetBrains IntelliJ IDEA

3.3.2 Frontend

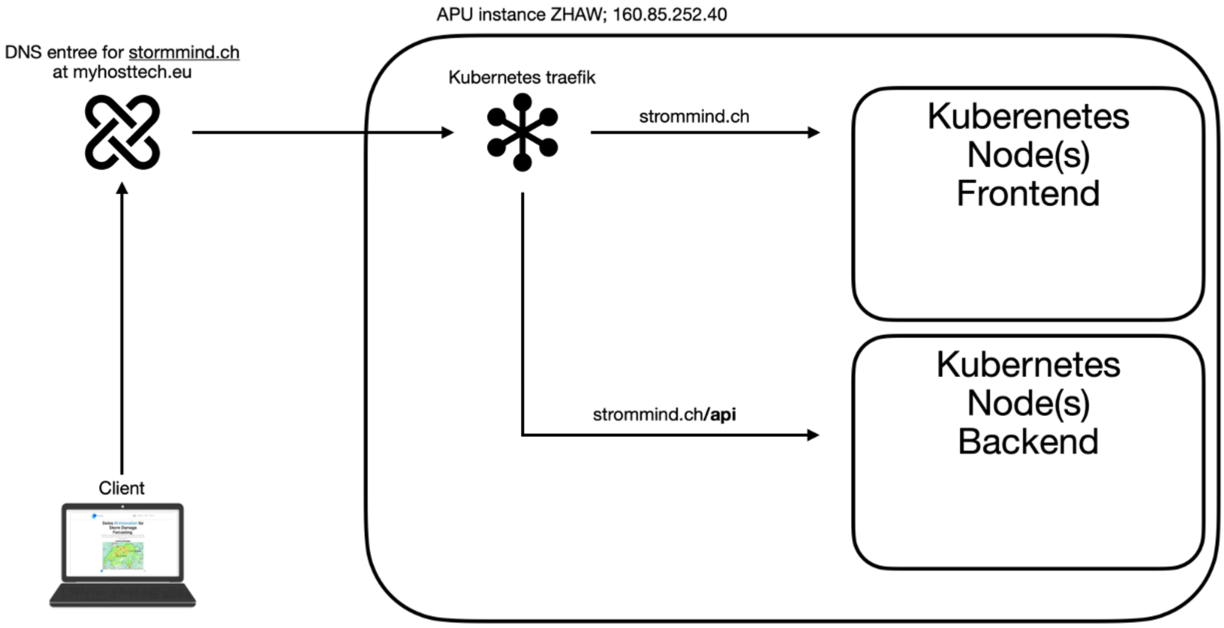

Figure 3.17 – web routing: created with apple freeform, laptop from chatgpt (prompt: "erstelle mir ein png eines minimalistischen laptops ohne hintergrund")

Technologies

The Frontend consists of a react/vite repository. The DNS Entree was made on Hosttech and references an instance on the Openstack cluster of ZHAW [39]

3.3.3 Test Concept

4 Results

This chapter presents the outcomes of both the deep learning experiments and the software engineering components developed during this project. Together, these results address the overarching research goal: to build a reliable and interpretable system for binary storm damage forecasting, supported by a user-oriented software solution.

The first part of this chapter focuses on the quantitative evaluation of machine learning models, where different deep learning architectures were compared with respect to their predictive performance on temporally and spatially structured weather data. These results provide empirical insight into how model complexity and spatial clustering affect classification accuracy and robustness under real-world forecasting conditions.

The second part of this chapter presents the outcomes of the software engineering effort. It details the implementation of the system's user interface and back-end components, designed to support decision-making and visualization for storm damage predictions. We report the results of functional testing, performance measurements, and usability considerations for the final application.

By combining rigorous model evaluation with software, this chapter demonstrates both the predictive capabilities and the practical deployment potential of the proposed solution.

