StormMind
Deep Learning based Web-Service for Storm Damage Forecasting

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 and Nils GÄMPERLI, declare that this thesis titled “StormMind - Deep Learning based Web-Service for Storm Damage Forecasting“ 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

Storms have caused over 11.2 billion Swiss francs in damages in Switzerland over the past five decades, yet the true cost may be even higher when considering the psychological toll on affected populations. This thesis explores whether Deep Learning models can forecast such storm-induced damages using only meteorological data. We compare three architectures: Feedforeward Neural Network (**FNN**), Long Short-Term Memory Neural Network (**LSTM**) and Transformer models, across various spatial granularities. To combat class imbalance, we introduced spatial clustering of municipalities and weekly temporal aggregation. Our best-performing model, a Transformer trained on six spatial clusters, achieved an accuracy of 70.25% in binary storm damage forecaesting. The final system was deployed as a publicly accessible web service (*stormmind.ch*), enabling real-time inference from current weather forecasts. This work not only demonstrates the feasibility of AI-driven storm damage forecasting, but also lays the groundwork for more comprehensive models integrating geospatial and infrastructural data in future research.

Key words: Deep Learning, Storm Forecasting, Transformer, Time Series, Switzerland, Web Application

Acknowledgements

We would like to thank Andreas Meier for his valuable support and guidance throughout the course of this project.

Furthermore, we would like to express our gratitude to: Dr. Käthi Liechti, for providing valuable insights into weather science and storm damage assessment. Dr. Ahmed Abdulkadir, for suggesting alternative approaches in the deep learning section. Dr. Iris Hübscher, for her helpful advice on academic writing and structural improvements. Werner Gämperli, Darius Ueltschi, and Dr. Andreas Ueltschi, for proofreading the manuscript and testing the website.

Finally, we extend our sincere thanks to the Swiss Federal Institute for Forest Snow and Landscape Research WSL (**WSL**) for collecting and providing the damage data, and to Patrick Zippenfenig for generously providing a free open-meteo API key, which made this project possible.

Preface

Growing up in Affoltern am Albis, Switzerland, a locality repeatedly battered by floods, we witnessed natural hazards as more than abstract news items. Water-logged fields and impassable forest tracks were fixtures of our childhood. These early encounters later converged with an academic fascination for Artificial Intelligence and Deep Learning which led to the question: Could algorithms trained on historical data forecast the event of storm damages?

A search for suitable datasets led us to the Storm Damage Database created by the **WSL**. When the institute gracefully granted access to five decades of geo-referenced storm-impact records, the idea of merging lived experience with algorithmic insight crystallised.

Table of Contents

1 Introduction

The objective of this project is to explore initial steps toward predicting or forecasting environmental damage. While environmental damage often involves complex geological interactions, this thesis focuses exclusively on meteorological factors to maintain a manageable scope. The analysis centers on weather data, with damage records extracted from regional newspaper reports. We applied three deep learning models: a **FNN**, a **LSTM**, and a Transformer, to evaluate how weather patterns relate to reported storm damage and assess the feasibility of data-driven forecasting.

To achieve this, we:

- Constructed a preprocessing pipeline that aggregates weather and damage data at weekly resolution and applies K-means clustering to group spatial locations.
- Trained and compare three neural network architectures of increasing complexity: a **FNN** as a baseline, a **LSTM** to capture short-term temporal dependencies, and a Transformer model to investigate the benefits of long-range attention mechanisms.
- Evaluated the performance of each model across different levels of spatial granularity (using cluster counts $k \in \{3, 6, 26\}$) and presented average metrics across multiple training runs to account for variance and ensure reproducibility
- Developed a software application that visualizes predictions and demonstrates how such a model could be applied in practice.

This thesis serves two purposes: to empirically evaluate deep learning methods for storm damage forecasting, and to provide an extensible framework for future research.

1.1 Comparable projects

The following projects demonstrate how AI is being applied in practice to address specific natural hazard challenges, from storm damage assessment to avalanche forecasting.

Swiss Re's Rapid Damage Assessment (**RDA**) leverages deep learning algorithms alongside existing Natural Catastrophes (**NatCat**) models, satellite imagery, weather, and property data to:

- assess the damage potential to properties
- prioritize property inspections following an event
- analyze property impacts to generate reports

Currently, the system is available only for tropical cyclones, tornadoes, and hailstorms in the US. However, there are plans to expand it to additional perils within the US and to extend coverage to other countries. [1]

RAvaFcast v1.0.0 [2] is an artificial intelligence system designed to improve avalanche prediction in high-risk regions in Switzerland. The research is based on historical avalanche records and weather data. The approach follows a three-stage model pipeline consisting of classification, interpolation, and aggregation. Evaluation results demonstrated a strong correlation between model predictions and human expert assessments. [3]

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 **RNN**, **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 [4]. In a conversation with 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. However, no definitive or commonly accepted values could be established. Given the timeframe of the thesis project, limited further research could not provide any conclusive informations.

Forest/Deforestation: Deforestation is widely recognized as a contributing factor to landslides, primarily due to the loss of root systems that stabilize the soil [5]. Liechti confirmed this statement

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

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.

Soil condition: This factor is discussed in detail within subsection where applicable.

Frozen ground: 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 builds the 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 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 detailed 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 soil conditions. [6]

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 cause significant snowmelt—can further augment the water load on the soil surface alongside rainfall. [6]

2.1.2 Reasons for Landslide

Of the potential causes for landslides, three were selected for consideration, as they were highlighted by Liechti [6] 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 ($0.83kg/dm^3$ [7]) to wet ($1.6 - 1.76kg/dm^3$ [8]). This gain in mass increases the instability of the soil.
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. [6]

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 **NN** architectures, as they form the core modeling approach used in this project.

2.2.1 Activation Functions

TODO

2.2.2 Feedforeward Neural Network (FNN)

FNN 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. **FNN** mimic this behavior by using computational units, also called neurons, connected by weighted links.

Architecture

A **FNN** trained with backpropagation, which is discussed in Section 2.2.3, can be illustrated as a directed acyclic graph with inter-connections. It contains a set of neurons distributed in different layers.

- Each neuron has an activation function.
- The first layer, shown on the left side in Figure 2.1, is called the input layer and has no predacessors. Furthermore, its 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 has no successors. Its 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 graph, are the weights, which represent an arbitrary number in $\mathbb{R}$ and are updated during the trianing process.

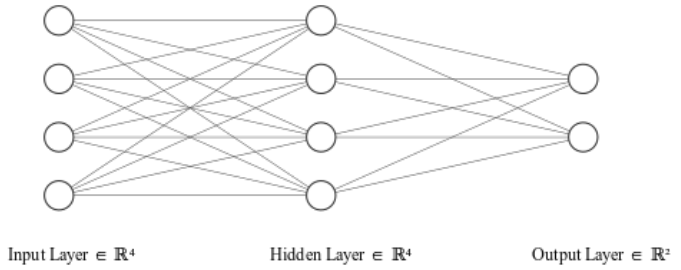

Figure 2.1 – Illustration of a Neural Network with 3 layers. Illustrated with [9]

Computation of the Output

The computation of the output in a **FNN** is referred to as a forward pass. Each neuron calculates its output by applying an activation function f to sum of its inputs. For input neuron i, the aggregated input (also called the potential) is denoted as ξ_i, yielding the expression $y = f\left(\sum_{i=1}^{n} \xi_i\right)$.

To complete a forward pass, this procedure is applied sequentially from the input layer through the hidden layers to the output layer. At each step, inputs are scaled by weight w_{ij} before being summed and passed through the activation function. This process is captured by the following equations:

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

[10] where j denotes the layer, ascending from input layer to output layer, L the number of layers, 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.

2.2.3 The Backpropagation Training Algorithm

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

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

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

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

[11]

The error function measures how far the actual output is from the desired output. 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.

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

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

[11]

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, w_{ij} the weight with index i at layer j and t to the timestep.

2.2.4 Recurrent Neural Network (RNN)

The feedforward **FNNs** models discussed in Section 2.2.2 can only handle inputs of fixed size, making them ill-suited for sequential data such as time series or language, where the input length can vary. To overcome this limitation, we introduce **RNNs**, a class of models designed specifically to handle sequences of arbitrary length.

Unlike FNNs, an **RNN** processes input one step at a time while maintaining a hidden state that carries information across time steps. This allows the model to "remember" relevant context from earlier in the sequence. Figure 2.3 shows a simple **RNN** with an input size of 1, while Figure 2.4 illustrates the same **RNN** unrolled over 4 time steps, highlighting how information flows through the sequence.

Architecture

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

- Input signal: The external data which is fed into the network at a time step t 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: $h_t = f(h_{t-1}, x_t)$, were x_t is the input at step 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.

