# Predicting Events of Hypoglycemia

## A Comparison of Long Short-Term Memory and Graph Attention Network Based Approaches

Bachelor Thesis

Faculty of Science, University of Bern


submitted by

**Fabian Hüni**

from Bern, Switzerland



Supervision:

PD Dr. Kaspar Riesen

Institute of Computer Science (INF)

University of Bern, Switzerland

**Abstract**

Diabetes is a condition that disrupts the regulation of blood glucose levels, leading to both high, known as hyperglycemia, and low, known as hypoglycemia, glucose levels. Hypoglycemia, specifically, can lead to cognitive impairment, seizures, and loss of consciousness. Therefore, predicting and avoiding hypoglycemia events is crucial to enhance the quality of life for patients. With the aid of machine learning and the widespread use of continuous glucose monitoring devices, intelligent systems are being developed to predict hypoglycemia events in advance. This thesis implements and evaluates both a Long Short-Term Memory (LSTM) model and a Graph Attention Network (GAT) on a real-world dataset comprising 37 patients with type 1 diabetes. While several studies have explored LSTM models for hypoglycemia prediction, this thesis introduces the novel application of Graph Neural Networks. To enable training of the GAT on continuous glucose monitoring measurements, the time series data is transformed into graphs using the concept of visibility graphs. This study compares the LSTM and GAT architectures in terms of their performance and stability across all datasets, considering different window sizes and prediction horizons. The LSTM excels in short-term predictions (15 minutes), while the GAT performs better for longer horizons (30 and 60 minutes). The GAT demonstrates stable performance across all horizons, whereas the LSTM's performance declines with longer horizons. Additionally, the GAT shows lower variability across different datasets compared to the LSTM. These findings suggest that Graph Neural Networks may offer superior performance in long-term hypoglycemia event prediction and should be investigated further to potentially improve the quality of life of patients suffering from diabetes.

# Acknowledgements

I would like to express my gratitude to my supervisor, PD Dr. Kaspar Riesen, for his time, trust, advice and support throughout this thesis. Additionally, I would like to extend a special thank you to Sarah Waber for her valuable medical advice regarding diabetes, my family and Dr. Andreas Ritter for their insightful feedback on the initial drafts, and the team at the Data Science Lab of the University of Bern for their patient assistance with issues related to TensorFlow and UBELIX. Last but not least, I would also like to thank my employer inpeek for allowing me to use the office infrastructure at any time during my bachelor thesis.

Calculations were performed on UBELIX (https://www.id.unibe.ch/hpc), the HPC cluster at the University of Bern.

# Contents

# Chapter 1

# Introduction

Diabetes mellitus is a disease characterized by hyperglycemia due to disorders in insulin secretion or insulin action. For the purpose of this work, diabetes can be divided into two broad categories: type 1 and type 2. A more detailed introduction can be found in the research literature on diabetes. In type 1 diabetes, the body does not produce insulin, leading to an absence of insulin in the blood. Conversely, in type 2 diabetes, patients are resistant to insulin and exhibit an inadequate compensatory insulin secretory response. The large majority of patients are affected by type 2 diabetes [1]. Type 1 diabetes is treated by injecting exogenous insulin to replace the missing endogenous insulin doses [2]. Insulin is necessary for muscles to absorb glucose from the blood. Without sufficient insulin, the blood glucose level can become too high, leading to hyperglycemia. Conversely, if too much insulin is administered, the blood glucose level can drop too low, resulting in hypoglycemia. The glucose concentration is measured in mg/dl and a glucose level below 70 mg/dl is considered dangerous for diabetes patients [3]. In the case of hypoglycemia, the symptoms can range from headaches and sweating to difficulty concentrating, unconsciousness, or even death.

Continuous Glucose Monitoring (CGM) devices support diabetes patients by providing real-time information about blood glucose levels. These devices are widely used and offer reliable measurements [4]. To aid patients in managing their health conditions effectively, it is essential to predict future hyper- and hypoglycemia events. The availability of CGM data empowers the development of intelligent systems capable of identifying patterns and forecasting blood glucose levels. One crucial application involves predicting precise glucose levels, while another one focuses on anticipating whether a patient is at risk of experiencing hyper- or hypoglycemia within specific time intervals. The prediction and detection of hypoglycemia events are crucial for patients to take timely actions to mitigate the consequences of such

events. For example, hypoglycemia events that occur during sleep, known as nocturnal hypoglycemia, can be particularly dangerous because individuals may not immediately perceive the symptoms [5]. A system that predicts a nocturnal hypoglycemia event in the evening would allow individuals to take preventive measures before going to bed, thereby reducing the risk of experiencing a hypoglycemic episode during the night [6].

This thesis focuses on the task of hypoglycemia event prediction and utilize methods from the field of Artificial Intelligence (AI). AI aims to enable machines to perform tasks that typically require human intelligence. In a broader sense, AI encompasses the research theories, methodologies, technologies, and applications that simulate, extend, and enhance human intelligence. Applications of AI span various domains, including Natural Language Processing, Computer Vision, and Machine Learning (ML). [7]

In ML, the objective is to train a system to learn from training data and subsequently use this acquired knowledge to solve similar tasks. A subfield of ML is deep learning, which employs multiple layers of Artificial Neural Networks (ANN). Deep learning models frequently outperform traditional ML models and conventional data analysis approaches. Tasks in ML and deep learning are generally classified into two categories: regression, which involves predicting numerical values, and classification, where the aim is to predict categorical outcomes. [8]

A classification task can be distinguished between multiclass classification and binary classification. For instance, the classification of images into one of several categories (*e.g.* food, vehicles, people) is a multiclass classification task, while predicting whether an X-ray image shows a broken bone or not is a binary classification task. This thesis focus on a binary classification task to determine whether the blood glucose value will fall below a given threshold within a certain future time frame.

In this work, the binary classification task is tackled with a Long Short-Term Memory (LSTM) model and a Graph Attention Network (GAT) model. The LSTM model is capable of operating on time series data, effectively handling both long- and short-term dependencies within the data. In contrast, the GAT model, a type of Graph Neural Network (GNN), operates on graph-structured data and leverages attention mechanisms to assign varying levels of importance to different nodes.

With the availability of real-time CGM data and the rapid advancement in deep

learning, new opportunities for hypoglycemia prediction can be explored. While LSTM models have already been studied, as for example by Iacono *et al.* [9], there has been so far no widely published research using GNNs and specifically GATs for this purpose. To address this gap, this thesis implements both LSTM and GAT-based approaches and compares their performance. Three different LSTM architectures are evaluated while for the GAT one variant with and another one without edge features is tested. For each model, different window sizes and prediction horizons are considered and evaluated to test the effect of the window size and prediction horizon on the performance. The window size determines the number of consecutive time stamps from the CGM time series that are considered together for one prediction. Prediction horizon, on the other hand, refers to the number of time stamps into the future for which a prediction is made. Both the window size and the prediction horizon can be expressed in terms of the number of included time stamps and the time span they cover. For instance, six time stamps correspond to a 30-minute time span, as the data is collected at five-minute intervals.
The objective of this work can be summarized into the following research question: How do GAT compare to LSTM models in predicting hypoglycemia events using only CGM data and what is the impact of different window sizes and prediction horizons on their performance?

This thesis is structured as follows. Chapter 2 provides the theoretical foundation of the core methods and reviews the current research landscape in hypoglycemia prediction. Chapter 3 outlines the CGM datasets utilized, describes the labeling process of samples, and explains the transformation of time series data into graphs. Chapter 4 comprehensively presents the conducted experiments and discuses the results. Finally, Chapter 5 concludes the thesis by summarizing the findings, discussing key insights, and proposing ideas for future research.

# Chapter 2

# Theory & Background

This chapter situates the thesis within the theoretical context. Section 2.1 provides an overview of the fundamentals of ANNs and then presents LSTMs as one of two different architectures used in this work. Section 2.2 introduces the fundamentals of basic graph theory and demonstrates how neural networks can be applied to graph structured data. As the second architecture the GAT is introduced. Finally, Section 2.3 presents an overview of current research in the field of hypoglycemia prediction.

## 2.1 Artificial Neural Networks

ANNs are a subfield of artificial intelligence (AI) and the basic building blocks for deep neural networks. ANNs are inspired by the structure of the human brain and consist of three layers. The input layer contains a node for each input variable, while the output layer consists of one or multiple output neurons that emit the final result of the ANN. Between the input and output layers, an ANN contains one hidden layer comprising multiple neurons. Each input node is connected to each neuron in the hidden layer, and each neuron in the hidden layer is connected to each neuron in the output layer. In contrast to ANNs, deep neural networks consist of multiple hidden layers. For more comprehensive details to ANNs beyond the scope of this introduction, the book *Dive into Deep Learning* by Zhang *et al.* [10] can be consulted.

#### Artificial Neurons

Neurons are the fundamental building block of a ANN. A neuron received various input values $x_0, x_1, ..., x_n$ from the input layer or, in case of deep neural networks, from the previous hidden layer. Each input value for a given neuron is multiplied

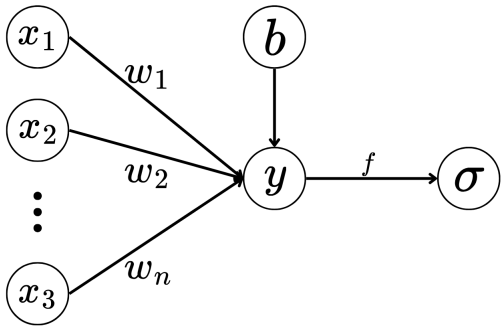



Figure 2.1: Structure of an artificial neuron with inputs $x_i$, weights $w_i$, bias $b$ and activation function $f$. The weighted sum of the inputs plus the bias results in $y$ which is then feed through the activation function $f$ resulting in the final output $o$ of the neuron.

with a corresponding weight $w_i$, which is optimized during training. The neuron calculates the weighted sum $y$ of all inputs and adds an additional bias term $b$.

$$y = \left( \sum_{i=1}^{n} w_i x_i \right) + b \tag{2.1}$$

The result $y$ of the calculation is then passed into an activation function $f$, which yields the final output of the neuron. Common examples for the activation functions include the Sigmoid function, the Rectified Linear Unit (ReLu) and the hyperbolic tangent (tanh) function.

$$o = f(y) = f\left( \left( \sum_{i=1}^{n} w_i x_i \right) + b \right) \tag{2.2}$$

**Forwardpass and Backpropagation**

As seen in the previous section, each neuron has weights $w_i$ and one bias term $b$, which are the parameters of an ANN model. During training, these parameters are optimized to minimize the model's loss on the training data. In the training process, samples are first fed through the network (forward pass), and then all weights are updated (backpropagation) according to the prediction loss.
After the forward pass, the loss is calculated using a loss function. For binary classification tasks, binary cross-entropy loss is often used, whereas mean squared error (MSE) is useful for regression tasks. The loss corresponds to the error the model made in predicting the right label on for the given training data. Given

the training loss, the gradient of the loss function is calculated and backpropagated through the network. Using the chain rule, the derivative can be calculated for each weight and bias, and the parameters are updated with an optimization algorithm such as gradient descent. This process of forward pass and backpropagation is repeated multiple times to incrementally reduce the loss until the training ends. For more details about optimizing a model, the *Deep Learning Book* by Goodfellow *et al.* [11] can be consulted.

