Neural network and graph neural network surrogate models for spatially resolved models in computational epidemiology

Master Thesis

submitted for the degree of Master of Science in Information Systems

by **Agatha Schmidt**

at the Faculty of Management, Economics and Social Sciences of the University of Cologne and in Cooperation with the German Aerospace Center (DLR)

Examiner: **Prof. Dr. Michael Felderer**
Second Examiner: **Dr. Martin Joachim Kühn**

Cologne, May 2024

UNIVERSITÄT
ZU KÖLN

DLR

Contents

List of Figures

1 Introduction

The SARS-CoV-2 virus that emerged in 2019 led to 760 million confirmed cases of the COVID-19 disease and 6,9 million deaths worldwide [1] and showed the crucial need for counter strategies to pandemic threads. Plenty of countries reacted, among others, with lockdowns [2] that included closures of schools, restaurants, offices and general curfews.

Expert models in the field of epidemiological modelling carry great potential for high quality predictions and assessment of containment strategies. MEmilio [3] is a high performance modular epidemics simulation software for the assessment of infectious disease spread and its related outcomes. Aside from agent-based models, it offers compartment models developed for SARS-CoV-2. The main principle of compartment models is the aggregation of individuals of a population based on their infection status. MEmilio leverages local models through generalized, semi-discrete graph approaches to realize realistic human mobility behavior. As these approaches need a variety of input from experts of different domains, we often call these models expert models, in contrast to pure data-driven models ignoring any mechanistic input knowledge.

One of the main characteristics of MEmilio's model is the implementation of non-pharmaceutical-interventions that change the contact patterns between individuals and therefore potentially the course of the pandemic. The contact reductions can be applied for specific age groups and localities at selected times and to a particular extent. The corresponding simulation outputs can become an important tool for decision makers to ensure the functioning of critical infrastructure such as hospitals or to find suitable interventions with optimized severity and, thus, reduced economic and individual damage. Multiple strategies can be evaluated in simulations, which show how the regulatory actions influence the course of the pandemic.

For the development of accurate compartment models disease-specific information such as transmission rates and infection rates has to be available or needs to be estimated from recorded data. In cases when this information is not known, supervised neural networks (NN) can be a useful tool for predicting the spread of the diseases efficiently. The strength of NNs lies in solving problems where the underlying mathematical connection between the input and the according labels is unknown. Furthermore, NNs also bear the potential in fields where knowledge about the existing mechanism is available. For example, physics-informed neural networks incorporate known mathematical functions into the NN [4]. Another application is using NNs as surrogate models for models based on differential equations [5]. In this case, the NN is employed as an computationally efficient approximator of the original model by using the original models data as input and label for the training process. A compromise between classic NNs and spatially resolved expert models is the use of graph neural networks (GNN). In GNNs, objects as well as the relationship between these objects can be modeled. This way, spatial features such as locality and proximity are represented.

The main goal of this thesis is to research on NNs and GNNs as data-driven surrogate models for particular ordinary differential equations (ODE-) and hybrid graph-ODE-based models in MEmilio [6]. The motivation for NNs and GNNs is to replace relatively costly expert models to enable decision makers just-in-time decisions that are necessary in critical situations such as epidemics. To this end, we will generate data with the expert models from MEmilio and train our machine learning models with this data for subsequent test predictions. In a first step, we will create surrogate models for a (partially) homogeneous population. These NNs are intended to replace particular ODE-based models from MEmilio. The graph-ODE-based model from MEmilio captures all 400 counties of Germany. We will introduce a GNN as a surrogate for the graph-ODE. Further, we are going to test the models for predictions up to 150 days and we will implement up to five changes in contact patterns.

This work is structured as follows: first, in Section 2, we are going to introduce compartment models and present examples of their application in the field of COVID-19. In Section 3, we will explain the working principles of NNs and present the NN architectures that will be realized in this work. A special focus will be dedicated to GNNs. Then, in Section 4, we are going to conduct several experiments with the goal to find the best model architectures for the approximation of the graph-ODE-based compartment model. The resulting model architectures will then be tested in multiple experiments in Section 5. These experiments will include the application of contact reductions to the model. Finally, the results will be discussed in Section 6.

2 Epidemiological Compartment Models

Infectious diseases can be modeled with various mathematical approaches such as agent based models, network models or compartment models. Agent based models, for example, simulate the behavior of individuals in a population. Agents with different characteristics can interact with each other. This way, populations with high heterogeneity can be modeled. Network models utilize graph structure by representing individuals as nodes and their connections and contacts as edges.
In this chapter, we will to introduce epidemiological compartment models, since they constitute the type of models we are going to replace by data-driven NNs. We will use the ODE-based and graph-ODE models, which are compartment models, for data generation. Consequently, a NN or GNN surrogate model has to predict the number of individuals per compartment.
In Section 2.1, we will explain the basic idea of dividing a population based on their stage of illness, which was introduced by Kermack and McKendrick [7]. Moreover, we are going to outline how their compartment model can be adapted for a more detailed analysis.
Then, in Section 2.2, we will provide some examples for the implementation of compartment models for COVID-19.

2.1 Overview of Epidemiological Compartment Models

The key-characteristic of infectious diseases like COVID-19 or influenza is the transmission, which can occur when an infected person has contact to a susceptible person. Within the time of contact, the transmission of the disease may occur via different ways such as transmission through air with aerosols or a smear infection through touch. The transmission process can be modeled with various model types on different levels of abstraction. Compartment models focus on a macroscopic view of the dynamic of the spread of the disease. This means that instead of considering single agents the models apply an abstraction, aggregating individuals into groups, sub-populations or even whole populations.

Kermack and McKendrick [7] developed the foundation for compartment models, which is known as the Susceptible-Infected-Removed/Recovered (SIR) model. The authors introduced the concept of dividing the population into categories based on the stage of disease. The Susceptible (S) compartment contains individuals who are susceptible to infection. Individuals who are currently infected are represented in the Infected (I) compartment. Individuals can transmit the disease to other susceptible individuals in their infectious period. The last compartment (R) represents recovered individuals, who are immune from infection and those individuals which were removed from the population because of death or by other means. The basic model assumes lifelong immunity, thus no reinfections are considered. The sum of all compartments denotes the total population (N). At this point it should be mentioned that Kermack and McKendrick originally introduced a model based on integro-differential equations, which are

equation that include derivatives and integrals of a function. Even though this original model was more general, their model based on ODEs is the most cited model in the field of epidemiological compartment models. The population at a given time t can therefore be described by: $N = N(t) = S(t) + I(t) + R(t)$. The idea of the SIR model is that individuals can change the compartment based on transition rates. In the simple SIR model with three compartments there are only two transition rates, namely the transmission rate and the recovery rate. The transmission rate ($\lambda(t)$) describes at which rates individuals become infected after coming into contact with an infectious person. In the original implementation of Kermack and McKendricks model, λ is not dependent on t. However, we adapt the λ by adding the time dependency in order to align with the ODE and graph-ODE-based model from [6], which constitutes the data basis for our surrogate NNs and GNNs. The number of daily contacts at time t is described as $\phi(t)$ and the risk of transmission is $\rho(t)$. In other words, $\lambda(t)$ determines at which rate the individuals move from the Susceptible compartment to the Infected compartment.
The recovery rate (T_I) defines the rate at which individuals move from the Infected compartment to the Removed compartment. It can also be described as the average time an individual stays in the Infected compartment. The population-based disease dynamics of an infectious disease can then be described through the following set of (nonlinear) ordinary differential equations:

$$\begin{aligned}
S'(t) &= -\lambda(t)S \\
I'(t) &= \lambda(t)S(t) - \frac{1}{T_I}I(t) \\
R'(T) &= \frac{1}{T_I}I(t) \\
\lambda(t) &= \rho(t)\phi(t)\frac{I(t)}{N(t)}
\end{aligned}$$

with initial conditions:

$$S(0) = S_0 > 0, \qquad I(0) = I_0 > 0, \qquad R(0) = 0.$$

The parameters T_I and $\lambda(t)$ are considered to be non-negative. As the model is formulated by ordinary differential equations, the epidemic process is assumed to be deterministic.

The model presented by Kermack and McKendricks is based on many assumptions and simplifications. It can be augmented by adding new parameters or compartments, to describe the course of the pandemic with more detail. Instead of homogeneous contact rates, which depend on the density of individuals, more specific contact rates, which may depend on age or occupation, can be introduced to the model. Another addition can be the inclusion of demographic data such as birth and death rates, which may lead to a varying population size. The simple SIR model disregards several different phases of infectious diseases, such as the incubation and latent periods [8]. After an individual is exposed to the pathogen, the pathogens need some time to multiply throughout the incubation period. When enough pathogens reach the target organ, the individual might develop symptoms. The period until the individual becomes infectious is called latent period and the infectious period describes the time where the individual is infectious. In order to better model this process of infection, researchers introduced an additional, latent compartment of Exposed (E) individuals. There are multiple use cases of this four-compartment SEIR model for multiple diseases such as malaria [9], ebola [10] and influenza [11].

Furthermore, the assumption of lifelong immunity is not applicable to many diseases, where a waning of immunity takes place after some time. This waning of immunity can be modeled by allowing transitions from the Recovered compartment to the Susceptible compartment. These models are called SIRS models.
Also, Kermack and McKendrick [7] assume a homogeneous population, where every individual is equally susceptible for infection. In reality, the individuals of a population are of different susceptibility due to differences in age or social groups. Implementing age groups in a compartment model allows the use of age group specific transmission rates.
Moreover, vaccination of individuals can be implemented by an additional compartment or adjusted infection rates. When no vaccination is available, other measures such as quarantine might be enacted. Analogously, it is possible to add a compartment with quarantined individuals.
Depending on the stakeholders for which the simulation of a compartment model can be of interest, other compartments can be introduced. For government officials and hospitals it can be of interest how many individuals are predicted to be hospitalized or in need of care in the intensive care unit (ICU).
The following section will show some of the mentioned model adaptions implemented in the context of COVID-19.

2.2 Compartment Models for COVID-19 Analysis

Since SARS-CoV-2 became of global significance, a great number of papers were published with the aim to model the spread of the virus, predict outbreaks and evaluate contact reducing measures with compartment models. The following mentioned papers do not represent an exhaustive overview, but represent chosen examples of particular research endeavors. Moreover, the listed examples show how the model design can be adjusted depending on the focus of the study.

The earliest publications used data from Wuhan in China, which was the source of the pandemic [12]. Tang et al. [13] designed a SIR-type model where quarantine, isolation and treatment were considered. This resulted in a model design with eight compart-

ments. Their study was conducted in the very early stages of the spread of the virus and showed how the reduction of contacts could influence the number of infected individuals. Ndaïrou [14] focused on the influence of super spreaders in an eight-compartment model. Their model did not incorporate quarantined or isolated individuals as compartments. A special feature of their work is the superspreader compartment. The authors were able to show the suitability of their model for the data from the outbreak in Wuhan.
As the virus spread, models for the surrounding regions of Wuhan were built, such as in the paper of Zhao et al. [15], who presented a compartment model for the Chinese Hunan province. Their model is made of five compartments, namely Susceptible, Exposed, Symptomatic, Asymptomatic and Recovered/Removed and they introduced four age groups to the model.
Spring 2020 was characterized by lockdowns in many countries of the world [2]. Acemoglulo et al. [16] analyzed the effectiveness of lockdowns based on data from Korea. An age structured SIR-model was used to analyze the effects of lockdown strategies for specific age groups. The results showed that stricter protective measures for vulnerable groups such as older people are more beneficial than lockdowns for all individuals of a population. The influence of age on the course of infection was also part of the study of Balabdaoui et al. [17], who divided the population into 17 age groups. The authors created a model with nine compartments for Switzerland and took different meeting places into consideration such as home, work and school.
In October of 2020 Batistela et al. [18] published a paper which focused on immunity and reinfection. A four-compartment model was constructed and the authors included immunity, reinfection, birth and death to create scenarios for the course of the pandemic considering different reinfection rates. Further, the authors provided estimates for duration and peaks of outbreaks and drew scenarios for up to one and a half years for the state of São Paulo of Brazil.

For Germany, Barbarossa et al. [19] developed a compartment model in the early stage of the pandemic. Their aim was to predict more than one year from the beginning of the epidemic for different scenarios. Scenarios with different types of non-pharmaceutical control measures were analyzed in the study. In [6], a spatially resolved hybrid graph-ODE-based model has been proposed to include realistic human mobility and to reliably predict infectious disease dynamics locally. In April 2021, when vaccines were available, Dashtbali and Mirzaie [20] presented a model that involved vaccinations by establishing a semi-susceptible compartment of individuals which are less susceptible for infection than the rest of the individuals in the susceptible compartment. Their model integrating vaccination was able to provide a better fit for the real world data, including Germany. To maintain mobility as a social good while vaccination was still in progress, the authors of [6] researched on advanced testing strategies to control the virus dynamics on a local scale

3 Artificial Neural Networks

Artificial Intelligence (AI) encompasses a variety of methods which can be useful for a multitude of problems. In this chapter, we will introduce basic knowledge about the operating principles of Artificial Neural Networks, also called Neural Networks (ANN or NN) and Graph Neural Networks (GNN). These model types constitute a subset of AI technologies. We will introduce a selection of specific model architectures in order to create a basis for understanding the deployment of these models in this work. The most common NN structure, the Multilayer Perceptron (MLP) and its learning algorithm will be presented in Section 3.1.1, as an introduction to NNs. In Section 3.1.2, we are going to characterize the Convolutional NN (CNN), and in Section 3.1.3, the Long Short-Term Memory (LSTM) NN. In Section 3.2, we will introduce GNNs and exemplify multiple GNN types and describe their strengths and exemplary applications. In Section 3.3, we will present some examples of NNs for epidemiological COVID-19 modelling.

3.1 Neural Networks

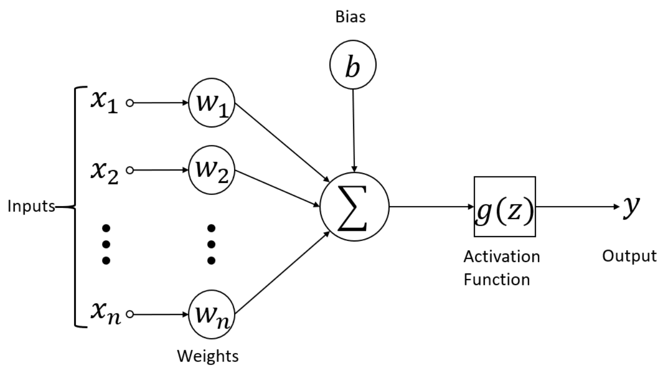
NNs are a tool in machine learning and can be developed for multiple types of tasks, including non-linear problems. Their advantage compared to classical statistical methods is that only little prior knowledge about the data is needed [21]. A task in machine learning can be supervised or unsupervised. Supervised learning involves training a model on labeled data, where the algorithm learns to make predictions based on input-output pairs, while unsupervised learning involves training on unbalanced data, where the algorithm discovers unknown pattern and structures. NNs can be applied for both types of task, but the application in supervised learning is more common. In the following introduction of different types of NNs, we will assume a supervised task for all algorithms, since in this thesis only supervised learning will be applied. In the supervised learning task in this thesis, the ODE and graph-ODE compartment models will produce the input data as well as the labels for the NN and GNN surrogate models.

3.1.1 Multilayer Perceptron

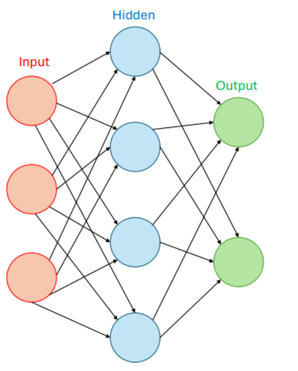
Structure of the Multilayer Perceptron

The most common and basic type of NNs are feed-forward NNs, also called MLPs. The fundamental objective of a feed forward network lies in approximating a given function, denoted as f^* [22]. For instance, in a classification task, $y = f^*(x)$ signifies the mapping of an input x to a corresponding category y. In a regression task y is a continuous value instead of a category. These models are termed "feed forward" because information travels through the evaluated function, progressing from the input x, through the intermediary computations defining f, and ultimately arriving at the output y. The term "network" deduces from the model architecture, which is composed of multiple layers. Each layer can be described as a function. In an example with three layers, we would have $f^{(1)}$ for the first layer, which is called input layer, and $f^{(2)}$ and $f^{(3)}$ for the second and third layer respectively. The overall number of layers defines the depth of the network. The final layer of a NN is called output layer and it must produce an output value

$f(x) = f^{(3)}(f^{(2)}(f^{(1)}((x)))$ that is close to the provided label y. Consequently, during training, the goal is to push $f(x)$ to match $f^*(x)$. The other layers, except the output layer, are also called hidden layers and do not have a objective function they have to approach. Only the output value is evaluated in training. Figure 1(b) illustrates the layer architecture and shows how the layers are connected sequentially in a feed forward NN. Finally, these networks are called "neural" networks because they are inspired by the way the biological neural system works. Each layer is composed of artificial neurons, which are interconnected and the quantity of them per layer determines the width of the NN. A neuron receives signals from multiple other neurons and summarizes these signals by calculating a weighted sum. The weights w, needed for this calculation are assigned to the connection between two neurons, are randomly initialized in the beginning and subsequently updated in the training process. As a weight is assigned to a pair of neurons, it can be interpreted as an indicator for the strength of a connection between two neurons. A neuron-specific value called bias b is added to the sum of weights. Consequently, the neuron is always activated, even if the input is zero. To be precise, the hidden unit obtains a vector of inputs $\mathbf{x}$ and uses the vector of weight $\mathbf{w}$ and neuron-specific bias b to calculate $z = \mathbf{wx} + b$. Weight and biases are learned parameters and updated throughout the training process. The resulting value z is then forwarded to a threshold function $g(z)$, which is called activation. The neuron dispatches an output value based on the activation function, which can be forwarded to neurons in the following layer. Popular activation functions are for instance the linear activation function $g(z) = z$, the relu activation $g(z) = max(0, z)$ or the tanh activation $g = tanh(z)$. The structure of a single neuron is illustrated in Figure 1(a).



(a) Architecture of a single neuron



(b) Layer structure

Figure 1: Visualization of MLP

These are own illustrations based on Figure 2 in [23] and Figure 8 in [24].

Learning Algorithm of the Multilayer Perceptron

As described before, the goal in a regression task is to build a model that is capable of producing an output value $\hat{\mathbf{y}}$, that is as close as possible to the continuous label value $\mathbf{y}$.

To reach this goal, the NN has to undergo a training process. A common method is the back-propagation algorithm [25]. The back-propagation algorithm alone is not sufficient to train a model, because it only calculates the gradient. Further, an optimizer is needed to perform learning using the gradient.

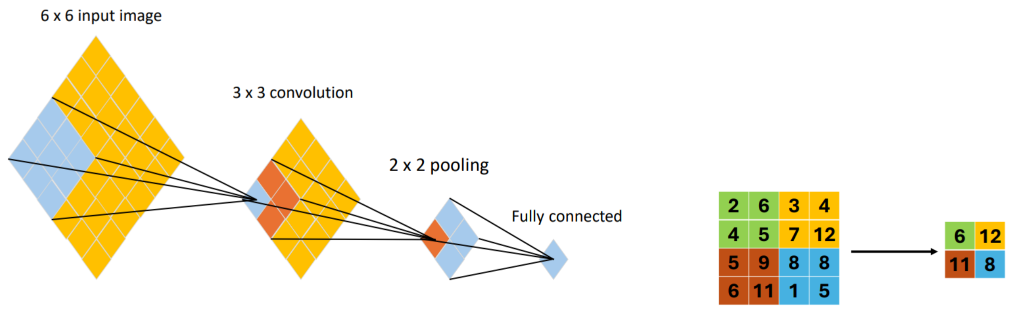
The back-propagation algorithm can be divided into two phases, namely the forward pass and the backward pass. In the forward pass, weights and biases are initialized randomly and a training set is passed to the first layer of the NN, called input layer. The information is passed from neuron to neuron through the layers of the network as described before. Then the difference between the output from the output layer $\hat{\mathbf{y}}$ and the label $\mathbf{y}$ is calculated, which is called error or loss. Depending on the task, different loss functions are applied as ways to determine the loss. A classification task would use accuracy, which describes the percentage of correct predictions, while a regression task could use mean absolute error, mean squared error or mean absolute percentage, just to name a few. The optimization goal of the NN is to minimize this loss function. As described before, the output of the activation function of neurons z depends on its weights and biases as well as on the outputs of the previous neurons which likewise depend on their weights and biases. Despite all weights and biases needing to be updated, the neurons of the output value are the only ones that can be evaluated with labels. The data does not provide any information about the values the neurons in the hidden layers should attain. This is the reason why back propagation is needed in order to update the neurons of all layers. Therefore, the backward pass starts at the output layer. First, the gradient of the loss function $L(\hat{\mathbf{y}}, \mathbf{y})$ with respect to the output of the last layer is computed. The gradient indicates, in which direction the value of the neuron would have to change, in order to obtain more accurate results. Then, this gradient is used by an optimization algorithm such as stochastic gradient descent (SGD) or Adam, to update the weights and biases of the output layer. Next, we want to update the weights and biases of the hidden layer right before the output layer. A specific chain rule of the derivative that can be applied on computational graphs, makes it possible to compute the gradient of the neuron and therefore determine, what influence the weights and biases of this neuron have on the output of the output neuron. This way, we can calculate the gradients of all neurons in all layers and change their weights ans biases so that we obtain an improvement of the output value of the NN. This process is repeated iteratively after each forward pass, until no improvement can be achieved by adjusting weights and biases.

The performance of a neural network is evaluated on a test dataset, which contains data samples that were not involved in the training process. Depending on the task, there are several options to determine the goodness of fit of the models predictions to the provided labels, based on different performance measures.

3.1.2 Convolutional Neural Networks

CNNs [26] are a specific type of feed forward NNs and are specialized on data with a grid-like topology. Images can be considered grid-structured because images are composed of pixels that are arranged in a grid-like topology. The term "convolutional" describes that a convolutional operation takes place. The convolutional operation is performed by a filter, also known as kernel. The filter has a predefined receptive field [27] which determines how many pixel the filter scans at once. In Figure 2(a) we can see the receptive field in the grid on the very left highlighted in light blue color. The process of convolving over an image describes the filter sliding over the image, always capturing the pixels in its receptive field. For example, it could be a 3x3 filter, that analyzes nine pixels at a time. Mathematically, the filter is a multidimensional array of parameters K that applies the convolutional operation on multidimensional array of input data I. In a two dimensional setting we have $S(i, j) = (I * K)$, where the asterisk $(*)$ denotes the convolutional operation and S is the resulting feature map [22]. A feature map usually has smaller dimensions than the input, because the filter is usually smaller than the input. It is called feature map, because it represents the presence of certain features or patterns in the input. Thanks to the convolutional operation, the CNN can share parameters and benefit in efficiency. In contrast to a MLP, where all neurons are densely connected, meaning that a neurons receives signals from all neurons from the previous layer, the filter reduces the number of signals that have to be processed. In case of an image as input, not every pixel is assigned to a neuron. Instead, the filter scans all pixels in its receptive field and outputs a value based on its function, which summarizes information about the pixels and its respective neighbors [28].

After the input data was processed by the input layer and the filter which is positioned in a so called convolutional layer a pooling layer can follow. The task of the pooling layer is to reduce dimension of the feature map, which is outputted by the convolutional layer. Pooling operations replace a local region of the input data with a single representative value. The most common pooling operation is max pooling, where the maximum value within each local region is retained, effectively downsampling the feature map. An example of the max pooling is illustrates in Figure 2(b), where each 2×2 square is summarized by a single value representing the highest value of the 2×2 square.



(a) Visualization of CNN structure **(b)** Max Pooling operation

Figure 2: Visualization of CNN
These are own illustrations based on Figures 10.22 and 10.8 in [27].

CNNs are applicable to graph-structured data. Instead of pixels, the graph data contains nodes and the CNN summarizes the information of all neighboring nodes of a central node. The structure is not perfectly grid-like like in an image, but the filter can identify neighboring nodes by following the edges. More details about graph convolution will be presented in Section 3.2. Moreover, CNNs can be applied to time series data. There, a one-dimensional convolutional layer performs a convolution along the time axis of the data. Even though other NN architectures that were specially designed for time series data exist, CNNs applied on time series data have the advantage of a small number of trainable weights, which results from the convolutional structure [29].

3.1.3 Long Short-Term Memory Neural Networks

The LSTM as proposed by Hochreiter and Schmidhuber [30] is a type of recurrent NN. There are multiple design option for recurrent NNs (RNN) and we will shortly introduce one option. Recurrent NNs are NNs that are designed to handle sequential data $\mathbf{x}^{(1)}, ...\mathbf{x}^{(T)}$ by handling the input data step by step, storing a hidden state $\mathbf{s}^{(t)}$ which is based on previous seen information and is updated after each new input: $\mathbf{s}^{(t)} = f(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t)}; \theta)$ where θ represents the trainable parameters, namely the weights and biases [22]. In a RNN the particular parts of the sequential input data are fed to the input layer consecutively. One key element of RNNs is a feedback loop that directs the output value of the activation to a function that sums all of these values. This process allows all seen values to influence the prediction. As the number of values in the sequential input data increases, the gradient is propagated over many stages which includes multiplying a weight by it self multiple times. That can lead to the occurrence of a vanishing or exploding gradient problem, which makes it impossible to capture long-term dependencies.

LSTMs were designed to address the vanishing gradient problem of RNNs [31]. One of the main characteristics of LSTMs is that they are able to "forget" information, which makes them more efficient. This makes them suitable for time series prediction with input that consists of long sequences.

Hochreiter and Schmidthuber [30] implement their LSTM using a so called memory cell and gate units. The set of memory cells and their gates is placed in the hidden layers of the NN. A single memory cell is composed of the following components. The cell state C_t represents the memory of the LSTM and can store information over long sequences [32]. It can be updated based on the calculations of the forget gate f_t, which determines how much of the long term memory should be remembered, by applying a sigmoid activation. That sigmoid activation outputs a value between 0 and 1, given the input at time t X_t and the output of the last LSTM unit h_{t-1}. Thus, we get the following formula for the forget gate at time step t: $f_t = \sigma(W_f[h_{t-1}, X_t] + b_f$ where b_f are the biases and W_f is the weight matrix of the forget gate. In a next step, the input X_t is passed to two layers, namely a sigmoid and a tanh layer. Figure 3 illustrates the whole process happening in the memory cell. The step just described corresponds to the second and third blue activations counting from the left. The sigmoid layer outputs a value between zero and one indicating whether the information should be updated or ignored and the tanh function assigns a weight to the values which indicates the importance of the information. The resulting new information is added to the old memory C_{t-1} and yields the new memory cell state C_t. A similar process is applied on the output of the last LSTM unit h_{t-1} to determine the output of the current LSTM unit h_t. A sigmoid gate is applied to the last output h_{t-1} and a tanh layer creates new values based on the cell state C_t, which are then combined to calculate the output h_t. These elements form the output gate of the memory cell and are represented by the two blue activation boxes on the right side of the memory cell in Figure 3.