[12], [10]

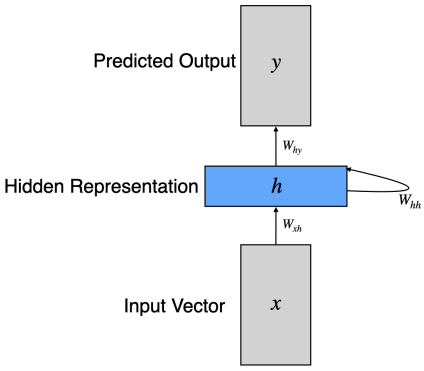

Figure 2.3 – **RNN**

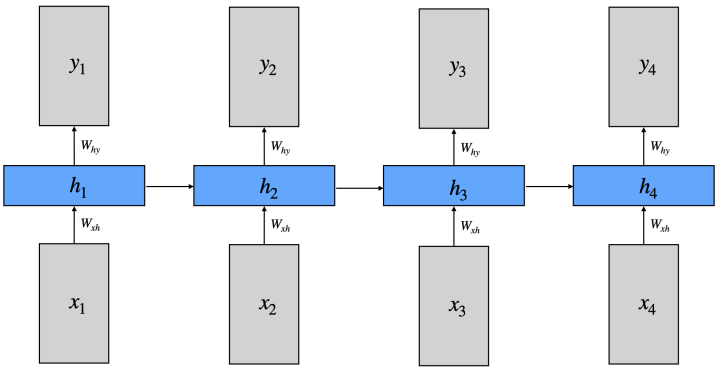

Figure 2.4 – 4 times unrolled **RNN**

An **RNN** processes input sequences one element at a time. As shown in Figure 2.3, the RNN takes a single input vector and produces an output. This corresponds to the case where the RNN processes the first element x_1 of an input sequence $\{x_1, x_2, ..., x_n\}$. In contrast, Figure 2.4 illustrates the **RNN** unrolled over four time steps. By the time the RNN reaches input x_4, it has already processed inputs x_1 through x_3. The output at this step, y_4 is influenced not only by x_4 but also by the hidden state h_4 that carries information from the previous inputs.

Vanishing Gradient Problem

The Vanishing Gradient Problem(**VGP**) is a challenge encountered during the training of **RNNs**, particullary dealing with **RNNs** and long input sequences. It arises from the way how gradients are updated during the backpropagation algorithm (discussed in Section 2.2.3), 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 previous parameters and impairing the network's ability to learn long-range

dependencies. The same principle arise, when the gradients become too large. In this case, the problem is called the exploding gradient problem.

[10]

2.2.5 Long Short-Term Memory Neural Network (LSTM)

LSTMs are a special form of **RNNs** designed to address the **VGP** while having a more fine-grained control over the previous input data and were introduced for the first time by Sepp Hochreiter in 1997 [13]. They are an enhancement because the reccurent mechanism that controls 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.

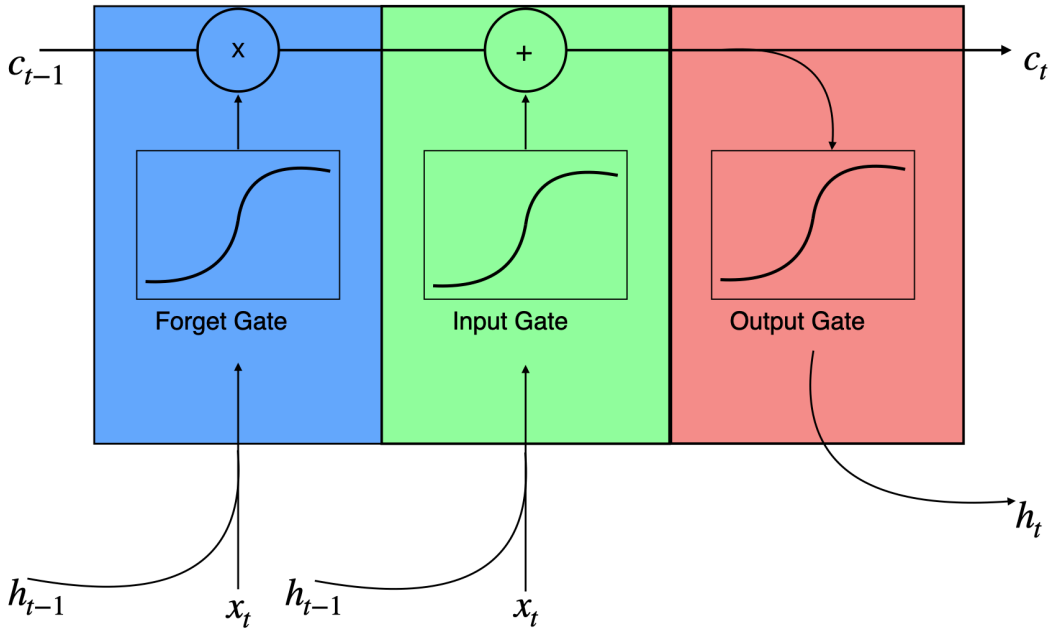

Figure 2.5 – Schematic illustration of an **LSTM** cell highlighting the internal gating structure. The colored blocks represent the three core gates—Forget (blue), Input (green), and Output (red)—and show how they interact with the cell and hidden states to regulate information flow.

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

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

[14], [15]

2.2.6 Transformer

With the advent of Large Language Models and influential works such as Attention Is All You Need [16], Transformer architectures have gained significant traction in the field of Deep 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" [17] due to their ability to model long-range dependencies.

In the following section, the core components and mechanisms of the Transformer architecture are outlined.

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 expressed with:

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

[2] [16]

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. 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. [16]

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 $x_{\leq n}$ through x_n, but not to any $x_{>n}$.

[2] M^T represents the transposed matrix of M

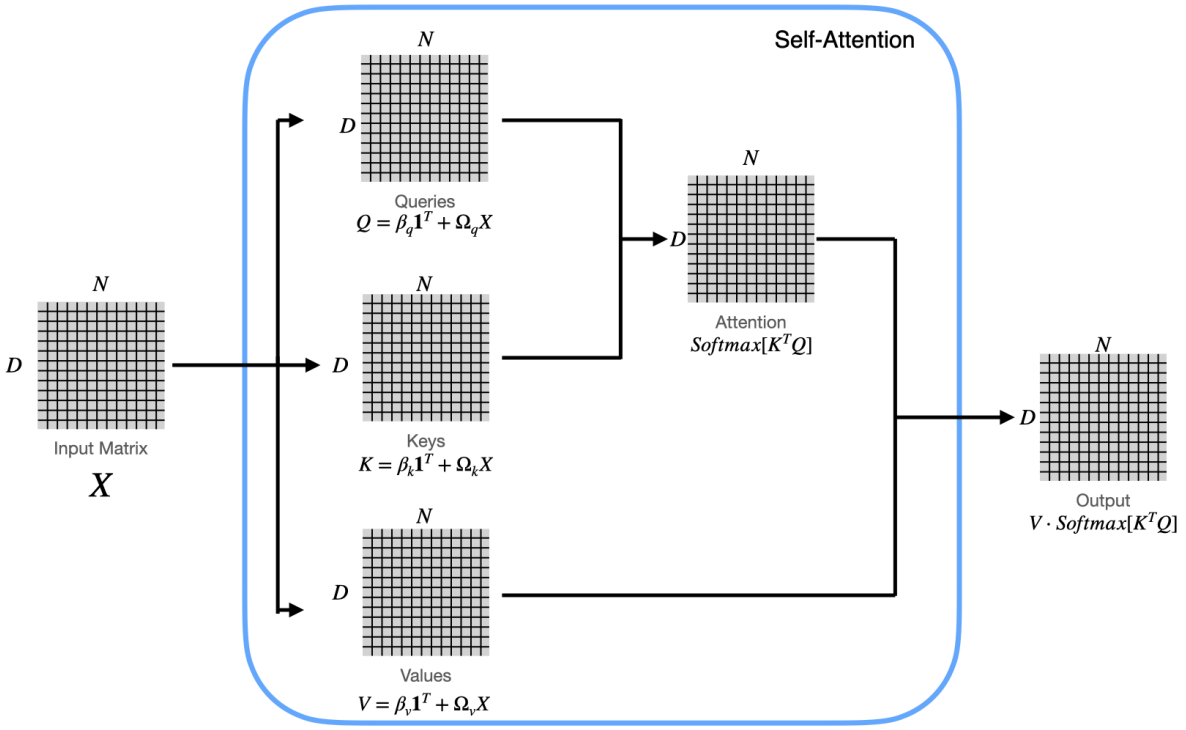

Figure 2.6 – Self-attention mechanism illustrated with matrices. All matrices have shape $D \cdot N$, where D is the sequence length and N is the feature dimension. The input matrix is projected into three separate matrices: Queries (Q), Keys (K), and Values (V). The attention weights are computed by multiplying Q with the transpose of K, followed by applying the Softmax function. The result is then used to weight the V matrix, producing the final output as $\text{Softmax}(QK^T) \cdot V$. [18]

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" [16] 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 $q < D$, where D is the original input size. Each set of projections corresponds to a separate attention head, allowing the model to focus on different semantic or temporal aspects of the sequence. These n 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 calculated using the following

equation, where W^O is a learnable output weight matrix, and W^Q, W^K, and W^V are learnable weight matrices corresponding to the matrices Q, K, and V, respectively:

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

[16]

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. [10]