#### Architectures

Multiple architectures exist for more advanced deep neural networks. As shown in Figure 2.2, the basic architecture employing multiple hidden layers, where each neuron is connected to each neuron of the previous and next layers (fully connected neurons), is called a Multilayer Perceptron (MLP). A more advanced architecture is the Convolutional Neural Network, where a neuron receives inputs only from a subset of neurons in the previous layer. Convolutional Neural Networks are particularly used in the domain of computer vision to process images or videos [12]. Another family of networks is the Recurrent Neural Networks (RNN), which can be used for sequential data. Recurrent Neural Networks allow signals from previous steps to be fed back into the network at the current step [13]. This enables RNNs to learn patterns over multiple steps of a sequence.

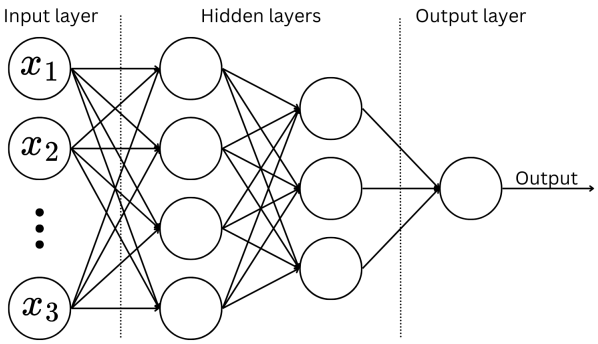



Figure 2.2: An example of a MLP with one input layer, two hidden layers and an output layer. All neurons are fully connected and the network take $x_1, x_2, ..., x_n$ as inputs and outputs a single value.

### 2.1.1 Recurrent Neural Networks

As introduced in the previous section, RNNs can capture patterns over multiple time steps. The RNN contains a hidden state that memorizes information from the

last timestep, making the current output dependent on previous outputs.

One major drawback of RNNs is the vanishing and exploding gradient problem. During backpropagation through a deep neural network, the gradient is computed using the chain rule. For activation functions (*e.g.* tanh) with gradients in the range $[-1, 1]$, the gradient can be a small number. This leads to the effect where small numbers are multiplied together repeatedly for early layers, resulting in a vanishing gradient. Consequently, the weights of the early layers might not be updated effectively. Conversely, for activation functions that can produce large gradients, large numbers are multiplied together repeatedly, leading to an exploding gradient. [14]

### 2.1.2 Long Short-Term Memory

The LSTM [15] is an improved RNN architecture that also addresses the vanishing and exploding gradient problem. An LSTM network consists of one or multiple LSTM cells. An LSTM cell, as shown in Figure 2.3, contains three gates: input, output, and forget gate. The input gate regulates the amount of information from the current timestamp that flows into the LSTM cell. The output gate controls the amount of information that is output from the cell. The forget gate determines the amount of information from the previous timestamp that is retained in the current cell. [16]

To train a LSTM model several hyperparameter are available. Due to limited computing resources, only the **number of LSTM layers** and the **units per LSTM cell** are optimized. The number of units reefers to the dimensionality of the hidden state in the LSTM cells.

In addition to the hyperparameters of the LSTM cells, a model also contains more general hyperparameters. The **batch size** corresponds to the number of samples in one forward and backward pass during training, while the **learning rate** sets the step size for parameter adjustments in the direction of the gradient during the training with the Adam optimizer. The **weight decay** is the second tuned parameter of the Adam optimizer, serving as a type of L2 regularization to improve model generalization. The **activation function** of a layer introduces non-linearity to the model and maps the values of one neuron to a specific interval. To regularize and prevent overfitting, dropout is used. In a dropout layer, some neurons are randomly deactivated, and the **dropout rate** indicates the probability of deactivating a certain neuron.

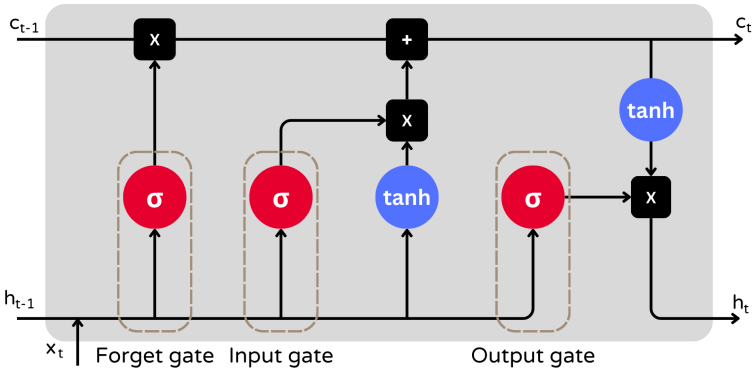



Figure 2.3: The architecture of a single LSTM cell. While $c_{t-1}$ and $h_{t-1}$ are the cell state and output of the previous cell respectively, $x_t$ is the input of the current timestamp. The gates use $h_{t-1}$ and $x_t$ to determine with the sigmoid activation function $\sigma$ a value in $(0, 1)$ which then regulates the amount of information that passes the gate.

## 2.2 Graphs

In this thesis, time series data are transformed into graphs and then a GNN is applied. Therefore, in this section the concept of graphs is introduced.

A graph is a pair $G = (V, E)$ consists of a set of nodes $V$ and a set of edges $E$, where each edge connects two nodes. Let $v_i \in V$ be a node and $e_{ij} = (v_i, v_j) \in E$ denote an edge pointing from the node $v_i$ to $v_j$. Nodes and edges in a graph can have attributes that represent the features of the corresponding entity. A graph is either directed or undirected. In a directed graph, all edges have a direction from one node to another. An undirected graph can be considered a special case of a directed graph where, for each directed edge, the inverse directed edge also exists [17].

Two important properties of graphs in this thesis are that they are non-Euclidean [18] and lack a natural node order relation [19]. This means that graphs do not have a natural mapping to a standard coordinate system, and nodes cannot be organized in a default order.

### Graph Neural Networks

The number of applications where data is represented as graphs (*e.g.* social networks) is increasing. As explained by Wu *et al.* [17], deep learning effectively captures patterns in Euclidean data but existing architectures do not work well on graph structures. Since graphs are irregular and contain a variable number of nodes and

edges, operations such as convolutions are challenging to apply. Furthermore, the assumption that instances are independent, common in traditional machine learning, does not hold for the highly interconnected nodes in graph data.

The outputs of GNNs can relate to the node, edge or graph level. At the node level, the output relates to node classification or regression tasks. At the edge level, the GNN can classify an edge or predict a link between two nodes. At the graph level, tasks involve the classification of the entire graph. [17]

Since the time series data is transformed into a graph, and then a prediction is made to classify whether the graph belong to a hypoglycemia or nonhypoglycemia event, only the graph classification task is utilized. The GNN model in this work comprises embedding generation, message-passing, pooling, and an output activation function. First, the features of nodes and edges are transformed into node and edge embeddings. Next, multiple rounds of message passing are applied, where connected nodes exchange information. Then, the pooling layer reduces the number of nodes in the graph for a more compact representation [20]. Finally, the output activation function maps the output of the pooling layer to a prediction for the classification task.

### Message Passing

Message Passing Neural Networks (MPNN) were proposed by Gilmer *et al.* [21], introducing a general framework for supervised learning on graphs. The Message Passing consists of three steps, performed for $T$ time steps:

1. Each node in the graph collect the embedding (also called messages) from all neighbours.

2. Each node aggregates the received messages by an aggregation function like summation.

3. The aggregated messages are then passed through an update function and then stored as the new hidden state of the node.

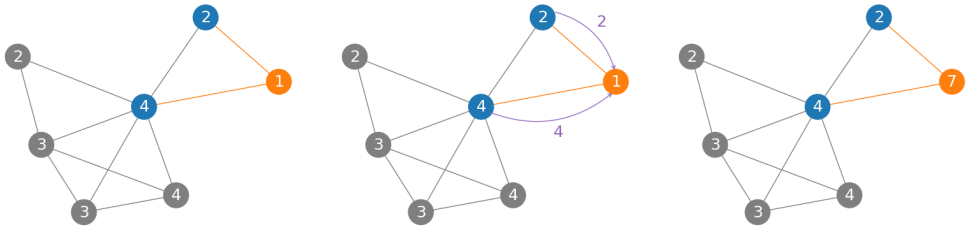
The same message passing process can be done for edges instead of nodes. This basic message passing mechanism is the key to leveraging the connectivity of graphs. [22]

A formal definition can be found in the original work by Gilmer *et al.* [21]. Let $v \in V$ be a node, $t$ the current time step and $N(v)$ the set of neighbours of $v$. Then

the hidden state $h_v$ of a node $v$ at the time step $t$ is calculated as in Equation (2.3).

$$h_v^{(t)} = \text{UPDATE}_t \left( h_v^{(t-1)}, \sum_{w \in N(v)} \text{MESSAGE}_t \left( h_v^{(t-1)}, h_w^{(t-1)}, e_{vw} \right) \right) \tag{2.3}$$

$\text{UPDATE}_t$ is the update function for the node embedding and $\text{MESSAGE}_t$ is the function that retrieves the message from a given neighboring node.




(a) (b) (c)

Figure 2.4: Visualization of the simplified message passing process in GNNs for intuition. In panel (a), a graph is depicted where the two neighbors of the orange node are colored in blue. Panel (b) illustrates the message passing process, where each neighbor transmits its current embedding to the orange node. In panel (c), the orange node is updated using the formula $h_{\text{orange}}^{(1)} = \text{ReLU}(2+4+1) = 7$, where ReLU is the update function, and $2, 4$ and $1$ are the messages received from the neighbors and the orange node's own embedding, respectively.

### Graph Attention Network

The GAT [23] architecture is a type of GNN that utilizes an attention mechanism. The general message-passing framework is adjusted by incorporating attention for each embedding. This approach allows GATs to assign individual weights to each neighboring node, providing a more flexible and expressive model. A GAT model iterates over each time step $t = 1, ..., T$ and compute for each node $v \in V$ the new embedding $h_v^{(t)}$ by applying Equation (2.4).

$$h_v^{(t)} = \underbrace{g^{(t)}}_{\text{activation function}} \left( \underbrace{W^{(t)}}_{\text{weight matrix}} \cdot \left[ \underbrace{\sum_{u \in N(v)} a_{vu}^{(t-1)} h_u^{(t-1)}}_{\substack{\text{Weighted average of v's} \\ \text{neighbours embeddings}}} + \underbrace{\alpha_{vv}^{(t-1)} h_v^{(t-1)}}_{\substack{\text{node v's weighted} \\ \text{embeddings}}} \right] \right) \tag{2.4}$$

While the activation function $g^{(t)}$ and weight matrix $W^{(t)}$ are shared across one time step, the attention weights $a_{vu}^{t-1}$ are individually computed for each combination of nodes $v, u \in V$ and the corresponding edge.

To get the attention weights $a_{vu}^{t-1}$, Veličković *et al.* [23] use an arbitrary attention mechanism $a : \mathbb{R}^{\mathbb{N}} \times \mathbb{R}^{\mathbb{N}} \to \mathbb{R}$ to compute unnormalised coefficients $e_{ij}$ for two nodes $v_i$ and $v_j$ based on their embeddings:

$$e_{ij} = a(h_i, h_j) \tag{2.5}$$

To get comparable attention weights, the coefficients are normalized over the neighbourhood $N(v)$ of node $v$ using the softmax function:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N(v)} \exp(e_{ik})} \tag{2.6}$$

To achieve a more stable learning process for the self-attention, multi-head attention is applied. In multi-head attention the operations of one time step is independently replicated multiple times and the outputs are featurewise aggregated. [23]

In this thesis, several hyperparameters of a GAT model were used and optimized. Starting with the **units of the dense layer** on the embeddings, which correspond to the output dimensionality of this layer. The **number of message-passing layers** (GAT layers) corresponds to the number of time steps $T$ in Equations (2.3) and (2.4). The **number of heads** indicates how many attention heads the GAT layer consists of, while the number of **channels per head** specifies the dimensionality of the output for each head. Next, the **edge dropout** is the probability that a particular edge is not used in the message-passing process. Finally, the **reduce type** of the pooling function defines which aggregation function is used to reduce the dimensionality of the graph. In addition to the GAT-specific hyperparameters, the batch size, output activation function, weight decay and learning rate are also optimized as presented in Section 2.1.2.