To summarize we can say that the input gate controls which information should be stored in the memory cell, the forget gate decides what information from the previous cell state should be retained and the output gate decides how and when information from the memory cell is passed to other memory cells. Further, the cell state can be characterized as a long term memory which analyzes output of other cells, evaluates the importance and influences the output of the current memory cell.

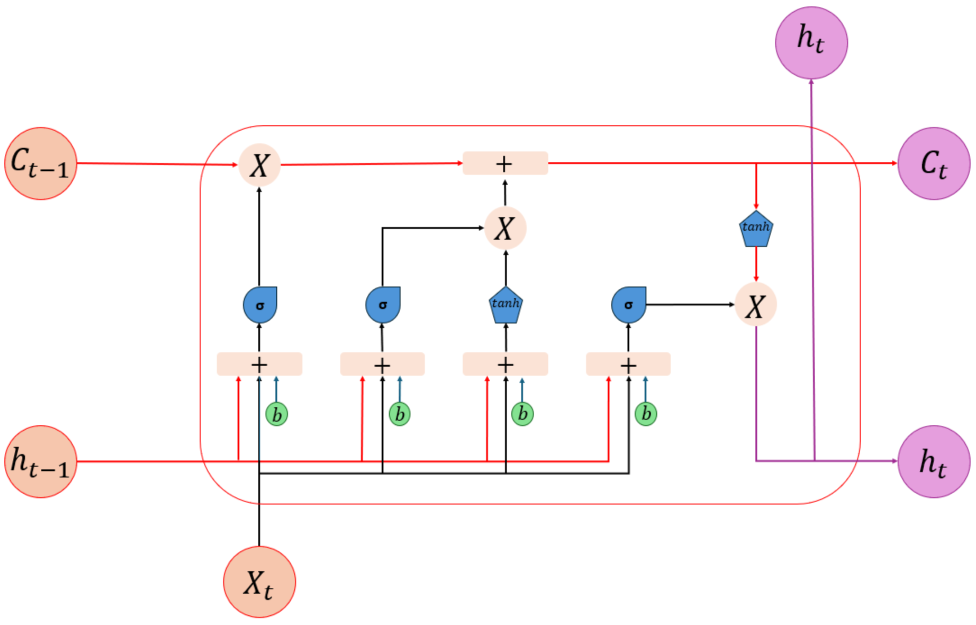
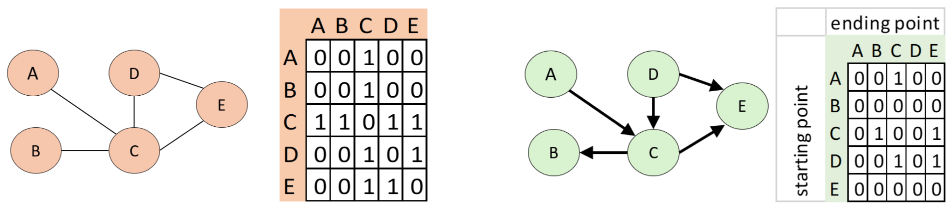


Figure 3: The memory cell of a LSTM
Own illustration based on Figure 4 in [32].

3.2 Overview of Graph Neural Networks

Many phenomena and tasks can be modeled with a graph. Social networks between individuals, biological networks like molecules, which can be described as a networks of atoms, or recommendation systems can all be modeled with a graph. The main idea of a graph representation of data is that nodes represent objects and edges represent the relationship between those objects. A graph G is defined as $G = (N, E)$, where N is a set of vertices, also called nodes, and E is a set of edges. Edges are defined by a pair of nodes, representing the connection between these nodes. In undirected graphs the edges do not have a specified direction associated, indicating a symmetric relationship between two nodes. in directed graphs the edge $e = (x, y)$ with $x, y \in N$ can be described as an arc directed from node x to node y. The adjacency matrix is a square matrix used to represent the connectivity of a graph. The rows and columns of the matrix correspond to the nodes of the graph and the entries indicate whether an edge exists between two nodes. For a graph with n nodes, the adjacency matrix A_{ij} is defined as a $n \times n$ matrix where a_{ij} is either 1 or 0, depending on whether there is an edge from node i to node j. The matrix is symmetric for undirected graphs. An example for an directed graph and an undirected graph with their respective adjacency matrices is illustrated in Figure 4. A specific type of graphs is a weighted graph, where a number is assigned to each edge as the weight.



(a) Undirected graph **(b)** Directed graph

Figure 4: Visualization of adjacency matrices
These are own illustrations based on Figure 13.2 in [27].

Often, classic NN approaches are not the best fit for handling graph structured data because irregular graph structure, dependencies between nodes and message passing between nodes constitute a complex challenge. NNs on graphs, which are often called GNNs, are able to capture the spatial information of the nodes and can be applied for multiple tasks, such as node classification [33], graph classification [34], link prediction [35] and node clustering [36].

Before the GNN as known today was introduced, there were first attempts of applying NNs on graph structured data [37]. A key difference between NN on graphs and GNNs is that GNNs are specially designed to operate on graph-structured data and particular algorithms such as message passing or graph attention are used to propagate information across the graph. The message passing algorithm builds the foundation for many GNNs, hence, we introduce its functionality by presenting the work that first adopted the algorithm into their GNN.

The concept of message passing was introduced by Gori et al. [38] when the authors proposed one of the first GNNs. Their model was based on recursive neural networks and their work laid the mathematical foundation for GNNs. The most significant difference compared to other attempts of processing graph structured data, the data not being forced into the form of vectors of real numbers, which is associated with information loss [39]. Instead, the graph structured data is preserved to a certain degree by keeping

the information about the graph topology. Information about the graph topology is included into the data processing step.
Gori at al.'s model can be described as a recurrent GNN. It is assumed that a node in a graph constantly exchanges information with its neighbors until a stable equilibrium is reached [40]. They differentiate between graph-focused and node-focused problems with the main difference being the type of output the model generates. For graph-focused tasks, the task of the model is to predict the labels on graph level, allowing dimension reduction throughout the training process, whereas for node-focused tasks, an output for each node has to be produced.

Even though Gori et al. [38] introduced the concept of GNNs in 2005, the paper of Scarselli et al. [39] from 2009 popularized the GNN architecture. Their work is based on Gori et al.'s concepts from 2005 and provides a more comprehensive explanation and multiple applications on real-world datasets. Scarselli et al. [39] added the idea of edge features to the GNN concept and present a more detailed explanation of the learning algorithm. We will explain the concept of the GNN based on their work.
A graph $G = (N, E)$ is composed of nodes $n \in N$ and edges $e \in E$. The neighbors of node n are denoted as $ne[n]$, and $co[n]$ represents all edges that have n as a vertex. Nodes as well as edges can have assigned labels and are represented by vectors $\boldsymbol{l}_n \in \mathbb{R}^{l_N}$ for node n and $\boldsymbol{l}_{n_1,n_2} \in \mathbb{R}^{l_E}$ for edge $e = (n_1, n_2)$, where $\boldsymbol{l}_N$ is a vector containing all node labels and $\boldsymbol{l}_E$ is a vector containing all edge labels. The notation for other vectors can be derived analogously to the presented notations. The vector with all labels of the neighbors of node n, for example, are represented by vector $\boldsymbol{l}_{ne[n]}$. The model's state is composed of the node's states $\boldsymbol{x}_n$ attached to node n which is based on the information of the node's neighbors $ne[n]$. The states are updated iteratively by the transition function and the last state is then fed into the the output function. The output of the model is denoted as $\mathbf{o}_n$. The authors define a local transition function f_w, which calculates the node state based on the node labels, edge labels as well as the node labels and node states of all nodes in the neighborhood. The node state $\boldsymbol{x}_n$ can be calculated with

$$\boldsymbol{x}_n = f_w(\boldsymbol{l}_n, \boldsymbol{l}_{co[n]}, \boldsymbol{x}_{ne[n]}, \boldsymbol{l}_{ne[n]}). \tag{1}$$

A local output function g_w calculates the output of the node from the nodes state and the node labels. The output is therefore determined as:

$$\boldsymbol{o}_n = g_w(\boldsymbol{x}_n, \boldsymbol{l}_n). \tag{2}$$

When the next iteration of the state is computed, f_w takes the current node's labels, neighbors' node labels, neighbors' edge labels and the previous states of all neighbors into consideration. The output of a subsequent iteration is based on the previous states as well as the current node labels.
The learning algorithm is based on the gradient descent algorithm and consists of two phases. First, the node state is updated until a stable point is reached. In the second step the gradients are calculated to update the models weights by gradient descent. The learning process repeats the two phases until a stopping criterion is reached.

Several other models are based on the message passing algorithm as presented before. One popular GNN that implements the algorithm is the message passing neural net (MPNN) introduced by Gilmer et al. [41]. The MPNN was build for solving tasks in the domain of quantum chemistry, where molecule properties need to be predicted. On molecular level, geometric characteristics such as atomic distance and bond angles are of particular importance. Thus, capturing these characteristics in a graph can be very useful. The forward pass of the MPNN consists of two phases. First, in the message passing phase, a hidden state function updates the hidden state of the node based on the node's current hidden state and the passed messages. The passed messages are based on the hidden states of the neighboring nodes and on the edge weight between all neighbors of the vertex. This phase is very similar to the mechanics of the GNN proposed by Scarselli et al. [39], as described before. The second phase is the read out phase. A read out function is applied to compute a feature vector for the whole graph using the hidden representations of every node in the graph.

Generally, GNNs can be categorized into different types depending on the task they have to solve or on the way data is processed. One type of GNNs are recurrent GNNs (RecGNN) that allow sequential information to propagate across the nodes. The GNN based on the work from Scarselli et al. [39] described before is a type of RecGNN. Another type of GNNs are graph attention networks that use a so called attention mechanism in the learning process.

In this work we will implement four types of GNNs as an AI based surrogate model for the graph-ODE model in Section 4.4. One of the four GNN layers we will deploy is based on the Graph Attention Network from Veličković et al. [42]. The other three models can be assigned to the group of Graph Convolutional Networks (ConvGNNs).

First, we will focus on the models that can be categorized as ConvGNNs. The main idea of ConvGNNs is to leverage from the powerful qualities of CNNs [27]. CNNs are capable of solving problems on grid-like structures, like the pixel-grid of images. CNNs calculate an aggregated representation of the data by first extracting features from the grid with a filter. Then a pooling layer is applied to downsample the information as described in Section 3.1.2. For ConvGNNs, the grid is replaced by a weighted graph [43]. The pooling operation from CNNs is replaced by graph pooling, which reduces the number of nodes in a graph [44]. Graph pooling includes all operations that reduce the dimension of the graph, meaning the pooling layer reduces the number of nodes by aggregating information of a node and its neighbors, similar to the pooling operation in CNNs. The locality property of images implies that nearby pixels are related to each other and can represent a semantic concept. Images can be considered as a special form of graphs, where each pixel represents a node [40]. Pixels are connected with its direct neighbors and a filter is applied by calculating the weighted average of pixel values of the central node and its neighbors. ConvGNNs generalize the concept of locality to the graph by using the edge weights to represent locality. The edge weight represents the relationship between two nodes. By setting a threshold for the values of the weight matrix, the neighborhood of a node can be defined. This way, not all adjacent nodes are considered neighbors, only the nodes that meet the edge weight criterion.

Kipf and Welling [33] introduced a GNN designed for a semi-supervised node classification task. The model belongs to the class of ConvGNNs and the authors named their model "Graph Convolutional Network" (GCN). This model builds the foundation for the GCNConv layer we will use in Section 4.4. The GCN is a spectral GNN. while the GNN from Scraselli et al. [39] is a spatial GNN. The two groups of GNNs, namely the spectral GNNs and spatial GNNs differ in the way of handling graph data. At this point we should explain the difference between spatial and spectral graph convolutional networks. Spectral GNNs handle the graph from the perspective of graph signal processing. This means that they utilize eigenvalues and eigenvectors of the *Laplacian* matrix to process graph data. The Laplacian L provides useful information of the graph and is calculated by $L = D - A$ where A is the adjacency matrix and D is the degree matrix. The degree matrix D is a diagonal matrix where each diagonal element $d_{i,i}$ represents the degree of the vertex i, i.e. the number of edges incident to vertex i. The convolutional operation can be interpreted using a filter to remove noise from graph signals [40].
Spatial GNNs on the other hand focus on the local connectivity and relationship between nodes, which leads to concepts such as message passing. The convolutional operation in spatial GNNs propagates node information along edges [40]. The representation of the central node is convolved with the representation of the nodes neighborhood to update the central node representation as we have seen in the description of Scraselli's GNN [39].

Now that the difference between spatial and spectral approaches is explained, we show how Kipf and Welling [33] define the information propagation process in their GNN. The layer wise propagation rule is defined as

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}), \tag{3}$$

where $\tilde{A} = A + I_N$ with A being the adjacency matrix and I_N the identity matrix. The summation of the adjacency matrix and the identity matrix leads to an adjacency matrix with added self-loops. Further, σ represents an activation function, $\tilde{D}$ the degree matrix, $W^{(l)}$ the layer specific trainable weight matrix of layer l and $H^{(l)}$ is the matrix of acivations of layer l. The adjacency matrix with added self-loops is used in order to take the node features into account. The degree matrix is used to normalize the adjacency matrix. With their work, they bridged the gap between spatial and spectral GNNs. Because even though the authors GCN is a spectral based method, it can also be interpreted as a spatial based method [40]. In the spectral approach, their calculation of the node hidden feature matrix can be described as the application of a filter which utilizes a modified version of the adjacency matrix on the node features. This expression can be translated into a spatial based approach where, for each node, a hidden representation is calculated from the own node features and the information from neighboring nodes, and then these hidden node representations are aggregated [40]. One limitation of their model is that edge features and directed graphs cannot be modeled.

The next layer we will present is the ARMAConv layer based on the paper from Bianchi et al. (2021) [45]. In their work the authors substituted the common polynomial filter used in GNNs with a filter based on the auto-regressive moving average (ARMA). The ARMAConv GNN utilizes the technique of auto-regressive moving average in the training process. An ARMA model utilizes values and errors from the past to predict the next value of a time series. In the ARMAConv GNN the ARMA principles are implemented into the filter.

The implementation of an $ARMA_1$ filter is realized by a Graph Convolutional Skip (GCS) layer and is based on spectral graph operations. The GCS can be be described by

$$\bar{\mathbf{X}}^{(t+1)} = \sigma(\tilde{\mathbf{A}}\bar{\mathbf{X}}^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V}) \tag{4}$$

where $\mathbf{W}$ and $\mathbf{V}$ are trainable weight matrices, X are the node features and $\tilde{\mathbf{A}}$ is the normalized adjacency $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. $\mathbf{D}$ represents the degree matrix of the graph and $\mathbf{A}$ the adjacency matrix.

A GCS layer recursively updates and iterates until convergence. The authors limit the number for iterations by $T \in \mathbb{N}$. The proporsed GNN consists of K GCS stacks. $K \in \mathbb{N}$ denotes the order of the $ARMA_K$ filter. Each k-th ($k \in \{1, \ldots, K\}$) GCS stack is initialized with random values for $\mathbf{W}_k$ and $\mathbf{V}_k$. This way, it is ensured that each GCS stack outputs a different result. The result is a node feature representation. The output of a GCS stack k is the last node feature representation at iteration T and is denoted as $\bar{\mathbf{X}}_k^{(T)}$ The output of the $ARMA_K$ filter is based on summarizing all k GCS stacks:

$$\bar{\mathbf{X}} = \frac{1}{K}\sum_{k=1}^{K}\bar{\mathbf{X}}_k^{(T)}. \tag{5}$$

The authors made several experiments to test their GNN. They applied it on node classification, graph classification and graph regression tasks. All datasets were not temporal data. The MINST dataset for example is a dataset that contains handwritten images. The temporal aspect of ARMA is not related to the type of input data. Instead, the ARMAConv model uses the ARMA principles in the learning process in form of a filter. The model does not iterate through the input sequence learning the prediction of next output features via ARMA. Instead, it updates the node representations $\bar{\mathbf{X}}$ in the learning process by using past node representations.

The APPNPConv Layer is the third layer we present and the last layer that is categorized as a ConvGNN. It is based on the work from Gasteiger et al. [46]. They proposed a method for personalized propagation of neural predictions (PPNP), and its fast approximation, APPNP. Contrary to many other approaches, the authors did not implement a message passing algorithm. Instead, they proposed a novel propagation algorithm based on PageRank [47]. PageRank was originally developed to rank web pages in search engine results based on their importance and relevance. The system of web pages is graph-structured, thus the algorithm can be a tool for GNNs. The PageRank algorithm is based on the random walk algorithm, which is a process that describes that a path of edges in a graph is formed by randomly choosing the next adjacent edge. Gasteiger et al. [46] implemented a method that includes a possibility of returning to the original node while performing the random walk. The process of returning is called teleportation and the teleportation portability α aids in balancing the preservation of proximity to the starting node while leveraging information from larger neighborhoods.

The authors describe one propagation step as a power iteration which is related to a random work with restarts. A power iteration is calculated via:

$$\begin{aligned}\mathbf{Z}^{(0)} &= \mathbf{H} = f_\theta(\mathbf{X}),\\ \mathbf{Z}^{(k+1)} &= (1-\alpha)\tilde{\mathbf{A}}\mathbf{Z}^{(k)} + \alpha\mathbf{H},\\ \mathbf{Z}^{(K)} &= \text{softmax}\left((1-\alpha)\tilde{\mathbf{A}}\mathbf{Z}^{(K-1)} + \alpha\mathbf{H}\right).\end{aligned} \tag{6}$$

Here, H is the prediction matrix that serves as the starting vector and teleport set. It is calculated by passing the feature matrix $\mathbf{X}$ to a MLP NN f_θ with parameter set θ. K denotes the number of following power iteration steps and Z represents the respective predicted node features. $\tilde{\mathbf{A}}$ is the normalized adjacency matrix with added self loops. So, the initial feature matrix is passed to a MLP which determines the starting vector H, to which the model can return at a probability determined by α. With a probability of $(1-\alpha)$ the model performs a random walk on the current set of predicted node features Z_K, which are multiplied by the adjusted adjacency matrix $\tilde{\mathbf{A}}$. Finally, the node features are passed to a softmax activation function.

Lastly, we want to introduce the basis for the last of the four GNN layers we will implement in this work, which is the Graph Attention network (GAT) published by Veličković et al. [42]. The proposed model is based on an attention based architecture designed to perform node classification. An attention mechanism is utilized to focus on the most relevant part of the input to make decisions by computing a hidden representation of each node in the graph by attending over its neighbors. The mechanism is implemented as a graph attention layer and takes a set of node features $\mathbf{h} = \vec{h_1}, \vec{h_2}, ..., \vec{h_N}, h_i \in \mathbb{R}^F$ as input and outputs a transformed set of node features $\mathbf{h}' = \vec{h'_1}, \vec{h'_2}, ..., \vec{h'_N}, h'_i \in \mathbb{R}^{F'}$, where F is the number of features in each node. In a first step, a linear transformation which is parameterized by a weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$ and shared between nodes is applied to every node. Then the self attention mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'}$ is applied tn the node. The attention coefficients are calculated by the attention mechanism a with

$$e_{ij} = a(\mathbf{W}\vec{h_i}, \mathbf{W}\vec{h_j}), \tag{7}$$

with e_{ij}indicating the importance of node j's features to node i. The GAT is a spatial approach and parallelization across node pairs makes it efficient compared to other approaches.

Although we have covered the GNN types to be used in this study, it is important to explore other GNNs used for similar prediction tasks, enhancing our comprehension of the field. The task of our GNN surrogate models is of spatio-temporal nature, since we have to predict future node values, based on node values from the past, while incorporating the spatial dependencies between the nodes. Spatio-temporal GNNs (STGNNs) are a specific category of GNNs, which process graphs with node attributes that change over time. One possible task of STGNNs is forecasting future node values. In spatio-temporal graph modelling the basic assumption is that the node's future value is dependent on its own historical values as well as on the historical information of its neighbors [48]. STGNN approaches can be categorized into two types: RNN based methods and CNN based methods [40], [48]. Nevertheless, most of the approaches combine both types.

Jain et al. [49] presented a structural RNN, where each node and edge is represented by a distinct RNN. Node RNNs and edge RNNs are connected to form a bipartite graph. A bipartite graph $G = (N, E)$ is a graph were the set of nodes N can be partitioned into two disjoint sets N_1 and N_2 such that every edge in E connects from a Vertex from V_1 to V_2. In the structural RNN, nodes can share parameters which enhances flexibility and compactness. The resulting model is applied to multiple tasks, such as human motion modeling and driver maneuver anticipation. One structured RNN graph represents the features at time t and the goal of the authors was to predict the features at time $t+1$.

A larger number of works with the aim to capture spatio-temporal relationships on graphs with RNNs combine RNNs and ConvGNNs, by integrating ConvGNNs into RNNs [48]. For example, convolutional operations could process graph information before the data is passed to a recurrent unit [50]. Generally it can be said that the convolutional structures are used to capture spatial dependencies while the recurrent structures analyze the temporal context [50], [51].
The Spatio-Temporal Graph Convolutional Network (ST-GCN) designed by Yan et al. [52] is a CNN based spatio-temporal GNN and was built for skeleton based action recognition. The model includes two types of edges, namely spatial and temporal edges. Through multiple layers, the convolutional operations of the ST-GCN are applied and gradually generate higher-level feature maps on the graph [52]. Yu et al. [53] created their ST-GCN for a traffic forecasting task. The model was built completely from convolutional structures. The authors' motivation was to avoid the resource consuming iterations of the RNN. The corresponding approach combines graph convolution with 1D-convolution. In the convolutional operation, gated CNNs are utilized for temporal feature extraction and a spatio-temporal convolutional block is applied to process graph-structured time series. This model structure can be generalized to other spatio-temporal tasks.

3.3 Examples for Machine Learning Applications for COVID-19 Prediction

AI has been applied in multiple forms with the aim to gain knowledge about the SARS-CoV2 virus [54]. A literature review [54] shows how wide the application field of AI in the context of studying COVID-19 is. AI tools for clinical applications like monitoring [55] or diagnosis [56] exist in a wide variety. Patients benefit from AI thanks to supporting symptom monitoring and prediction of recovery, mortality and severity.
ANNs are a subgroup of AI and in the context of COVID-19 they can help experts to

classify CT scans and X-Rays of lungs [57]. Moreover, studying vaccine effectiveness [58] and even the identification of new potential drugs [59] are tasks where ANNs can support research.
In this section, we will present work which has been published in the research field of machine or deep learning for epidemiological modelling and prediction of Sars-CoV-2 cases. Ahead of this, we will mention a selection of papers which presented models for other infectious diseases or more general models, which can be applied to any infectious disease.

Wu et al. [60] combined the strengths of convolutional and recurrent neural networks in order to predict cases of illness based on real world datasets from Japan and the USA. The authors define the task as a time series problem with the goal to predict the epidemiological profile for up to eight weeks. The utilized data was not grid-structured, so in order to apply a CNN layer to the corresponding data, the authors created a new type of filter to apply on the node structure.

Spatial information is of great importance in epidemiology, as most diseases are transmitted through contact. This means that at the level of individual agents, the proximity of individuals plays a role, but also in the larger spatial context, the proximity of objects can influence the process of the spread of the disease. For example, diseases would usually spread from one city to the neighboring cities and then further spread to neighboring cities of higher hierarchies. This process can be modelled with a graph, which is why GNNs can be a useful tool in epidemiological forecasting. One example of GNNs for epidemic forecasting is the cross-location attention based graph neural network (Cola-GNN) from Deng et al. [61]. They propose one of the first GNNs for epidemic forecasting.
Cola-GNN can be described as a message passing GNN with a RecNN module and a location awareness attention. The latter method is applied to model the relationship between nodes, which represent locations. The utilized data is on population level and the input for each node can be described as the number of cases of illness in the location. The input is a time series for multiple weeks and the outputs are long term predictions for up to 20 weeks. Real world data from three different epidemics in Japan and the USA builds the database for their work. Two of the datasets capture only cases of influenza, while the third dataset contains information about cases of influenza-like illnesses which are defined as respiratory illness that includes fever plus a cough or sore throat. The authors compare their models' performance with other methods for time series prediction such as auto-regressive moving average, recurrent neural nets and long short-term-memory models.

Kapoor et al. [62] presented a GNN for the prediction of COVID-19 cases on county level in the US. Mobility data from two Google datasets was included into their work in order to model inter- and intra-regional movement of individuals. Nodes represent US counties and can be connected to other nodes. They are attributed with node features such as state, county, day, past cases and past deaths. Spatial as well as temporal aspects are modeled with edges. In the spatial domain, edges represent direct location-to-location movement. In the temporal domain, edges simply represent binary connections to past days. This leads to a stacked graph structure, where each layer represents the county connectivity graph for that day. The task of the model is to predict COVID-19 cases for the next day, based on input data from the previous seven days.

Gao et al. [63] published the spatio-temporal attention network (STAN), which achieved better results than Cola-GNN [61], Kapoor's CovidGNN [62] and other methods such as differential equation based SIR and SEIR models. The task for their model was to predict the number of COVID-19 cases for a fixed number of days into the future on county- and state-level in the USA. The input consists of positive cases and static and dynamic statistical information about the population of each county. The authors built a graph where nodes represent counties and each node is associated with an input as described. The edges were built based on geographical proximity and demographic similarity between nodes. By using graph attention layers [42], they extracted spatio-temporal features from the attributed graph. The graph attention mechanism is applied on every node and captures information from neighboring nodes and generated graph embeddings based on this information. Graph embeddings represent nodes or entire graphs as low-dimensional vectors. To incorporate the temporal aspect of disease transmission a gated recurrent unit calculates hidden states for each specific location. In their experiments they predicted up to 20 days. Their model outperformed the Cola-GNN [61] as well as the GNN for COVID-19 prediction from Kapoor et al [62].

Another GNN for COVID-19 forecasting was developed by Panagopoulos et al. [64]. The task of the model is to predict future cases of COVID-19 for each node. The time horizon for prediction is up to 14 days. Data from four European countries, namely Italy, France, Spain and England were used for training and testing. In their work, the authors built a sequence of graphs, where each graph represents the spread of the virus among the nodes, which represent location, at only one point in time. Moreover, mobility data as edge weights were implemented in the model with the goal to capture the influence of mobility on the spread of the virus. Their model structure is a combination of message passing NN (MPNN) [41] and LSTM. First, the MPNN is applied on each graph of the input sequence. The representation of each node was updated based on messages received from its neighbors. The resulting representations were then fed into the LSTM layer to extract time dependent information. The results of the GNN of the authors outperformed classic statistical methods such as the auto-regressive moving average.