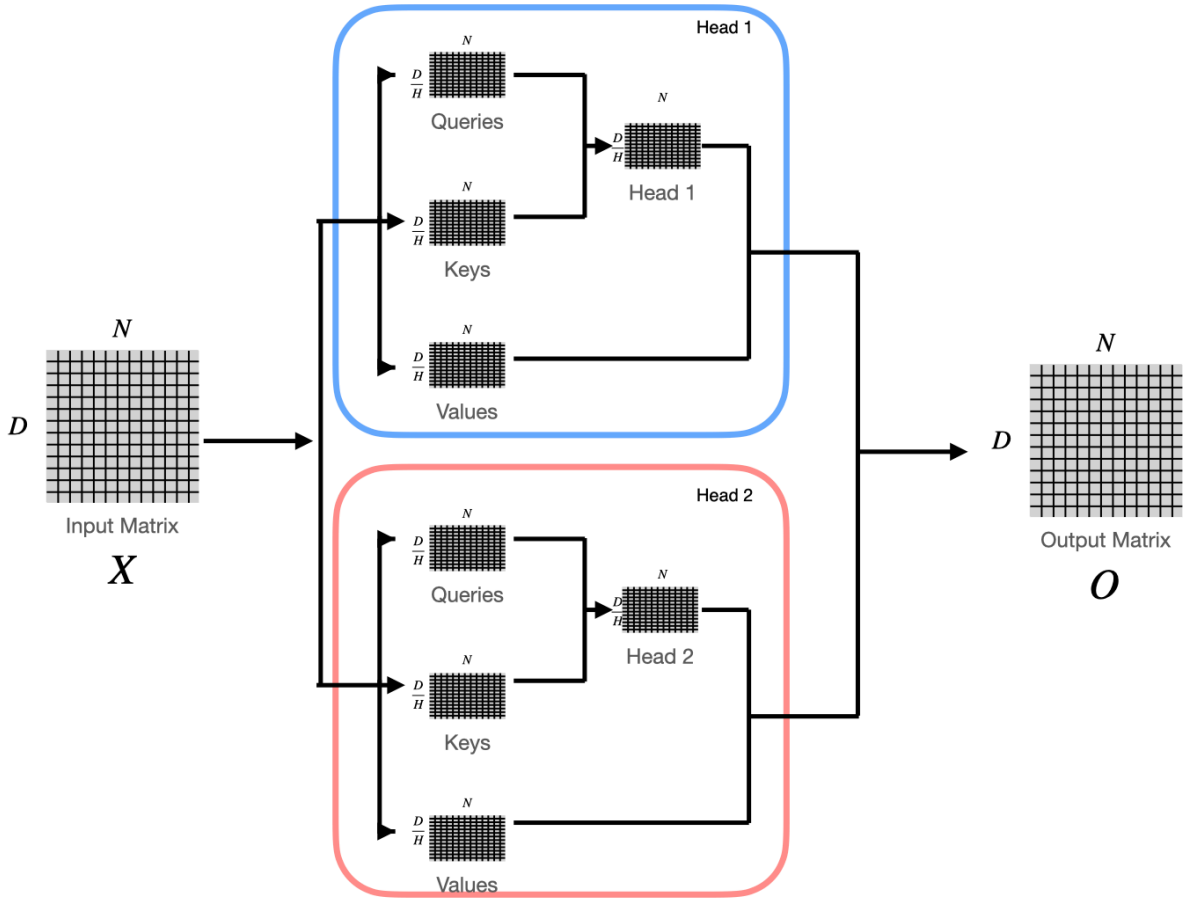

Figure 2.7 – Multi-head attention mechanism. The input matrix X of shape $D \times N$, is linearly projected into multiple sets of Queries, Keys, and Values. Each set defines an individual attention head (e.g., Head 1, Head 2), which independently computes scaled dot-product attention. The outputs from 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 $D \times N$. [18]

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. [19]. 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 a 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.

The positional encoding matrix P_i can either be fixed—using functions such as sine and cosine, as in Attention Is All You Need [16]—or it can be learned during training as part of the model parameters. 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 are based on an encoder–decoder architecture, as illustrated in Figure 2.8. The encoder processes the full input sequence and generates a contextual representation, known as the encoder vector (shown in grey). The decoder then uses this representation to generate the output sequence token by token. Both components consist of stacked layers with a shared modular structure, including multi-head attention, feedforward sub-layers, residual connections, and normalization.

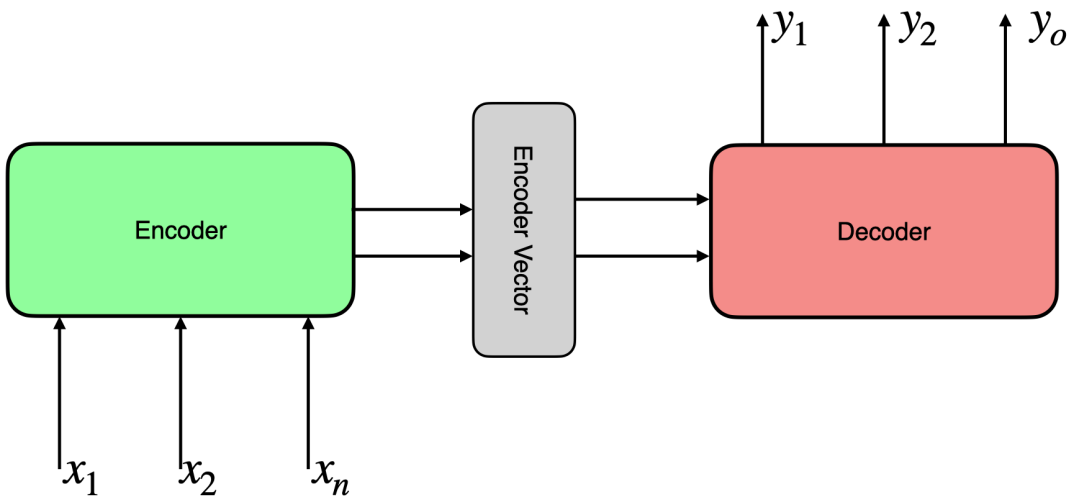

Figure 2.8 – High level illustration of the encoder–decoder architecture. The encoder receives an input sequence $x_1, ..., x_n$ and transforms it into a sequence of contextualized representations, here called the Encoder Vector, which is symbolically represented by the grey box. The encoder vector are passed to the decoder, which generates an output sequence $y_1, ..., y_o$, where the output length o may differ from the input length n.[10]

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. [10]

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.

[16], [20], [18]

3 Methodology

This chapter outlines the methodological approach used to design, implement, and evaluate the system developed in this project. It is divided into three main sections. First, we describe the data sources, preprocessing steps, and preparation techniques applied to support the deep learning experiments. Followed by presenting the architecture and implementation details of the deep learning models. Finally, we discuss the software engineering aspects of the project, including the design of the backend and frontend components, as well as the implementation of a continuous integration and deployment (CI/CD) pipeline.

3.1 Dataset

3.1.1 Data

The underlying data was provided by the **WSL**, with 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 **WSL**s 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.

[3] fall, slide, water/debris flow; the damage-causing process

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"[3], and "Schadensausmass", which were identified as the most relevant variables related to damage.

Based on the inputs of Liechti, the relevant meteorological variables were identified as sunshine duration, 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.
The temperature at 2 meters above ground (from [21]) was used, as it provides a more meaningful indication of potential snowmelt. In this context, the influence of frozen ground was considered less significant and therefore not explicitly taken into account.
Snowfall can contribute to snowmelt processes later in the seasonal cycle.
Rainfall directly contributes to the potential for flooding.

In a subsequent experiment, snowfall was found to have no significant short-term impact or correlation with the data and was therefore completely removed from the dataset.

3.1.2 Data Cleaning

The damage data referenced in Section 3.1.1 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 [22]. For approximately 300 out of 2759 municipalities, no coordinates could be retrieved. Manual analysis of these cases revealed the following recurring issues.

For some incidents, as described in Section 3.1.1, **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 occurrence, 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.

3.1.3 Availability of Sources and Data Collection

The data currently in use was collected with relatively little difficulty. The damage data was kindly provided by Liechti from the **WSL** following a formal request via email [23].

For the collection of weather data, the initial approach was to use official government data provided by MeteoSwiss [24]. 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 [21] was discovered. The open-meteo API is an open-source project that aggregates weather data from various national meteorological services [25]. To avoid excessive costs, a free academic access key was requested and kindly provided [26].

To obtain information on soil conditions, the first resource consulted was the Swiss federal geoportal map.geo.admin [27]. However, the format of the data was mostly incompatible with the tools available for this thesis. An alternative considered was the GIS Browser [28], which is the cantonal equivalent of map.geo.admin [27]. 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 [29], it was assumed that public institutions must maintain relevant geotechnical data. First, the building construction office of Affoltern am Albis was contacted [30]. 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 [31]. The contact person from the Office for Spatial Development [32] was likewise unable to provide relevant data or further contacts. Their suggestion was to consult the GIS Browser [28] or map.geo.admin [27]. After these repeated unsuccessful attempts, the GIS Helpdesk was contacted [33]. The proposed solution [34] was again to use the GIS Browser or map.geo.admin, which had already proven inadequate. Due to time constraints, efforts to retrieve soil data were ultimately discontinued.

3.1.4 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 3.1.1 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 [35] 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 reported, 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 [35]. 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

What stands out in particular is the uneven distribution. In addition to the imbalance between damage and no-damage cases, the distribution of damage severity itself is also highly skewed. As a reference, a distribution following a Poisson process would be expected.