## 2.3 Research on Hypoglycemia Prediction

In this section, the current research conducted on hypoglycemia prediction is presented and discussed. The number of studies using approaches from machine learning and in particular deep learning has increased due to the recent development in the field of AI.

### 2.3.1 Hypoglycemia Prediction Variations

In the current research on hypoglycemia prediction, several important variations can be found, reflecting different approaches and objectives. If no other reference is provided, this section is based on the review paper on hypoglycemia prediction by Mujahid *et al.* [24]. Firstly, the research can be categorized into regression and classification tasks. Regression tasks aim to predict the actual blood glucose level, whereas classification tasks focus on determining whether a sample indicates a hypoglycemia or non-hypoglycemia event. Secondly, the prediction horizons used in studies vary widely. Most research focuses on short-term predictions (less than 60 minutes), while some investigate mid-term (60 to 240 minutes) or long-term (over 240 minutes) predictions. Another distinction lies in the type of diabetes the models are trained for. While the majority of patients suffer from type 2 diabetes, this type is more challenging to predict due to its high variability in blood glucose levels. Consequently, most research so far focuses on type 1 diabetes [25]. Furthermore, the data can be collected in clinical trials using CGM devices from patients or generated artificially using a diabetes simulator like UVA/PADOVA [26]. Additionally, a variety of machine learning models are employed, ranging from statist ical methods to advanced deep neural networks. Lastly, the studies differ in the features utilized. While some work uses only blood glucose levels, other studies combine CGM values with insulin doses or even measurements from other sources, such as electrocardiograms. In this work, the focus is on a classification task for short-term predictions using LSTM and GNN models, exclusively on CGM data from real type 1 diabetes patients.

### 2.3.2 Hypoglycemia Prediction

Bertachi *et al.*[27] applied a MLP and Support Vector Machine to predict nocturnal hypoglycemia events. Utilizing CGM data in conjunction with data from a FitBit activity tracker for 12 patients, they achieved a median recall of 78% and a precision of 82% with the better-performing Support Vector Machine model.

Iacono *et al.* [9] developed a personalized alarm system for hypoglycemia using an LSTM model. The personalization involved optimizing the hyperparameters of the LSTM for each patient in the study. Their research was based on an in silico dataset from the UVA/PADOVA simulator, incorporating past CGM values along with information on meals and insulin as inputs. The system first predicts future glucose values and subsequently feeds these predictions into an alarm system that classifies the predictions. The results demonstrate a recall of 76% and a precision

of 83% for hypoglycemia detection.

Lee *et al.* [25] implemented a transformer-based approach to predict glucose levels and classify hypoglycemia events. Their dataset primarily consisted of type 2 diabetes patients, though some experiments included type 1 diabetes data. In addition to real-world patient data, they generated additional data using a Generative Adversarial Network (GAN). The study tested various window sizes and prediction horizons. Furthermore, they evaluated their model on the frequently used type 1 diabetes dataset, OhioT1DM [28], in a transfer learning setting, achieving F1-scores above 0.7 for the classification task.

In many studies, decision trees are applied to the classification problem. For example, Reddy *et al.*[29] used decision trees to predict hypoglycemia in the context of aerobic exercise. They also implemented a Random Forest model (an ensemble of decision trees) for comparison and the more complex Random Forests outperformed the simple decision tree. Dave *et al.*[30] extracted 26 features from a type 1 diabetes CGM dataset, including demographic data, and then applied Random Forest for hypoglycemia classification. Their model predicted hypoglycemia for 30 and 60-minute prediction horizons with a recall of 91% and a precision of 90%.

In summary, various combinations of models, datasets, and settings have been studied, resulting in a wide range of achieved precision, recall, and F1-scores. In this work, a new dataset is used, and in addition to the LSTM model, a novel GNN approach is evaluated.

# Chapter 3

# Data

This chapter introduces the dataset used in the thesis. Section 3.1 presents the original dataset. Section 3.2 outlines the methodology employed for sample labeling, while Section 3.3 presents the applied preprocessing techniques. Section 3.4 provides further details and statistics on the resulting datasets and defines an average dataset. Section 3.5 finally explains the process of transforming the time-series data to graphs.

## 3.1 Dataset

In this thesis, the dataset of Garcia-Tirado *et al.* with CGM data from 37 individual patients is utilised [31]. From now on, the term dataset is used for a single set of CGM measurements from one patient. For each dataset the CGM measurements are collected over a timespan of approximately 30 days in an interval of five minutes. Table 3.1 shows two CGM measurements from the patient 85101 as example of the raw dataset. An illustrative example of a single day of CGM measurements is shown in Figure 3.1.

| **Id** | **Reading taken at** | **Reading** | **Patient** | **Created** |
|---|---|---|---|---|
| 1 | 05/07/2021 00:00:00 | 155 | 85101 | 6/9/2021 11:30 |
| 2 | 05/07/2021 00:05:00 | 152 | 85101 | 6/9/2021 11:35 |

Table 3.1: Two samples from the raw CGM dataset with all columns. The column *Reading* contains the CGM measurement. Only the columns *Reading taken at* and *Reading* are used in this work.

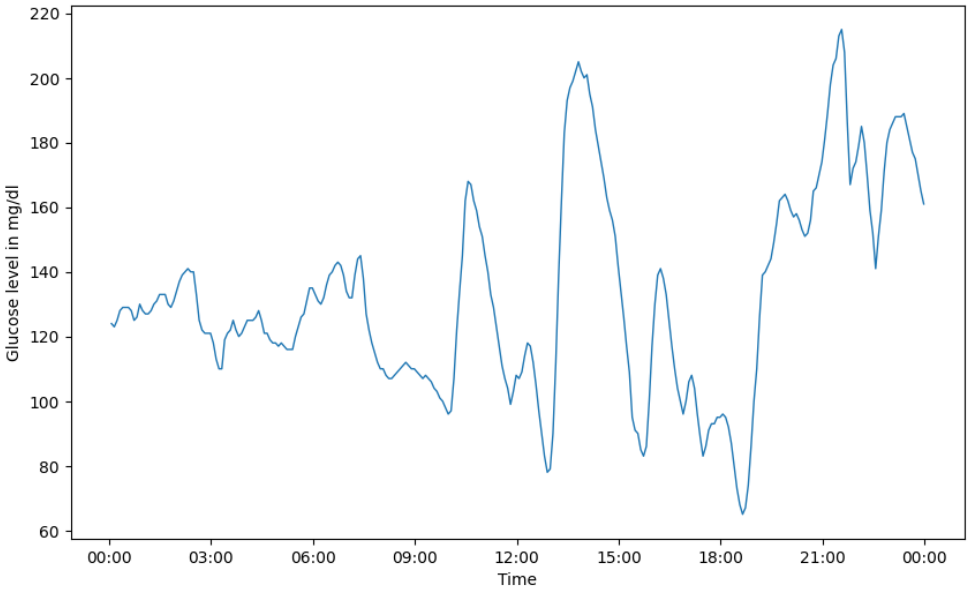



Figure 3.1: 24 hours of CGM values from one patient as an example. There is significant variability throughout the day, with glucose levels starting around 120 mg/dl, spiking up to 200 mg/dl, dropping to around 65 mg/dl, and then rising again to nearly 220 mg/dl.

## 3.2 Labeling of the Data

In order to run supervised deep learning algorithms on the data, it is necessary to provide a label for each timestamp. Given that the objective is to predict whether the glucose level of a patient in the future is below the threshold of 70 mg/dl, the labels are added in accordance with this criterion.

In general, the labels are assigned with the following meanings:

- A **positive label (True)** signifies that the patient will be affected by hypoglycemia in the prediction horizon.
- A **negative label (False)** signifies that there is no hypoglycemia event in the considered future.

There are several approaches to label the samples, three of which are presented below:

1. **Point prediction:** The only timestamp of the glucose values that is considered is the one that is precisely X minutes into the future. This is then checked against the threshold of 70 mg/dl.
2. **Within interval prediction:** Instead of considering only a single timestamp, a fixed interval in the future is analyzed. If at least one timestamp within

this interval falls below the threshold of 70 mg/dl, the sample is labeled as positive. For example, with a prediction horizon of six and an interval width of seven, the interval considered for labeling the sample is $[3, 9]$, where the value of 0 represents the current timestamp, 1 represents the next, and so on.

3. **Within prediction horizon:** All CGM measurements from the current time until the prediction horizon are examined to determine if any fall below the threshold of 70 mg/dl. If at least one timestamp meets this criterion, the sample is labeled as positive.

If a hypoglycemia event is predicted to occur, a point prediction will trigger an alarm only when the exact future timestamp contains a glucose level below the threshold. In contrast, the second approach considers a longer period but may stop the warning as the time until the event shortens. Therefore, this thesis adopts a third approach: an alarm is raised if any hypoglycemia event is predicted to occur within the entire prediction horizon. In Figure 3.2 a example is provided with a visual explanation of the window size and prediction horizon. This example would be labeled as a positive hypoglycemia event since multiple measurements within the prediction horizon fall below the threshold.

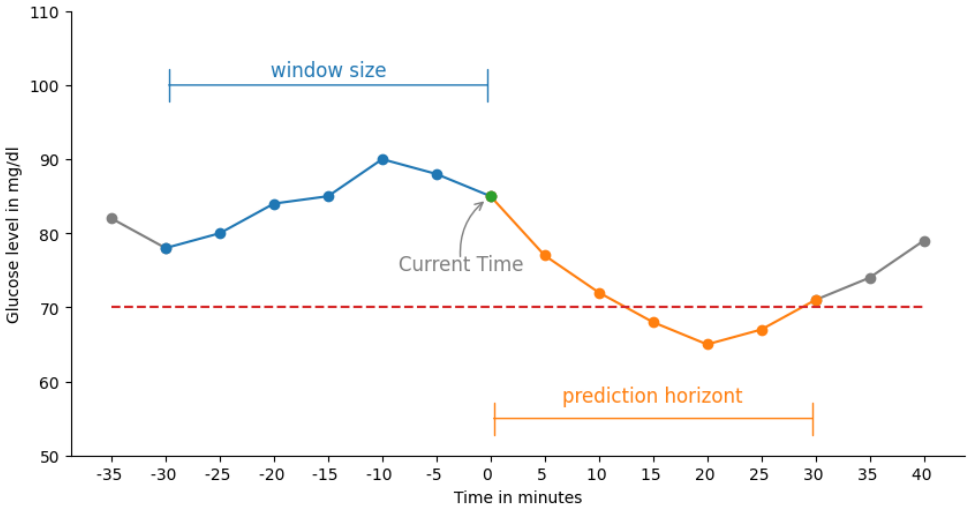



Figure 3.2: Illustration of an example of CGM data with annotations for the window size and prediction horizon, along with the current time. The blue points represent the measurements within the window size, which are used as input for the prediction. The orange points represent the measurements within the prediction horizon and are relevant for label generation. The red dashed line indicates the hypoglycemia threshold of 70 mg/dl.