4 Surrogate Modelling for Epidemiological Compartment Models

In this chapter we will present surrogate models for ODE-based epidemiological compartment models. In Section 4.1, we are going to outline the motivation for the implementation of surrogate models in general as well as the motivation for our specific application.
In Section 4.2, we will describe the ODE model that will be approximated by the NNs. We are going to introduce the compartments and their interconnections as well as the characteristics of the model. We will also describe how to we generate the data which will be processed by the NN.
At the core of this chapter stands the creation of the surrogate models. The aim is to find the most suitable model type and architecture. For this reason we will test some of the model parameters in grid searches and tune further parameters afterwards as well.
In Section 4.3, we will implement models for population-based models without spatial resolution. There, we will differentiate between models where only one age group is considered and models with age-resolution. The first type of model will be addressed in Section 4.3.1, and the model with age groups in Section 4.3.2. In the final step of this section, in Section 4.4, we will derive advanced GNNs for the entirety of all 400 German counties.

4.1 A Motivation for Surrogate Models

Surrogate Modelling is a technique to approximate the behavior of complex systems. Usually, the more complex the original model is the higher is the requirement of computational resources. The goal of the surrogate model is to approximate the original model's behavior more efficiently, while only loosing an acceptable amount of accuracy. Many mathematical techniques can be applied for surrogate modeling [65] including Gaussian process, regressions and radial basis functions, just to name a few. Furthermore, more advanced surrogate models can be provided by using machine learning techniques such as support vector machines and ANNs [65].

NNs can be used as function approximators and perform well for interpolation as well as for fitting [66]. Traditionally, NNs are characterized as useful tools for problems, where the underlying mathematical relationship between the inputs and the labels is unknown. Utilizing the power of neural networks in a field where some specific underlying physical laws or principles are known and should be included in the models implementation is one of the main motivations for physics-informed NNs (PINN). PINNs were introduced by Raissi et al. [4] for solving "supervised learning tasks while respecting any given law of physics described by general nonlinear partial differential equations" [4]. The authors implement information of the differential operator into the activation function and loss function of the model. In their data driven approach the NN with incorporated information about the dynamics in the data is applied on training data in order to solve the underlying equation in the training data. Famous applications are for example

Navier-Stokes based computations of fluid mechanics [5], [67] or traffic flows [68].

Another field where surrogate models can be applied is agent-based modelling [69]. In this application, machine learning techniques are employed to create statistical models that mimic the behavior of the original agent-based model (ABM). For the data driven approach the authors create the input data for the surrogate models from the simulation of the ABM, where each run constituted a sample for the machine learning application.

For ODE-based problems, NN can be applied as well. The focus of research in this area was towards the application of NNs as ODE solvers. The idea was initially discussed by Lagaris et al. [70], who used a simple MLP to solve ODEs. Further, neural network structures were applied by Chen et al. [71] when they introduced neural ordinary differential equations. Likewise, their contribution was not a surrogate model, but rather a solving technique for ODEs. These types of contributions are often based on physics-informed neural networks and implement information about the ODE into the network (e.g. into the loss function) and emerge frequently in the field of solving ODEs [72], [73].

The target model we want to substitute is implemented in the HPC software library MEmilio. Although it is implemented efficiently and even runs in less than a second, a large number of ensemble runs might need an HPC system to provide results with minimal waiting time. In critical epidemic situations, scanning a large set of alternative reactions is mandatory and swift responses are crucial. Furthermore, simulations might be needed for stakeholders that do not have access to HPC systems. To overcome these barriers and to provide just-in-time answers to urgent questions, pre-trained AI surrogate models can become of great aid for decision makers.

4.2 Data Generation

We used MEmilio [74] for data generation. MEmilio is a high performance Modular Epidemics simulation software developed by different researchers from different institutions – with the main contributors coming from the Institute for Software Technology of the German Aerospace Center (DLR) and the the department of Systems Immunology (SIMM) of the Helmholtz Center for Infection Research (HZI). It offers a variety of models for infectious disease dynamics, from compartment models through Integro-Differential equation-based models to agent- or individual-based models.

We utilize a SECIR-type model from MEmilio which uses eight compartments and ordinary differential equations to simulate the disease dynamics in the population over time. Its modular design allows for age stratification and age-dependent parameters such as contact patterns, susceptibility, death rates, and much more. In most use cases, the population is divided into six age groups: 0-4, 5-14, 15-34, 35-59, 60-79 and 80+ years olds since the official data to compare against is reported in these age groups.

All healthy individuals which are susceptible to infection are grouped into the Susceptible (S) compartment. After an individual gets in contact with the virus it is considered

to be in the Exposed (E) compartment. These individuals are not infectious to others nor have symptoms, yet they carry the virus. Carriers that do not show any symptoms because they are pre- or asymptomatic belong to the InfectedNoSymptoms (I_{NS}) compartment. They can be infectious to other individuals. As soon as a carrier is infectious and shows symptoms of disease, the carrier is assigned to the InfectedSymptoms (I_{Sy}) compartment. Infected individuals whose disease development is severe and a hospitalization is necessary are in the InfectedSevere (I_{Sev}) compartment. The InfectedCritical (I_{Cr}) compartment contains infected individuals in a critical state. Those individuals generally require intensive care. Lastly, the model has a Dead (D) compartment and a Recovered (R) compartment, the latter of which with individuals that are immune and cannot be infected again.
We call the set of all compartments $\mathcal{Z} := \{S, E, I_{NS}, I_{Sy}, I_{Sev}, I_{Cr}, R, D\}$. The number of individuals in a compartment $z \in \mathcal{Z}$ at time t is written as $z(t)$, for example $E(t)$ for the Exposed compartment.
Figure 5 as adopted from [6] illustrates the context of the compartments and clarifies which compartment transitions are possible. It shows that the model assumes immunity after an infection, since no arrow leads from the Recovered compartment back to the Susceptible compartment. Further model developments, with models for later phases of the pandemic, using multiple immunity layers [75] or waning immunity [76], have also been integrated in MEmilio but are out of the scope of the current study.
The age-resolved design leads to age dependent transmission rates and a contact matrix, which describes the mean number of daily contacts of a person from one age group with a person from another age group. The contact patterns are split up into the four categories "Home", "School", "Work", and "Other". In their paper [6], the authors provide detailed information about how they derived the resulting contact values. When we utilize contact matrices in our models, we do not differentiate between the different contact places. Instead, we take the sum of the contact matrices for Home, School, Work and Other.
Generally, all parameters are based on prior publications and expert knowledge, which is why those type of models are often called expert models.

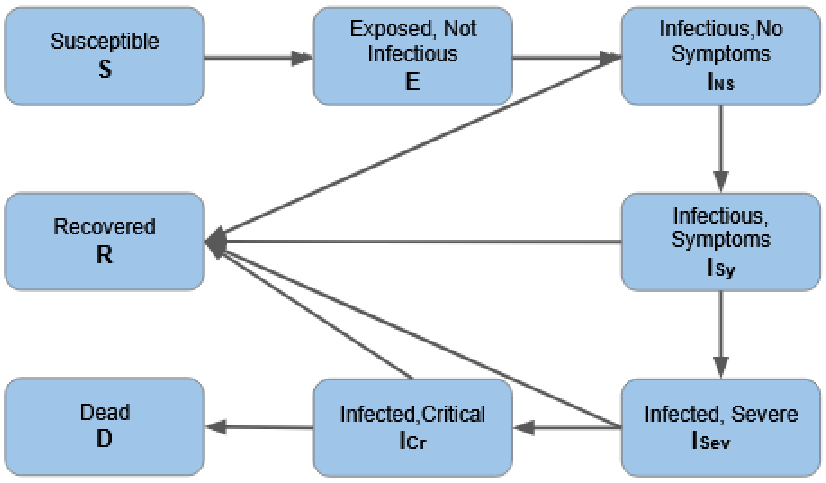


Figure 5: Overview of MEmilio's compartment structure
The boxes represent the compartments and the arrows indicate the possible transitions individuals can make.

For a better understanding of the ODE model and therefore the task our machine learning models have to solve, we also provide the underlying ODEs as introduced in [6], [77]. Fist, we need to introduce the parameters which are used. The age specific transmission risk for age group $i = 1, ..., n$ is denoted by ρ_i. It describes the probability of transmission when a contact between individuals takes place.
The contact frequency matrix $\Phi = (\phi_{i,j})_{i,j=1,...,n}$ describes the number of daily contacts. An entry in the matrix is denoted as $\phi_{i,j}$ and contains the average number of daily contacts of individuals of age group i with individuals of age group j [6].
N_j represents the total population of age group j. In [77] the concept of isolation is introduced and this concept is implemented by two variables. Both variables describe individuals which are not isolated nor in quarantine and therefore capable of infecting other individuals. We write $(\xi_{I_{NS}})$ for individuals in the InfectedNoSymptoms and $(\xi_{I_{Sy}})$ for individuals in the InfectedSymptoms compartment.
The remaining parameters can be described by two generic variables. T_{z_1} is the time spent in state $z_1 \in \mathcal{Z}$ before moving to any other state. For instance, the variable $T_{I_{Cr,i}}$ describes the time period an individual in age group $i = 1, ...n$ spends in a critical infection state before it either goes to the Recovered or the Death compartment. Lastly, the probability of a patient to transit from state z_1 to state z_2 is denoted as $\mu_{z_1}^{z_2}$, where $z_1, z_2 \in \mathcal{Z}$. Hence, we have introduced all parameters necessary to describe the ODE model:

$$\frac{dS_i}{dt} = -S_i \rho_i \sum_{j=1}^{n} \phi_{i,j} \frac{\xi_{I_{NS,j}} I_{NS,j} + \xi_{I_{Sy,j}} I_{Sy_j}}{N_j - D_j} \tag{8}$$

$$\frac{dE_i}{dt} = S_i \rho_i \sum_{j=1}^{n} \phi_{i,j} \frac{\xi_{I_{NS,j}} I_{NS,j} + \xi_{I_{Sy,j}} I_{Sy,j}}{N_j - D_j} - \frac{1}{T_{E_i}} E_i \tag{9}$$

$$\frac{dI_{NS,i}}{dt} = \frac{1}{T_{E_i}} E_i - \frac{1}{T_{I_{NS,i}}} I_{NS,i} \tag{10}$$

$$\frac{dI_{Sy,i}}{dt} = \frac{\mu_{I_{NS,i}}^{I_{Sy,i}}}{T_{I_{NS,i}}} I_{NS,i} - \frac{1}{T_{I_{Sy,i}}} I_{Sy,i} \tag{11}$$

$$\frac{dI_{Sev,i}}{dt} = \frac{\mu_{I_{Sy,i}}^{I_{Sev,i}}}{T_{I_{Sy,i}}} I_{Sy,i} - \frac{1}{T_{I_{Sev,i}}} I_{Sev,i} \tag{12}$$

$$\frac{dI_{Cr,i}}{dt} = \frac{\mu_{I_{Sev,i}}^{I_{Cr,i}}}{T_{I_{Sev,i}}} I_{Sev,i} - \frac{1}{T_{I_{Cr,i}}} I_{Cr,i} \tag{13}$$

$$\frac{dR_i}{dt} = \frac{1-\mu_{I_{NS,i}}^{I_{Sy,i}}}{T_{I_{NS,i}}} I_{NS,i} - \frac{1-\mu_{I_{Sy,i}}^{I_{Sev,i}}}{T_{I_{Sy,i}}} I_{Sy,i} + \frac{1-\mu_{I_{Sev,i}}^{I_{Cr,i}}}{T_{I_{Sev,i}}} I_{Sev,i} + \frac{1-\mu_{I_{Cr,i}}^{D_i}}{T_{I_{Cr,i}}} I_{Cr,i} \tag{14}$$

$$\frac{dD_i}{dt} = \frac{\mu_{I_{Cr,i}}^{D_i}}{T_{I_{Cr,i}}} I_{Cr,i} \tag{15}$$

In their work [6], the authors focus on the influence of non-pharmaceutical interventions (NPIs). Two levels of interventions are considered. The first level includes avoiding contacts and the second level covers measures such as wearing masks and ventilating closed spaces. Many other parameters as for instance influence of the seasons, quarantine and isolation as well as epidemiology-specific parameters like asymptomatic individuals, incubation period and recovery time are defined based on scientific results from the respective fields.

The enactment of NPI and therefore the reduction of the daily contacts of individuals is called damping. Some examples of NPIs are working from home, school closures, closing of public places such as bars and restaurants or measures like face masks and social distancing [6]. In the ODE model from MEmilio, the dampings are applied individually for each contact category "School", "Home", "Work" and "Other", because each measure influences these categories differently. Therefore, the contact reduction is implemented as a damping factor α which is applied to the contact matrix Φ, reducing all entries in the matrix and therefore reducing the contacts that take place, what results in a damped contact matrix:

$$\Phi' = \Phi(1-\alpha). \tag{16}$$

The quality of the predictions of the model was evaluated by a retrospective analysis where the models prediction was compared to real-world data [6]. Despite the difficulty to predict mass-spreading events, the model was capable of reproducing overall infection dynamics, considering static as well as dynamic, incidence-dependent NPIs. Overall, the median of the simulated infections and deaths did not differ significantly from the real-world data [6].

The MEmilio approach leverages simple ODE-based models to spatially resolved models by using a novel graph-based approach. It thus defeats a major limitation of many ODE-based models, which is the lack of representation of heterogeneous spatial features [63]. In our application, it applies a SECIR-type model to each spatial entity. Consequently, it realizes geographical information such as the spatial distribution of individuals and commuting and travelling activities. Commuter data is derived from the German Federal Employment Agency for work commutes and geo-referenced Twitter data is utilized for leisure time travel information.
Moreover, data on the population of Germany is needed. Germany has 400 counties of different sizes. The distribution of county size is illustrated in Figure 6. The average size is 197,993 and the median is at 145,768. 50% of the cities and counties have between 104,071 and 233,222 inhabitants. In both diagrams we can see the four outliers with more than one million individuals per county with the biggest county, Berlin, with approximately 3.2 million residents.

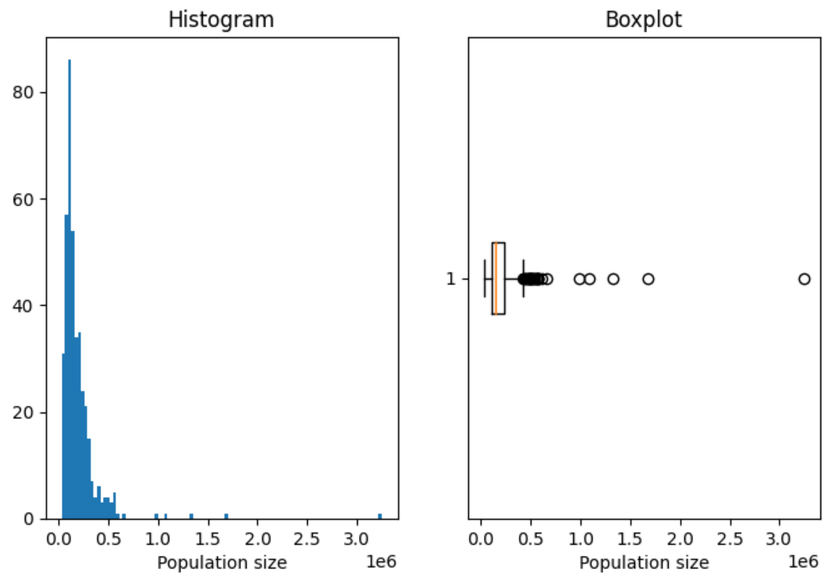


Figure 6: Illustration of the distribution of the 400 populations

4.3 Modelling the Spread of COVID-19 in a Homogeneous Population with Neural Networks

Before building a GNN that can predict the spread of COVID-19 in all of Germany's 400 counties, we want to test the ability of NNs to model the COVID-19 cases for one population, first in a completely homogeneous setting and second assuming stratification in six age groups. The aim is to determine boundaries of the model, e.g maximum number of days or included changes in contact patterns which can be predicted within an acceptable error range. We denote the models for one population without age-resolution SimpleNN and the models for one population with age groups GroupsNN.

4.3.1 Predictive Models for One Population and One Age Group

For our very first experiments, we want to create a simple model with no age-resolution and no changes in contact patterns. Using MEmilio we generate a dataset consisting of 10.000 simulations. Each simulation starts with a random number (within a predefined range as described in Table 1) of individuals in each of the eight compartments. The simulations start at the beginning of an epidemic. This means, the model has to predict when the number of infected individuals will peak. The population size was fixed at 50.000. The data was normalized with a logarithmic transformer.
We chose mean absolute percentage error (MAPE) as the performance measure, since the compartments are unbalanced. The Susceptible and Recovered compartments contain most of the individuals, depending on the stage of the epidemic. In the case of COVID-19, the Death and InfectedCritical compartments are smaller compartments. However, these compartments are of special interest, because the goal of stakeholders could be to minimize the number of individuals in those critical stages of the infection. The MAPE measure ensures that accurate predictions in compartments with many individuals do not lead to an overall good performance. We force the model to aim for accurate predictions for all compartments, regardless of the number of individuals it contains. The MAPE is calculated by the following formula:

$$MAPE = \frac{100}{n} \sum \left| \frac{y_{true} - y_{pred}}{y_{true}} \right|$$

where y_{true} is the label and y_{pred} is the predicted value and n is the number of data samples.
The input data contains the distribution of individuals along the eight compartments for five days. It is passed to the model as an array with the dimensions: number of samples, number of input days and number of compartments. The task of the model is to predict the distribution of individuals for the next 30 days.

We generate two different datasets that differ in the initial distribution of individuals along the compartments. This is achieved by adjusting the ranges from which the percentages which determines the fraction of the population that is assigned to each compartment are drawn. We adjust the ranges of random initialization as described in

Table 1. Figure 7 illustrates the distribution of the datasets. It is clearly visible that the compartments from dataset 2 take on more variable values while the data from dataset 1 is concentrated in very small boxes. Figure 8 offers a better visualization of dataset 1 by showing the percentiles of the data distribution. The y-axis of the plots indicates that most of the data remains in a smaller range of possible inital values.

	dataset 1	**dataset 2**
E	$60 \times x_0$	$1800 \times x_1$
I_{NS}	$55 \times x_0$	$1650 \times x_1$
I_{Sy}	$50 \times x_0$	$1500 \times x_1$
I_{Sev}	$12 \times x_0$	$360 \times x_1$
I_{Cr}	$3 \times x_0\ x_1$	$90\ \times x_1$
R	$50 \times x_0$	$1500 \times x_1$
D	0	$30 \times x_2$
S	all remaining	all remaining

Table 1: SimpleNN: Data generation with different ranges
x_0 is drawn from $U(0.2, 1)$ and x_1 is drawn from $U(0.002, 1)$, where U denotes a uniform distribution.

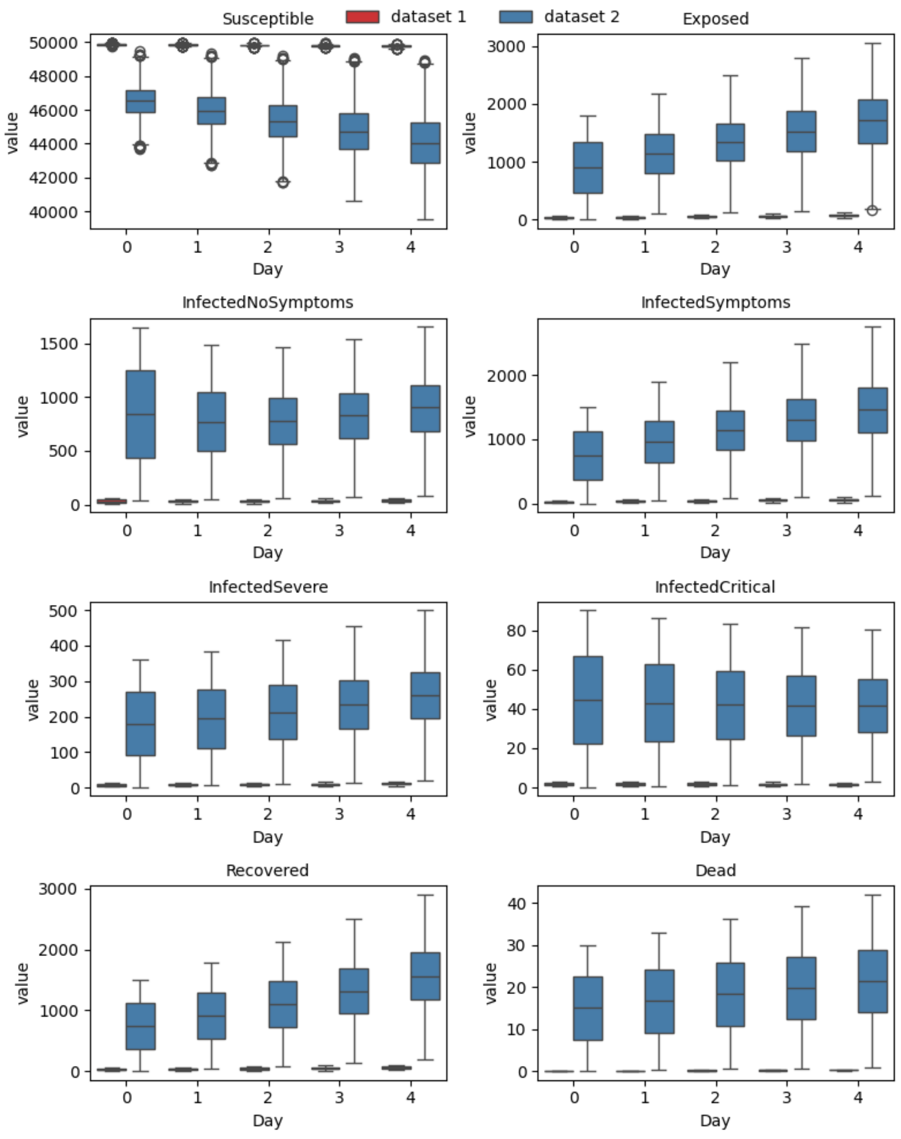


Figure 7: SimpleNN: Distribution of different inputs illustrated by box plot. The red boxes represent the data from dataset 1. The blue boxes represent data from dataset 2.

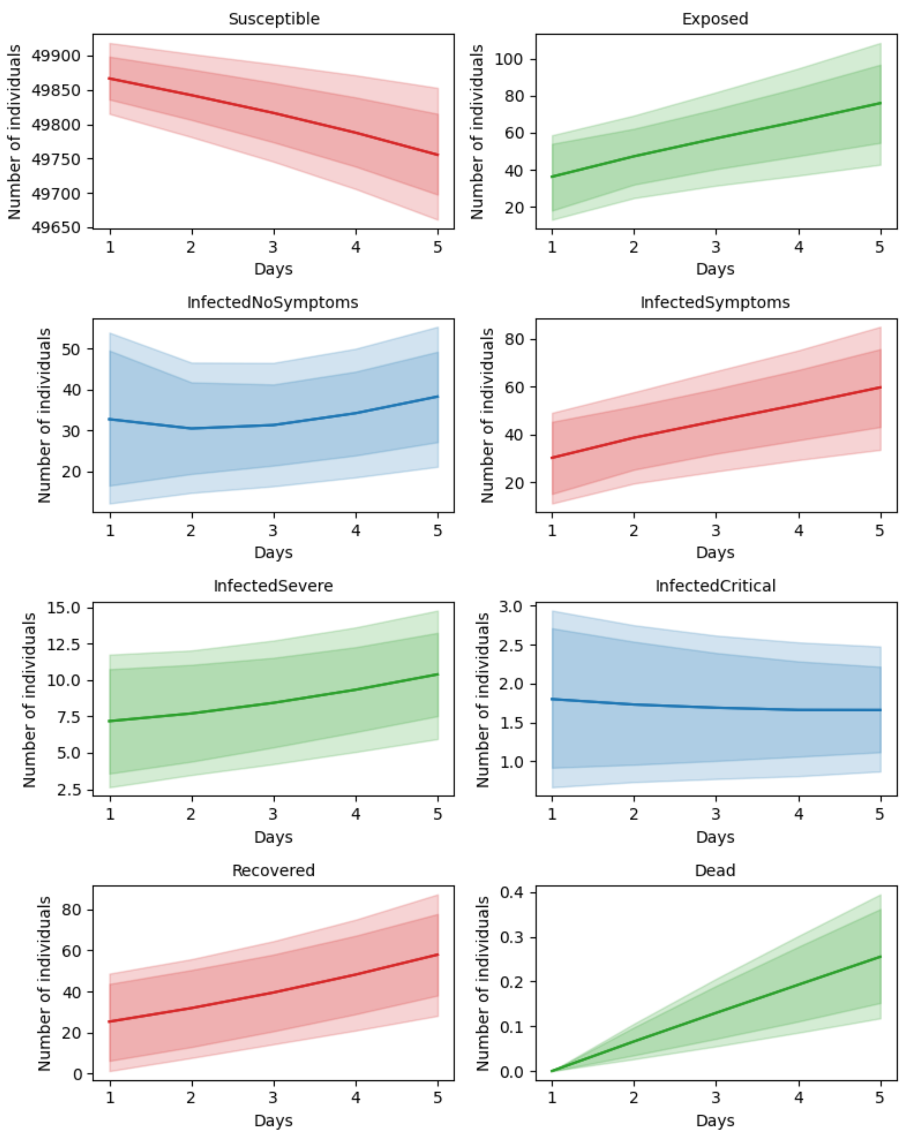


Figure 8: SimpleNN: Input data with 5, 25, 75, 95 percentiles
The bold line represents the 50% percentile. The darker area depict the 25 and 75 percentiles respectively. And the lighter outer areas are the 5 and 95 percentiles.

We will use dataset 1 for all of the experiments in this section. The aim of this section is to find a suitable model architecture for our task. Then, in Section 5.1, we will test the model on dataset 2.

Long term prediction problems can be categorized into two categories: iterative and direct models [61]. Direct models predict future values at one shot, based on past input values. Iterative models predict one time step at a time, meaning they recursively invoke short term prediction to make long term predictions. We decided to implement a direct model that predicts all labels belonging to multiple days in one step.