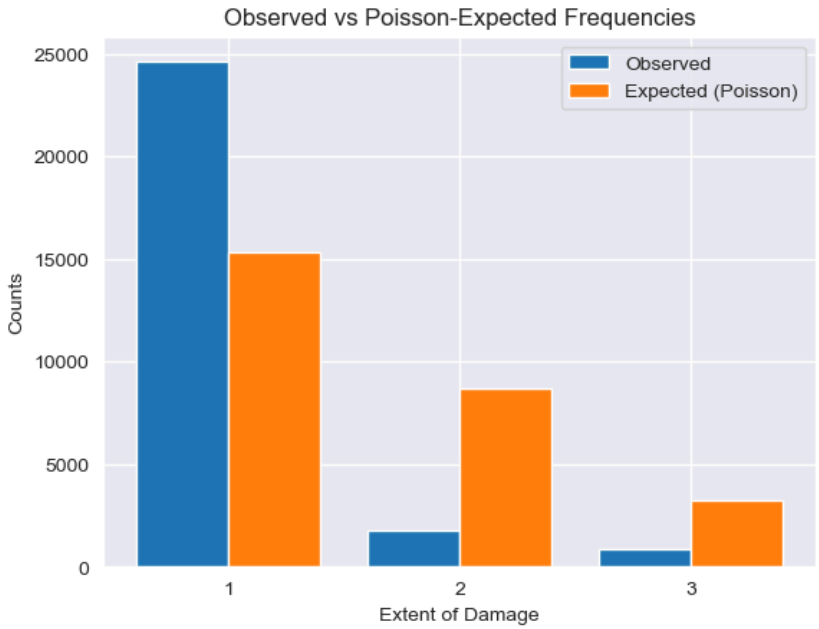

Figure 3.9 – distribution of damages compared to the expected Poisson distribution.

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 λ and longitude φ). Let $x_i = (\lambda_i, \varphi_i)$ be the coordinates for municipality i. The clustering objective was to minimize:

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

[36] where μ_j denotes the centroid of cluster j. This was implemented using the *KMeans* algorithim from SciKitLearn [37].

To ensure deterministic behavior of the *KMeans* algorithm from SciKitLearn [37], 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. [37] Figure 3.10 presents an illustrative example of the spatial clustering of all municipalities into $k = 6$ clusters.

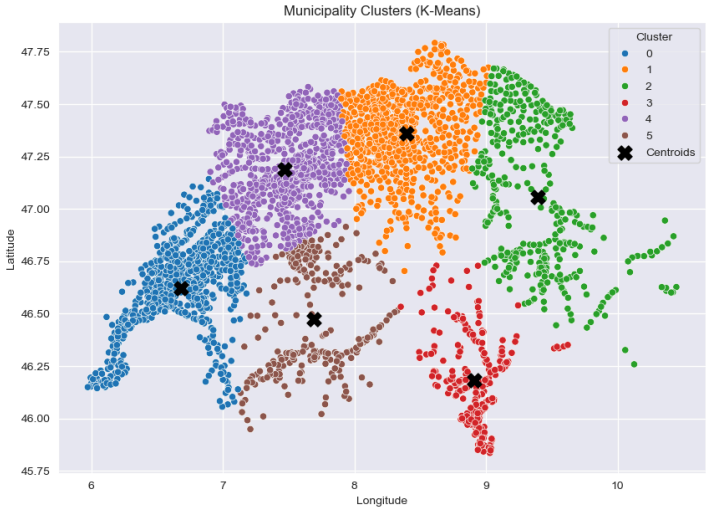

Figure 3.10 – Example clustering of all Swiss municiaplities with $k = 6$. The black crosses indicate the centroids of the respective clusters.

Determining the optimal number of clusters proved to be challenging, as no clear "elbow point" could be identified in the curve shown in Figure 3.11. 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. Furthermore, we will use a more finer granularity of $k = 26$.

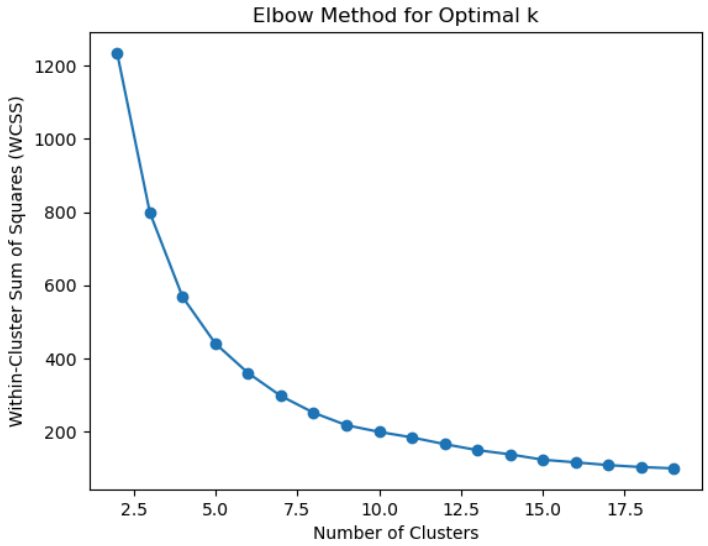

Figure 3.11 – Elbow plot showing the number of clusters on the x-axis and the corresponding within-cluster sum of squares (WCSS) on the y-axis

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 n time series, where $n = k$.

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:

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

$\text{MeanDamage}_{\text{class(day)}}$ 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:

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

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: $(0, T_{\text{low}}]$
- Class 2: $(T_{\text{low}}, T_{\text{mid}}]$
- Class 3: (T_{mid}, ∞)

Weeather Data Interpolation For each cluster and week, the corresponding weekly sum of rain, average temperature, and average sunshine duration were computed based on the weather at the cluster centroid. These values were then assigned to all municipalities within the respective cluster.

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 **FNN**, and compared their performance on a held-out test set. The result of the different models are discussed in Section 4.

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.

All features were normalized using Z-score normalization, defined as $Z = \frac{X-\mu}{\sigma}$ [38], where X denotes the value to be normalized. 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.

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
26	Train	52′376	2013–2023	4′590	2′222
26	Test	13′572	2013–2023	11′955	1′617

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

Training Pipeline

The complete training workflow is illustrated in Figure 3.13. 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 *BaseCross-Validator* [37]. This custom method, *ClusteredTimeSeriesSplit*, is shown in Figure 3.12. It ensures chronological consistency by keeping validation data strictly later in time than training data within each geographic cluster.

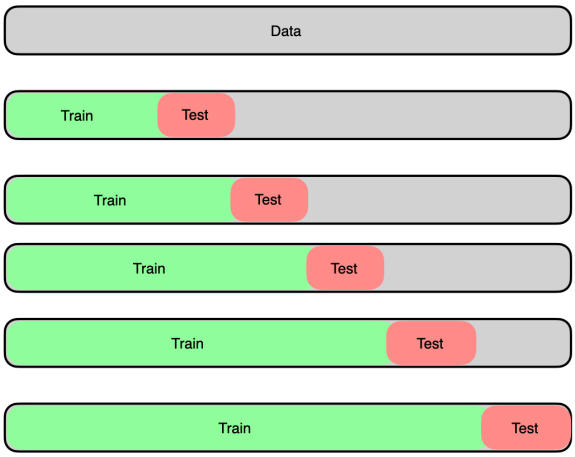

Figure 3.12 – Chronological 5-fold cross-validation. Each fold validates on a later time window, preserving the time series structure.

Within each fold, the model was trained over multiple epochs, which is shown in Figure 3.13 on the 4th line. We used the Adam optimizer [39], as it provides adaptive learning rate updates and has been shown to work well in practice for deep learning tasks. 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 3.13, 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.

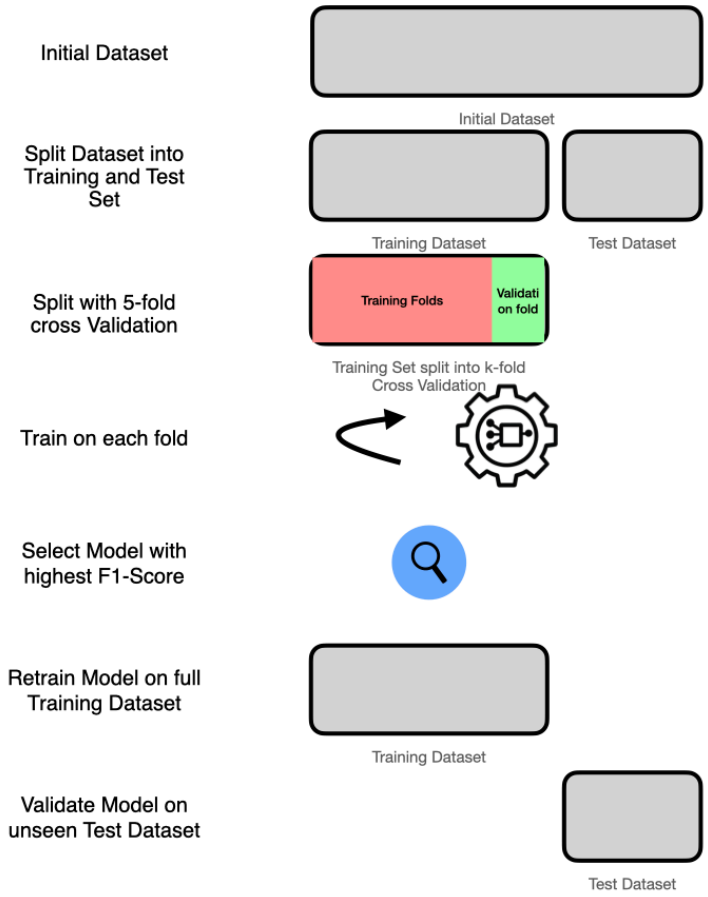

Figure 3.13 – End-to-end training pipeline, from dataset preparation through cross-validation and final testing.