## 3.3 Preprocessing of the Data

In this thesis, only two preprocessing steps are performed on the raw data. First the unused columns are removed from the datasets. Secondly, after generating samples using a sliding window, any samples containing gaps due to missing data is removed from the dataset to ensure that only complete samples are used.

## 3.4 Statistics of the Data

In order to gain a deeper insight into the 37 datasets, some statistical calculations are performed on the available data. Furthermore, one dataset is defined as the average datasets, which will be used later for hyperparameter tuning.

### 3.4.1 Characteristics

A dataset contains on average 8472 CGM measurements, with a range of values between 39 mg/dl and 654 mg/dl. The mean measured value is 150 mg/dl. One outlier dataset contains just 3782 CGM measurements, while all other datasets contain between 7332 and 9001 measurements.

The proportion of positive labels in the dataset is dependent on the prediction horizon. As observed in Table 3.2, the percentage of positive labels increases with a higher prediction horizon. Nevertheless, in all datasets and across all prediction horizons, the percentage of positive labels is very small, resulting in highly imbalanced datasets.

| **Horizon** | **Minimum (%)** | **Median (%)** | **Maximum (%)** |
|---|---|---|---|
| **15** | 0.4 | 3.1 | 14.0 |
| **30** | 0.6 | 4.3 | 16.7 |
| **60** | 1.0 | 6.7 | 21.6 |

Table 3.2: The minimum, median and maximum of the percentage of positive labels per prediction horizon calculated across all datasets. For instance, for the prediction horizon of 30 minutes a dataset contains only 0.6% positive labels while the median dataset contains 4.3% and the dataset with the most positive labels contains 16.7%. As the prediction horizon increases, the percentages rise because a given positive timestamp is included in a greater number of samples. For every prediction horizon, the data is significantly imbalanced.

Since the dataset will be split into a training and validation set by maintaining the time series ordering, it is important to ensure that the positive labels are evenly distributed over the full dataset. This is evaluated by splitting the dataset at 70%

of the measurements and then calculating the difference between the train and validation set in terms of positive label proportion. In Table 3.3 one can see that the deviations are in a low single digit area and therefore no further actions are required.

| Horizon | Minimum (%) | Median (%) | Maximum (%) |
|---|---|---|---|
| **15** | -3.8 | 0.3 | 6.7 |
| **30** | -4.5 | 0.4 | 7.8 |
| **60** | -6.6 | 0.5 | 9.5 |

Table 3.3: The difference in percentage points between the training and validation sets regarding the percentage of positive labels per prediction horizon. The median of all differences is near zero, indicating that the positive labels are well distributed between the training and validation sets.

### 3.4.2 The Average Dataset

We select one of the 37 datasets as the average dataset for later hyperparameter tuning. To identify the best matching dataset, we use the statistics from Section 3.4.1 and select the dataset as follows:

1. Filter all datasets to ensure that the difference in the percentage of positive labels between the training and validation sets is less than 1%.
2. From the remaining datasets, we select the top three with the lowest deviation from the mean percentage of positive labels.
3. From the top three remaining datasets, we select the one with the highest number of measurements.

The resulting average dataset belongs to the patient with the ID 85105_20210503_20210603 and can be described with the values in Table 3.4.

| Statistic | Value |
|---|---|
| # Measurements | 9001 |
| Overall Positive Labels | 4.56% |
| Train Positive Labels | 4.83% |
| Validation Positive Labels | 3.92% |

Table 3.4: Selected statistics from the average dataset. The dataset is heavily imbalanced, with overall positive labels comprising only 4.56%. The difference in the percentage of positive labels between the training and validation sets is slightly less than one percentage point.

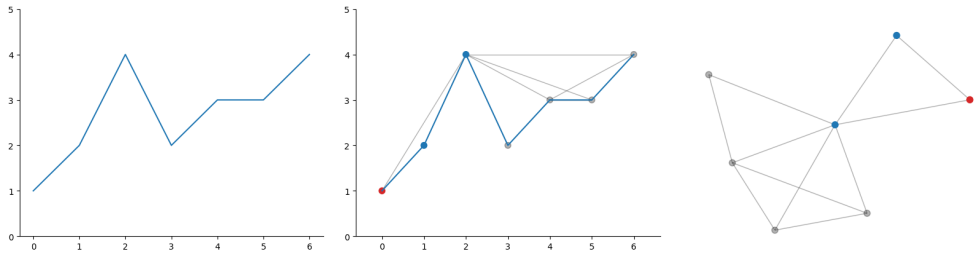
## 3.5 Time Series to Graphs

Thus far the datasets are used as time series data, with each sample comprising a timestamp and a corresponding CGM value. The next step involves transforming this time series data into graphs. This transformation is essential for utilizing GNNs in the classification task.

In this thesis, the concept of visibility graphs [32] is used to transform the time series to graphs. In the visibility graph, each CGM value is interpreted as a node. The visibility relationship between each pair of nodes is then evaluated, and an edge is added between them if they are visible to each other. Two arbitrary data points $(t_a, y_a)$ and $(t_b, y_b)$ are visible to each other if any other data point $(t_c, y_c)$ placed between them fulfills the following equation:

$$y_c < y_b + (y_a - y_b)\frac{t_b - t_c}{t_b - t_a} \tag{3.1}$$

A visual introduction of the transformation can be found in Figure 3.3. In this thesis, natural visibility is employed to construct the graphs from the time series data.



(a) Time Series (b) Time Series with nodes and edges (c) Visibility graph

Figure 3.3: A visual introduction to the transformation of a time series into a visibility graph. In panel (a), the time series is plotted within a coordinate system where the x-axis represents the time and the y-axis represents the value at each timestamp. Panel (b) illustrates the addition of nodes and edges to the 2D representation of the time series. Finally, in panel (c), the coordinate system is removed, and an arbitrary representation of the graph is displayed. The red and blue nodes help identify the correspondence between time stamps and nodes.

### 3.5.1 Create Visibility Graphs

This section shows in more detail how the visibility graphs are generated.

Each dataset is processed by iterating over all combinations of prediction horizon and window size. For each combination, all samples with the corresponding labels are obtained and each time series sample is transformed into a single visibility graph using the ts2vg Python package.

Each node is augmented with two features: the CGM value and its position in the time series data sequence. The inclusion of the node's order enables the model to capture the temporal relationships between nodes. Additionally, the euclidean distance between two nodes, connected by an edge, is included as a feature of that edge.

Since TensorFlow GNN (TF-GNN) [33] is used for the GNNs, several library specific adjustments were made. TF-GNN is a Python library specifically designed for developing GNNs within the TensorFlow platform.

- Since TF-GNN utilizes directed edges and the visibility relationship is undirected, two opposite directed edges were added for each edge between a pair of nodes.
- The GAT in TF-GNN learns attention weights for each edge, but by default, it does not include a node's own embedding. To incorporate the embedding of a node in the attention mechanism, a loop is added to each node.

### 3.5.2 The Graph Dataset

The resulting graph dataset includes a train and validation record for each combination of window size, prediction horizon, and dataset, structured as graphs. These records are stored in the TensorFlow .tfrecord format. The complete dataset is available on the GitHub page of the Pattern Recognition Group at the University of Bern.

# Chapter 4

# Experiments

This chapter presents the conducted experiments and the corresponding results. Section 4.1 introduces the general setup of the experiments. Section 4.2 explains the goals of the experiments. Sections 4.3 and 4.4 present the LSTM and GNN implementations, respectively, including their hyperparameter tuning and results. Finally, Section 4.5 compares the results of the LSTM and GNN models.

## 4.1 Experiment Setup

This section present the overall setup of the experiment, the process for hyperparameter tuning, the used evaluation metrics and the technology stack.

#### General Setup

In this thesis a single model is trained for each combination of dataset, prediction horizon and window size separately. Thus tyyyyyyyhe following steps are done for each trained model.

First, the CSV file containing all CGM values is loaded, and the unused columns are removed. Each time series data point is then labeled as explained in Section 3.2. Then the ratio of positive to negative samples is calculated and later used to adjust the weighting of the positive samples in the binary cross-entropy loss function. To convert the labeled data points into samples $(x, y)$, where $x$ consists of several data points as input and $y$ is the corresponding label for the supervised learning task, the sliding window technique is applied to the dataset.

The window size of the sliding window corresponds to the number of data points considered for one sample, while the prediction horizon defines the time offset be-

tween the input window and the target data point. Since the dataset labeling accounts for the prediction horizon, the label of the data point after the prediction horizon offset can be directly used. To evaluate the performance on unseen data, the samples are split into disjoint training (70%) and validation (30%) sets. The first 70% of the samples are used for the training set, while the last 30% are used for the validation set.

After generating all samples using the sliding window technique, the model is defined, built, and compiled. The compiled model is then fitted to the training data and evaluated on the validation set. For training, the Adam optimizer [34] is used with learning rate and weight decay as hyperparameters.

#### Hyperparameter Tuning

To improve the performance of the models a hyperparameter tuning is performed on several hyperparameters.

To evaluate the performance of a trial, the $F_2$-Score is used. The $F_2$-Score emphasizes recall, making it more important to correctly predict hypoglycemia events rather than non-hypoglycemia events. The tuning is performed using Bayesian Optimization [35], with 200 trials executed for each run, and each trial is executed twice to reduce variance. The number of trials is limited to 200 due to constraints on computing resources.

As defined in Section 3.4.2, only the average dataset is used for tuning. In addition, both the window size and the prediction horizon are fixed at 30 minutes to limit the required computing resources. Various approaches for tuning could be used, some of which are briefly introduced here:

1. Run the hyperparameter tuning for each dataset separately. This method would require creating a train, validation, and test set for each dataset.

2. Perform hyperparameter tuning on a combined set comprising all validation sets. This approach may enhance generalization but necessitates a distinct test set encompassing all datasets.

3. Tune the model using the validation set of a single dataset and subsequently apply the same hyperparameters to all other datasets.

All of these variants assume that the window size and prediction horizon are fixed. For a more exhaustive hyperparameter tuning, the parameters could be optimized

for each combination of window size and prediction horizon. Due to limited computational resources and time constraints, the third approach is applied in this work.

### Evaluation Metrics

To evaluate the results, we use precision, recall, and $F_2$-Score as metrics. Precision is the proportion of true hypoglycemia events among all predicted positive cases. A precision of 1 means that no false positive predictions are made and all predicted events are actual hypoglycemia events. Recall, on the other hand, is the proportion of correctly detected hypoglycemia events among all actual events. A recall of 1 means that all hypoglycemia events are predicted, while a value of 0 means that no real event could be classified correctly. To combine these two metrics, the $F_\beta$-score can be used, where $\beta$ is replaced by some positive number. In this work, it is more important to detect as many real cases as possible, even at the cost of predicting some false positives. To give more weight to recall than precision, the value of $\beta$ is set to 2 and the $F_2$-Score is obtained. The best possible performance is achieved when all three metrics are 1, indicating perfect classification. Conversely, the worst case occurs when the metrics are 0, indicating no correct predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{4.1}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.2}$$

$$F_\beta\text{-Score} = \frac{(1 + \beta^2) \times \text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}} \tag{4.3}$$

### Technology Stack

The experiments are implemented in Python using TensorFlow, Tensorflow GNN, and Keras libraries. Hyperparameter tuning utilizes the KerasTuner library [36], and parameter analysis is conducted using TensorBoard. The training is executed on UBELIX, the high-performance computing cluster at the University of Bern.