We performed experiments for three types of models and all models are implemented using the Keras [78] library. The first model is a simple MLP, consisting of a minimum of two dense layers, one of them serves as an input layer and the second one as the output layer of the model. The number of hidden layers as well as the number of units per layer can be altered. For our model implementations and for building an LSTM, the library offers an LSTM layer that embodies an LSTM model based on [30]. The LSTM model for our experiments is built with a minimum of one LSTM layer and one dense layer as an output layer. The last model is a CNN which is constructed with a minimal architecture of one 1D-Conv layer and one dense layer. Before feeding the data to the 1D-Conv layer, it is pre-processed in a lambda layer. This layer brings the input into the right shape for the next layer.

To ensure that we determine the best possible model architecture, we conduct a grid search for each model. In the course of this, we vary the number of hidden layers between zero and four and the number of units per layer takes the values of 32, 62, 128, 512 and 1024. The number of units is the same for each layer, including the input layer. The hidden layers are always dense layers. For each layer, the number of units describes the dimension of the output space. For dense layers, the number of units can be interpreted as the number of neurons and for the 1D-Conv layer of a CNN they can be interpreted as the number of filter applied in the convolution

All hidden layers as well as the 1D-Conv layer and the dense input layer in the MLP utilize the relu activation function. All dense layers that serve as an output layer employ the linear activation function, so the output is adequate for our regression problem. The optimizer is set to Adam and we conduct 5-fold cross validation. This means that we split the dataset five times into training and validation dataset. Each time, we utilize $\frac{4}{5}$ of the data to train the model and the remaining $\frac{1}{5}$ for validation. This is repeated five times, so that each data sample was utilized in the training set as well as in the validation dataset once. This is how we can ensure that the results are independent from the split.

The presented MAPE scores are the mean of of the validation MAPE scores of all folds. The maximum number of epochs was 1500 with a stopping criterion, that takes effect when the monitored metric stopped improving for 100 epochs.

The heatmaps in Figure 9 show the inter-dependencies of the number of units and number of layers for each model type. One insight we obtain is that more layers do not necessarily lead to better performance. Instead, the simpler models often show a lower MAPE. Nevertheless, the best model architecture for each model type has to be chosen individually. The heatmaps allow to directly derive the best model architecture depending on number of layers and units. For the MLP the best average validation MAPE of 0.2% is achieved by multiple model architectures. On the one hand, smaller models with zero hidden layers and 62 units per layer or a model with one hidden layer but only 32 units reach this score. On the other hand, the most complex model tested

with four hidden layers with 1024 each, results in the same MAPE. The CNN model performs best with one hidden layer with 62 units reaching a MAPE of 0.17%. The overall best performance of a 0.13% MAPE of the LSTM model occurs two times, each for a model without hidden layer, with varying number of units.
In summary, we have best scoring models of each model type with a lean architecture with no or few hidden layers and a small amount of units per layer. The task of predicting the distribution of individuals of one population among compartments without age-resolution can be modeled with multiple model types with small architectures.

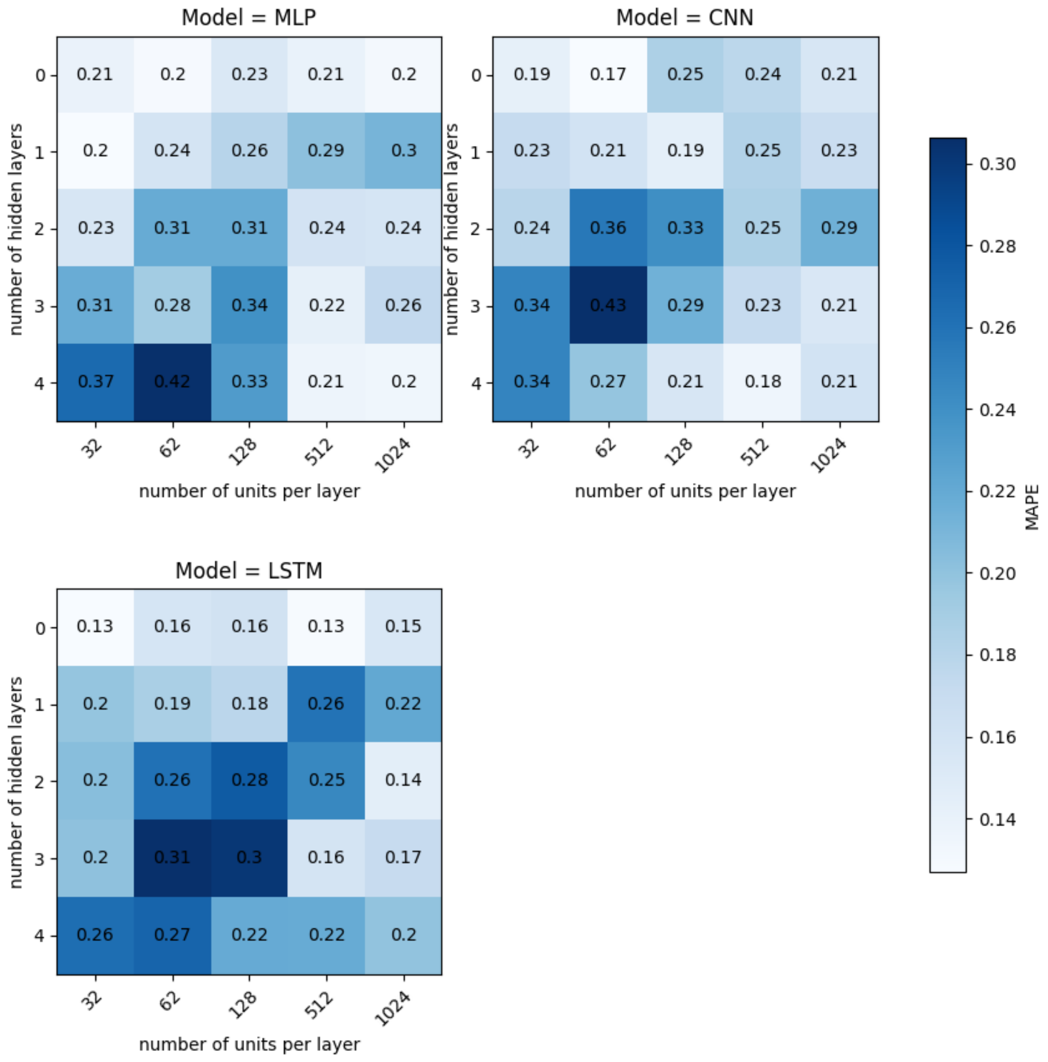


Figure 9: SimpleNN: Heatmaps grid search

The best model for this task is the LSTM model with no hidden layer and 32 or 512

units per layer with an average validation MAPE of **0.13%**. For the following experiment we chose the smaller model with 32 units per layer.
In a next step, we want to determine the optimal optimizer for this model. In the grid search, we utilize Adam optimizer and relu activation function in hidden layers. Since our optimal LSTM model features no hidden layers, there is no need for finding the best activation function for those layers. The LSTM layer, which serves as the input layer for our model, traditionally utilizes tanh and sigmoid activation and those specific activation functions are the most critical component of the LSTM layer [79] which is why they are not altered in our experiments. Customized to our regression task, the output Dense layer should use the linear activation function.
We test the optimizers Adam, Nadam, SGD, Adagrad or RMSProp on a newly generated dataset. The parameters of the data generation were the same as before in dataset 1. As depicted in Figure 10 the best results were achieved by Adam optimizer with an average test MAPE of 0.1162%. Nadam is close to Adam's performance, while the other three optimizer degrade the scores so much that they are worse than any combination tested in the grid search. The worst optimizer seems to be SGD with a test MAPE of 1.8059%.

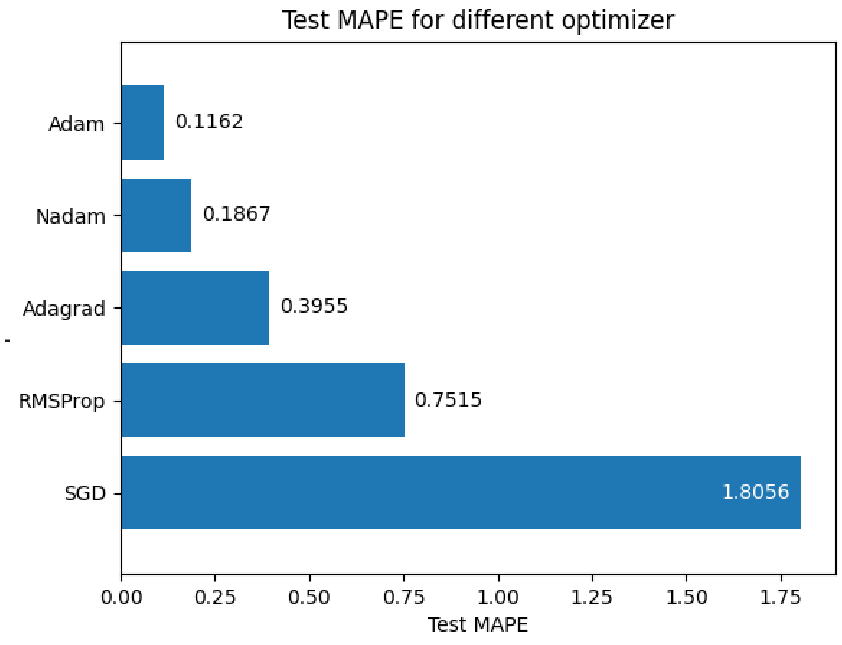


Figure 10: SimpleNN: Optimizer for LSTM
The LSTM with zero hidden layers and 32 units in the LSTM layer was chosen for these experiments.

To conclude the results for the first simple model without age-resolution and only one

population, it can be said that all three models can achieve average MAPE scores lower than 0.2% with a best score of **0.1162%**. The best score is achieved by an LSTM model with zero hidden layers, 32 units in the LSTM layer and Adam optimizer.
Figure 11 illustrates the resulting model architecture for our dataset with 10.000 samples for the task of predicting the distribution among the eight compartments for the next 30 days, given five days as input.

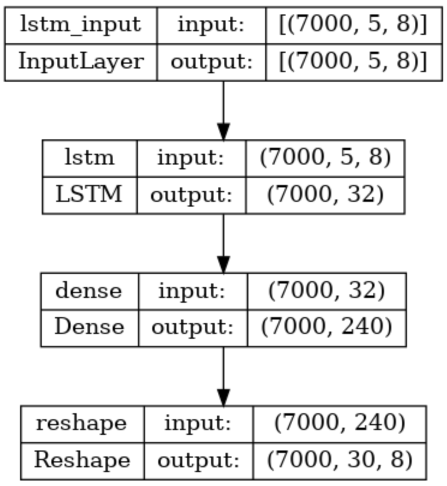


Figure 11: SimpleNN: Model architecture of best model (LSTM)
The LSTM with zero hidden layers and 32 units in the LSTM layer was chosen for these experiments.

4.3.2 Predictive Models for a Single Population with Age-Resolution

As a next step, we resolve the population by age-groups, according to the six age groups in MEmilio, to out model.
The data generation for these models is slightly different from the models with one age group. The original data that is available for Germany is stratified into six age-groups and therefore van be used directly instead of choosing an arbitrary population size. We draw a random population from the list with all 400 population and assign random percentages of individuals of each age-group to the compartments. Again, we create two datasets, namely groups-dataset 1 and groups-dataset 2. Groups-dataset 1 will be used in this section for the experiments with the aim to find the best model architecture. In Section 5.1, we will use groups-dataset 2 for the experiments with varying number of prediction days and contact reductions. Table 2 shows the exact ranges that are used for data generation.

	groups-dataset 1	**groups-dataset 2**
E	pop[i] $\times$ U(0.00025,0.0005)	pop[i] $\times$ U(0.00025,0.005)
I_{NS}	pop[i] $\times$ U(0.0001,0.00035)	pop[i] $\times$ U(0.0001,0.0035)
I_S	pop[i] $\times$ U(0.00007,0.0001)	pop[i] $\times$ U(0.00007,0.001)
I_{Sev}	pop[i] $\times$ U(0.00003,0.00006)	pop[i] $\times$ U(0.00003,0.0006)
I_{Cr}	pop[i] $\times$ U(0.00001,0.00002)	pop[i] $\times$ U(0.00001,0.0002)
R	pop[i] $\times$ U(0.002,0.008)	pop[i] $\times$ U(0.002,0.08)
D	0	pop[i] $\times$ U(0, 0.0003)
S	$N - E - N_{NS} - I_S - I_{Sev} - I_{Cr} - R - D$	$N - E - N_{NS} - I_S - I_{Sev} - I_{Cr} - R - D$

Table 2: GroupsNN: Data generation for GroupsNN with adjusted ranges
U is a uniform distribution.

In this setting, we compare the same three types of models: MLP, LSTM and CNN as before. In a grid search, we alter the model architecture by varying the number of hidden layers between zero and four. In addition to the depth of the models we investigate the effect of the model width by modifying the number of units per layer. It can take the values of 32, 62, 128, 512 and 1024. We evaluated the performance on a dataset with 10.000 samples with 5-fold cross validation. The presented MAPE scores are the mean of of the validation MAPE scores of each fold. The maximum number of epochs was 1500 with a stopping criterion, that takes effect when the monitored metric stopped improving for 100 epochs.

Figure 12 illustrates the influence of the number of hidden layers on the validation MAPE. The depicted values are the mean of all models of the same type with the same number of of hidden layers. For example, the very first blue bar with 1.32% MAPE represents the MLP models with zero hidden layers and 32, 62, 128, 512 and 1024 units per layer, by calculating the mean of all these models. We can see that the number of hidden layers has a different influence on performance depending on the type of model. For the LSTM model, which achieves the best results by far with 0.51% average test MAPE, the best models have no hidden layer. Each additional layer deteriorates the test score. The CNN models benefits from more hidden layers, improving the performance compared to zero hidden layers by 0.18%, resulting in a lowest test score of 0.99% for the models with three hidden layers. Otherwise, the model with four hidden layers has higher MAPE scores than the model without hidden layers. The behavior of the MLP is similar to the one of the CNN.

Additionally to the average values as presented in Figure 12, Figure 13 visualizes the distribution of the validation error for the different kinds of models, depending on the number of hidden layers in form of a box-plot diagram. The boxes contain 50% of all samples, and the whiskers stretch from the box to the most distant data point within 1.5 times the interquartile range (IQR) from the box. Outlier points beyond the whiskers are illustrated as a dot. The boxes and whiskers are relatively compact for the LSTM. This indicates little variation in MAPE scores for all models. The increase of error with rising number of hidden layers is also clearly visible for LSTM. MLP and CNN have more outliers that yield higher validation MAPE scores and the median does not increase linearly with increasing number of hidden layers. In the heatmap as presented in Figure 15, we can see that the outliers are the models with only 32 units per layer.

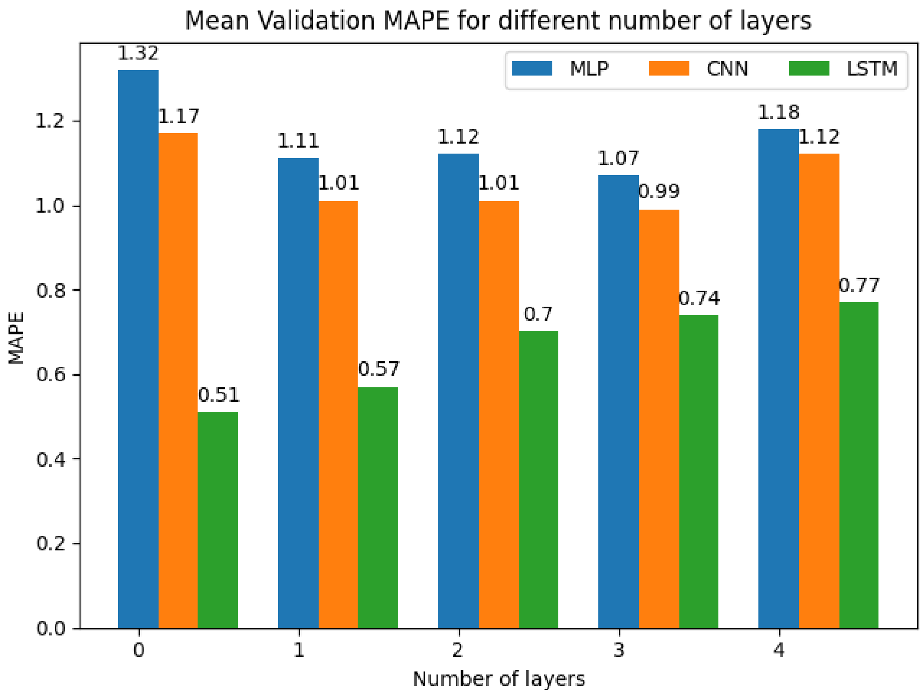


Figure 12: GroupsNN: Influence of hidden layers bar plot

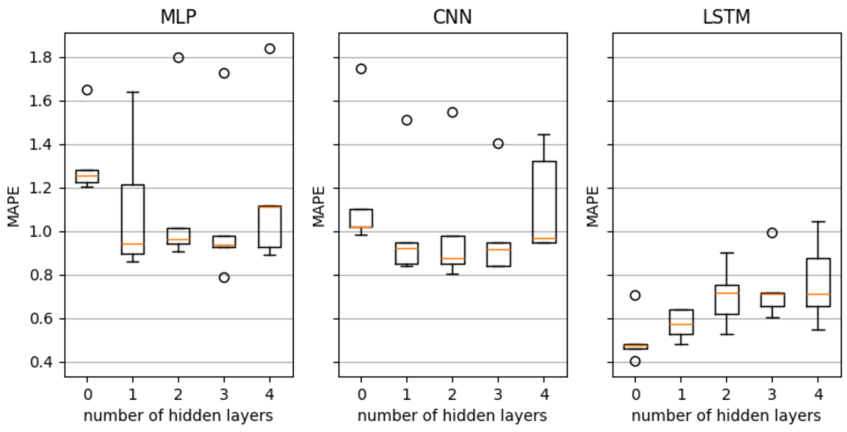


Figure 13: GroupsNN: Influence of hidden layers boxplot

Next, we want to describe the influence of the number of units per layer on the validation MAPE. We altered the number of units in the hidden layers as well as in the input layers.

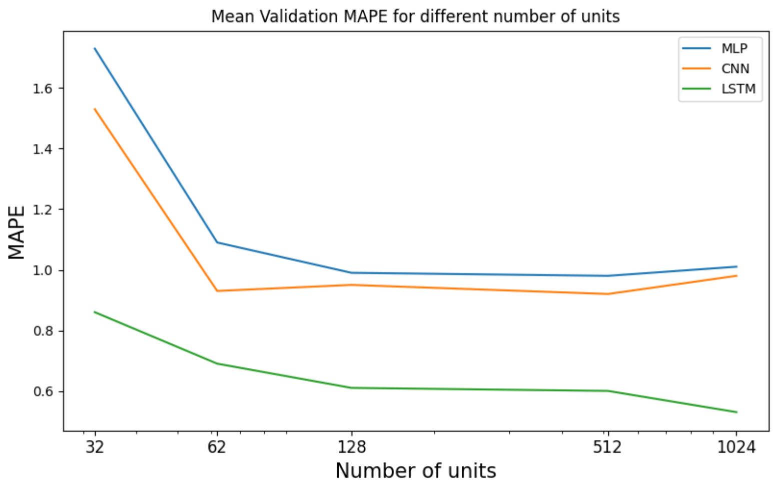


Figure 14: GroupsNN: Influence of number of units

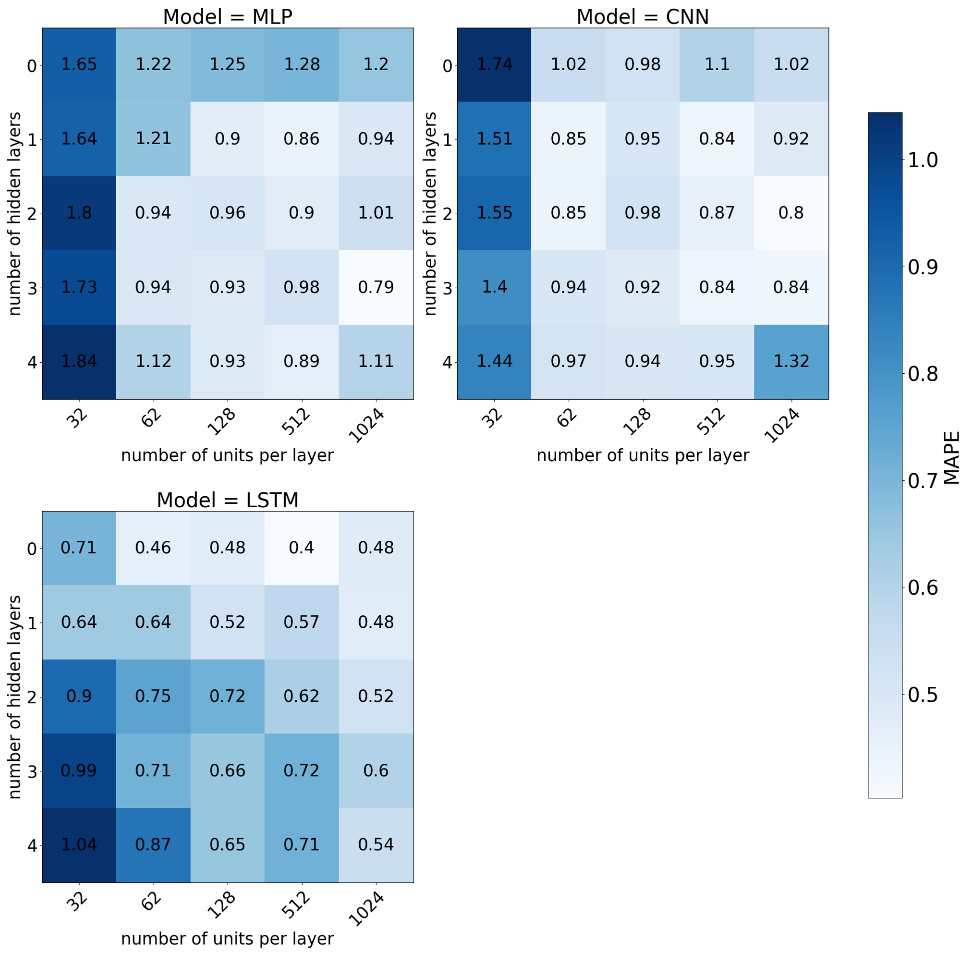


Figure 15: GroupsNN: Grid Search Results

All diagrams discussed before can be summarized with the heatmap in Figure 15. It shows the validation MAPE scores of every combination of number of hidden layers and number of units per layer for each model type. The best combination for the MLP model is three hidden layers with 1024 units per layer with a test MAPE of 0.79%. The CNN reaches an almost identical result with 0.8% test MAPE for the model with two hidden layers and 1024 units per layer. The overall best model is the LSTM model with zero hidden layers and 512 nodes, achieving 0.4% validation MAPE.
As explained before, the specific activation functions are the most critical component of the LSTM layer [79] which is why they are not altered in our experiment. Thus, we are solely testing the influence of optimizer and choose between Adam, Nadam, SGD, Adagrad and RMSProp. Here we perform 5-fold cross validation like before, but utilize a newly generated dataset which has the same parameters as dataset 1. Figure 16 shows that the Adam optimizer is the best optimizer with an average MAPE of 0.4364%, closely followed by Adagrad with only 0.0032% difference. Once more, SGD is the

poorest optimizer with more than 5% MAPE.

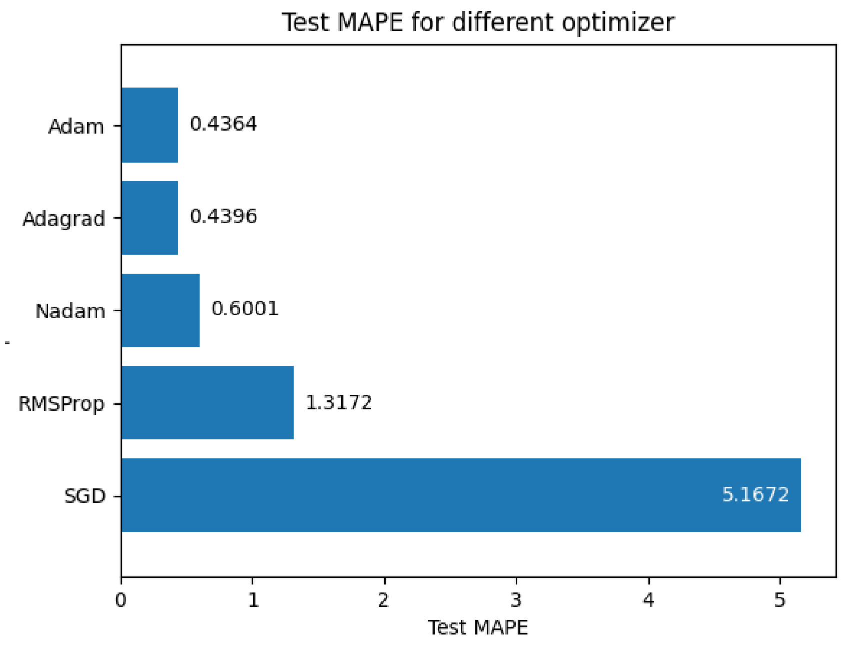


Figure 16: GroupsNN: Optimizer for LSTM

In conclusion, the best model architecture for our task of predicting the number of individuals from each age group per compartment for one population is the LSTM model with 512 units in the input layer and without hidden layers, utilizing the Adam optimizer. This model reaches an average test MAPE of **0.4%** for the task of predicting the distribution of individuals among compartments for 30 days, while considering six age groups. The architecture is visualized in Figure 17.

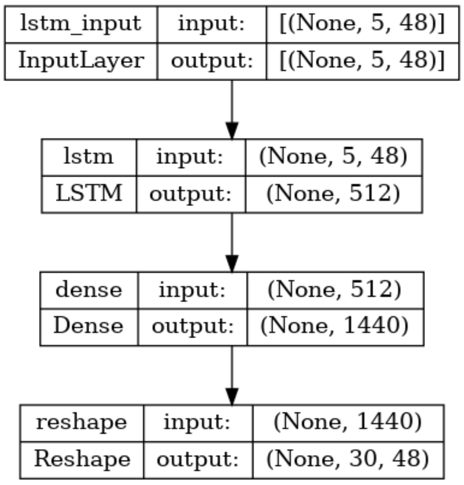


Figure 17: GroupsNN: Model architecture of best model (LSTM)

4.4 Building a Graph Neural Network for Germany on County-Level

In the previous section, we derived optimal NN configurations to replace the ODE-based models for a single population, either homogeneous or stratified by six age groups. In this section, we aim to develop a GNN-based surrogate model replacing the combined expert model for all counties of Germany. Building an independent NN for each county would not be sufficient, since we want to model the influence of proximity of population and mobility between different counties. This type of problem can be modeled with a GNN.