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 3.1.1, 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.

3.2.2 Feedforeward Neural Network (FNN) based forecasting model

Our first experiment employed a baseline **FNN**, whose architecture is illustrated in Figure 3.14. 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 Rectified Linear Unit (**ReLU**) activation functions. This depth was chosen to for a sufficient level of non-linearity 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.

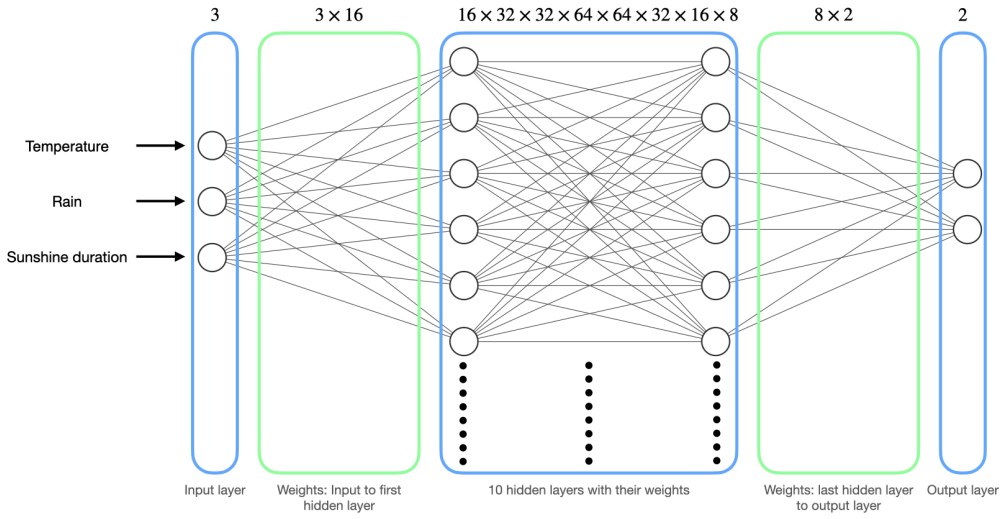

Figure 3.14 – Illustration of the **FNN**: 3 Input Neurons, 2 Output Neurons. 8 hidden layers with Neurons variing between 8 and 64 as shown in the illustration respectively.

3.2.3 LSTM 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 3.15 and consists of a stack of 10 **LSTM** layers followed by a fully connected output layer.

The LSTM block, shown in the middle of Figure 3.15, receives as input a multivariate time series of weather features, such as temperature, rainfall, and temperature, over a fixed sequence window.

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 [14].

The last layer of the model is a **FNN** 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 3.15. This architecture was chosen based on the findings of Steven Elsworht et. al in "Time Series Forecasting Using LSTM Networks: A Symbolic Approach" [40]. 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 historical time span.

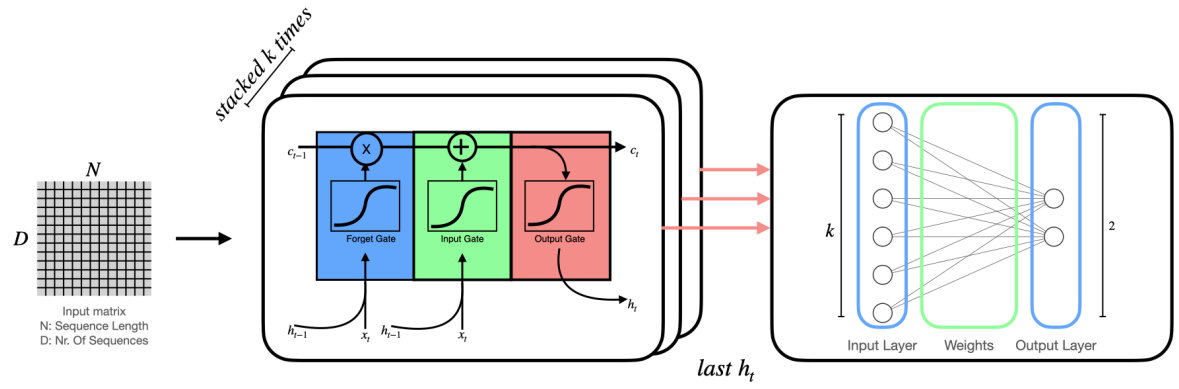

Figure 3.15 – Architecture of the LSTM-based forecasting model. The model consists of stacked LSTM layers followed by a fully connected output layer.

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 3.16 and is composed of a sequence embedding layer, positional encoding, stacked Transformer encoder layers, and a feedforward output layer. This results in an encoder-only Transformer architecture, similar to the one used in the well-

known language model BERT [41], which also addresses a classification task—though in the domain of textual data. Since our task is likewise a classification problem, we adopted a similar architecture.

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. [16]

The core of the architecture is a Transformer encoder block consisting of *6* stacked layers of multi-head self-attention and feedforward sub-layers. Here we used the built in Transformer module of the PyTorch Library [14]. 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 used in a self-contained manner, where the same input matrix x is used as both the query and context for attention. Since our task does not involve sequence generation, no autoregressive decoder is required.

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.

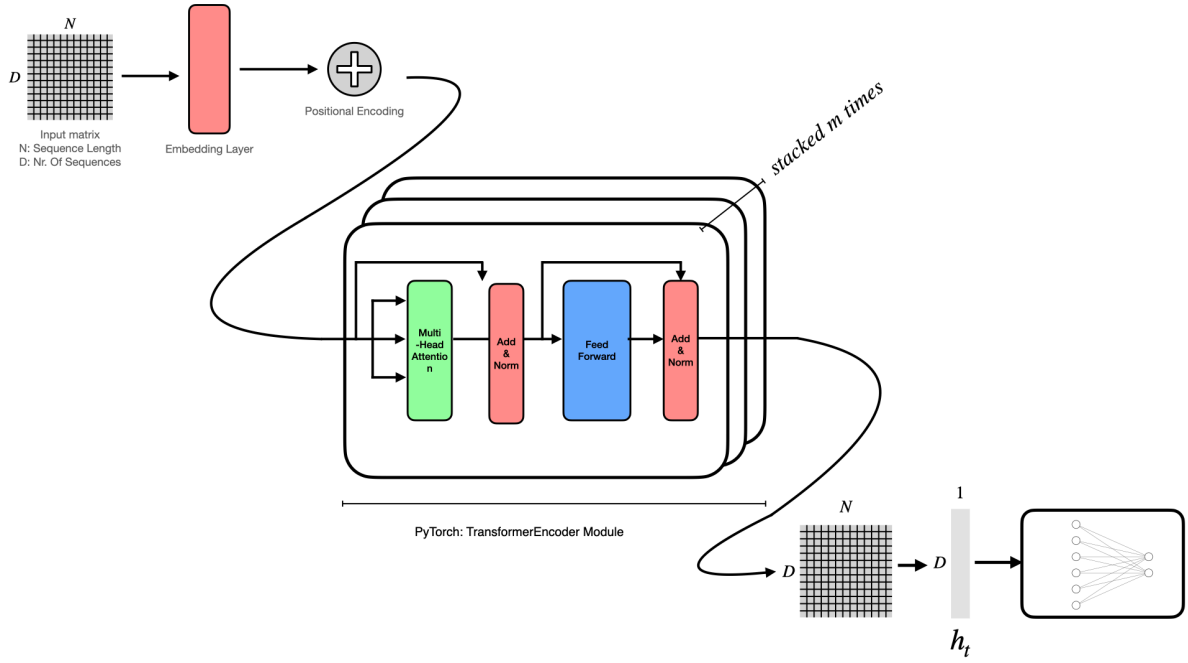

Figure 3.16 – Architecture of the Transformer-based forecasting model. The model includes input embeddings, positional encoding, stacked self-attention layers, and a feedforward output module.

3.2.5 Model Comparison Setup

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. [42]

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

Table 3.2 – Hyperparameter search space used during model sweeps.

In addition, to assess the impact of spatial aggregation on model performance, each architecture was trained using different cluster counts ($k \in \{3, 6, 26\}$), which determine the number of geographic regions derived from the spatial clustering process (see Section 3.1.4). 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

The backend was implemented in Java, a type-safe language. To accelerate development and reduce boilerplate code, we adopted the Spring Boot framework, which offers a convention-over-configuration paradigm and seamless integration with web, data, and security components.

Data persistence is handled by PostgreSQL, an open-source relational database.