## 4.2 Goal of the Experiments

The goal of the experiments is to evaluate the performance of the LSTM and GNN models on the hypoglycemia prediction task and compare the results. Additionally, the performance across different window sizes and prediction horizons is tested to

assess the impact of varying prediction horizons and window sizes on the performance of the models.

## 4.3 Long Short-Term Memory

This Section presents the implementation details of the LSTM models, the corresponding hyperparameter tuning and the final results.

### 4.3.1 Implementation

Through preliminary experiments, three different LSTM architectures were defined and implemented. Hyperparameter tuning and evaluation are performed for each of these architectures, and the final results are compared. Figure 4.1 presents the three architectures. While the first and third architectures contain two LSTM layers, the second consists of four consecutive LSTM layers. All three architectures include a dropout layer for regularization and a dense output layer with a sigmoid activation function to map the values to probabilities. Additionally, the second and third architectures contain an extra dense layer.

The training was conducted for a maximum of 80 epochs with an early stopping mechanism. Early stopping was activated starting at epoch ten, with a patience of ten epochs. The weights of the positive and negative classes were set according to the ratio of positive to negative labels to address class imbalance.

| **Architecture 1** | **Architecture 2** | **Architecture 3** |
|---|---|---|
| 2 x LSTM-Layers | 4 x LSTM-Layers | 2 x LSTM-Layers |
| 1 x Dropout Layer | 1 x Dense Layer | 1 x Dense Layer |
| 1 x Dense Output Layer | 1 x Dropout Layer | 1 x Dropout Layer |
| | 1 x Dense Output Layer | 1 x Dense Output Layer |

Figure 4.1: The three different LSTM architectures evaluated in this work. Architectures 1 and 3 each contain two LSTM layers, while Architecture 2 consists of four LSTM layers. Additionally, architectures 2 and 3 each include an extra dense layer following the LSTM layers and before the dropout layer.

### 4.3.2 Hyperparameter Tuning

The hyperparameter tuning for the LSTM model is performed over a fixed search space, as shown in Table 4.1. Focal cross-entropy loss [37] is used to address class imbalance in the dataset. Tuning is performed for each architecture, and the best

performing combination of hyperparameters is selected for evaluating the architecture across all datasets, window sizes, and prediction horizons. Best-performing hyperparameters per architecture are shown in Table 4.2.

As defined in Section 4.1, the hyperparameter tuning is performed on just one patient with a fixed window size and prediction horizon. Since only 200 trials are conducted, only a small subspace of the search space is evaluated. Future work could expand this to a more extensive search of hyperparameters. While most trials for architectures 2 and 3 were stopped after approximately 30 epochs, Architecture 1 showed more diversity in the number of epochs.

| **Hyperparameter** | **Values** |
|---|---|
| Batch Size | 16, 32, 64, 128, 256, 512, 1028 |
| Learning Rate | 0.00001, 0.0001, 0.001, 0.01 |
| Weight Decay | 0.00001, 0.0001, 0.001, 0.01 |
| # Units per LSTM-Layer | 32, 64, 128, 256, 512 |
| # Units per Dense-Layer | 16, 32, 64, 128 |
| Dense Activation Function | ReLu, Elu, Tanh |
| Output Activation Function | ReLu, Sigmoid, Linear |
| Dropout Rate | 0.3, 0.4, 0.5 |
| Loss Function | Binary Crossentropy<br>Binary Focal Crossentropy |
| Focal Crossentropy Alpha | [0.1, 0.9] |
| Focal Crossentropy Gamma | 1, 2, 3, 4, 5 |

Table 4.1: All considered hyperparameters for the LSTM models along with their respective search spaces.

### 4.3.3 Results

Each architecture is once evaluated over each combinations of dataset, window size and prediction horizon. The results are then aggregated over all datasets and summarized in terms of precision, recall and $F_2$-Score. Architecture 1 has outperformed Architecture 2 and 3 in terms of the $F_2$-Score and the results are shown in Table 4.3. All complete results can be found in Appendix A.

The results show that the $F_2$-Score decreases from 0.503 to 0.255 as the prediction horizon increases. Furthermore, it can be concluded that the window size does not significantly affect the performance and the model predicts significantly better for low prediction horizons.

| Parameter | Architecture 1 | Architecture 2 | Architecture 3 |
|---|---|---|---|
| Batch Size | 16 | 16 | 32 |
| LSTM Units 1 | 512 | 256 | 512 |
| LSTM Units 2 | 32 | 512 | 32 |
| LSTM Units 3 | - | 512 | - |
| LSTM Units 4 | - | 64 | - |
| Dropout Rate | 0.4 | 0.3 | 0.3 |
| Dense Units | - | 32 | 128 |
| Dense Activation | - | ELU | relu |
| Output Activation | Sigmoid | Linear | linear |
| Focal Loss | False | True | True |
| Focal Alpha | - | 0.6 | 0.9 |
| Focal Gamma | - | 2 | 1 |
| Learning Rate | 0.0001 | 0.0001 | 0.0001 |
| Weight Decay | 0.0010 | 0.0001 | 0.0100 |
| **Validation $F_2$-Score** | **0.469** | **0.478** | **0.466** |

Table 4.2: The hyperparameters of the best-performing trial for each LSTM architecture. During tuning, Architecture 2 achieved the highest $F_2$-Score on the validation set with a score of 0.478. Architecture 1 and 3 have almost the same $F_2$-Scores, with 0.469 and 0.466 respectively.

Figure 4.2 presents the distribution of the metrics across all datasets. The median recall is approximately 0.800, while the median precision is below 0.200. The $F_2$-Score has a median around 0.400 with a standard deviation of 0.175.

| | **$F_2$-Score** | | | **Precision** | | | **Recall** | | |
|---|---|---|---|---|---|---|---|---|---|
| **horizon** / **window** | **15** | **30** | **60** | **15** | **30** | **60** | **15** | **30** | **60** |
| **30** | **0.503** | **0.376** | **0.264** | 0.211 | **0.146** | 0.103 | **0.868** | **0.745** | **0.626** |
| **60** | 0.501 | 0.365 | 0.262 | 0.214 | 0.144 | **0.105** | 0.846 | 0.703 | 0.568 |
| **120** | 0.502 | 0.348 | 0.255 | **0.224** | 0.143 | **0.105** | 0.805 | 0.650 | 0.502 |

Table 4.3: The aggregated results of the LSTM Architecture 1 across all 37 datasets for each combination of window size and prediction horizon. The best results per metric and window size are emphasized in bold. The $F_2$-Score indicates a noticeable decline in model performance with increasing prediction horizon. Conversely, the impact of increasing window sizes on performance is minimal, except for the lowest prediction horizon of 15 minutes where no change is evident.

## 4.4 Graph Neural Network

This Section presents the implementation details of the GNN model, the corresponding hyperparameter tuning and the final results. The GNN model are based on the GATv2 [38] architecture, which is an improved version of the GAT, and

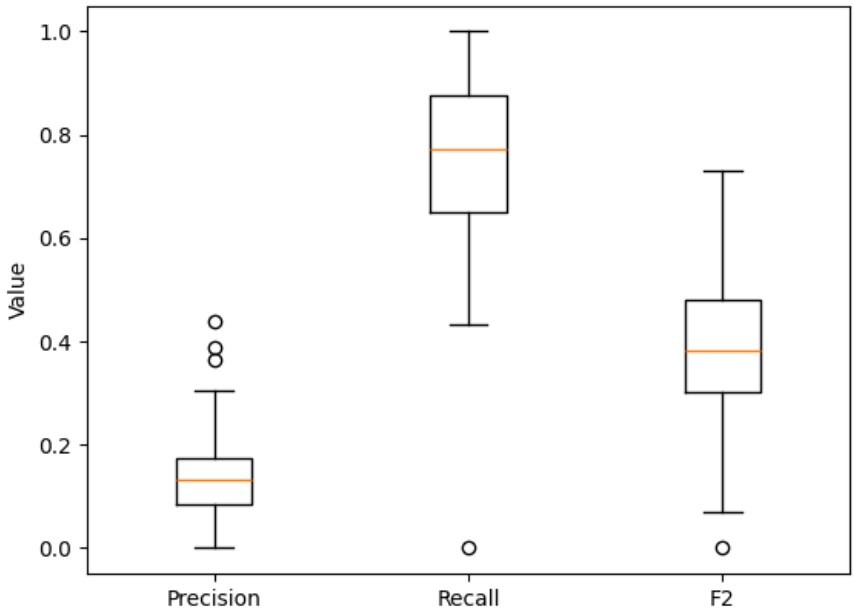



Figure 4.2: The distribution of the metrics from the LSTM Architecture 1, for a window size and prediction horizon of 30 minutes, across the 37 datasets. The precision values are generally low, ranging from 0 to 0.500. In contrast, the recall values are higher, ranging from 0.500 to 1. The $F_2$-Score varies significantly, with a median around 0.400 and values spanning from 0 to nearly 0.800. It is visible that the standard deviation is quite high, indicating a wide distribution of values across a large interval.

trained on the visibility graph data as introduced in Section 3.5.1. The term GAT will refer to the enhanced GATv2 architecture from now on.

### 4.4.1 Implementation

Unlike the LSTM model, different architectures in terms of layers were not defined for the GNN model. Instead, one model use only node features, while another use both node and edge features of the visibility graph. This experiment evaluates the influence of the edge feature (euclidean distance of the edges) on the predictions.

The architecture begins with embedding layers for the node features and, if used, edge features, each followed by a dense layer. Next, several GAT layers with edge dropout for regularization and multiple heads and channels are applied. To reduce the feature representation of the graph to a single value, a pooling layer is applied, followed by a final dense layer with a sigmoid activation function.

### 4.4.2 Hyperparameter Tuning

The hyperparameter tuning for the GNN model is performed over the search space shown in Table 4.4. Preliminary experiments have shown that, unlike the LSTM model, the focal cross-entropy loss does not improve performance for the GNN model. Therefore, the loss function is fixed to binary cross-entropy. Further, a hyperparameter for the number of GAT layers is defined and included in the hyperparameter tuning. The "Node/Edge Dense Unit" corresponds to the units of the dense layer applied after the generation of the node/edge embedding. For both variants (with and without edge embedding), the tuning is performed and evaluated. The best performing combinations of hyperparameters are fixed and can be found in Table 4.5.

| **Hyperparameter** | **Values** |
|---|---|
| Batch Size | 16, 32, 64, 128, 256 |
| Learning Rate | 0.00001, 0.0001, 0.001, 0.01 |
| Weight Decay | 0.00001, 0.0001, 0.001, 0.01 |
| # Node Dense Units | 8, 16, 32, 64, 128, 256 |
| # Edge Dense Units | 8, 16, 32, 64, 128, 256 |
| # GAT-Layers | 2, 3, 4, 5, 6 |
| # Heads per GAT-layer | 3, 4, 5, 6 |
| # Channels per head | 6, 7, 8, 9, 10 |
| Edge dropout | 0.1, 0.2, 0.3, 0.4, 0.5 |
| Pool reduce type | Mean, Min, Max |

Table 4.4: All considered hyperparameters for the GNN models along with their respective search spaces.