Zhou et al. [80] define a design process for GNNs, which we will walk through in order to define our GNN. For defining the graph type, the authors differentiate between directed vs. undirected graphs, heterogeneous vs. homogeneous graphs and static vs. dynamic graphs. Our mobility data indicates how many individuals travel from one node to another node. The direction of travel is important, which is why we have a graph with directed edges. Originally the graph-ODE model is a special graph type with multiple edges between two nodes, called multi graph. For our study, we generalize the graph model by simplifying the edge structure so that we have a standard graph with only one edge between two nodes. Node and edge features represent the same type of data, namely the number of individuals, so we have a homogeneous graph. Because the node features contain COVID-19 cases for different days, our graph is considered to be dynamic. Zhou et al. [80] define three task types, namely node-level, edge-level and graph-level tasks. We want to predict the epidemiological development for each node, so we operate on node-level. Node level tasks include node classification, node regression and node clustering. Since we predict specific values for each node, we perform node regression. From the supervision perspective we have a supervised learning problem, as node labels are provided. To summarize, we have a **supervised node-level regression problem on a directed, homogeneous and dynamic graph**.

4.4.1 Datasets and Tools

When generating data, we create data for all of Germany's 400 counties. The data is age-resolved as described in previous sections. Additionally to the distribution of individuals among compartments, we utilize information about mobility. As described in [6], the mobility data is extracted from a combination of data from the German Federal Employment Agency [81] and geo-referenced Twitter data. Details about the exact construction of mobility data can be looked up in [6].
For each county, we assign random percentages of people to the compartments for each run.

One simulation results in one data sample. This sample is then split into input and output. We use five days as input for the GNN and the task of the GNN then is to predict all of the remaining days.
The simulation can be performed for each of Germany's 400 counties or for a subset of those. As a result we have an input dataset of size [number of runs, number of counties, number of input days, number of age groups$\times$number of compartments] and an output dataset of size [number of runs, number of counties, number of output days, number of age groups$\times$number of compartments].
For an exemplary dataset consisting of 1000 runs, 400 counties, 5 input days, 30 output days, 6 age groups and 8 compartments, the input dataset has the dimensions [1000, 400, 5, 48] and the output dataset has the dimensions [1000, 400, 30, 48].
As in the Section before, we utilize a dataset that is based on the ranges as depicted in Table 2 for groups-dataset 1.

Spektral [44] is an open-source Python library for building graph neural networks. Other existing libraries are PyTorchGeometric[82] and DeepGraphLibrary [83], both based on PyTorch. Spektral on the other hand is based on TensorFlow. We use Spektral for defining the graph data, building, training and testing the neural network. The data is represented by a graph, where each node corresponds to one county. The node features are the distribution of the counties population among the epidemiological compartments for 5 days. The population is split into the mentioned six age groups. Between the nodes, we define directed edges. An entry a_{ij} of the adjacency matrix A is non-zero if an edge exists that goes from node i to node j. The edge features describe the normalized number of people commuting daily from node A to node B. The graph structure is illustrated in Figure 18.

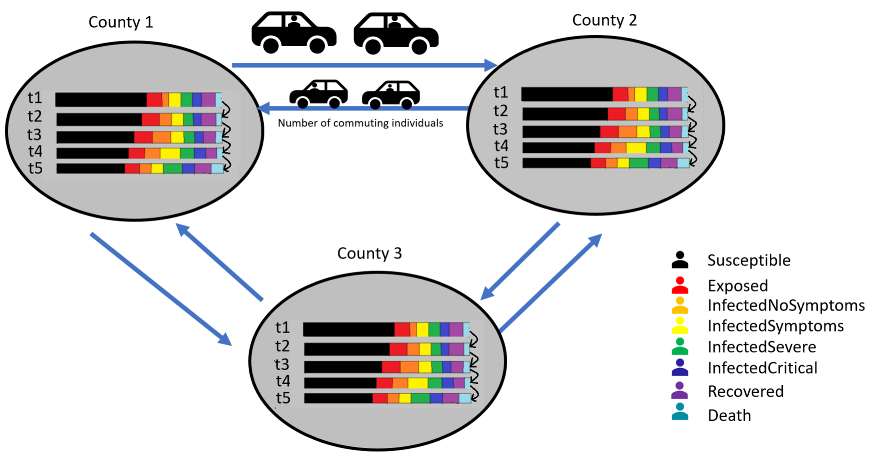


Figure 18: GNN: Graph Structure

The dataset consists of n (n = number of runs) graphs. In many other spatio-temporal graphs [52], [53], [84], each graph represents one moment in time and the goal is to predict the graph and its labels in the next time step. In our case, the graph contains information of the last five days represented by the node features. Further, instead of predicting only the next time step, we predict the distribution of each county's population along the compartments for multiple days at once. This information is represented by the graph's label, which contains information about the distribution of the population along the compartments for all 400 nodes.
Spektral implements specific data types. In our first experiment, we want to make predictions without changing any of the edge features, meaning the mobility data stays the same. This data type is described as mixed mode and is for models where the graph structure and adjacency matrix are the same for all data samples, which is the case for our graph.

4.4.2 Parameter Testing

Spektral offers a wide variety of GNN layers, which can be categorized into two groups. The first group is composed of GNN layers that utilize node features and an adjacency matrix for their computations. The second group additionally includes edge features into their calculations which makes the input larger and model more complex. This is why they need a lot more computational resources than those available for this study. Due to this limitation, we focus on the first group of models. The input for those GNNs consists of a tuple with two elements, namely the adjacency matrix and the node features. Figure 19 illustrates the input composition for the advanced example of a model with one damping. Dampings are changes in the contact patterns of individuals, which can be implemented in the model and will be included in experiments in Section 5.2. The objective of the current section is to find the optimal GNN model architecture which we will then test on datasets with and without dampings in Section 5.2. The input structure as depicted in Figure 19 is adapted intuitively for the cases without dampings, simply by not appending information about the contact reductions to the input array. The main difference to the models describing one population is that the node features cannot remain splitted by day like a time series. Instead, they are expected to follow the shape [(batchsize), number of nodes, number of node features]. This is probably the case, because temporal GNN models implement the aspect of time usually by creating a graph for each point in time. Nevertheless, we wanted to preserve our non-recursive approach and implement a direct model regarding input and output. This implies that we provide all input information at once, even though it covers multiple days, and that the model has to predict all remaining days at once. Thus, the input consists of a flat array that contains the distribution of all six age groups among the 8 compartments for five days ($6 \times 8 \times 5 = 140$) for each node.

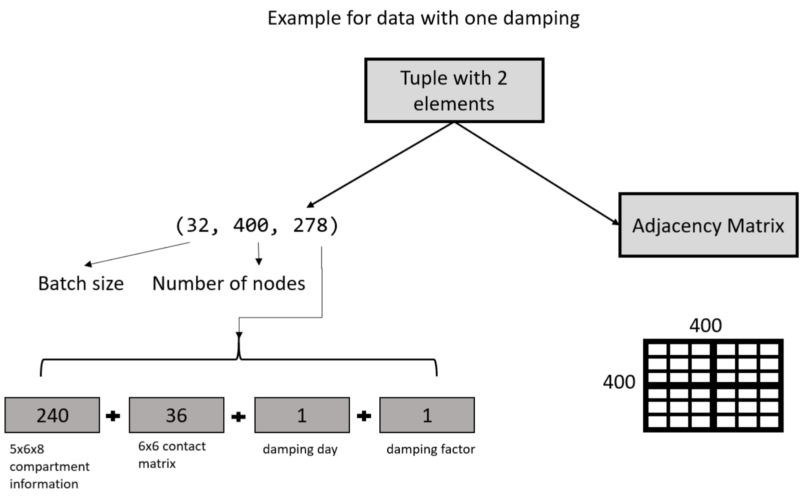


Figure 19: GNN: Illustration of Input with damping

We select four types of layers and perform a grid search in order to find the best model architecture. All models are composed of one, two or three layers of the specific graph convolutional layer and a dense layer as the output layer. The number of output channels of the convolutional layer represents the width of the model, which can be 32, 128, 512 or 1024. The dataset for this experiment contains 1000 samples with five days of input for all 400 nodes, labels for the next 30 days and no changes in contact patterns. We specifiy Adam as optimizer and relu as activation function. We should mention that cross validation is omitted in the grid search, due to reasons of computational efficiency. The maximum number of epochs is 1500 with a stopping criterion, taking effect when the monitored metric has stopped improving for 100 epochs. The data was split into training, validation and test set according to a 80/10/10 split. All models had the same data in each set. We use the MAPE as performance measure. The compartments are unequally distributed, therefore a percentage measure ensures a fair comparison of performance. We shortly introduce the four layers, which were all described in detail in Section 3.2.

Th first layer is the GCNConv layer, which is based on the Graph Convolutional Network (GCN) as proposed by Kipf and Welling [33]. It is a spectral approach with a modified adjacency matix with self loops.
The next layer is the ARMAConv layer based on the paper from Bianchi et al. (2021) [45]. Like the GCNConv layer it utilizes a spectral approach and is implements convolutional structures while applying concepts from auto-regressive moving average to the learning process.
The APPNPConv Layer is based on a work from Gasteiger et al. [46] and can also be categorized as an apprach with convolutional and spectral structures.
Lastly, the GATConv layer which is based on Veličković et al.'s introduction of Graph Attention Networks [42] is not a spectral model but a spatial model implementing an attention mechanism.

Figure 20 shows the results of the grid search considering number of layers and number of channels of the five different convolutional layers. The number of channels refer to the number of output channels and denotes the width of the model.
The GCNConv layer remains around 33% test MAPE for each of the model architectures with a best score of 32.26%. Likewise, most architectures of the GATConv layer reach around 33% test MAPE. The best GATConv model is the smallest one with 20.54% MAPE. The two other models show more variance in their performance. The APPNPConv varies between 13.14% and 28.72% test MAPE. The best model is the model with only one layer and 1024 channels. The ARMAConv model is the only model that reaches test scores below 13%. The best performing architecture is the biggest tested architecture with three layers and 1024 channels per layer. Since the best architecture is the one with the most layers and the highest number of channels, it could be possible that larger models perform even better. This is why we conduct a further experiment, enhancing the number of layers and number of channels per layer.
Figure 21 depicts the results. We can see that the changes after adding more than 1024 and more that four layers to the model become small and affect the second or third decimal of the mean absolute percentage error.

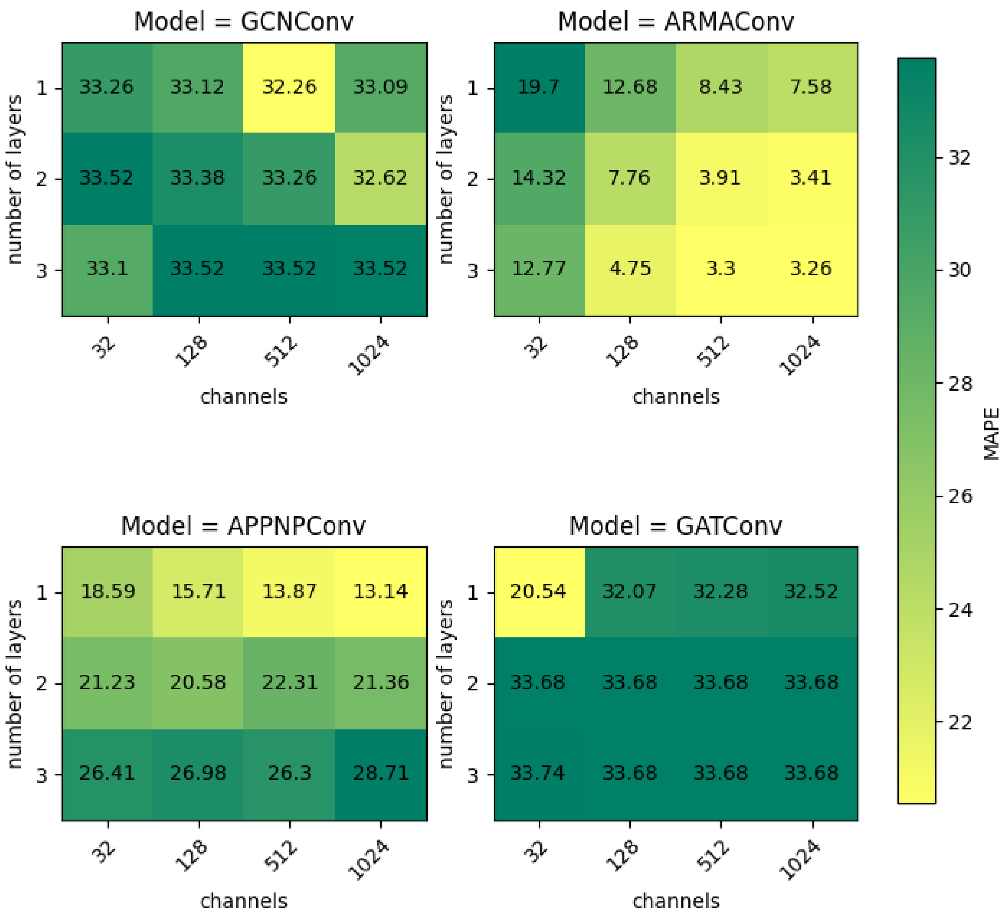


Figure 20: GNN: Heatmap for number of layers and number of channels

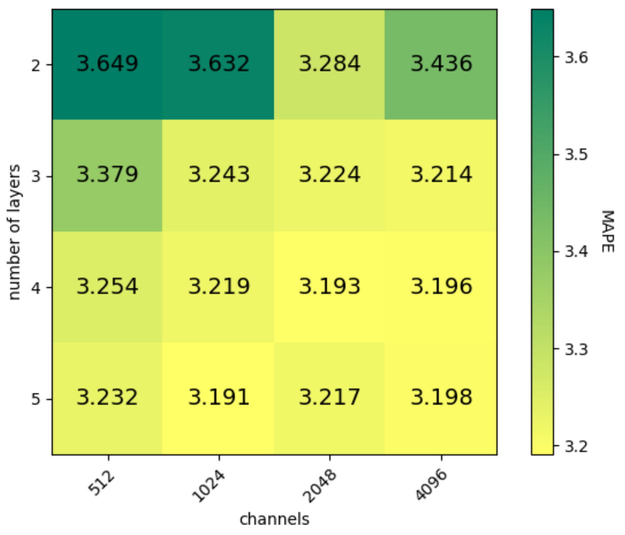


Figure 21: ARMAConv: enhanced grid search

As described previously, cross validation was excluded from the grid search. Nevertheless, we want to ensure that we find the best model architecture without over fitting to a specific data split. Therefore, we conduct an experiment with five-fold cross validation for specific ARMAConv models. We choose all architectures that had at least two and at most five layers and at least 124 channels and at most 1024 channels. This way we exclude the larger models which are more time consuming in training without improving the performance significantly. The results as shown in Figure 22 expose that the split that was used in the grid search without cross validation was extremely beneficial regarding the performance of the models. Now, the mean of the performance on all five test-fold is higher than before for every model architecture. Before, the models were evaluated on test data which is now in in the first split. Some models that performed similar on that first split, such as the 4-512 model (4 layers and 512 channels) and the 4-1024 model, which had a difference of 0.035%, have a mean difference of 6.7% in the case with cross validation. The best model with a mean test performance of 4.239% is the 4-512 model.

Upon closer inspection of the distribution of the performances on each of the five test datasets in Figure 23 we can see that some models experience high variance in performance depending on the data which occurs in the test split. Some models achieve up to 33% MAPE for specific splits. The best model has the lowest variance between splits. It is interesting that increasing the number of layer or the number of channels has such an distinct influence on the performance on specific data splits.

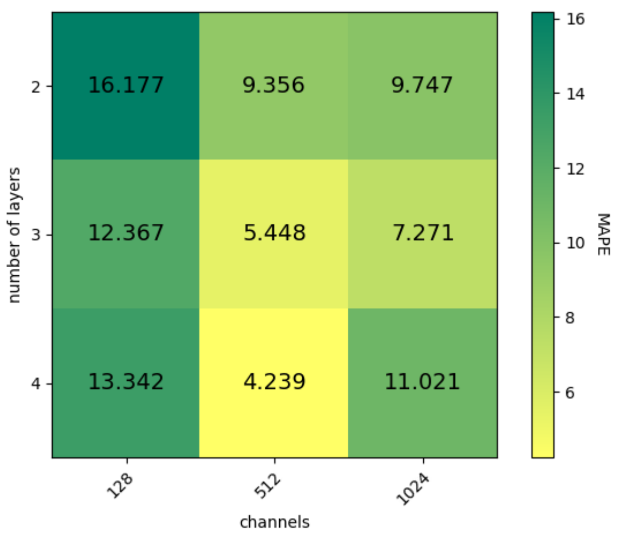


Figure 22: ARMAConv: with cross validation
The depicted values are the mean validation values of the five folds.

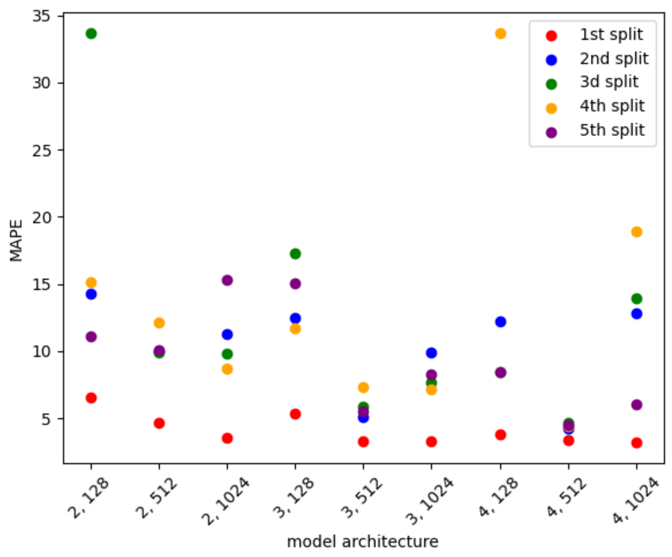


Figure 23: ARMAConv: scatterplot for all cross validation folds

To optimize hyper-parameters, we tested combinations between the optimizers Adam, Nadam and Adagrad and the activation functions elu, relu and linear. To save computational resources, we reduced complexity and only conducted these experiments for the best architecture of the ARMAConv layer which is the model with 4 layers and 512 channels. Due to reasons of computational efficiency we conducted the grid search without cross validation, meaning that we only consider the first split. In Figure 24 we can see that the Adagrad optimizer yields four times higher MAPE scores than the other optimizers. All other combinations are between 3.19% for the combination of Adam and linear activation function and 3.38% MAPE for the combination of Adam and relu activation. Since we only evaluate the data on one test dataset, taking an other measure into consideration is reasonable, as the results are not as reliable to the decimal places, when cross validation is excluded. Another criterion that can be relevant for the identification of the most favorable optimizer and activation function is the efficiency regarding training time. Figure 25 illustrates the training time in minutes for each of the optimizer-activation combinations. One combination stands out particularly compared to all others, with a training time of 53 minutes, which is close to half as much as the other models require. In order to save computational resources in the following experiments, we decided to select the model with four layers and 512 channels with Adam optimizer and relu activation function as the baseline model architecture.

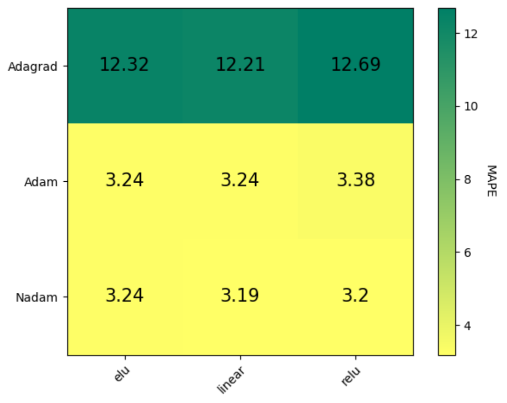


Figure 24: ARMAConv: test MAPE for grid search for optimizer and activation for the 4-512 model

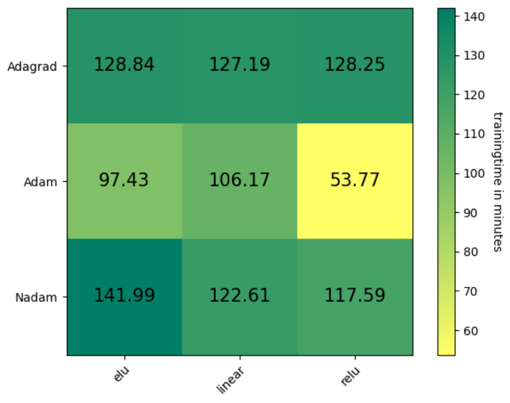


Figure 25: ARMAConv: training time for grid search for optimizer and activation for the 4-512 model

All five convolutional layers expect a binary or normalized adjacency matrix or a modified Laplacian matrix, which is based on a binary adjacency matrix as well. In the grid search, we implemented a binary adjacency matrix or the respective conversions of it as input. The entry a_{ij} of the adjacency matrix A is 1 when there is an edge between nodes i and j and 0 when there is no edge.
For specific applications it is possible to use a variant of the adjacency matrix, which we denote as non-binary adjacency matrix, where the entries of the matrix are equal to the weight on the edges [85]. In that case, a_{ij} does not only indicate whether there is an edge or not. Instead, the entry in the non-binary adjacency matrix equals the weight assigned to the edge. In our case, the edge weight represents the number of individuals commuting between node i and node j.
In an experiment we tested whether replacing the binary adjacency matrix with an

non-binary adjacency matrix with edge weights affects the performance for the best ARMAConv model. Changing the entries of the adjacency from binary values to the actual edge weights impairs the performance of the model from 4.24% to 6.78%. On the one hand it is surprising because a binary adjacency provides no information about the mobility for the model, yet it performs better. On the other hand the result is not surprising because the ARMAConv layer expects a binary adjacency and providing other data than expected can lead to computational problems.

Several characteristics of the ARMAConv model as described in Section 3.2, are implemented as parameters in the ARMAConv layer from Spektral. All experiments before were conducted with the default setting. These include the order of the $ARMA_K$ filter, which is one. The order indicates the number of parallel stacks in the layer. The number of iterations to compute each $ARMA_1$ approximation is 1 per default and weight sharing within each $ARMA_1$ stack is set to False. Within a GCS stack $ARMA_1$ stack, dropout can be implemented, but by default the dropout rate is zero.
We performed 5-fold cross validation for each of these experiments on dataset 1 like before. The results are presented in Figure 26 and a comparison to the baseline which is the 4-512 model with relu, Adam and binary adjacency is included. Enabling weigh sharing and increasing the number of iterations lead to higher mean validation MAPE. On the other hand, increasing the order improves the baseline model with a lowest MAPE of **3.46%** for the model with 3 stacks.

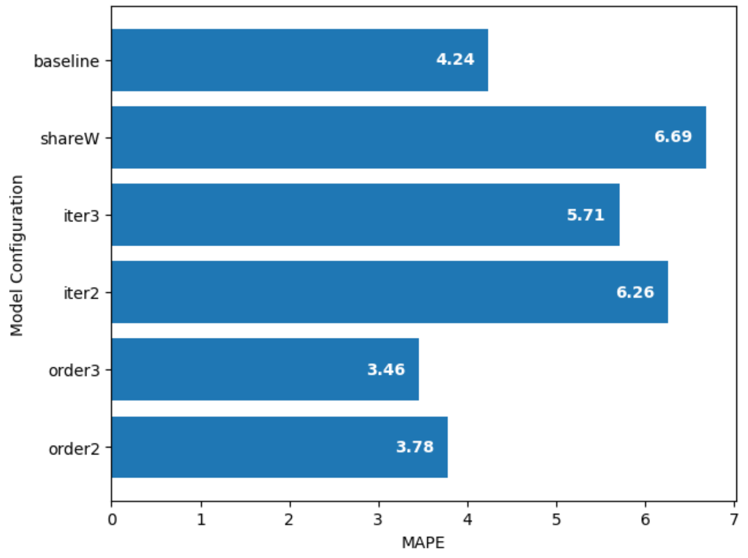


Figure 26: ARMAConv: validation MAPE for possible configurations
The order describes the number of parallel $ARMA_1$ stack per layer. Iterations describe the number of iterations that are applied to calculate each stack. Share weights is a boolean variable which allow weight sharing in each stack when set to True. Lastly, dropout indicates a dropout rate of 0.2 for skip connections.

Consequently, in the experiments in Section 5, we will only employ the ARMAConv model with four layers, 512 channels, Adam optimizer, relu activation function, an order of three and binary adjacency matrix since it is the most convenient model regarding performance and training time. Figure27 illustrates the model's architecture.

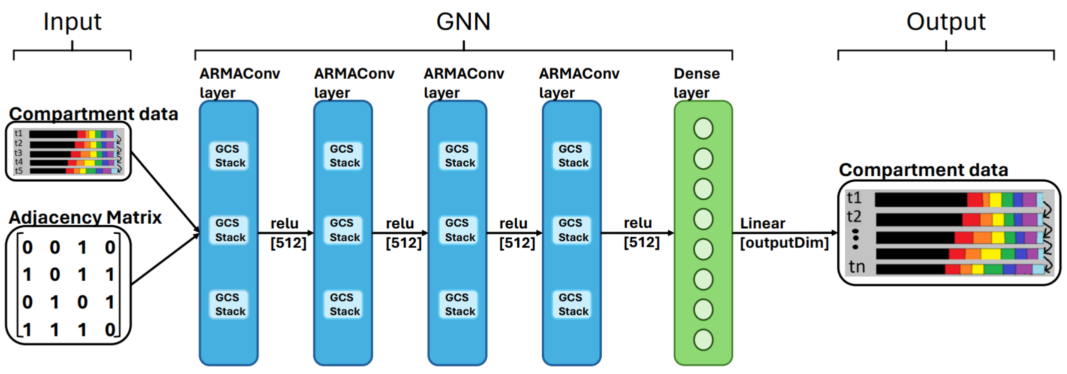


Figure 27: ARMAConv: resulting model architecture
The graphic illustrates the input data, the AGNN model and the output. The input consist of the node features which are the distribution of individuals along the eight compartments and the adjacency matrix of the graph. The GNN is build with four ARMAConv layers, each of those containing three GCS stacks. The input is passed to the layer then fed to the relu activation function. The output dimension of the ARMAConv layers is 512. Finally, the dense output layer transforms the data to the output dimensions required by the labels through a linear activation. The output data are the prediction for the next n days.

5 Numerical Results

5.1 Model Performance SimpleNN and GroupsNN

In this section, we will to present the performance of the SimpleNN and the GroupsMM model on different tasks with increasing difficulty. The difficulty of the tasks is altered by the number of days the model has to predict and the number of changes in contact patterns which have to be included in the calculations.