4.1 Results of AI Engineering

The following section presents the results of our comparative evaluation of three deep learning architectures, **FNN**, **LSTM**, and Transformer, applied to the task of binary storm damage forecasting. This experimental block aims to answer three key questions:

- Which architecture achieves the best performance on the held out test set?

- Does increased model complexity (e.g., through temporal modeling in LSTM or long-range dependency modeling in Transformers) lead to better forecasting performance compared to the baseline FNN?
- How does spatial granularity, operationalized via cluster sizes $k \in \{3, 6\}$, affect model performance? 3 and 6 were chosen as number of clusters due to the highest decreasing within sum of cluster centroid as described in Section 3.1. The number of 26 was chosen as it aligns with the number of Cantons of Switzerland and we wanted to test on a more finer granularity.

To assess these dimensions, each model was trained independently on datasets grouped by different cluster counts. For every architecture–cluster configuration, we performed 20 independent training runs to capture the effects of stochasticity in optimization and initialization.

To ensure statistically robust comparisons, we report the mean and variance of all evaluation metrics across the 3 best runs over the 20 independent runs, following the best practices recommended in X. Bouthillier et. Al. "Accounting for Variance in Machine Learning Benchmarks" [40]. The reason for only choosing the top 3 runs per model and cluster, is to eliminate any negative outliers. We are more interested in what the model is capable to do in its best configuration, rather than accounting for worse configurations. For inter-model comparison, we primarily use the macro-averaged F1 score on the test set, as summarized in Table 4.3. This metric is particularly appropriate given the class imbalance described in Section 3.1, as it gives equal importance to both classes. Since the minority class (damage) represents the critical target, macro-F1 is better suited than accuracy alone. Additional metrics, including accuracy, AUC, precision, recall, and specificity, are discussed in detail in the model-specific result subsections.

TODO add rest of the values

Table 4.3 – Average test macro F1-score and variance for each model across different spatial cluster configurations.Each value represents the mean F1-score over the top 3 runs from the 20 independent training runs, with the corresponding variance shown. Results are grouped by the number of spatial clusters $k \in \{3, 6, 26\}$ used during data preparation.

Clusters	3	6	26
FNN	$0.67 \pm 9.5e{-6}$		
LSTM	$0.67 \pm 2.2e{-6}$		
Transformer	$0.68 \pm 1.4e{-6}$		

4.1.1 Feedforward Neural Networks (FNNs): Results

Cross Validation Results

Test Set Performance

Table 4.4 – Performance Metrics of the **FNN** model. We provide the mean over the 20 runs and the variance

Cluster	3	6	26
Accuracy	$67.55\% \pm 0.35$		
AUC	$0.71 \pm 2.2e{-5}$		
F1	$0.66 \pm 3.6e{-5}$		
Precision	$0.67 \pm 4.9e{-5}$		
Specificity	$0.66 \pm 3.5e{-5}$		

Figure 4.18 shows the ROC curve of the best performed **FNN** model in the 20 runs for $k = 3$. This visualization provides insight into the classification behavior of a single run, we emphasize that the average metrics over multiple runs are used for formal model comparison, as discussed in Section Section 4.1.

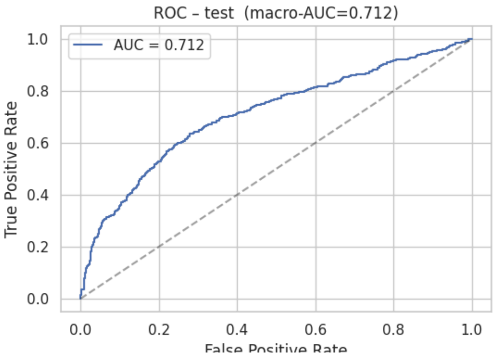

Figure 4.18 – ROC curve for the best-performing **FNN** model (cluster size = 3) across 20 runs, selected based on highest AUC score.

4.1.2 LSTM Neural Network

Test Set Performance

Table 4.5 – Performance Metrics of the **LSTM** model. We provide the mean over the top 3 of the 20 runs and the variance.