### 4.4.3 Results

Similar to the LSTM results, each architecture is evaluated for each combination of dataset, window size, and prediction horizon. The results are then aggregated over all datasets and summarized. The first architecture (using only node features) has an average $F_2$-Scores (overall datasets, window sizes and prediction horizons) of 0.378, while the second architecture (with both node and edge features) achieves an average $F_2$-Score of 0.401. In contrast to the hyperparameter tuning results, where Architecture 1 achieved an $F_2$-Score of 0.557 and outperformed Architecture 2 by 6.5 percentage points, the conducted experiments showed that Architecture 2 performed better. One explanation could be, that the hyperparameter tuning is conducted on just one dataset and Architecture 2 did better generalize to all other datasets.

| Parameter | Variant 1 | Variant 2 |
|---|---|---|
| Batch Size | 128 | 128 |
| Node Dense Units | 32 | 256 |
| Edge Dense Units | - | 8 |
| Message Passing Layers | 2 | 2 |
| Graph Number Heads | 3 | 6 |
| Graph Per Head Channels | 9 | 8 |
| Edge Dropout | 0.3 | 0.4 |
| Pool Reduce Type | min | min |
| Output Activation | sigmoid | sigmoid |
| Learning Rate | 0.00100 | 0.00100 |
| Weight Decay | 0.00001 | 0.00001 |
| **Validation $F_2$-Score** | **0.557** | **0.492** |

Table 4.5: The hyperparameters of the best-performing trial for each GNN variant are presented in the table. Variant 1 includes only node features, while Variant 2 incorporates both node and edge features. Variant 1 achieved a higher $F_2$-Score on the validation set during tuning compared to Variant 2.

The results of Architecture 2 are presented in Table 4.6, while those from Architecture 1 can be found in Appendix A. The results indicate that the performance of the GNN model does not decrease with an increasing prediction horizon. Additionally, the window size does not appear to affect the performance of the GNN model.

Figure 4.3 presents the distribution of the results for a window size and prediction horizon of 30 minutes in a boxplot. The graphic illustrates that the $F_2$-Score distribution is tight, with a computed standard deviation of 0.0342. This indicates that the GNN model is able to achieve similar performance for all datasets.

| | **$F_2$-Score** | | | **Precision** | | | **Recall** | | |
|---|---|---|---|---|---|---|---|---|---|
| **horizon** / **window** | **15** | **30** | **60** | **15** | **30** | **60** | **15** | **30** | **60** |
| **30** | 0.388 | 0.408 | **0.403** | 0.133 | 0.147 | 0.142 | **0.779** | 0.755 | 0.770 |
| **60** | **0.412** | 0.400 | 0.398 | **0.150** | 0.141 | 0.140 | 0.767 | **0.772** | **0.779** |
| **120** | 0.397 | **0.409** | 0.396 | 0.139 | **0.148** | **0.143** | 0.770 | 0.760 | 0.761 |

Table 4.6: The aggregated results of GNN Variant 2 across all 37 datasets for each combination of window size and prediction horizon. The best results per metric and window size are emphasized in bold. The $F_2$-Score remains stable at approximately 0.400 across all combinations of prediction horizon and window size.

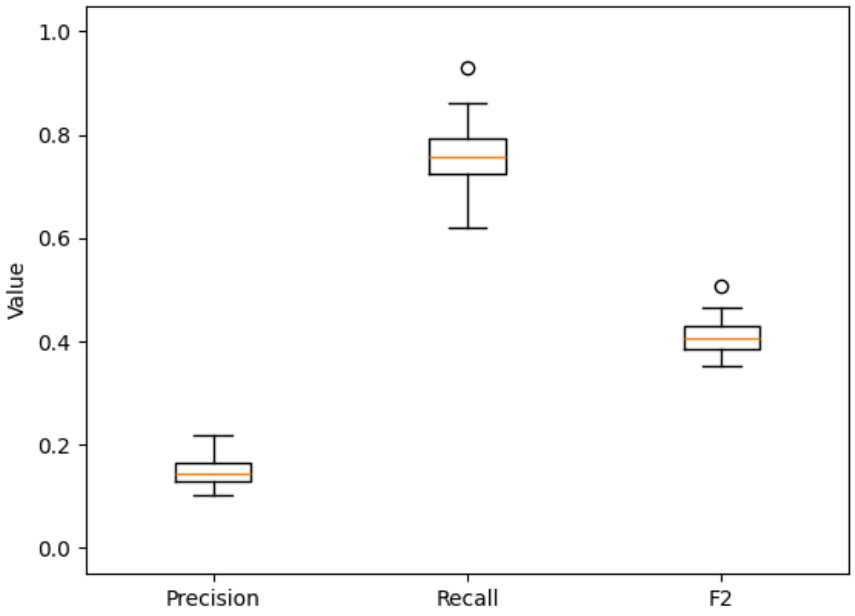



Figure 4.3: The distribution of metrics from GNN Variant 2, using a window size and prediction horizon of 30 minutes across all 37 datasets. The figure shows that precision values are generally low, whereas recall has a median slightly below 0.800. The $F_2$-Score exhibits a compact distribution with a median around 0.400. Notably, the standard deviation is very low, indicating a dense distribution of results and thus a highly stable model across different datasets.

## 4.5 Result Comparison

The previous sections presented the best results for the LSTM and GNN models. In Table 4.7 the $F_2$-Scores of both models are presented for each horizon and window size. The LSTM model achieves an average $F_2$-Score of up to 0.503 for a window size of 30 minutes and a prediction horizon of 15 minutes. In contrast, the best average performance of the GNN model is significantly lower, with an $F_2$-Score of 0.412 on the same settings. However, while the performance of the LSTM model decreases with increasing prediction horizons, the GNN model remains very stable, achieving almost the same performance across all tested prediction horizons. Already for the prediction horizon of 30 minutes the GNN slighly outperforms the LSTM model and for 60 minutes the GNN performs significantly better, with a nearly 15 percentage point advantage. This suggests that the LSTM model excels at short-term predictions, whereas the GNN model produces more consistent results for longer-term predictions.

| horizon | 15 | | 30 | | 60 | |
|---|---|---|---|---|---|---|
| **window** | **LSTM** | **GNN** | **LSTM** | **GNN** | **LSTM** | **GNN** |
| **30** | 0.503 | 0.388 | 0.376 | 0.408 | 0.264 | 0.403 |
| **60** | 0.501 | 0.412 | 0.365 | 0.400 | 0.262 | 0.398 |
| **120** | 0.502 | 0.397 | 0.348 | 0.409 | 0.255 | 0.396 |

Table 4.7: The $F_2$-Scores of the LSTM and GNN models for each window size and prediction horizon. While the LSTM model performs better for the 15-minute horizon, the GNN achieves better scores for the 30 and 60 minute horizons

The second interesting insight concerns the standard deviation of the $F_2$-Score distribution over all datasets as shown in Figure 4.4. In the examined combination of window size and prediction horizon both 30 minutes, the LSTM model exhibits a standard deviation of 0.175 while the distribution of the GNN model has a standard deviation of only 0.034. This indicates that the GNN model is significantly better in producing stable results on all datasets than the LSTM model.

A reasonable comparison of our results to other scientific work is difficult, since in this thesis a different approach for the data labeling is used than in most other papers (point prediction).

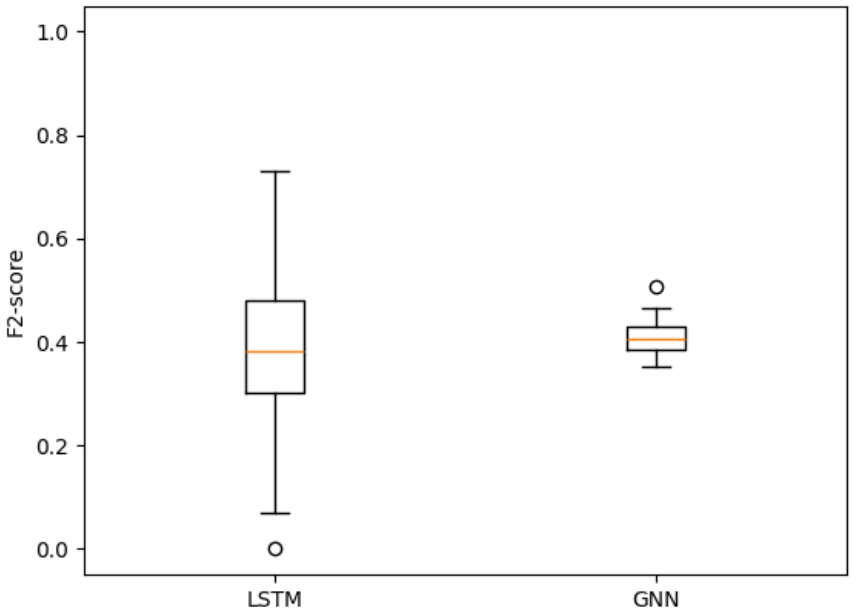



Figure 4.4: The distribution of the $F_2$-Scores across all datasets for both the LSTM and GNN models. It is clearly visible that the distribution of the GNN is much more compact than that of the LSTM.

# Chapter 5

# Conclusions and Future Work

The reliable prediction of hypoglycemia events with sufficient advance notice can significantly enhance the quality of life for diabetes patients. With the rapid advancements in CGM technology and artificial intelligence, new opportunities for intelligent systems have emerged and can be explored. Researchers have proposed various architectures for hypoglycemia prediction, including LSTM and transformer-based models.

In this thesis, an LSTM and a GNN model are implemented and evaluated on a real-world dataset of type 1 diabetes patients. Specifically, we propose three LSTM architectures and two variations of GNNs, selecting the best variant from each model type for comparison. For the LSTM model, the simplest architecture, comprising two LSTM layers, a dropout layer, and a dense output layer, performed slightly better than the other architectures. In the case of the GNN, the variation that included edge features outperformed the one without edge features.

In the experiments, each model was evaluated across 37 datasets from real-world patients with window sizes of 30, 60, and 120 minutes, and prediction horizons of 15, 30, and 60 minutes. The results were aggregated over all datasets to produce a table with averages for each combination of window size and prediction horizon. The results lead to the following conclusions regarding the $F_2$-Score of the evaluations:

1. The LSTM outperforms the GNN for the prediction horizon of 15 minutes while the GNN performs better on horizons of 30 and 60 minutes.

2. The GNN model demonstrate stable performance across all prediction horizons, suggesting its robustness for longer-term predictions. On the other hand the performance of the LSTM clearly decreases with increasing prediction horizon.

3. The standard deviation of the $F_2$-Score across all datasets is significantly lower for the GNN model compared to the LSTM model. This indicates that the GNN model is able to produce more stable results across different datasets, suggesting that it generalizes better than the LSTM model.

4. The inclusion of edge features, specifically the Euclidean distance, positively impacts the performance of the GNN model.

5. Even the best-performing LSTM model, with a recall of 0.868 and a precision of 0.211 for a 15-minute prediction horizon, falls short of being production-ready. This model accurately predicts 86% of actual hypoglycemia events but only 21% of its predictions correspond to real hypoglycemia events. Such a false positive rate would render the model less supportive and reliable for patients.

Comparing the performance of our models directly with other studies is challenging due to variations in labeling processes, data sources, and diabetes types. However, we can state that the best-performing models for hypoglycemia prediction, which use multiple features in addition to CGM data, achieve precision and recall scores above 0.990 and 0.940, respectively [39]. Although they used different approaches, these results are far more production-ready than our findings. The reasons for this performance difference can be varied. One possible explanation is that models incorporating more features extracted from the CGM data are able to capture a wider range of patterns, leading to better performance. Another potential explanation is that the architectures or hyperparameters of the models in this work were not optimally chosen, preventing the models from effectively capturing the patterns in the CGM data. The lack of features could be addressed by incorporating feature generation techniques from other works as for example by Dave *et al.* [30]. To truly support diabetes patients in managing their blood glucose levels and preventing hypoglycemia events, the models developed in this thesis did not provide significant assistance. However, it is noteworthy that the GNN model demonstrated more stable performance across different prediction horizons. Therefore, this work might support the development of new GNN-based hypoglycemia prediction models and achieve stabler performance for the current state-of-the-art hypoglycemia prediction systems. Further the GNN models might be interesting for long-term prediction horizons (over 240 minutes) which is for example the case for nocturnal hypoglycemia prevention.