First, we will analyse how our models perform with increasing number of predicted days. In the previous experiments we predicted the next 30 days. Now, we increase the number of days by multiples of 30, so we have experiments for 30, 60, 90, 120, and 150 days prediction. We use the best performing models with and without without age-resolution. For the simple model, it is the LSTM model with 32 units, zero hidden layers and Adam optimizer. For the model with age groups, it is the LSTM with zero hidden layers and 512 units per layer.
We utilize the data with the wider input ranges from dataset 2 for this experiment. The data was generated with the ranges as depicted in Table 1. We split the data according to a 70/20/10 split. 70% of the data will be used for training, 20% for validation and 10% for testing. The results show the evaluations on the test dataset.
Figure 28 illustrates the MAPE for both models. For the simple models, the lowest score of 0.0702% is achieved in the prediction of 30 days and the error is rising when the number of days is increased.
The model with age-resolution reaches the lowest error when it predicts 60 days with 0.3225% test MAPE. The worst models for both, SimpleNN and GroupsNN are the models with a 150 day prediction horizon.
A trend of rising error with rising number of days that have to be predicted is visible, as expected, because we perceive longer prediction as more difficult. Meanwhile, the slight drop of MAPE for the SimpleNN for 60 days in comparison to the model for 30 days is surprising.

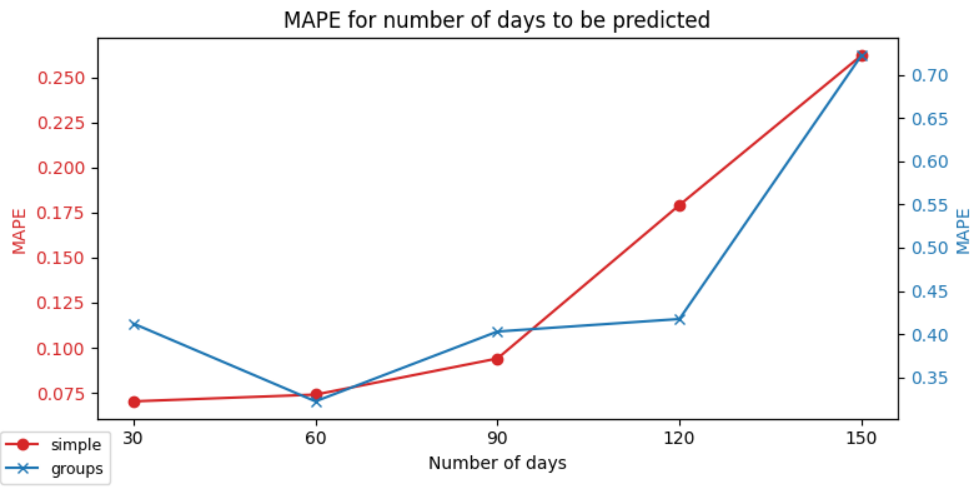


Figure 28: SimpleNN and GroupsNN: MAPE depending on prediction horizon

To gain a better understanding of the model we explore the characteristics of the loss in more detail. Additionally to the MAPE metric, we take the absolute error in Figure 29 and Figure 33 into consideration. Even though the MAPE has useful characteristics which can support balanced training of all features regardless of their size, absolute error can provide additional important information. The mean absolute error (MAE) indicates by how many individuals the prediction of the model deviates from the test labels. Before we calculate the MAE we have to reverse the normalization we implemented beforehand. Figure 29 provides the error for each day. We utilize the model with age-resolution that predicts 150 days. We can see that the MAE varies between 100 and 200 individuals for most of the time, with some extreme peaks up to over 600 individuals on day 35. This plot shows that the loss per day varies, but does not follow a specific trend.

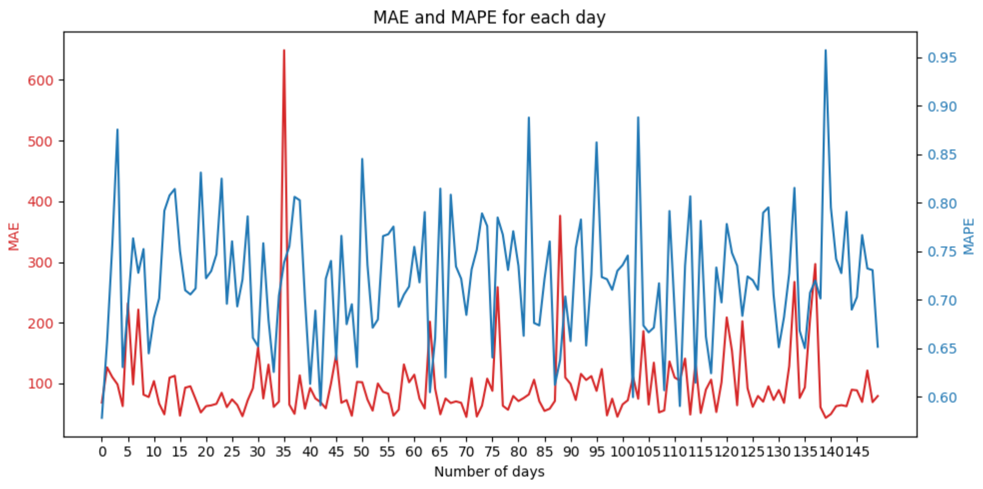


Figure 29: 150-day-GroupsNN: MAE and MAPE

As we have seen in Figure 29, the average error for the 150-day-model varies for each day, but no significant trend is visible. So, from Figure 29 alone, it cannot be deduced that the prediction for farther days yields higher errors.
We want to to take a closer look on the performance differences of the various models (30 day model, 60 day model, etc.). Our objective is to answer the question whether they perform differently in the first 30 days of prediction. Figure 30 shows the test MAPE for the first 30 days for all five models without age-resolution (30, 60, 90, 120, 150 days) per compartment. The results represent the mean of the results on a test dataset with 1000 data samples. For most compartments, the lines overlap. This means that the MAPE is in similar ranges for every model. Also, there is visible variance in each of the models. The 120-day model seems to perform worse in the early days for the Exposed, InfectedNoSymptoms and InfectedSymptoms compartments. Even though the 150-day-model overall MAPE is high, it performs good in the first 30 days.
However, in some compartments, the models seem to have higher test MAPE in the first days. There is a slight trend indicating better test performance on later days, which is surprising and should be analyzed in more detail.

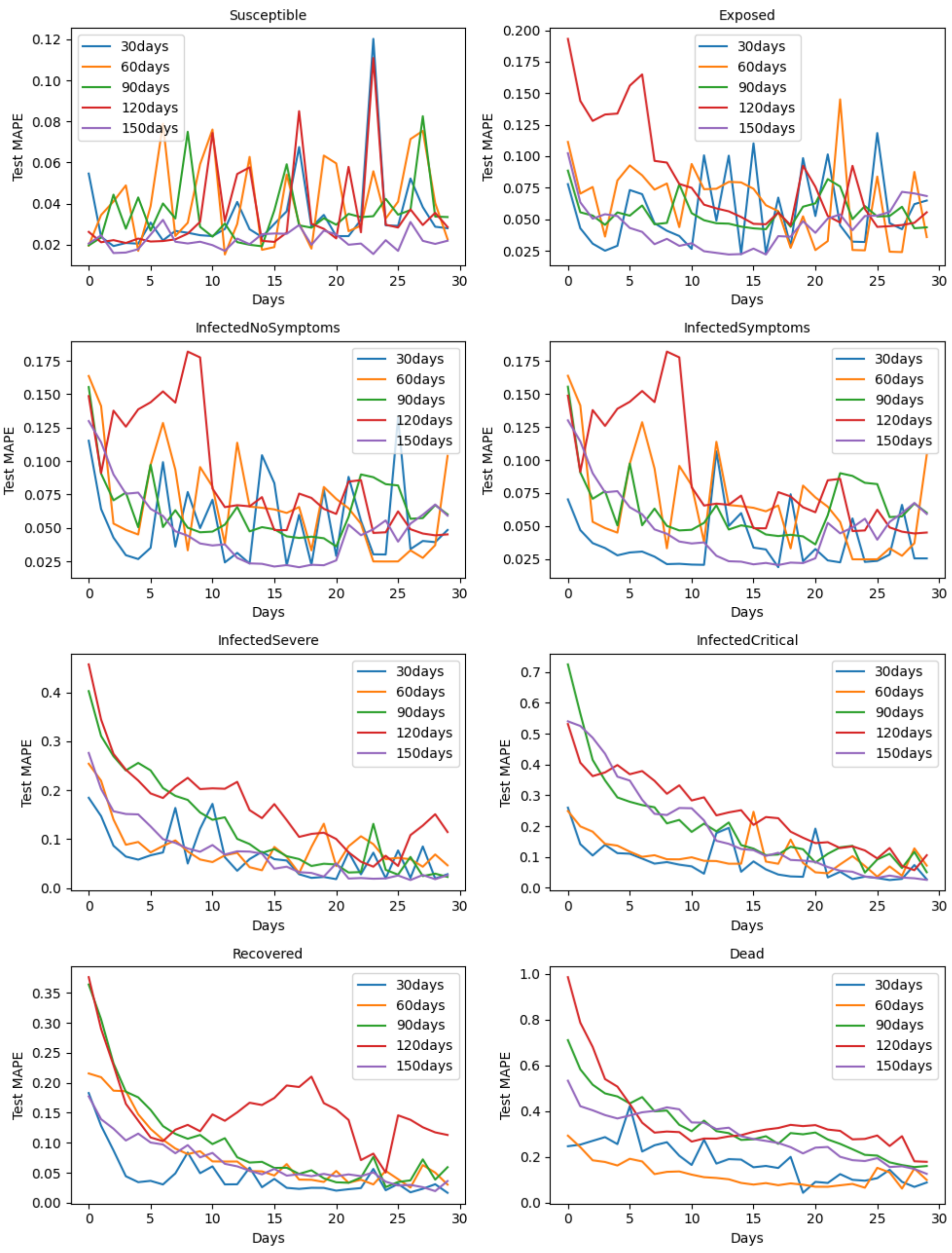


Figure 30: SimpleNN: Comparison of models trained for specific number of days on performance in first 30 days

To better understand the error we additionally visualize the course of the disease for each compartment. Figure 31 depicts the predicted values and the test labels for the simple model without age groups for a prediction horizon of 150 days for one element from the generated sample set.

Both curves are difficult to distinguish from one another, because the error in this task is small. The SIR model assumes that the number of infected individuals approaches the value zero ($I(\infty) \to 0$) when t approaches infinity [86]. Since we do not consider reinfections in our model, there will, by default, occur a time when no infections take place anymore. We can see that the point in which the curves start to become flat differs for each compartment. Compartments like Susceptible, Exposed or InfectedNoSymptoms flatten after approximately 70-80 days. Otherwise, the compartments of severely and critically infected individuals as well as the Death compartment experience notable decline or increment of individuals until 120-130 days. Consequently, it is not suggested or necessary to predict more than 150 days in our model without reinfections.
If we compare this plot with Figure 53 in the Appendix, which is based on dataset 1, we can see that due to the wider range at the input generation, the curves in Figure 31 represent another epidemic starting situation. As we have described in Table 1, random number of individuals are assigned to all compartments except the Susceptible compartment and the Susceptible compartment contains all remaining individuals. Since more individuals can be assigned to other compartments less individuals are are in the Susceptible compartment. This is why the number of individuals in that compartment declines faster. Other compartments such as the Exposed or all of the Infected compartments reach their peak about 20 days earlier in dataset 2 compared to the plots based on dataset 1.

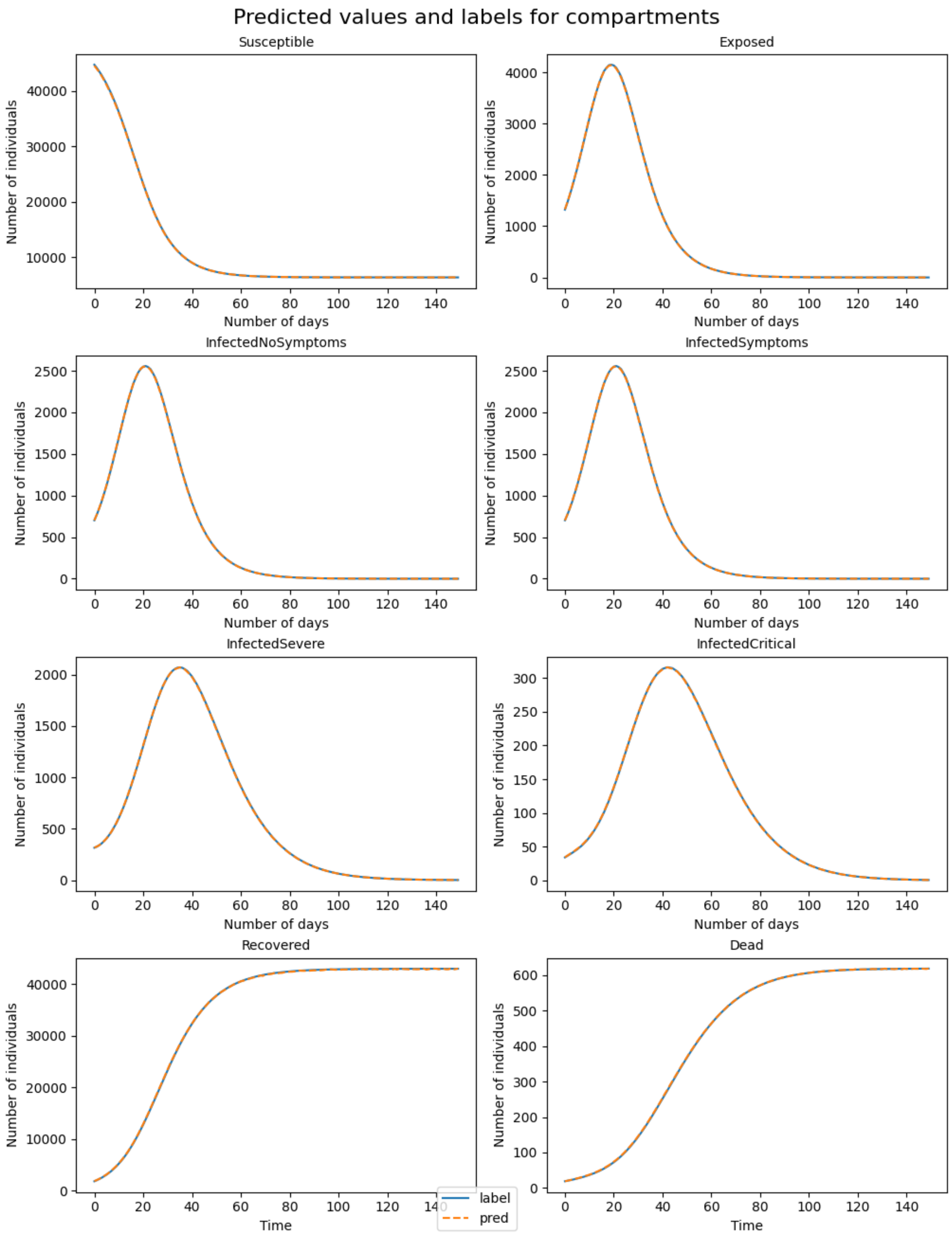


Figure 31: 150-days-SimpleNN: Number of individuals per compartment

Furthermore, the size of the error in each compartment is of interest. Figure 32 illustrates the test MAPE of each compartment for the SimpleNNs for a 30, 90 and 150 day prediction horizon. The data represents the average MAPE for the 1000 test samples of dataset 2. We can see that the Susceptible compartment yields the smallest MAPE for all three models. The Dead compartment and the InfetedCritical compartment have the highest errors for the 30-day-model while the 150-day-model reaches the highest MAPE scores in the InfectedNoSymptoms, InfectedSymptoms and Exposed compartment. These errors are exceptionally high with more than 0.5% MAPE.

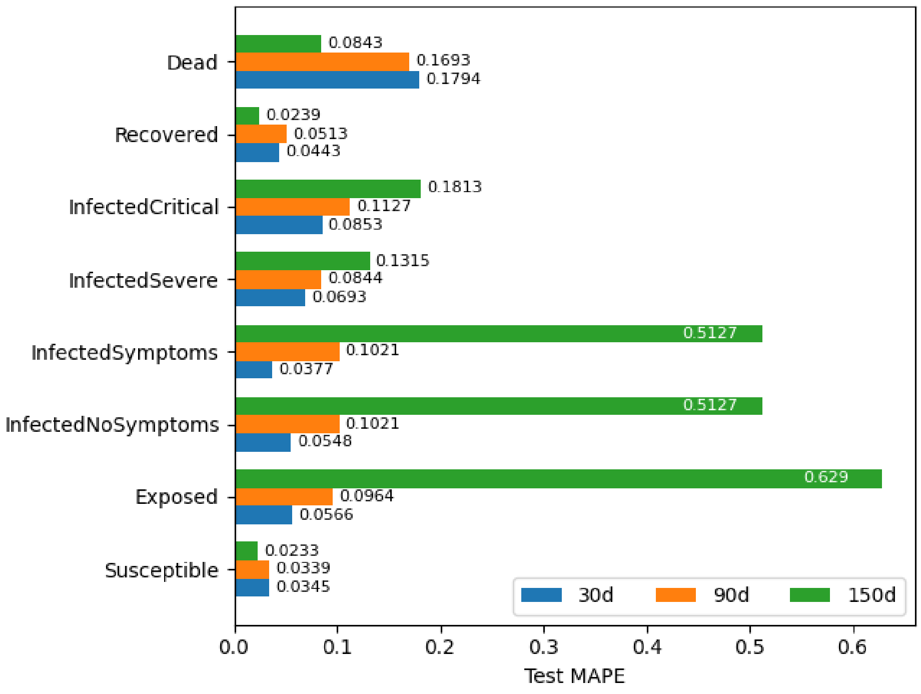


Figure 32: SimpleNN: MAPE per compartment for 30, 90 and 150 days

Figure 33 illustrates the average test MAPE and MAE per day for each compartment. Again, the lines represent the average test MAPE for 1000 test samples for the SimpleNN on dataset 2. We can see that the MAPE and MAE curves run rather opposed to each other. The absolute error has some similarities with the course of infection as depicted in Figure 53. When the number of infected individuals rises, the absolute error peaks. Even though the absolute error is high, because of the high number of individuals in the respective compartment, the percentage error is small. Contrary, the MAPE is higher when the absolute error is low. Even though the predicted values differ only slightly from the labels, the percentage error is high, because the numbers are very small so that even a slight difference makes a significant impact on the MAPE.
In Figure 34 we provide a variant of Figure 28 with a change from MAPE to MAPE that changes the source of the curves. As presented in Figure 34, the MAE is the highest for the 120-day model. The MAE for the 150-day SimpleNN is around 11 which is half of

the error for the 30-day-model.
The model with age groups has the highest error for the 30-day-model, a smaller peak for the 90-day-model and the second highest MAE is reached by the 150-day-model.

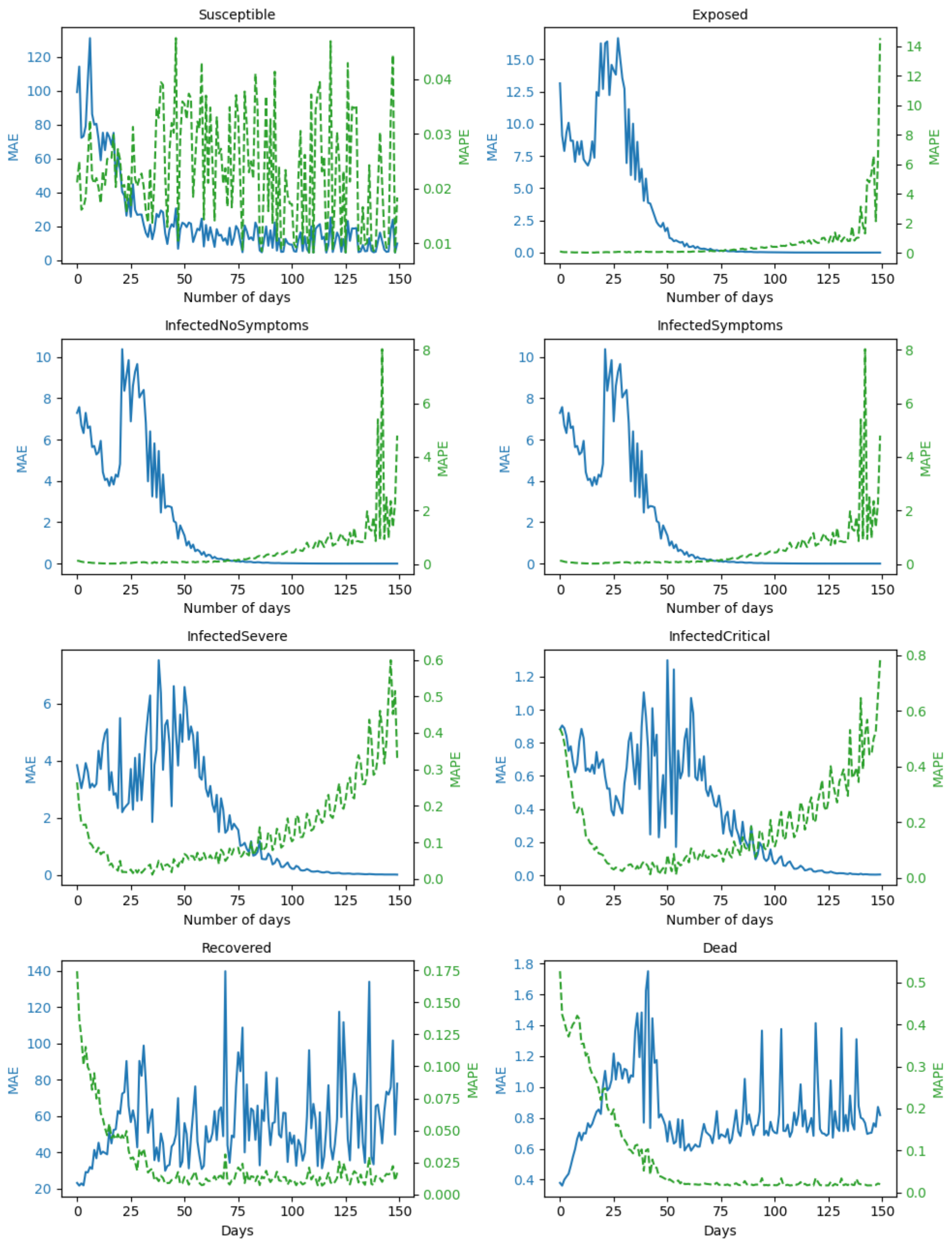


Figure 33: 150-days-SimpleNN: MAE and MAPE for each compartment

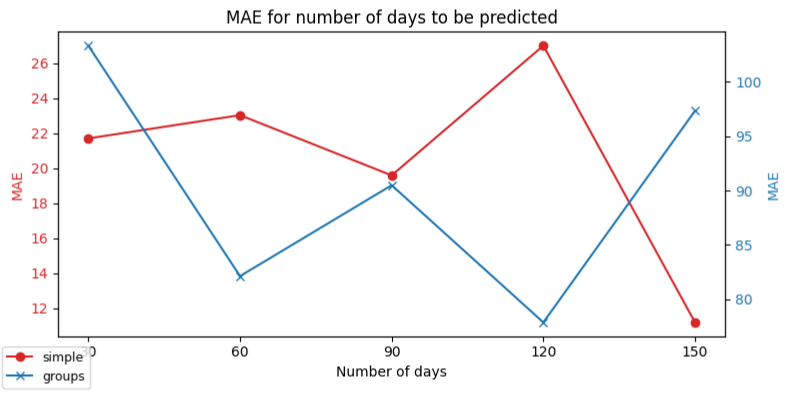


Figure 34: SimpleNN and GroupsNN: MAE depending on prediction horizon

Predictive Models with One Change in Contact Patterns

As a next step we introduce dampings to our model. We only apply damping factors to the whole contact matrix, so no age-specific contact reductions are analyzed. Furthermore, no differentiation between the place of contact (i.e. ”Home”, ”School” etc) is considered. The damping can take place on any day between day five and day 35. The damping factor is randomly generated between 0 and 0.5.
First, we analyze how the models described before perform on a dataset with one damping, without any further information (Model 1 in Figure 35). We take the best model architectures as detected in the grid search and evaluate them on the same test data set. The task is to predict the next 30 days.

Compared to the test MAPE of the models without any changes in contact patterns as described in the heatmap (Figure 15) the error increased for all models, when a change in contact patterns takes place but no information is provided as input to the model. Instead of MAPE errors between 0.4% and 1.9%, the values range between 4.8% and 6.8%. The ranking of the models changes, with the LSTM model still making the most accurate predictions, followed by the MLP model and the CNN. In the experiment without damping the CNN performed better than the MLP.

Next, we try to adjust the input for the models with the aim to improve the performance of the models by providing information about the changes that take place. Previously, our input was an array, which contains the numbers of individuals in each compartment for five days. In the case of the LSTM model, appending the damping date to the array improves the average test MAPE from 4.84% to 3.01%. The date does not provide any information about the extent of the contact reduction. When generating data, we store the damping date and the contact matrix with reduced contacts which results from applying the damping to the matrix.

In a next step, we appended the flattened 6×6 damped contact matrix to the input array. This additional information led to major improvement in the LSTM model yielding an average test MAPE of 0.55%, which is about nine times better than the model without information about the damping and only around 0.05% worse than the LSTM performance on a dataset without dampings.

We apply this input composition of compartmental data, damping date and damped contact matrix to the best LSTM, CNN and MLP model, as derived from the grid search (Model 2 in Figure 35). As presented in Figure 35, the LSTM remains the model with lowest MAPE. For all models the MAPE increases by adding a change in contact patterns to the data. Even though the information about the time of the damping is not directly represented by a time series, but simply as one entry of an array, the models seems to be able to utilize the provided information meaningfully.
The behavior of the CNN is very striking as the MAPE decreases drastically when information about the damping is appended to the input.