To enable model inference within the backend, we integrated the Deep Java Library (**DJL**) [43]. DJL provides a high-level Java API for loading and running deep learning models, allowing seamless integration of our trained PyTorch models into the Spring Boot service. It also supports GPU acceleration via CUDA, significantly reducing inference latency on compatible hardware.

Architecture

We adopted the Clean Architecture pattern [44], visualized in Figure 3.17 to ensure a modular, maintainable, and testable codebase. This architecture clearly separates domain logic from external concerns and enforces a unidirectional dependency rule: inner layers must not depend on outer ones.

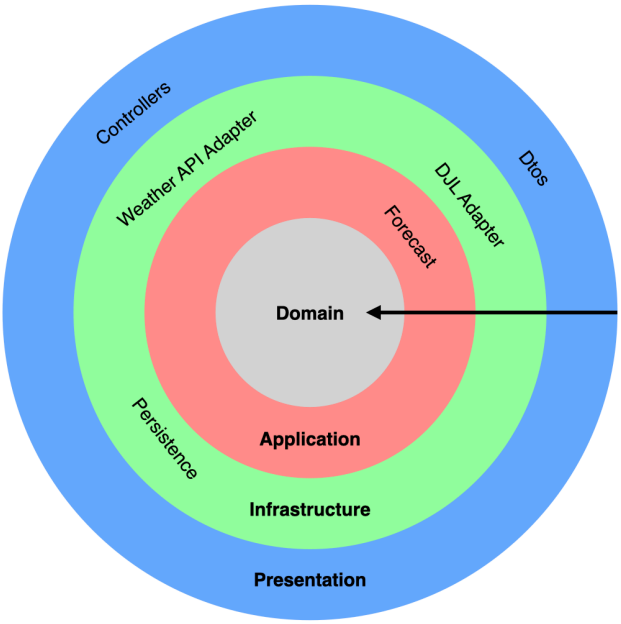

Figure 3.17 – Illustration of the applied Clean Architecture of the Backend

The architecture is organized as follows:

- **Domain Layer**: Encapsulates the core business entities and domain logic. It is entirely decoupled from technical concerns and external frameworks, as illustrated by the inner circle of the Figure 3.17. The following core entities have been defined:
  - Municipality: Represents a geographic administrative unit.
  - MunicipalityToCluster: Maps each municipality to its corresponding cluster, based on the chosen number of clusters k.
  - Damage: Represents a damage event recorded in the **WSL** database. It is also persisted in our PostgreSQL database.
  - GroupedDamage: Aggregated damage records grouped by municipality. This entity is materialized as a view in the PostgreSQL database.
  - Inference: Encapsulates the input data required for making predictions using the deep learning models.
  - Forecaset: Represents the output produced by a deep learning model.
  - WeatherData: eteorological data required for performing inference with the deep learning models.

- **Application Layer**: Defines the system's use cases and orchestrates business rules by coordinating entities. It is responsible for implementing application-specific logic while remaining independent of external technologies. This layer also defines interfaces, named *Ports* that describe the required functionality from the infrastructure layer.

A central part of this layer is the inference orchestration logic, which involves multiple sequential steps. To manage this complexity, we adopted the Chain of Responsibility design pattern [45]. This allows each step in the inference process to be encapsulated in a dedicated handler that can pass the request along the chain. The chain is hold by the *ForecastService* class, which is shown on the bottom left corner of Figure 3.18.

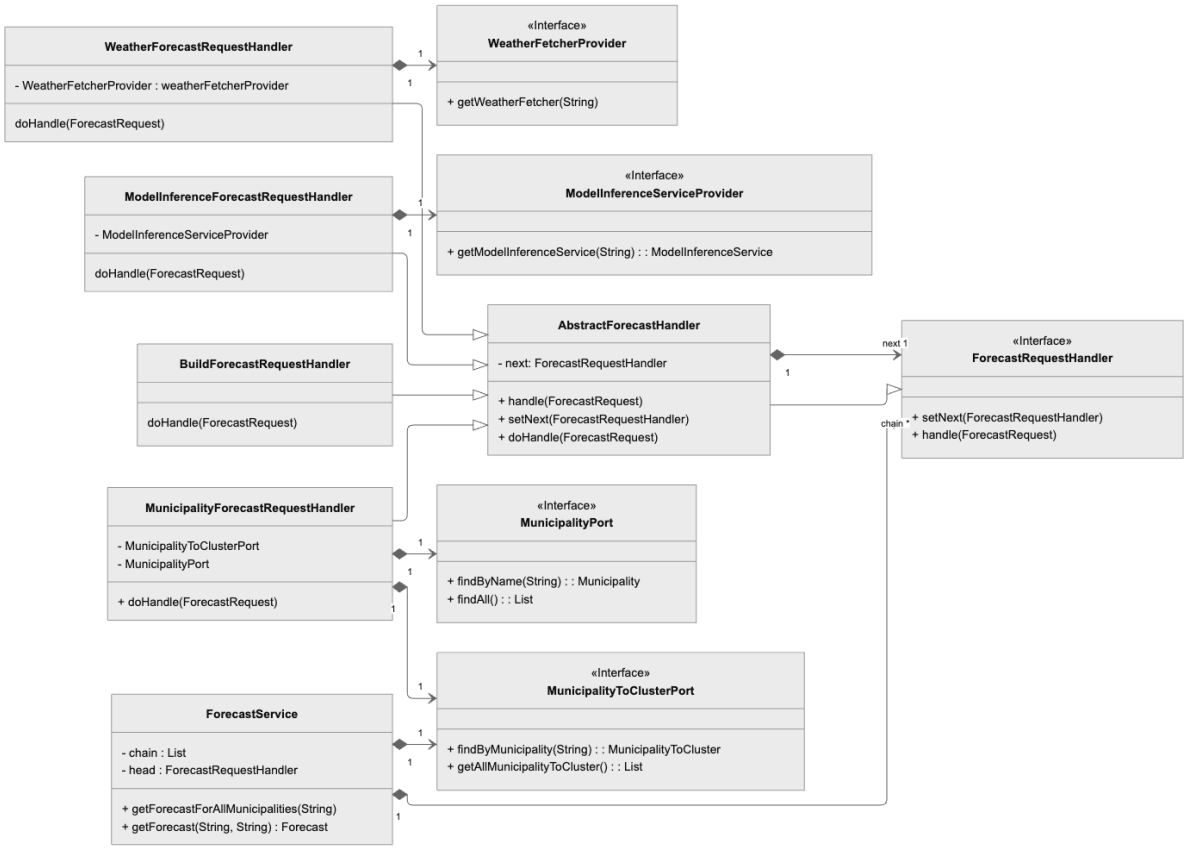

Figure 3.18 – Class Diagram for Application Layer of the Backend

- **Infrastructure Layer**: The Infrastructure Layer provides concrete implementations of the interfaces (*Ports*) defined in the Application Layer. It is responsible for «integrating external systems and technologies, such as:
  - PostgreSQL, using Spring Data JPA for data persistence,
  - the Deep Java Library (DJL) for running deep learning model inference,
  - and Open-Meteo APIs for weather data retrieval.

  This layer encapsulates all technical details and external dependencies, keeping the rest of the system decoupled from implementation concerns.

  To support modular weather data retrieval, the Factory Pattern is employed in the *OpenMeteoWeatherFetcherFactory*. This allows dynamic instantiation of the appropriate *WeatherFetcher* implementation based on the request context.

  In addition, the Decorator Pattern is used to compose weather fetchers with different temporal scopes: The base fetcher retrieves current-week data. It is then wrapped by decorators to add previous-month and previous-year data, respectively, forming a flexible and extensible weather data pipeline.

  Persistence adapters implement the required interfaces by delegating to Spring Data JPA repositories. These adapters act as bridges between the domain model and the database, handling entity retrieval and storage.

  The *ModelInferenceServiceFactory* uses a simple factory mechanism to return the appropriate model-specific inference service (e.g., FNN, LSTM, Transformer) depending on the requested type.

  The most important infrastructure classes are illustrated in Figure 3.19

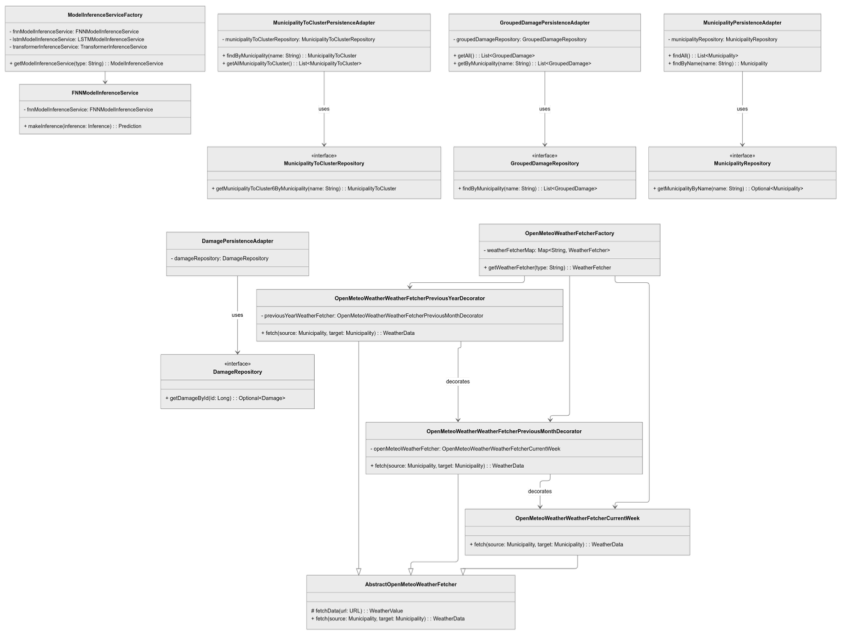

Figure 3.19 – Class Diagram for Infrastructure Layer of the Backend

- **Presentation Layer**: Exposes the application's functionality to external clients via RESTful HTTP APIs, implemented using Spring MVC. It is responsible for handling incoming HTTP requests, delegating execution to the appropriate application services or adapters, and formatting responses using Data Transfer Objects (DTOs).