It would be valuable to investigate whether the insights gained from the GNN models in this study can be applied to other datasets and labeling processes. To

facilitate further research and validation, the generated graph dataset has been published and is available for use in future research activities.

Throughout the project, several ideas for future research activities have emerged and are presented below. These ideas encompass both extensions of the current project and potential improvements aimed at enhancing the reliability of the experiments conducted and the overall results.

### Labeling Process

As discussed in Section 3.2, various approaches exist for labeling the data. For point or interval prediction, more advanced labeling processes could be implemented by considering the trend of the time series. In future work the different labeling process could be compared to each other.

### Hyperparamter Tuning

As delineated in Section 4.1, hyperparameter tuning was constrained to 200 trials, with each trial executed twice. Future research could benefit from a more comprehensive hyperparameter search and evaluation. Moreover, in this thesis, the tuning process was conducted using data from an average patient, and the resulting hyperparameters were subsequently applied to all other models. As discussed in Section 4.1, alternative approaches for hyperparameter tuning include conducting the process for each dataset individually or using combined validation sets. Future research could investigate whether the overall performance would significantly improve if the hyperparameters were tailored to each dataset individually.

### Visibility Graph Generation

To generate the visibility graphs from the time series data in this thesis, the natural visibility method is utilized. Various approaches exist for generating graph representations, some of which are outlined below. First, it would be valuable to evaluate whether other algorithms, such as horizontal visibility [40], might yield better-performing graph representations. Second, the embeddings of the nodes and edges could be enriched with additional features. For instance, node features could include the degree of the node, and edge features could incorporate the slope of a connection. Finally, to limit connections to the local neighborhood, only edges with a certain maximal distance could be added to the graph.

### Data Augmentation

Each dataset contains a finite number of CGM values. Since the generalization and performance of machine learning models can often be improved with more data, one approach could be data augmentation. Based on the existing time series data, additional data could be generated to train or evaluate the model. A similar approach would be to use artificially generated data from a diabetes simulator like the UVA/PADOVA [26]. This method allows for generating a large number of samples in a cost- and time-efficient manner.

### Feature Engineering

In future work, additional features could be incorporated into the training process. In addition to the raw CGM values, properties of the time series could be included as features. For instance, features such as the slope or the difference in CGM values between two consecutive time steps could be considered. More potential features that can be extracted from CGM data are detailed in the work of Dave *et al.* [30].

### Repeated Runs

In this work the model evaluation was performed just once and therefore the results are susceptible to random variability. To mitigate the influence of these random factors and improve the reliability of the findings, multiple evaluations and averaging the results across all runs could be considered.

### Multiclass Classification

Instead of binary classification that distinguishes hypoglycemia and nonhypoglycemia, a multiclass task could be explored. Alongside the existing positive and negative classes, introducing a third class could smooth the transition between positive and negative. This approach might help mitigate false positives.

### Expand Prediction Horizons

To assess whether the GNN model consistently maintains stable performance across increasing prediction horizons, additional values could be tested. Furthermore, examining the standard deviation of the $F_2$-Score across all datasets would provide insight into the consistency of results.

# Appendix A

# Results of all models

| horizon window | $F_2$-Score 15 | 30 | 60 | Precision 15 | 30 | 60 | Recall 15 | 30 | 60 |
|---|---|---|---|---|---|---|---|---|---|
| **30** | **0.503** | **0.376** | **0.264** | 0.211 | **0.146** | 0.103 | **0.868** | **0.745** | **0.626** |
| **60** | 0.501 | 0.365 | 0.262 | 0.214 | 0.144 | **0.105** | 0.846 | 0.703 | 0.568 |
| **120** | 0.502 | 0.348 | 0.255 | **0.224** | 0.143 | **0.105** | 0.805 | 0.650 | 0.502 |

Table A.1: Aggregated results of the LSTM architecture 1.

| horizon window | $F_2$-Score 15 | 30 | 60 | Precision 15 | 30 | 60 | Recall 15 | 30 | 60 |
|---|---|---|---|---|---|---|---|---|---|
| **30** | **0.449** | **0.346** | **0.236** | 0.185 | **0.139** | 0.100 | **0.864** | **0.712** | **0.579** |
| **60** | 0.459 | 0.314 | 0.228 | **0.191** | 0.125 | 0.105 | 0.850 | 0.688 | 0.485 |
| **120** | 0.415 | 0.330 | 0.230 | 0.162 | 0.136 | **0.109** | 0.838 | 0.668 | 0.514 |

Table A.2: Aggregated results of the LSTM architecture 2.

| horizon window | $F_2$-Score 15 | 30 | 60 | Precision 15 | 30 | 60 | Recall 15 | 30 | 60 |
|---|---|---|---|---|---|---|---|---|---|
| **30** | 0.462 | 0.350 | 0.261 | 0.190 | 0.135 | 0.106 | **0.892** | **0.729** | **0.594** |
| **60** | **0.501** | **0.357** | 0.256 | **0.213** | **0.138** | 0.102 | 0.853 | 0.717 | 0.592 |
| **120** | 0.477 | 0.340 | **0.273** | 0.196 | 0.133 | **0.108** | 0.860 | 0.703 | 0.570 |

Table A.3: Aggregated results of the LSTM architecture 3.

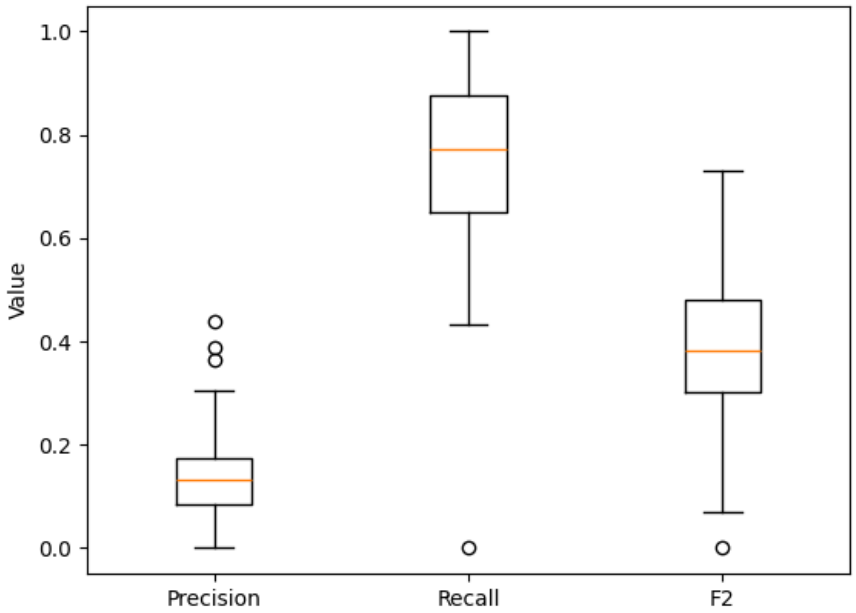



Figure A.1: Distribution of the metrics from the LSTM architecture 1 for window and prediction horizon 30 minutes over the 37 datasets.

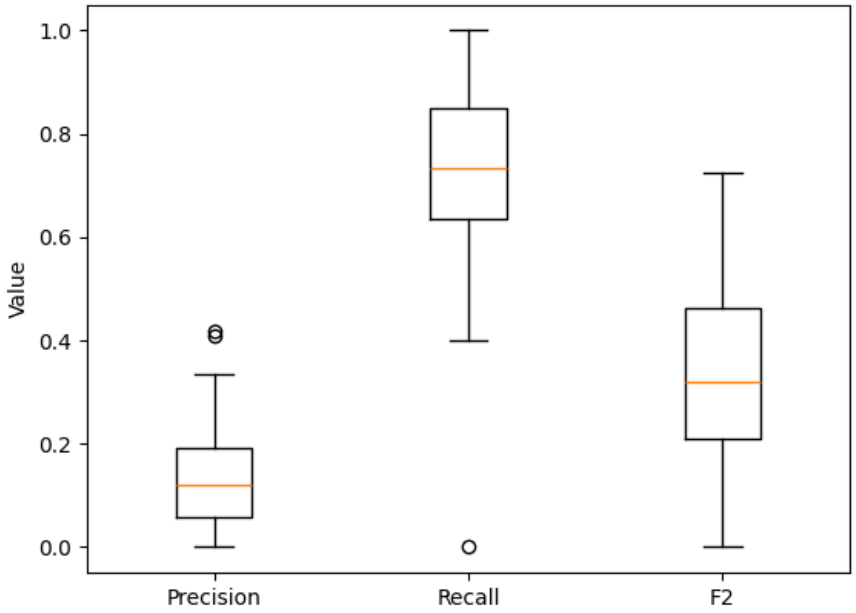



Figure A.2: Distribution of the metrics from the LSTM architecture 2 for window and prediction horizon 30 minutes over the 37 datasets.

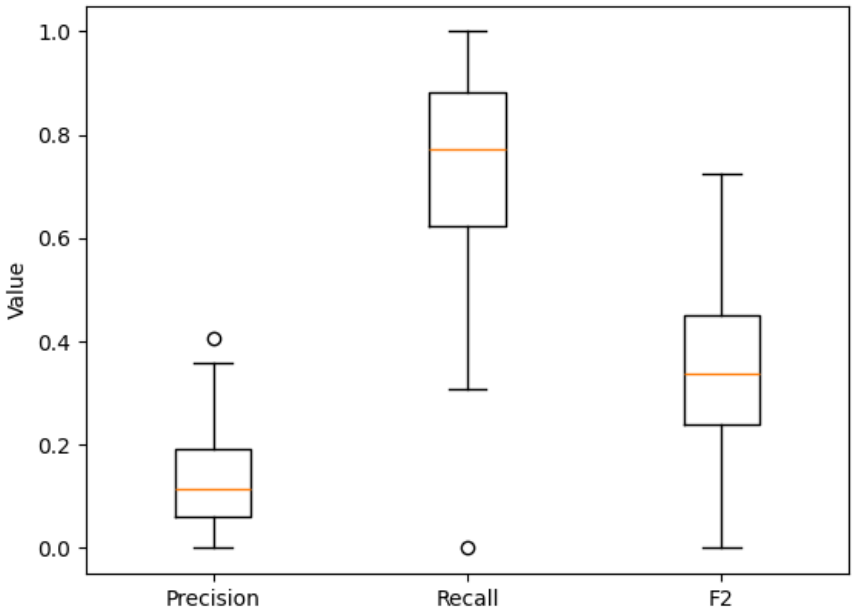



Figure A.3: Distribution of the metrics from the LSTM architecture 3 for window and prediction horizon 30 minutes over the 37 datasets.

| | $F_2$-Score | | | Precision | | | Recall | | |
|---|---|---|---|---|---|---|---|---|---|
| **horizon** / **window** | **15** | **30** | **60** | **15** | **30** | **60** | **15** | **30** | **60** |
| **30** | **0.383** | 0.375 | **0.382** | **0.129** | 0.123 | **0.127** | 0.796 | **0.809** | 0.789 |
| **60** | 0.371 | **0.382** | 0.382 | 0.120 | **0.129** | **0.127** | **0.814** | 0.799 | 0.793 |
| **120** | 0.374 | **0.391** | 0.365 | 0.122 | 0.133 | 0.117 | 0.799 | 0.792 | **0.806** |

Table A.4: Aggregated results of the GNN variant 1.

| | $F_2$-Score | | | Precision | | | Recall | | |
|---|---|---|---|---|---|---|---|---|---|
| **horizon** / **window** | **15** | **30** | **60** | **15** | **30** | **60** | **15** | **30** | **60** |
| **30** | 0.388 | 0.408 | **0.403** | 0.133 | 0.147 | 0.142 | **0.779** | 0.755 | 0.770 |
| **60** | **0.412** | 0.400 | 0.398 | **0.150** | 0.141 | 0.140 | 0.767 | **0.772** | **0.779** |
| **120** | 0.397 | **0.409** | 0.396 | 0.139 | **0.148** | **0.143** | 0.770 | 0.760 | 0.761 |

Table A.5: Aggregated results of the GNN variant 2.