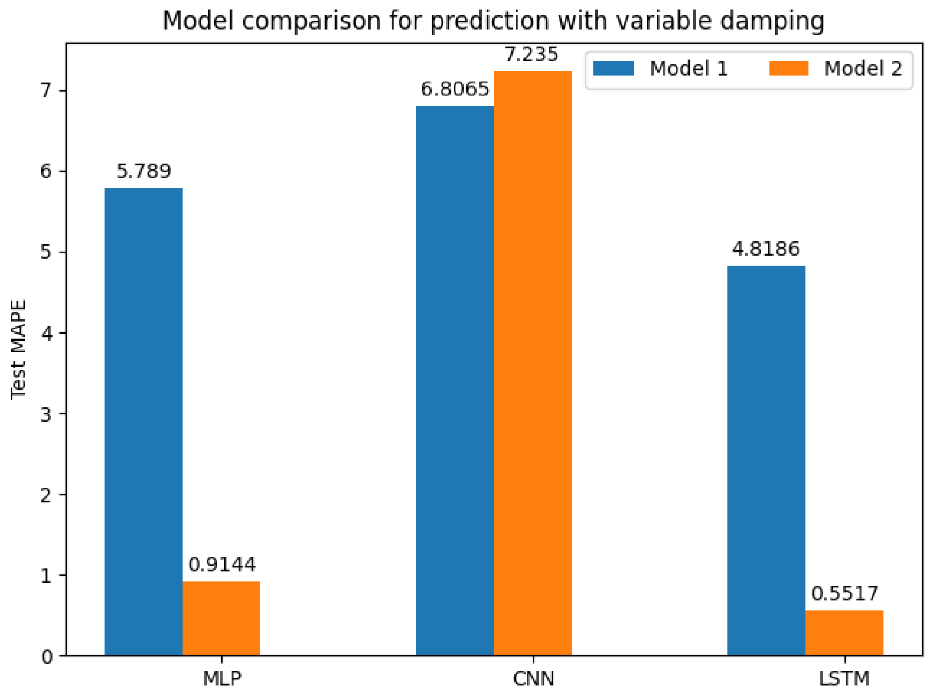


Figure 35: GroupsNN: Performance for models with one damping
Model 1 describes the models that only have data about the compartments available and no information about the damping that takes place is provided. Model 2 has compartmental data, damping date and damped contact matrix available.

Predictive Models with Multiple Change Points

When considering time spans of multiple weeks and months, it can be relevant to model more than one change in contact patterns. NPIs are enacted depending on the development of the pandemic and official rules can change regularly. In our experiments we always draw the days of contact changes randomly, but keep a a minimum distance of

seven days between two contact reductions. Further, we adjust the sampling, so we approach a uniform distribution of the damping days in the training data.
When multiple dampings are implemented, the damping factor is always applied to the baseline matrix. This means that in the course of time-span we examine, the contacts are not always constantly reduced. Instead, when the contact reduction is smaller in $t + 1$ than the contact reduction in t, the damping in $t + 1$ operates as a relaxation of NPIs, allowing more contacts at time $t + 1$ than at time t.
When we want to implement more than one contact reduction, it is reasonable to predict more than 30 days. With respect to the course of the curves as illustrated in Figure 31, we decided to predict 100 days. This way, the model has to predict the peak for every compartment.

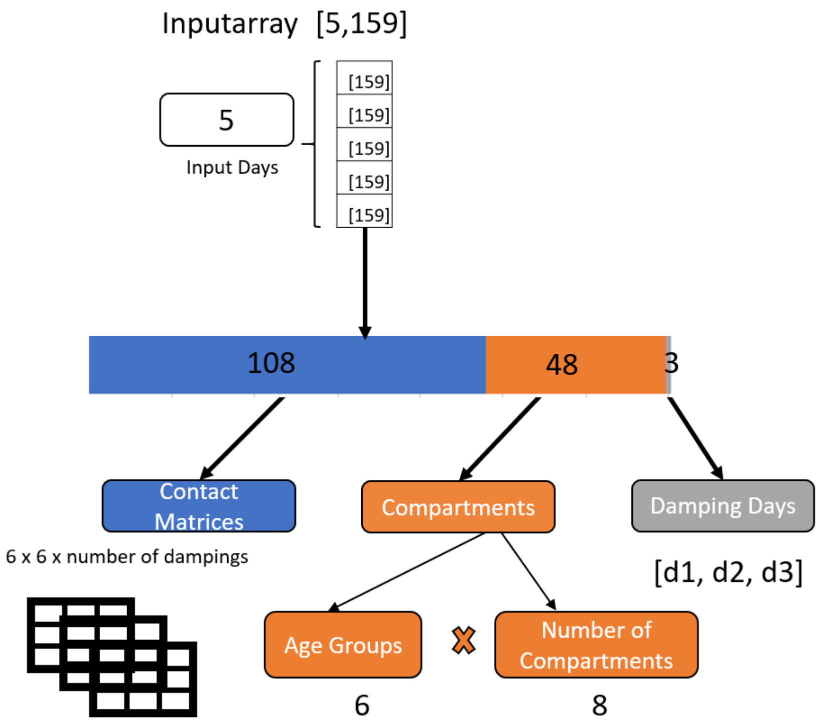


Figure 36: GroupsNN: Illustration of input composition: 3 damping example

We generate datasets with two, three, four, and five dampings with 10.000 samples in each dataset. The input data contains five days and the task is to predict the following 100 days. Figure 36 shows how the input is composed for the case where three dampings are implemented. As described in the example with one damping, additionally to the information about the distribution of individuals among the compartments, information about the dampings is appended. Now that multiple dampings are considered, multiple contact matrices and multiple damping days are appended to the array. The result is a two dimensional array with 159 (for the case of three dampings) entries for each of the five input days. The array contains the compartmental data, the contact matrices and the damping days. For each day the respective distribution of individuals among compartment is given as well as the damping information. The additional damping information is the same for each of the five entries.

To create a baseline, we evaluate how MLP, CNN, and LSTM perform in the prediction of 100 days on a dataset without dampings. We employ the best models from the grid search for the prediction for one population with age groups for that task.
The MLP and LSTM model remain exactly the same as in the task before without dampings. The CNN with the architecture as before yields a high error. This is why we adjust the CNN slightly to adapt to the larger input array. Even though a Pooling Layer is a substantial component of a CNN, we omitted it in the tasks before. The reason was that we apply the CNN, which is originally designed for visual data such as images, to a regression task. Therefore, we wanted to create a model with minimum architecture without sticking to established best practices, which might not fit our task. Now, as the input size increases, we can profit from the complexity minimizing characteristics of a Pooling layer. It increases training speed noticeably. We reduce the number of filters used in the convolutional layer from 1024 to 64, apply a 50% dropout to reduce complexity and to avoid over-fitting and feed the data to a Max-pooling layer. Next, the data is flattened, before it proceeds to two Dense layers with 1024 units, like before.

The results as pictured in Figure 37 show a monotonically increasing trend for the relationship between the test MAPE and the number of dampings for the LSTM model. The more changes in contacts patterns are applied, the higher the MAPE becomes. The LSTM is the best model for every number of dampings except in the case with five dampings where it reaches 1.47% while the MLP has a MAPE of 1.28%. The CNN performs worst with an error more than five times higher than the LSTM. This is noteworthy, because the CNN was better than the MLP in all previous tasks. The MLP remains below 1.28% for all scenarios with the highest error in the five-damping case and the lowest MAPE of 0.91% for the model with one damping.
As in all experiments before, the depicted results are based on the more variable dataset 2.

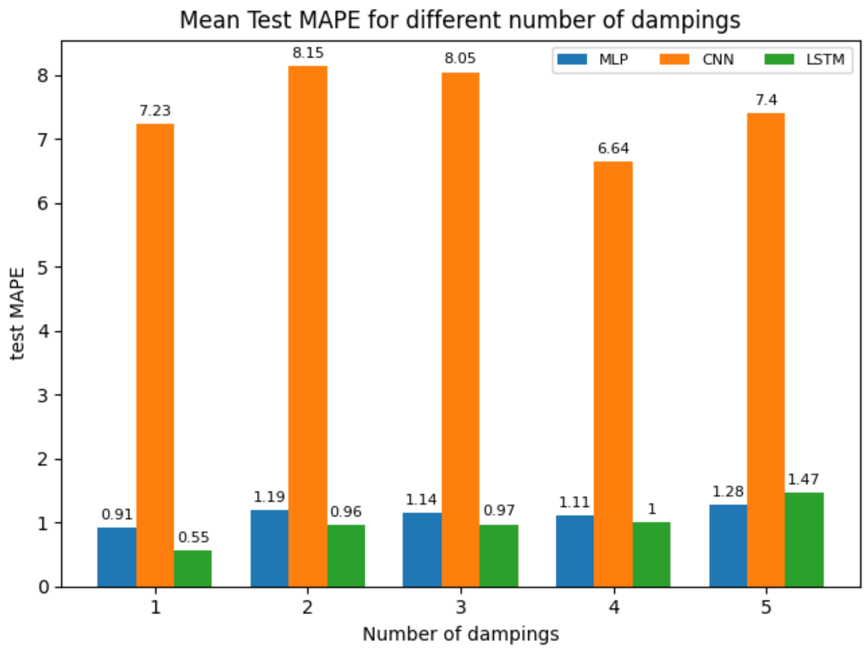


Figure 37: GroupsNN: Model performance depending on number of dampings

5.2 Model Performance GNN

In the following experiments we always utilize the best ARMAConv GNN from Section 4.4, which was a ARMAConv model with 4 layers, 512 channels, 3 parallel stacks per layer, binary adjacency matrix, Adam optimizer and relu activation. The data used in this experiments is generated with wider ranges as depicted in Table 2 for the case of groups-dataset 2.

For our first experiment we create datasets with 30, 60 and 90 days. We omit the 150-day and 120-day experiment, since the previous illustrations, like Figure 31, showed that after approximately 90 days, no significant epidemiological action takes place. We perform 5-fold cross validation in all three experiments.
In Figure 38 we can see that the performance on the separate splits of each model are close to each other, indicating relatively similar data in each split. The results in Figure 39 show a clear trend of decreasing error with increasing number of days in the prediction. The 30-day model has the highest mean validation MAPE with 6.18%, the 60-day model has 3.86% MAPE and the best model is the 90 day model with 2.97% MAPE. These results are surprising as we would expect higher errors for longer predictions.

Moreover, it is interesting that even though the input length of five days remains the same and the prediction output is generated in one step, long-term predictions can be executed with such a low error.

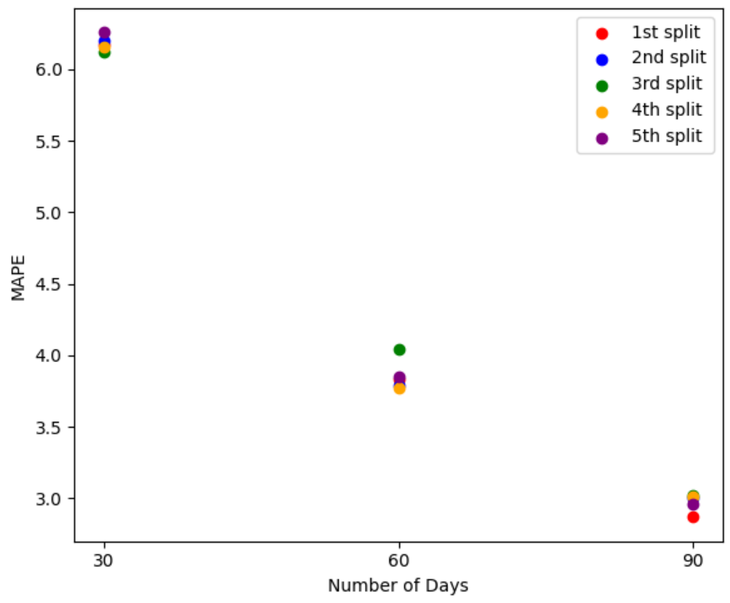


Figure 38: ARMAConv: validation MAPE for cross validation folds for models with variable number of prediction days

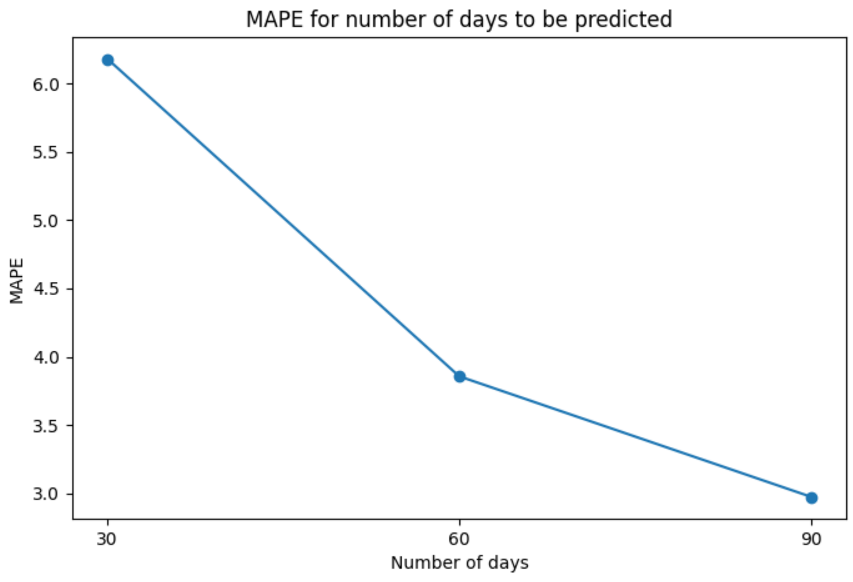


Figure 39: ARMAConv: MAPE performance for models with varying number of days in prediction

These results depict the mean validation loss of all folds of the cross validation.

Figure 40 depicts the MAPE and MAE for models that were trained on 80% on the data, validated on 10% and then tested on the remaining 1000 samples. The MAPE curve follows exactly the curve as presented in Figure 39. The 60-day model yields the highest MAE of 241. The 30-day model as well as the 90-day model achieve better scores around 190.

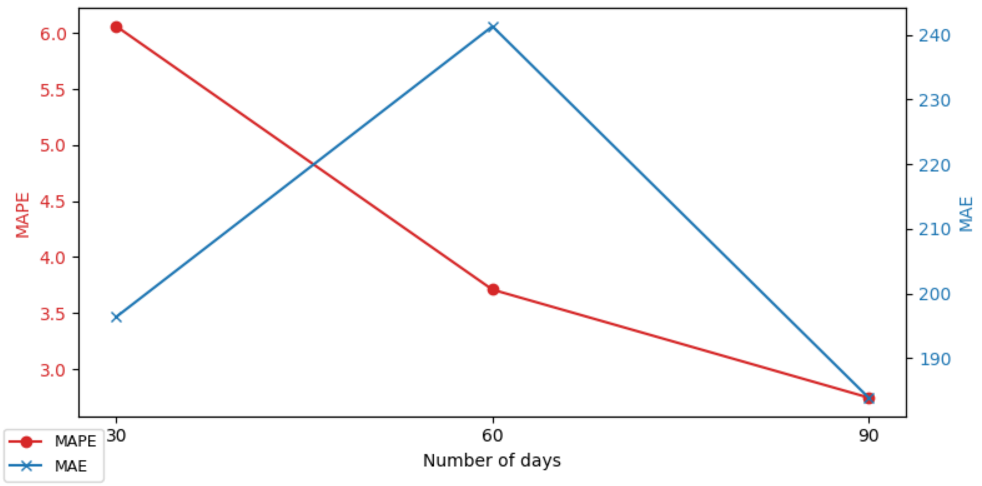


Figure 40: ARMAConv: MAPE and MAE performance for varying number of days in prediction

The depicted results are based on one test dataset with 100 samples.

Further, we want to investigate the error and analyze its characteristics. We split the data into training, validation and test dataset in a 80/10/10 split, meaning that the following plot contains information about 100 test samples.
Figure 41 illustrates the occurrence of each error for the 60-days-GNN example. The plots illustrate the distribution of the errors MAPE and MAE for 400 nodes. The two metrics follow different distributions. Most nodes have a very low MAPE and the higher the MAPE becomes on the x-axis the less occurrences can be counted. Generally, most nodes achieve MAPE below 2.5% but there are still multiple nodes with more than 7.5% with up to about 12.5%. This means that there is a high variance of MAPE depending on the node. If we consider MAE the distribution resembles a normal distribution which lightly right-skewed. Most nodes have an MAE about 150-350 and much smaller or higher error up to 500 are less common.

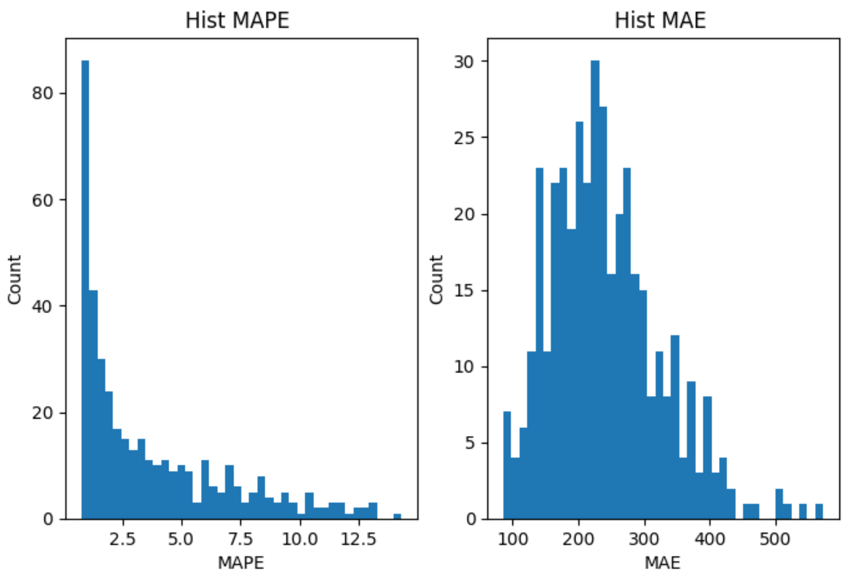


Figure 41: 60-days-GNN: Histogram for MAPE and MAE

The distribution of the error among compartments is illustrated in Figure 42. The data represents the average performance of 100 test samples for the 60-days ARMAConv model on dataset 2. The plot includes MAPE as well as MAE and illustrates the error per compartment for three out of 400 population-nodes. We choose Berlin as the biggest city with more than three million residents, Mettmann county with 471.8 k residents and Havelland county with only about 200k residents. The average population size of all 400 nodes is 197,993. This could be the reason why the Havelland node has the best performance regarding the MAPE with a 0.897% average test MAPE. Mettmann has an average MAPE of 6.028% and surprisingly high absolute error of 404 on average and more than 2200 in the Susceptible compartment. Regarding the compartments, the distribution pattern remains the same, regardless of the population. For MAPE, the Dead and IndectedCritical compartments yield the highest errors. For MAE, the compartments that contain the majority of individuals, namely Recovered and Susceptible reach the highest errors.

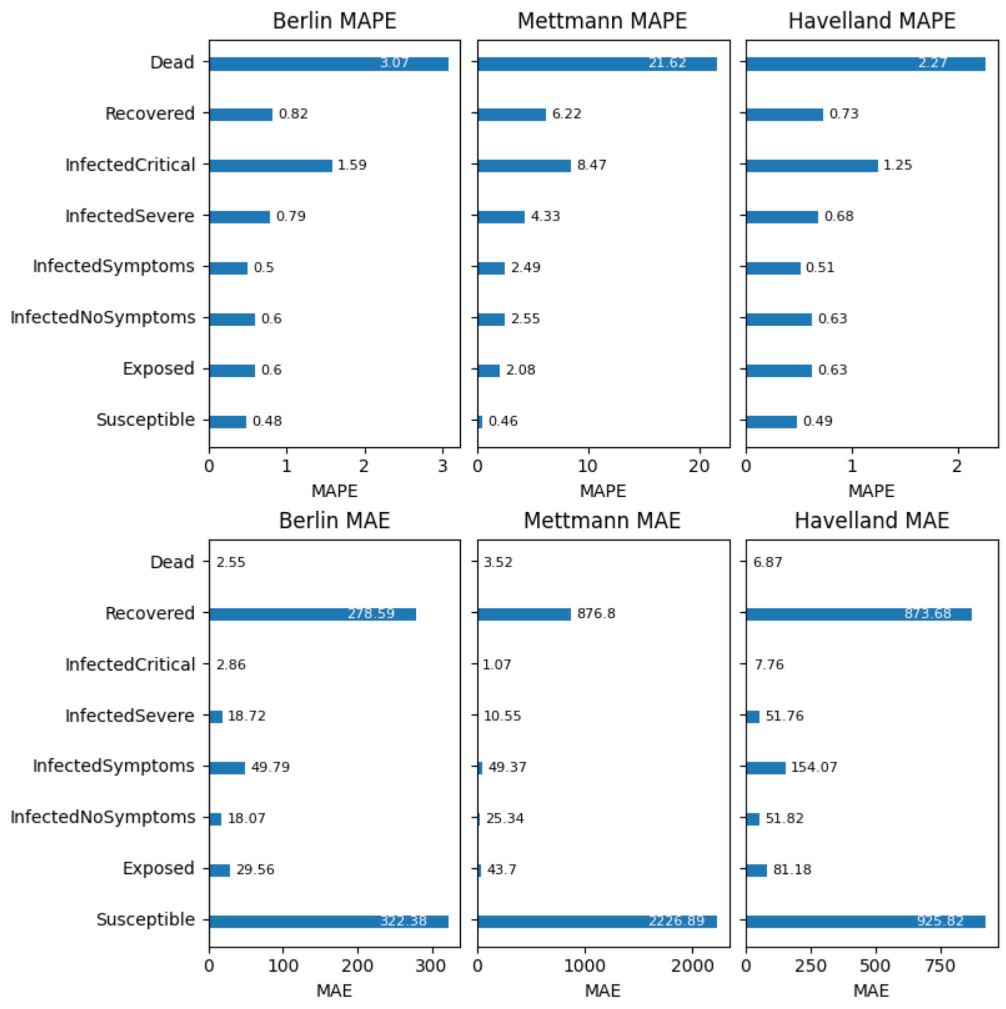


Figure 42: 60-days-GNN: MAPE and MAE per compartment

Figure 43 illustrates the relationship of the populations size and the MAE/MAPE for the test dataset. Since many counties have a population size between 100k and 250k individuals, the point cloud is very dense in that range. Figure 44 provides a more detailed view of this range. A relationship between population and MAE or MAPE is not clearly visible. A slight trend of larger populations leading to higher MAE could be visible.

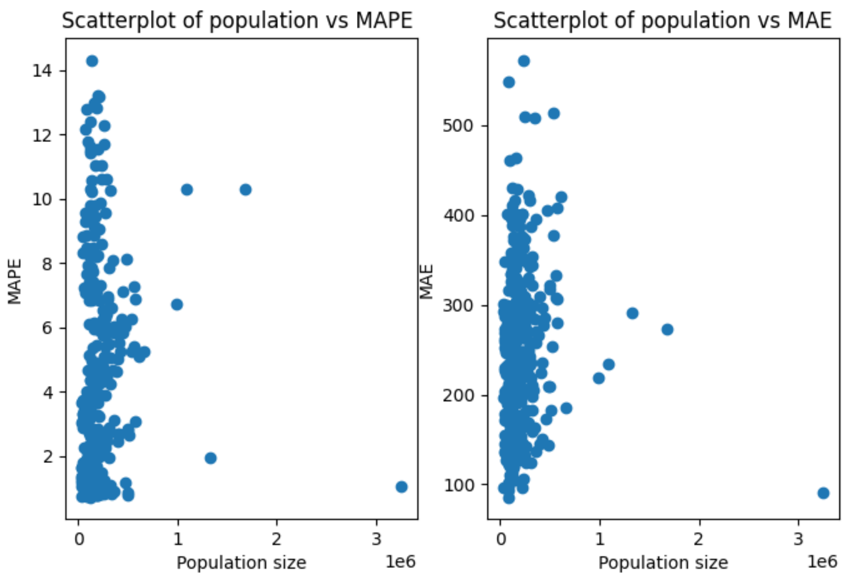


Figure 43: 60-days-GNN: Scatter plots for populations vs. MAE/MAPE

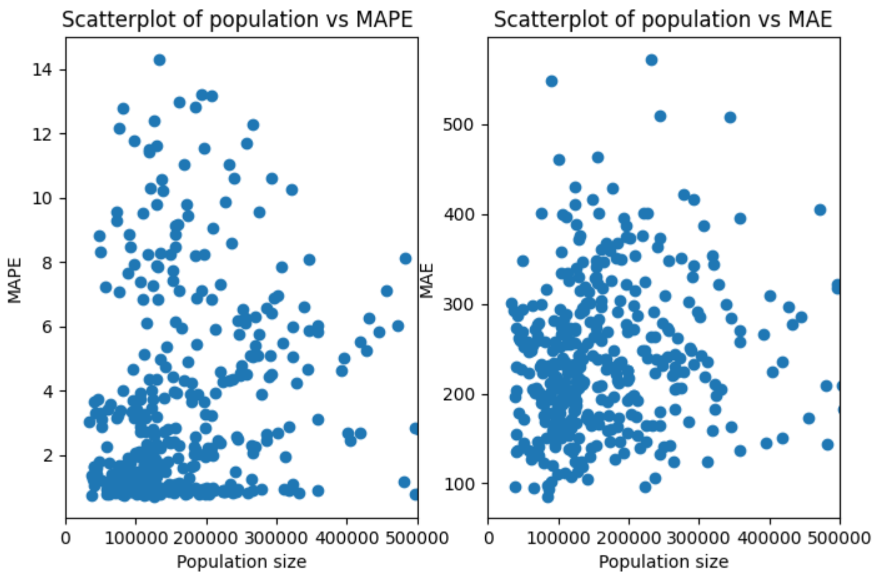


Figure 44: 60-days-GNN: Scatter plots for populations vs. MAE/MAPE for populations $<$ 500k

The ARMAConv layer we use in our GNN expects a binary adjacency matrix. As mentioned in Section 4.4, the model performance deteriorates if we define the adjacency

as a representation of the edge weights. Consequently, our model has no specific information on the travel and commuting activities that happen in the ODE-graph model. We want to study the relationship between the mobility activities that takes place in a node and the node specific MAPE of the surrogate GNN. For the sake of this analysis and the creation of a plot we want to summarize the commuting activities of each node into one representative number. For each node, we add all incoming and outgoing edge weights and we call this sum traffic. Additionally, we normalize the traffic by dividing it through the number of individuals in the population, in order to minimize the effect of population size and focus on the travelling and commuting activities. This way, we get the relative traffic. The relationship between the MAPE of a node and its absolute and relative traffic illustrated in Figure 45. Each data point corresponds to one node of the 60-days-GNN.
Both scatteplotst do not clearly indicate relationship between mobility and MAPE. However, we can derive some information from the Figure. We can see that most nodes have an aggregated traffic of less than 100k and a relative traffic between 20% and 40%. Busier nodes with more traffic have a multiple nodes with low MAPE and some nodes with higher MAPE, just as the nodes with less traffic.