Cluster	3	6	26
Accuracy	$68.41\% \pm 0.02$		

AUC	$0.71 \pm 2.4e{-8}$
F1	$0.67 \pm 2.2e{-6}$
Precision	$0.68 \pm 9.1e{-6}$
Specificity	$0.67 \pm 1.5e{-6}$

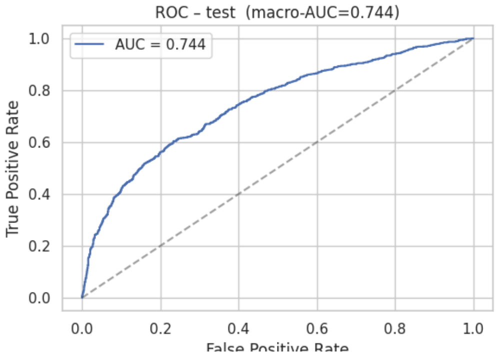

Figure 4.19 – Training of the **FNN**

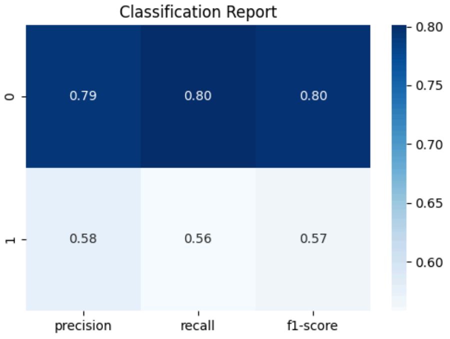

Figure 4.20 – Confusion Matrix of the **FNN**

4.1.3 Transformer

Cross Validation Results

Test Set Performance

Table 4.6 – Performance Metrics of the Transformer model. We provide the mean over the top 3 of the 20 runs and the variance

Cluster	**3**	**4**	**5**	**6**

Accuracy	$68.14\% \pm 0.00$
AUC	$0.72 \pm 9.6e{-5}$
F1	$0.68 \pm 1.4e{-6}$
Precision	$0.68 \pm 4.6e{-07}$
Specificity	$0.67 \pm 2.9e{-6}$

4.2 Conclusion

4.3 Software Results

5 Discussion and Outlook

Bibliography

[1] S. F. R. I. WSL, *USDB_1972_2023_ohneSchadenszahlen_ohneBeschriebe.* (2023).

[2] “Erdrutsch.” Jan. 13, 2025. Accessed: May 04, 2025. [Online]. Available: https://de.wikipedia.org/w/index.php?title=Erdrutsch&oldid=252204629

[3] “Austausch zu Unwetterschäden und Daten.” [Online]. Available: https://teams.microsoft.com/l/meetup-join/19%3ameeting_ZDk5ODM1MWQtN2E2Yi00NDZjLWFiNTEtOWE0ZWU4ODQ3YzA1%40thread.v2/0?context=%7b%22Tid%22%3a%225d1a9f9d-201f-4a10-b983-451cf65cbc1e%22%2c%22Oid%22%3a%225f85a69b-df7a-4a9a-acab-f5ad7d25b1a9%22%7d

[4] Maria, “Gewicht nasser Erde: Formel zum Umrechnen.” Accessed: May 06, 2025. [Online]. Available: https://hortica.de/gewicht-nasse-erde/

[5] Designerpart, “Schüttgut-Gewicht - Big Bag Puhm | Alles über Gewicht und Volumen.” Accessed: May 06, 2025. [Online]. Available: https://bigbag-puhm.at/handhabung/schuettgut-gewicht/

[6] “NN SVG.” Accessed: May 03, 2025. [Online]. Available: http://alexlenail.me/NN-SVG/

[7] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook.* Cham: Springer International Publishing, 2023. doi: 10.1007/978-3-031-29642-0.

[8] I. Mrázová, “Multi-Layered Neural Networks.”