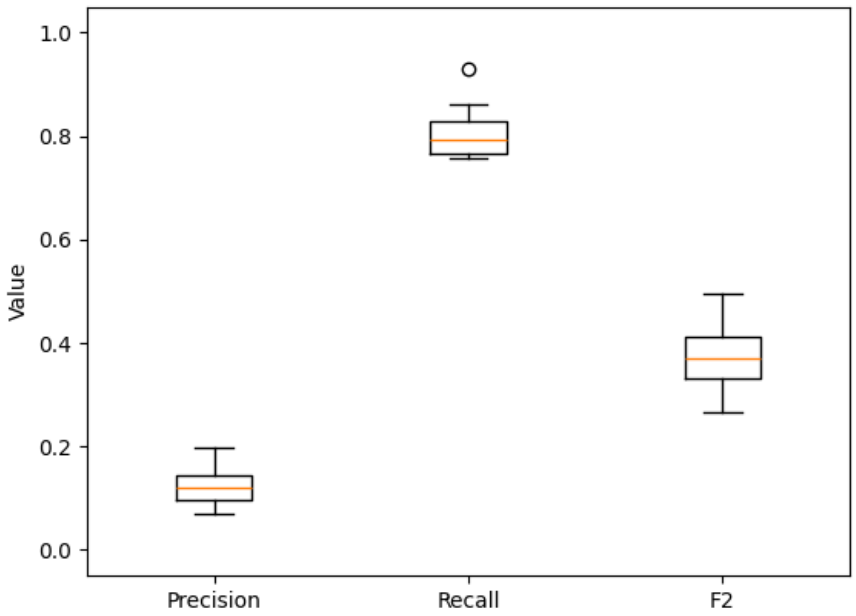



Figure A.4: Distribution of the metrics from the GNN architecture 1 for window and prediction horizon 30 minutes over the 37 datasets.

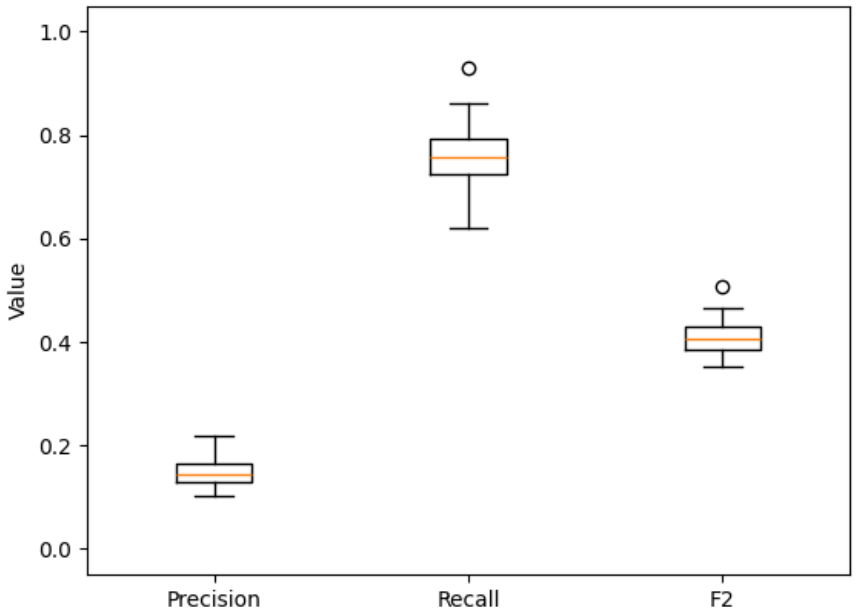



Figure A.5: Distribution of the metrics from the GNN architecture 2 for window and prediction horizon 30 minutes over the 37 datasets.

# Bibliography


[1] American Diabetes Association. Diagnosis and classification of diabetes mellitus. *Diabetes Care*, 33 Suppl 1(Supplement_1):S62–9, January 2010.

[2] Varun Pathak, Nupur Madhur Pathak, Christina L O'Neill, Jasenka Guduric-Fuchs, and Reinhold J Medina. Therapies for type 1 diabetes: Current scenario and future perspectives. *Clinical Medicine Insights: Endocrinology and Diabetes*, 12:1179551419844521, May 2019.

[3] Elizabeth R Seaquist, John Anderson, Belinda Childs, Philip Cryer, Samuel Dagogo-Jack, Lisa Fish, Simon R Heller, Henry Rodriguez, James Rosenzweig, and Robert Vigersky. Hypoglycemia and diabetes: a report of a workgroup of the american diabetes association and the endocrine society. *Diabetes Care*, 36(5):1384–1395, May 2013.

[4] David Rodbard. Continuous glucose monitoring: A review of successes, challenges, and opportunities. *Diabetes Technology and Therapeutics*, 18 Suppl 2(S2):S3–S13, February 2016.

[5] Stephanie A Amiel. The consequences of hypoglycaemia. *Diabetologia*, 64(5):963–970, May 2021.

[6] Morten H Jensen, Claus Dethlefsen, Peter Vestergaard, and Ole Hejlesen. Prediction of nocturnal hypoglycemia from continuous glucose monitoring data in people with type 1 diabetes: A proof-of-concept study. *Journal of Diabetes Science and Technology*, 14(2):250–256, March 2020.

[7] Yuchen Jiang, Xiang Li, Hao Luo, Shen Yin, and Okyay Kaynak. Quo vadis artificial intelligence? *Discover Artificial Intelligence*, 2, 12 2022.

[8] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, April 2021.

[9] Francesca Iacono, Lalo Magni, and Chiara Toffanin. Personalized lstm-based alarm systems for hypoglycemia and hyperglycemia prevention. *Biomedical Signal Processing and Control*, 86:105167, 2023.

[10] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. `https://D2L.ai`.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Xia Zhao, Limin Wang, Yufei Zhang, Xuming Han, Muhammet Deveci, and Milan Parmar. A review of convolutional neural networks in computer vision. *Artificial Intelligence Review*, 57(4), March 2024.

[13] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.

[14] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[16] Christian Bakke Vennerød, Adrian Kjærran, and Erling Stray Bugge. Long short-term memory rnn, 2021.

[17] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[18] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017.

[19] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[20] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 35(2):2708–2718, February 2024.

[21] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[22] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[23] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[24] Omer Mujahid, Ivan Contreras, and Josep Vehi. Machine learning techniques for hypoglycemia prediction: Trends and challenges. *Sensors*, 21(2), 2021.

[25] Sang-Min Lee, Dae-Yeon Kim, and Jiyoung Woo. Glucose transformer: Forecasting glucose level and events of hyperglycemia and hypoglycemia. *IEEE Journal of Biomedical and Health Informatics*, 27(3):1600–1611, 2023.

[26] Chiara Dalla Man, Francesco Micheletto, Dayu Lv, Marc Breton, Boris Kovatchev, and Claudio Cobelli. The uva/padova type 1 diabetes simulator: New features. *Journal of Diabetes Science and Technology*, 8(1):26–34, 2014. PMID: 24876534.

[27] Arthur Bertachi, Clara Viñals, Lyvia Biagi, Ivan Contreras, Josep Vehí, Ignacio Conget, and Marga Giménez. Prediction of nocturnal hypoglycemia in adults with type 1 diabetes under multiple daily injections using continuous glucose monitoring and physical activity monitor. *Sensors*, 20(6), 2020.

[28] Cindy Marling and Razvan Bunescu. The OhioT1DM dataset for blood glucose level prediction: Update 2020. *CEUR Workshop Proceedings*, 2675:71–74, September 2020.

[29] Ravi Reddy, Navid Resalat, Leah M. Wilson, Jessica R. Castle, Joseph El Youssef, and Peter G. Jacobs. Prediction of hypoglycemia during aerobic exercise in adults with type 1 diabetes. *Journal of Diabetes Science and Technology*, 13(5):919–927, 2019. PMID: 30650997.

[30] Darpit Dave, Daniel J. DeSalvo, Balakrishna Haridas, Siripoom McKay, Akhil Shenoy, Chester J. Koh, Mark Lawley, and Madhav Erraguntla. Feature-based machine learning model for real-time hypoglycemia prediction. *Journal of Diabetes Science and Technology*, 15(4):842–855, 2021. PMID: 32476492.

[31] Jose Garcia-Tirado, Patricio Colmegna, Orianne Villard, Jenny Diaz Castañeda, Rebeca Esquivel-Zuniga, Chaitanya Koravi, Charlotte Barnett, Mary Oliveri, Morgan Fuller, Sue Brown, Mark Deboer, and Marc Breton. Assessment of meal anticipation for improving fully automated insulin delivery in adults with type 1 diabetes. *Diabetes care*, 46, 07 2023.

[32] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuño. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13):4972–4975, April 2008.

[33] Oleksandr Ferludin, Arno Eigenwillig, Martin Blais, Dustin Zelle, Jan Pfeifer, Alvaro Sanchez-Gonzalez, Wai Lok Sibon Li, Sami Abu-El-Haija, Peter Battaglia, Neslihan Bulut, Jonathan Halcrow, Filipe Miguel Gonçalves de Almeida, Pedro Gonnet, Liangze Jiang, Parth Kothari, Silvio Lattanzi, André Linhares, Brandon Mayer, Vahab Mirrokni, John Palowitch, Mihir Paradkar, Jennifer She, Anton Tsitsulin, Kevin Villela, Lisa Wang, David Wong, and Bryan Perozzi. TF-GNN: graph neural networks in tensorflow. *Computing Research Repository*, abs/2207.03522, 2023.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[35] J. Mockus. On the bayes methods for seeking the extremal point. *IFAC Proceedings Volumes*, 8(1, Part 1):428–431, 1975. 6th IFAC World Congress (IFAC 1975) - Part 1: Theory, Boston/Cambridge, MA, USA, August 24-30, 1975.

[36] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. `https://github.com/keras-team/keras-tuner`, 2019.

[37] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[38] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022.

[39] Christopher Duckworth, Matthew J Guy, Anitha Kumaran, Aisling Ann O'Kane, Amid Ayobi, Adriane Chapman, Paul Marshall, and Michael Boniface. Explainable machine learning for real-time hypoglycemia and hyperglycemia prediction and personalized control recommendations. *Journal of Diabetes Science and Technology*, 18(1):113–123, January 2024.

[40] Bartolo Luque, Lucas Lacasa, Fernando Ballesteros, and Jordi Luque. Horizontal visibility graphs: Exact results for random time series. *Physical Review E*, 80:046103, 10 2009.