  Each controller corresponds to a specific use case or domain concept:

  - *DamageController* manages endpoints for recording and retrieving individual damage events.
  - *GroupedDamageController* provides access to aggregated damage data grouped by municipality.
  - *ForecastController* serves endpoints for requesting deep learning model forecasts, either for all municipalities or a specific one.

This separation of concerns enhances testability and makes it straightforward to substitute components (e.g., switch databases) without affecting core logic.

Caching:

After the initial deployment, user tests were conducted as outlined in the frontend test concept chapter. During these tests, a noticeable and user-unfriendly delay was observed. The issue was traced back to the request responsible for retrieving forecast data for all municipalities in Switzerland, which depends on the open-meteo API [21]. Since this request must aggregate weather data across a large number of locations, it involves considerable processing time.

To better assess the performance characteristics of this request, response times were measured and compared under different conditions. The following tables present both the individual request durations and the corresponding average values across varying caching and deployment scenarios.

request #	local cached	local uncached	production cached
1	54.69 s	55.34 s	133 ms
2	20 ms	52.75 s	57 ms
3	19 ms	52.39 s	52 ms
4	16 ms	49.41 s	56 ms
5	12 ms	49.74 s	35 ms
6	19 ms	54.50 s	34 ms
7	14 ms	52.75 s	76 ms
8	17 ms	54.00 s	43 ms
9	16 ms	52.01 s	44 ms
10	14 ms	51.46 s	35 ms

Table 3.3 – Comparison of forecast retrieval times for all municipalities: locally with caching, locally without caching, and via the production API with caching.
Note: For caching scenarios, the initial request populates the cache and thus exhibits similar latency to uncached retrieval. The cache is valid for 24 hours. In the case of the production system, the initial daily request had already been completed, resulting in no noticeable difference between the first and subsequent requests.

averages	local cached	local uncached	production cached
ø	$16.\overline{3}$ ms	$52'112.\overline{2}$ ms	48 ms

Table 3.4 – Average request times
Note: To account for cache warm-up, only requests two to ten were included in the calculation of the average request time.

Test Concept

All technically relevant logic components from the application package are covered by unit tests. These tests verify the behavior of each class in isolation from external dependencies by using mocks or stubs. The goal was to achieve a test coverage of 80% for these classes, ensuring correct handling of inputs, states, and error scenarios.

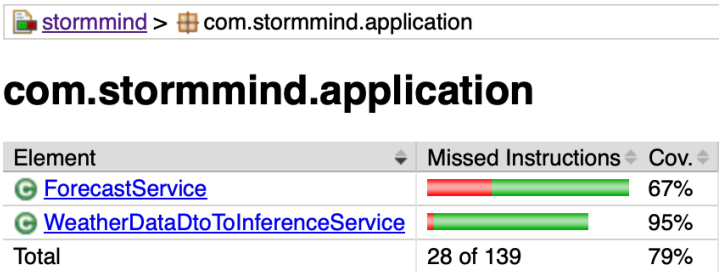

Figure 3.20 – Test coverage analysis of the application package using JaCoCo

3.3.2 Frontend

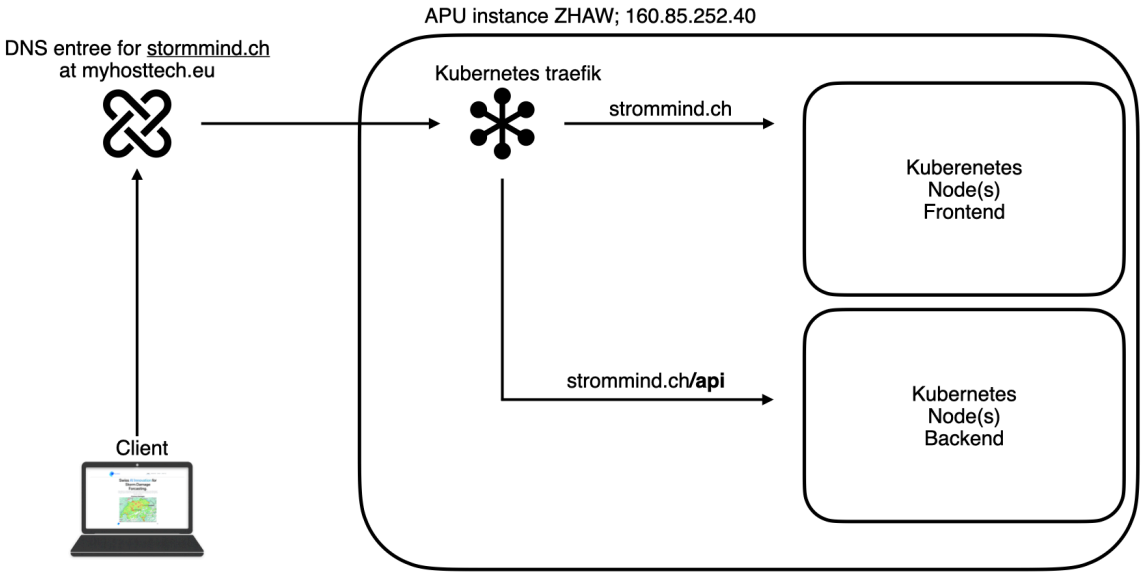

Figure 3.21 – web routing[4]

Technologies

The frontend of the application is implemented using **React** and structured as a separate repository based on the **Vite** build tool. It follows a modular and maintainable architecture, distinguishing clearly between application logic and user interface components.

Routing is handled on the client side, and the overall structure aligns with modern single-page application principles. The development setup emphasizes performance, scalability, and a clear separation of concerns.

Test Concept

Given the small scope of the frontend, automated testing was not conducted. Functional correctness was instead ensured through manual testing during development.

[4]laptop generated with chatgpt (prompt: “erstelle mir ein png eines minimalistischen laptops ohne hintergrund”)

3.3.3 CI/CD and Deployment

Deployment is managed via a virtual instance hosted on the **OpenStack** cluster of the ZHAW [46]. DNS configuration was performed using **Hosttech**, directing traffic to the appropriate backend infrastructure.

The application consists of two components: a backend and a frontend. Both are containerized using Docker and deployed on a Kubernetes cluster. The use of Kubernetes enables dynamic scalability, allowing the application to adapt to changing resource demands.

The frontend runs on port 80 within its pod and communicates over TCP. To manage and route incoming traffic, **Traefik** is used as an ingress controller. It is configured to forward requests to *stormmind.ch*, including the root path / and all subpaths, to the frontend pod. Requests to *api.stormmind.ch* are routed to the backend service, which listens on port 8080. All HTTP traffic is automatically redirected to HTTPS to ensure secure communication.

Both the backend and frontend deployments are configured to always pull the latest Docker image and to retain only one previous ReplicaSet. This setup simplifies the deployment process and reduces potential ambiguity when identifying the active version.

To enable HTTPS functionality, a *ClusterIssuer* and a *Certificate* resource were configured within the Kubernetes cluster. These components automatically request and manage TLS certificates from **Let's Encrypt**, making them available to Traefik for encrypted traffic handling.

For continuous integration and deployment, two separate GitHub Actions pipelines were implemented—one for the frontend and one for the backend. Both workflows are triggered on each push to the *main* branch. The pipelines establish an SSH connection to the machine running the Kubernetes cluster, pull the latest project state, build new Docker images, push them to Docker Hub, and trigger a rollout of the updated containers.

Given the moderate size of the project, this CI/CD approach was deliberately chosen over the integration of additional DevOps tools in order to keep operational overhead low.

Argo CD is used to monitor the state of the Kubernetes cluster and manage application deployments, including version control and scaling.

4 Results

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:

1. Which architecture achieves the best performance on the held out test set?
2. 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**?
3. How does spatial granularity, implemented through cluster sizes $k \in \{3, 6, 26\}$, 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 3.1.1. 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. It is explicitly stated that the constructed clusters do not precisely correspond to the canton borders, as municipalities are assigned to clusters using the K-means algorithm, as detailed in Section 3.1.4.

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" [47]. 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 4.5. This metric is particularly appropriate given the class imbalance described in Section 3.1.1, 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

Clusters	3	6	26
FNN	$0.67 \pm 9.5e{-6}$	$0.67 \pm 9.9e{-6}$	$0.67 \pm 3.7e{-7}$
LSTM	$0.67 \pm 2.2e{-6}$	$0.65 \pm 3e{-7}$	
Transformer	$0.68 \pm 1.4e{-6}$	$0.67 \pm 7.9e{-8}$	

Table 4.5 – 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.