[9] F.-P. Schilling and Z. Cai, “Lecture 05: Sequential Models,” Mar. 19, 2025.

[10] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[11] “PyTorch Foundation.” Accessed: May 06, 2025. [Online]. Available: https://pytorch.org/

[12] D. Thakur, "LSTM and Its Equations." Accessed: May 05, 2025. [Online]. Available: https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af°

[13] A. Vaswani *et al.*, "Attention Is All You Need." Accessed: May 26, 2025. [Online]. Available: http://arxiv.org/abs/1706.03762°

[14] Q. Wen *et al.*, "Transformers in Time Series: A Survey." Accessed: May 26, 2025. [Online]. Available: http://arxiv.org/abs/2202.07125°

[15] S. J. D. Prince, "Understanding Deep Learning."

[16] A. Dosovitskiy *et al.*, "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale." Accessed: May 31, 2025. [Online]. Available: http://arxiv.org/abs/2010.11929°

[17] S. Frank-Peter, "09_Transformers," Apr. 16, 2025.

[18] P. Zippenfenig, "Open-Meteo.Com Weather API." [Online]. Available: https://open-meteo.com/°

[19] "OpenCage - Easy, Open, Worldwide, Affordable Geocoding and Geosearch." Accessed: May 06, 2025. [Online]. Available: https://opencagedata.com/°

[20] K. Liechti, "RE: Anfrage Zur Nutzung Der Unwetterschadens-Datenbank Für Bachelorarbeit." Nov. 29, 2024.

[21] "MeteoSwiss IDAWEB: Login at IDAWEB." Accessed: May 06, 2025. [Online]. Available: https://gate.meteoswiss.ch/idaweb/login.do°

[22] P. Zippenfenig, "Professional API Key for Research Purposes."

[23] "Maps of Switzerland - Swiss Confederation - Map.Geo.Admin.Ch." Accessed: May 06, 2025. [Online]. Available: https://map.geo.admin.ch/#/map?lang=en&center=2660000,1190000&z=1&topic=ech&layers=ch.swisstopo.zeitreihen@year=1864,f;ch.bfs.gebaeude_wohnungs_register,f;ch.bav.haltestellen-oev,f;ch.swisstopo.swisstlm3d-wanderwege,f;ch.vbs.schiessanzeigen,f;ch.astra.wanderland-sperrungen_umleitungen,f&bgLayer=ch.swisstopo.pixelkarte-farbe°

[24] "GIS-Browser Geoportal Kanton Zürich." Accessed: May 06, 2025. [Online]. Available: https://geo.zh.ch/maps?x=2693065&y=1253028&scale=279770&basemap=arelkbackgroundzh°

[25] "Vergraben und vergessen." Accessed: May 06, 2025. [Online]. Available: https://www.derbund.ch/vergraben-und-vergessen-413137488243°

[26] "Phone call Hochbau und Umwelt Affoltern am Albis." Aug. 04, 2025.

[27] “phone call Hochbau Amt Zürich.” Aug. 04, 2025.

[28] “phone call Amt für Raum Entwicklung und Vermessung.” Aug. 04, 2025.

[29] D. Ueltschi, “Bodenbeschaffenheitskarte.”

[30] F. GIS, “[ARE-JIRA] GIS-2262 [EXTERN] Bodenbeschaffenheitskarte.”

[31] *Amtliches Gemeindeverzeichnis der Schweiz - MS-Excel Version*, no. 286080. Bundesamt für Statistik (BFS) / BFS. Accessed: Feb. 25, 2025. [Online]. Available: https://dam-api.bfs.admin.ch/hub/api/dam/assets/286080/master°

[32] “2.3. Clustering.” Accessed: May 13, 2025. [Online]. Available: https://scikit-learn/stable/modules/clustering.html°

[33] “Scikit-Learn: Machine Learning in Python — Scikit-Learn 1.6.1 Documentation.” Accessed: May 06, 2025. [Online]. Available: https://scikit-learn.org/stable/#°

[34] “Standard Score.” May 24, 2025. Accessed: May 29, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Standard_score&oldid=1291997598°

[35] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” Accessed: May 06, 2025. [Online]. Available: http://arxiv.org/abs/1412.6980°

[36] S. Elsworth and S. Güttel, “Time Series Forecasting Using LSTM Networks: A Symbolic Approach.” Accessed: May 29, 2025. [Online]. Available: http://arxiv.org/abs/2003.05672°

[37] H. Irani and V. Metsis, “Positional Encoding in Transformer-Based Time Series Models: A Survey.” Accessed: May 30, 2025. [Online]. Available: http://arxiv.org/abs/2502.12370°

[38] “Weights & Biases.” Accessed: May 31, 2025. [Online]. Available: httpss://wandb.ai/wandb_fc/articles/reports/What-Is-Bayesian-Hyperparameter-Optimization-With-Tutorial%E2%80%94Vmlldzo1NDQyNzcw°

[39] “Login - OpenStack Dashboard.” Accessed: May 06, 2025. [Online]. Available: https://apu.cloudlab.zhaw.ch/auth/login/?next=/°

[40] X. Bouthillier *et al.*, “Accounting for Variance in Machine Learning Benchmarks.” Accessed: May 31, 2025. [Online]. Available: http://arxiv.org/abs/2103.03098°

[41] S. F. R. I. WSL, “Infos_Daten_Unwetterschadensdatenbank_WSL_english.” 2023.

List of Figures

List of Tables

A Appendix

A.1 Disclaimer

“ **Information on the Data of the Swiss flood and landslide damage database managed by WSL**

Please note the following when using the data:

- Names of municipalities refer to the state of 1996. I.e. some municipalities have merged.
- Media reports are the main source of information. There are local and regional differences in reporting. In addition, the focus of the media has changed over time.
- In some cases, it is difficult to assign the damage to a location and / or municipality (in a few cases of doubt, the damage is assigned to the respective canton capital or even the Swiss capital)
- The coordinates have been set based on information from media reports, images, etc. They can therefore deviate greatly from the real main point of damage.

The following must be observed when publishing:

- If the data values are cited or published, they must be provided with at least a reference to the source (“WSL Swiss Flood and Landslide Damage Database”).
- It must be clearly mentioned that the damage data are only estimates.
- Damage values must always be rounded in publications.
- No monetary damage may be published at community level, but only in aggregated form (regions, cantons).
- Monetary damage may only be published in a form that does not allow any conclusions to be drawn about individuals and individual objects.“[41]

A.2 Original Data Features

Gemeinde, Gemeindenummer, Weitere Gemeinde, Kanton, Prozessraum, MAXO Datum, Datum, MAXO Zeit, Zeit, Gewässer, Weitere Gewässer, Hauptprozess, Hauptprozess Rutschung Unterteilung, Hauptprozess Wasser/Murgang Unterteilung, Weitere Prozesse, Schadensausmass: gering [0.01-0.4]; mittel [0.4-2]; gross/katastrophal[>2] oder Todesfall [Mio. CHF], x-Koordinate, y-Koordinate, Schadenszentrum; Gemeindegebiet falls nicht bekannt, Grossereignisnummer; mehrere Ereignisse; welche aufgrund meteorologischer oder räumlicher Gegebenheiten zusammengefasst werden, Gewitterdauer MAXO, Gewitterdauer [Std.], Gewitter Niederschlagsmenge MAXO, Gewitter Niederschlagsmenge [mm], Dauerregen Dauer MAXO, Dauerregen Dauer [Std.], Dauerregen Niederschlagsmenge MAXO,Dauerregen Niederschlagsmenge [mm], Schneeschmelze MAXO, Schneeschmelze, Ursache nicht bestimmbar MAXO, Ursache nicht bestimmbar, ID