4.1 Feedforeward Neural Networks (FNNs): Results

In this section, the in dept results of the **FNN** model are reported. As shown in Table 4.6, the models was relativly stable across the different cluster sizes. Although accuracy improved with finer granularity, this may be attributed to increased class imbalance, which can artificially inflate accuracy by favoring the majority class.

Cluster	3	6	26
Accuracy	$67.55\% \pm 0.35$	$70.68\% \pm 0.56$	$68.71\% \pm 0.02$
AUC	$0.71 \pm 2.2e{-5}$	$0.72 \pm 5.5e{-5}$	$0.72 \pm 2.9e{-6}$
F1	$0.66 \pm 3.6e{-5}$	$0.67 \pm 9.9e{-6}$	$0.67 \pm 3.7e{-7}$
Precision	$0.68 \pm 3.8e{-5}$	$0.66 \pm 6.7e{-6}$	$0.67 \pm 7.7e{-8}$
Specificity	$0.66 \pm 3.5e{-5}$	$0.68 \pm 3.3e{-5}$	$0.67 \pm 4e{-6}$

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

4.2 Long Short-Term Memory Neural Network (LSTM): Results

Cluster	3	6	26
Accuracy	$68.41\% \pm 0.02$	69.25 ± 0.44	
AUC	$0.71 \pm 2.4e{-8}$	$0.71 \pm 1.9e{-05}$	
F1	$0.67 \pm 2.2e{-6}$	$0.65 \pm 3e{-7}$	
Precision	$0.68 \pm 9.1e{-6}$	$0.65 \pm 1.6e{-6}$	
Specificity	$0.67 \pm 1.5e{-6}$	$0.67 \pm 3.6e{-05}$	

Table 4.7 – Performance Metrics of the LSTM model. We provide the mean over the top 3 of the 20 runs and the variance.

4.3 Transformer: Results

Cluster	3	6	26
Accuracy	$68.14\% \pm 0.00$	$70.25\% \pm 0.00$	
AUC	$0.72 \pm 9.6e{-5}$	$0.74 \pm 6.2e{-6}$	
F1	$0.68 \pm 1.4e{-6}$	$0.67 \pm 7.9e{-8}$	
Precision	$0.68 \pm 4.6e{-7}$	$0.66 \pm 9.5e{-8}$	
Specificity	$0.67 \pm 2.9e{-6}$	$0.69 \pm 1.8e{-6}$	

Table 4.8 – Performance Metrics of the Transformer model. We provide the mean over the top 3 of the 20 runs and the variance

4.4 Key Findings and Interpretation

This section summarizes the key findings in direct response to the research questions outlined at the beginning of the results chapter.

1. The Transformer consistently achieved the highest macro-averaged F1 scores across the cluster configurations $k \in \{3, 6\}$. While the performance gains were modest, the Transformer outperformed both the FNN and the LSTM models, indicating its superior capability in capturing relevant patterns for storm damage forecasting.
2. Increased complexity did not uniformly translate into better performance. The LSTM, although more complex than FNN, performed worse in several configurations. In contrast, the Transformer —which introduces even greater complexity through self-attention mechanisms—demonstrated slight but consistent improvements over both FNN and LSTM. This suggests that not all forms

of complexity are equally beneficial, and that architectural innovations like attention may offer more value than merely increasing parameter count or depth.

3. Model performance tended to decline slightly as the number of spatial clusters increased. This may be attributed to increased class imbalance at finer granularities, where the proportion of "no-damage" samples rises, making storm damage harder to detect.

In summary, while the Transformer emerged as the most effective model, its advantage over the simpler **FNN** was relatively small. This makes the **FNN** a strong candidate for practical applications requiring lower computational overhead. At the same time, the success of the Transformer supports further exploration of self-attention mechanisms for this forecasting task.

5 Discussion and Outlook

This thesis presents a first step toward deep learning-based storm damage forecasting. Although the evaluated models—**FNN**, **LSTM**, and Transformer—are not yet suitable for real-world deployment, their performance marks an important foundation for future research. The planned demonstration and expert assessment by Liechti, could not be conducted due to her unavailability during the project's final phase. The final version of this work will be submitted to her for review, with a concluding summary published on *stormmind.ch*.

All models were trained solely on weather data, yet storm damage depends on a complex interplay of environmental, infrastructural, and geographical factors, many of which were not included in the current modeling approach. This shortcoming points to a key opportunity for future research: incorporating additional data sources such as vegetation cover, infrastructure types, topography, and soil saturation. We believe that with an expanded feature set and increased domain knowledge, the predictive accuracy of these models can be significantly enhanced. As discussed in Section 1.1, major industry players are actively pursuing comparable solutions, underscoring both the relevance and the inherent difficulty of this problem. It is our hope that the methods and insights presented in this thesis will provide a meaningful foundation for future work in this area. In particular, we see promising potential for collaboration with **WSL**, whose datasets and expertise could substantially support continued development. Further data collection and careful feature engineering will be critical in identifying the most predictive variables for storm damage forecasting.

- List of Abbreviations

DJL	Deep Java Library	**RNN**	Recurrent Neural Network
FNN	Feedforeward Neural Network	**ReLU**	Rectified Linear Unit
LSTM	Long Short-Term Memory Neural Network	**VGP**	Vanishing Gradient Problem
NN	Neural Network	**WANDB**	Weights & Biases
NatCat	Natural Catastrophes	**WSL**	Swiss Federal Institute for Forest Snow and Landscape Research WSL
RDA	Rapid Damage Assessment		

Bibliography

[1] "Swiss Re Rapid Damage Assessment | Swiss Re." Accessed: Jun. 03, 2025. [Online]. Available: https://www.swissre.com/reinsurance/property-and-casualty/solutions/property-solutions/rapid-damage-assessment.html

[2] "RAvaFcast - Eine Künstliche Intelligenz Zur Vorhersage Der Regionalen Lawinen-Gefahrenstufe in Der Schweiz - Opendata.Swiss." Accessed: Jun. 03, 2025. [Online]. Available: https://opendata.swiss/en/showcase/ravafcast-eine-kunstliche-intelligenz-zur-vorhersage-der-regionalen-lawinen-gefahrenstufe-in-de

[3] A. Maissen, F. Techel, and M. Volpi, "A Three-Stage Model Pipeline Predicting Regional Avalanche Danger in Switzerland (RAvaFcast v1.0.0): A Decision-Support Tool for Operational Avalanche Forecasting," *Geoscientific Model Development*, vol. 17, no. 21, pp. 7569–7593, Oct. 2024, doi: 10.5194/gmd-17-7569-2024.

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

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

[6] "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

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

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

[9] "NN SVG." Accessed: May 03, 2025. [Online]. Available: http://alexlenail.me/NN-SVG/

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

[11] I. Mrázová, "Multi-Layered Neural Networks," Nov. 2024.

[12] F.-P. Schilling and Z. Cai, "Lecture 05: Sequential Models," Mar. 19, 2025.

[13] 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.

[14] "PyTorch Foundation." Accessed: May 06, 2025. [Online]. Available: https://pytorch.org/

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

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

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

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

[19] 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

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

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

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

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

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

[25] "👋 About | Open-Meteo.Com." Accessed: Jun. 03, 2025. [Online]. Available: https://open-meteo.com/en/about

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

[27] "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.swisstlm

3d-wanderwege,f;ch.vbs.schiessanzeigen,f;ch.astra.wanderland-sperrungen_umleitungen,f&bgLayer=ch.swisstopo.pixelkarte-farbe°

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

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

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

[31] "phone call Hochbau Amt Zürich." Aug. 04, 2025.

[32] "phone call Amt für Raum Entwicklung und Vermessung." Aug. 04, 2025.

[33] D. Ueltschi, "Bodenbeschaffenheitskarte."

[34] F. GIS, "[ARE-JIRA] GIS-2262 [EXTERN] Bodenbeschaffenheitskarte."

[35] *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°

[36] "2.3. Clustering." Accessed: May 13, 2025. [Online]. Available: https://scikit-learn/stable/modules/clustering.html°

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

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

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

[40] 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°

[41] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." Accessed: Jun. 05, 2025. [Online]. Available: http://arxiv.org/abs/1810.04805°

[42] "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°

[43] "DJL - Deep Java Library." Accessed: Jun. 03, 2025. [Online]. Available: https://djl.ai/°

[44] R. C. Martin and R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. in Robert C. Martin Series. London, England: Prentice Hall, 2018.

[45] "Chain of Responsibility." Accessed: Jun. 04, 2025. [Online]. Available: https://refactoring.guru/design-patterns/chain-of-responsibility°

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

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

[48] S. F. R. I. WSL, "Infos_Daten_Unwetterschadensdatenbank_WSL_english." 2023.

List of Figures

[5]laptop generated with chatgpt (prompt: “erstelle mir ein png eines minimalistischen laptops ohne hintergrund”)

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.“[48]

A.2 Original Data Features

Gemeinde, Gemeindenummer, Weitere Gemeinde, Kanton, Prozessraum, MAXO Datum, Datum, MAXO Zeit, Zeit, Gewässer, Weitere Gewässer, Hauptprozess, Hauptprozess Rutschung Un-

terteilung, 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