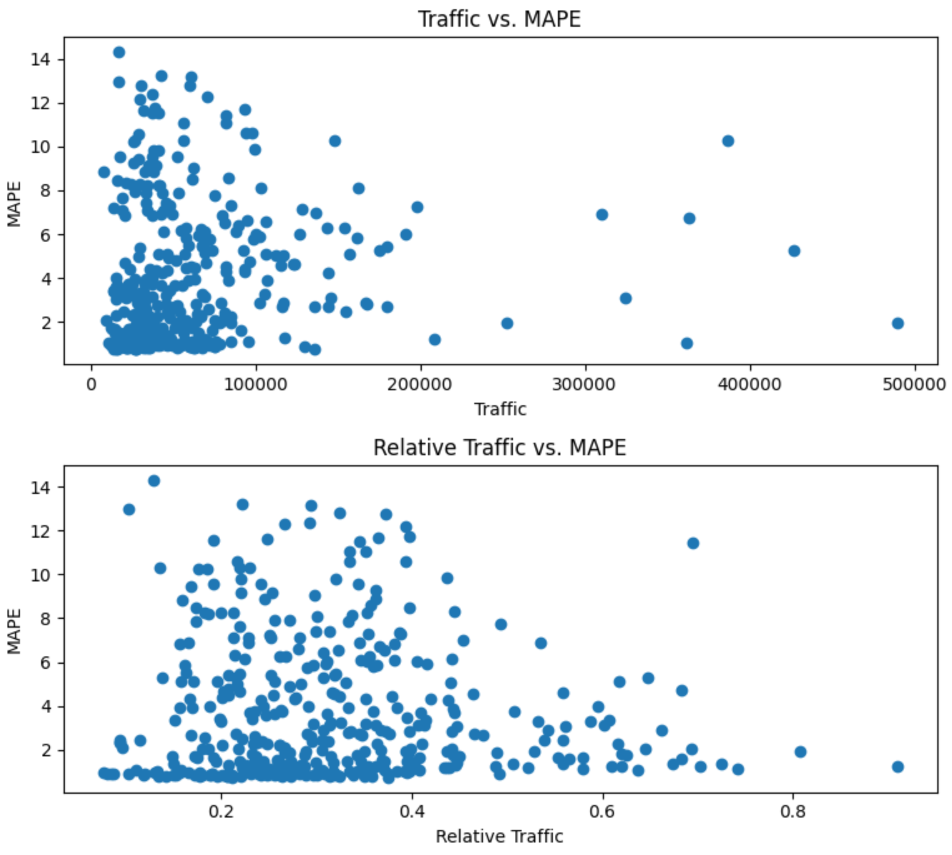


Figure 45: 60-days-GNN: influence of mobility on MAPE

GNNs with Multiple Changes in Contact Patterns

Now, we introduce dampings to our GNN. Likewise, as in the experiments for one population, the model input consists of five days of input and the model has to predict the following 100 days. We assume that the NPIs are enacted on a national level, so that all 400 counties experience the same contact restrictions.

Based on the input composition as depicted in Figure 19 we append damping day, damping factor and the modified and flattened contact matrix to the input. Since the input is implemented as the node features of the GNN, we append the information about the damping to every node's input. The information about the damping is the same for all counties, so we append the same values to each individual compartmental input data of the node.

We conduct experiments for one, two, three, four and five dampings. Analogously to the models for one population, the dampings are drawn with a minimum distance of seven days from each other. Figure 46 illustrates the results. The results represent the performance of the model on the more variable dataset 2 with a 80/10/10 split, meaning that the data in Figure 46 represents the mean MAPE from 100 test samples. The model with 1 damping performs has a MAPE of 2.73%. All other models have a MAPE at least twice as high.

Of course the performances are always dependent on the chosen dataset. A beneficial draw of samples can lead to deceivingly good performances and the other way round. If the datasets are large enough, positive as well as negative outliers can be balanced. As our test datasets are relatively small with only 100 samples, we wanted to conduct a cross validation.

The 5-fold cross validation gives us the opportunity to evaluate the models on a larger sample. These results are depicted in Figure 47. The results from the cross validation do not significantly differ from the results on the single test dataset. Most striking remains the good performance of the model with one damping, while all other models have between 6.49% and 7.14% test MAPE

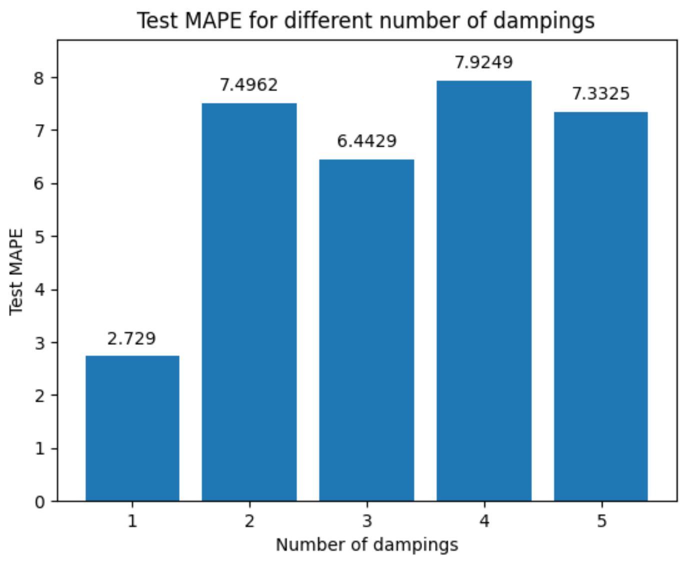


Figure 46: GNN: Test MAPE for varying number of dampings

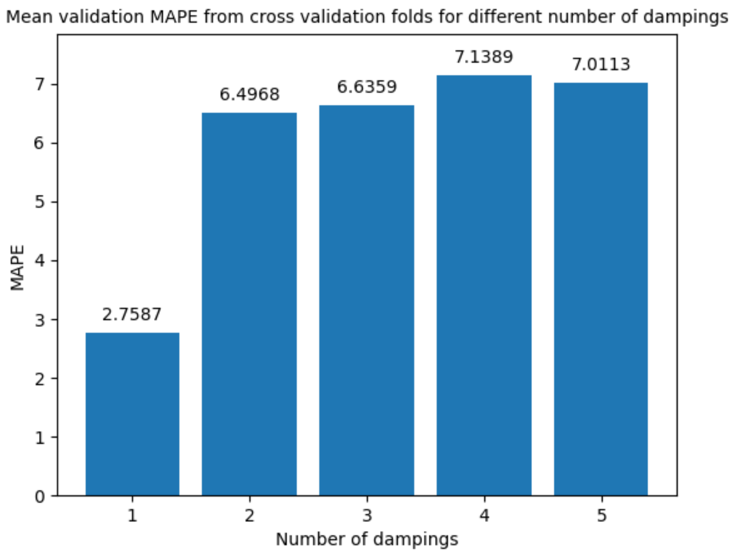


Figure 47: GNN: Average Validation MAPE of cross validation for varying number of dampings

As we can see in Figure 48 the variance between the folds is small, compared to the results from the grid search in Figure 23.

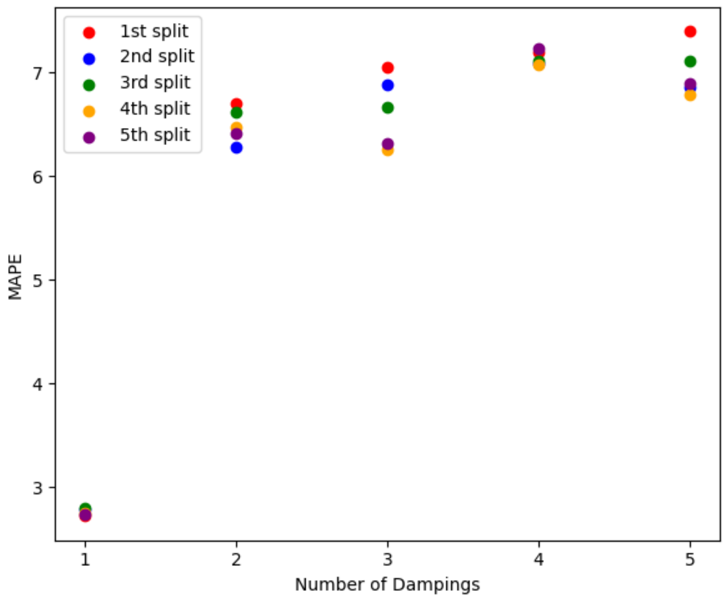


Figure 48: GNN: scatter plot for the validation MAPE of each cross validation fold for multiple dampings

5.3 Runtime Analysis

The main goal of our surrogate model is to replace the resource consuming calculations required to solve the differential equations of the ODE model. In the section before, we focused on creating a model with good performance measures regarding the prediction accuracy measured with the mean absolute percentage error. Now we want to test whether the prediction of test data can be executed faster than predicting the same task with the ODE model.
First, we will compare the performance for the models for one population. We consider the simplest models, excluding age-resolution (SimpleNN), models with age-resolutions (GroupsNN) as well as the models for various days and dampings.
One significant difference between the ODE model and the NNs is that the data generation process based on the ODE model produces one data sample at a time, because it is not vectorized for parallel computations yet. Meanwhile, the predict function for Keras models can processes a sequence of inputs.

The implementation of the numerical solver of the ODE model into python code is realized by a function named `run_secir_simulation()` and it generates one data sample at a time. In order to generate multiple samples with an ODE model we need to place the `run_secir_simulation()` into a for-loop. For that reason, we divide the experiment into two sub-experiments. In the first one we compare the time required to generate or predict one sample and in the second one for multiple data samples.
The difference of the two situations is shown in the code snippets below. The prediction with the NN is the same in both scenarios. We load the saved model with the best

weights resulting from a training process. Then we can feed multiple data samples from the test sample into the function.
For the data generation with the ODE-based model we only measure the time needed for the running the `run_secir_simulation()` function. All other functions, like the random selection of a population or the drawing of dampings are excluded from the time measurement.

Listing 1: Run ODE

```
# generating one sample
start = time.time()
data_run = run_secir_simulation(days)
end = time.time()

# generating mutliple samples
start = time.time()
for _ in range(0, num_runs):
    data_run = run_secir_simulation(days)
end = time.time()
```

Listing 2: Predict with NN

```
start = time.time()
pred = secirsimple_model.predict(test_inputs[:num_runs])
end = time.time()
```

Figure 49 illustrates the average time from 1000 runs that is needed to generate one data sample. We conduct three experiments. First, we run the experiment for the ODE model for one population without age groups versus our SimpleNNs. The number of days that have to be predicted varies between the values 30, 60, 90, 120 and 150. ODE as well as LSTM remain relatively constant for each of the prediction horizon. The ODE is six times faster than the LSTM in each case.
Next, we conduct the same experiment for the ODEs and NNs with age groups. There, the ODE becomes progressively slower, when more days have to be simulated. The LSTM on the other hand remains relatively constant. Nevertheless, the LSTM still needs approximately twice as much time as the ODE.
The models with dampings always predict 100 days and on the right side of Figure 49 we can see that the LSTM model is consistently is three times slower then the ODE-based model. Moreover it is noteworthy that the running time is not influenced by increasing number of dampings for both models.

The main advantage of the NN becomes visible when multiple test samples need to be analyzed. The `predict()` function of Keras can take an array with multiple test samples and make prediction for all of them in an efficient way. The ODE solver on the other hand has to handle one data sample of the test dataset after another. Figure 50 depicts the mean running times of 100 iterations of predicting 1000 samples. It is clearly

visible how the ODE becomes slower the more days have to be predicted while the LSTM experiences smaller fluctuations. There exists points where the LSTM performance is faster compared to the ODE for both, the SimpleNN as well as for the GroupsNN. For the SimpleNN this is at 90 prediction days and for the GroupsNN for 120 days.
Further, the ODE becomes significantly slower when contact reductions are implemented. For 1000 data samples, the ODE needs approximately 15 to 16 seconds. The LSTM solves this task in 0.2 seconds.

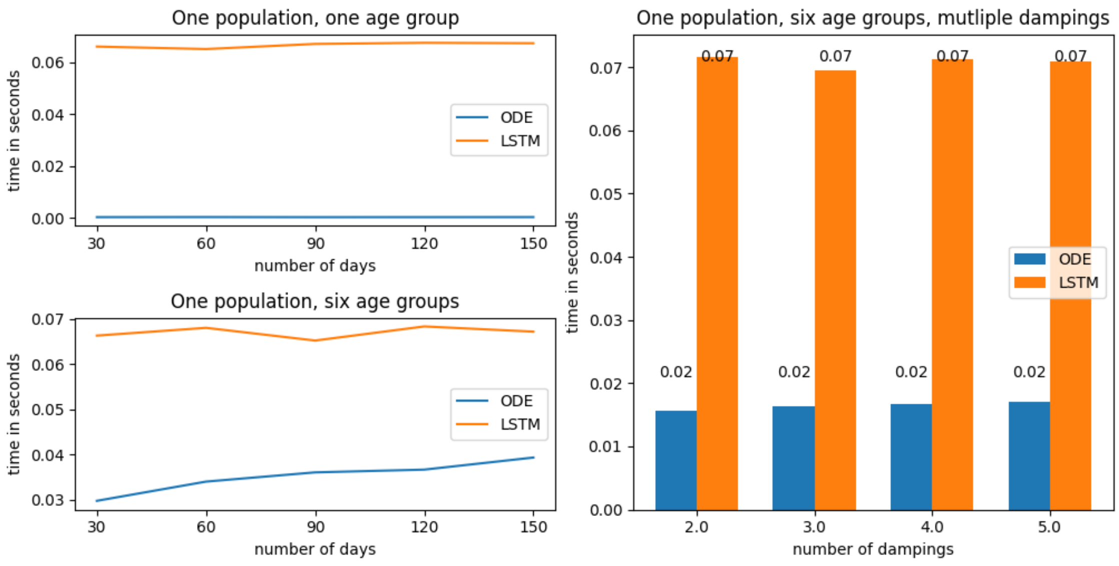


Figure 49: SimpleNN and GroupsNN: time analysis for prediction of one sample

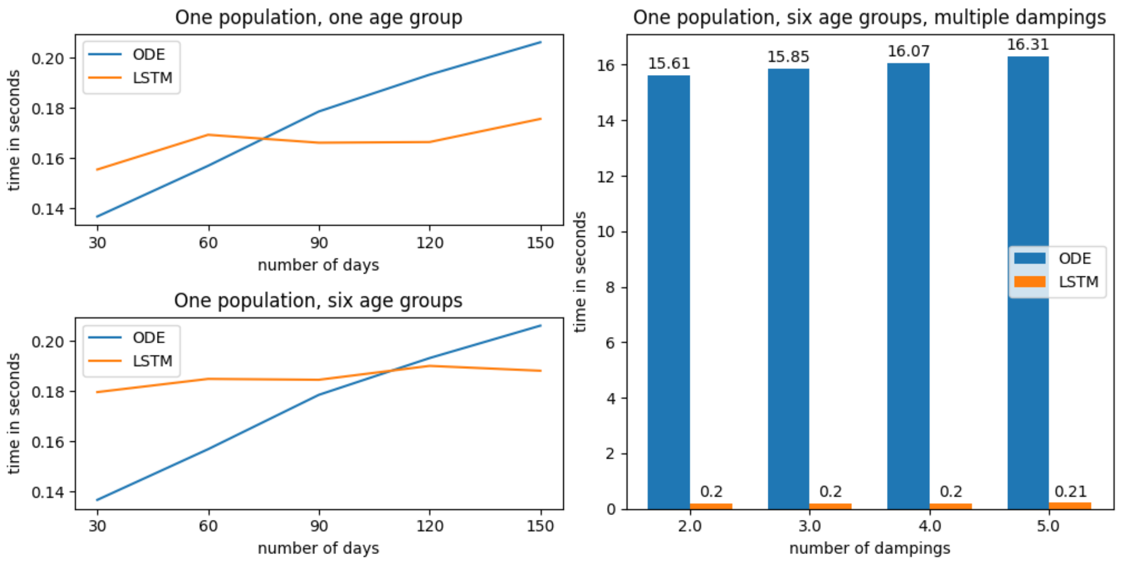


Figure 50: SimpleNN and GroupsNN: time analysis for prediction of multiple samples

We conduct the same experiments for our GNN surrogate models in comparison with the graph-ODE model. Figure 51 depicts the mean running time of 50 runs of predicting one data sample. We can clearly see that all models have higher running time compared to the models on one population. Especially the graph-ODE models require significantly more time. The number of days that have to be simulated has a strong influence on the running time for the graph-ODE model. While a run for 30 days requires 16 seconds, a run for 90 days takes 31 seconds. All models with dampings are designed for 100 days which is why the graph-ODE needs more than 30 seconds for all simulations that include dampings. The number of dampings does not have an impact on the running time.

The GNN surrogate models is able to make a prediction for every tested scenario in approximately 2.5 seconds. The number of days or the number of dampings included in the data has no significant influence on the running time.

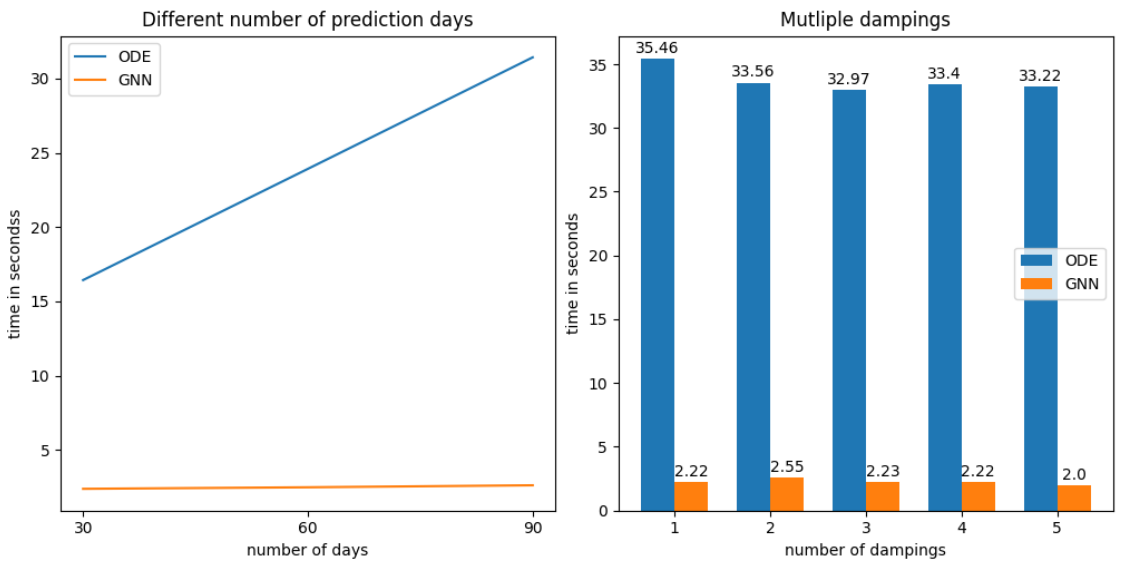


Figure 51: GNN: time analysis for prediction of one single sample
The depicted values of the mean time of 50 iterations with only one samples that has to be predicted.

In Figure 52 we can see similar results as for the simpler models. While the graph-ODE model increased linearly with the number of samples that have to be generated, the performance of the NN remains constant around 2.5 seconds. The time the ODE-model needs for 10 samples is 10 times higher than the time it need for 1 sample. Especially in this scenario, our GNN can contribute as a surrogate model with significant runtime improvement.

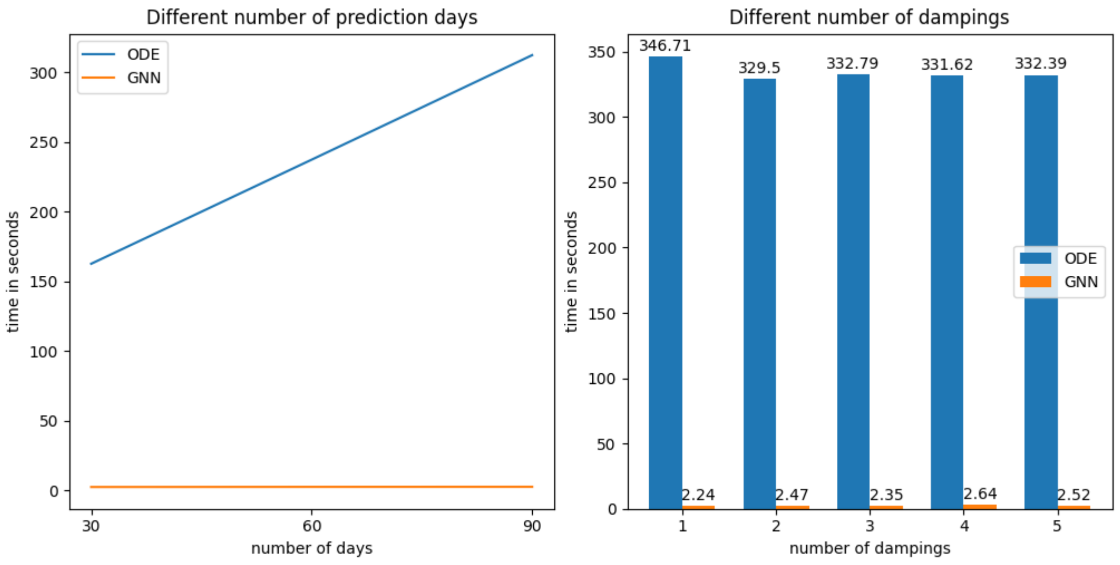


Figure 52: GNN: time analysis for prediction of multiple samples
The depicted values of the mean time of five iterations with 10 samples that have to be predicted.

6 Discussion, Limitations and Outlook

In this section, we will discuss previous results regarding specific notable features and characteristics as well as limitations and open questions. Since we have conducted multiple experiments, we will structure the discussion by topic.

Data: Discussion

As described before, the input for our model is generated by allocating a random percentage of individuals of the total population to each compartment. The randomness is controlled by a specific range. If the range is small, the data samples are similar to each other. If the range is large, the dataset contains more variety. If the data has more variety, more different pandemic situations are represented in the dataset. For our experiments, we wanted to create a dataset where each sample starts in the earlier stage of the pandemic. This way, the models have to predict the whole course of the pandemic and should aim to forecast the peak of each compartment.

We were able to show for all surrogate models that the models that were selected based on the experiments with dataset 1 were able to make predictions within acceptable error ranges for the more variable dataset 2. Table 3 summarizes the results and a significant difference between the models for one populations and the GNN is visible. The SimpleNN has a lower MAPE on dataset 2 and GroupsNN has almost the same MAPE as on dataset 1. The GNN on the other hand has a MAPE almost twice as high on dataset 2. This difference could occur due to the higher complexity of the GNN.

	dataset 1	**dataset 2**
SimpleNN	0.13%	0.07%
GroupsNN	0.4%	0.41%
GNN	3.46%	6.18%

Table 3: Comparison of results based on dataset
All results are based on the 30-day models.

Data: Limitations and Outlook

The dataset size was chosen randomly for our experiments. We reduced the size to 1000 samples when we trained the GNNs, with the aim to reduce training time. Moreover, the data generation is computationally expensive. For machine learning issues, a dataset of 1000 samples is a relatively small dataset size. It could be interesting to test how the model performs with other dataset sizes.
Also the number of five input days was selected randomly. We wanted to provide the models with enough data to capture some time-relative dependencies while not training surrogate models that need vast amounts of input data to provide good predictions. Further experiment with other input ranges could be interesting.
When generating data, we determined bounds for the random parameter assignment.

Even though the distribution among compartments is different for every run, they all rather represent the beginning of a pandemic, with many individuals in the Susceptible compartment. We can assume that if these ranges are extended, larger datasets will be required so that every scenario is represented sufficiently often for the model to learn.

SimpleNN: Discussion

The results for our simplest experiment with only one population and no age-resolution showed that small model architectures with few or no hidden layers and rather little number of units per layer perform best for this simple task for all three model types. Nonetheless, for MLP and CNN the best scores are achieved by multiple architectures. The best MLP score of 0.2% is obtained by one of the smallest models with zero hidden layers and 62 units per layer as well as by the largest tested model with 4 hidden layers and 1024 nodes. Usually, we would assume that larger models over-fit, but in this case, they can reach same MAPE as smaller models. The LSTM on the other hand only reaches its best performance of 0.13% with models that are composed without any hidden layer. Models with four hidden layers remain over 0.2%.

One of our main findings is that the LSTM is the best suitable model for the task of predicting the COVID-19 dynamic for the eight compartments. This approves the LSTM's capabilities to capture temporal dependencies.

SimpleNN: Limitations and Outlook

The resulting models are not the result of a complete analysis and complete hyper-parameter tuning. There are several factors such as batch size or learning rate which can improve training and prediction performance. Moreover, layer specific parameters could be varied. Many layers offer multiple binary and non-binary parameters that can be altered in future experiments. Our models utilize the default settings mostly.
Also, the model architectures we propose are kept as minimal as possible. Nevertheless, adding more layers could make the performance even more efficient. Pooling and Drop Out layers could minimize computational complexity. They are not necessary, since our models do not over-fit, but they have the potential to speed up training. Even if this means a trade-off with accuracy, it could make sense, depending on the task and application.

GroupsNN: Discussion

For the models with age-resolution, we decided to create additional plots to the heatmap with the aim to depict the influence of the number of layers and units more clearly. Figure 12 illustrates similar findings as we made for the models without age-resolution. Firstly, the LSTM becomes worse the more layers are added. Secondly, MLP and CNN behave similarly with the best results for the smallest models but the more complex architectures do not diverge substantially. Figure 14 consolidates the point that the number of units influences the performance to a certain point. While all models profit

significantly from an increase from 32 to 62 nodes per layer, the differences in performance are minimal for further augmentations.
To summarize, the influence of the amount of hidden layers and the amount of units per layers on the performance depends on the model type but is generally not clearly recognizable for the models for one population.

Moreover, likewise as the SimpleNN, the LSTM is the best model, positioning itself as the most suitable surrogate model for the ODE-based model with age groups.

GNNs: Discussion

The results for the grid search of the GNNs are very interesting as they show significant differences between the layer types applied. GCNConv and GATConv remain around 33% test MAPE for almost all model architectures. The GATConv model has an exception for the model with one layer and 32 channels which achieves a MAPE score of 20.45%. It could be assumed that the 33% MAPE are the score a model reaches when it is not able to substantially learn from the data. The score is reached after only five epochs of training and it does not improve further.
As depicted in Figure 41 the variance in MAPE is relatively high between the counties. We could not identify why some nodes yield high errors and which perform well. The size or traffic of the nodes did not provide enlightment to that question.

Also the question, why the MAPE declines with a rising number of predicted days remains unanswered.

Generally, is very interesting that the resulting GNN model is able to extract temporal information from a flat array without any context. We do not specify which number belongs to which moment in time. Nevertheless, it is able to process the information and create accurate results. Moreover, different types of information e.g. compartment information, damping day or contact matrices can be processed by the model.
The successful implementation of a direct model that predicts all 30, 60 or 90 days simultaneously is a major result. Comparable models for the same type of spatio-temporal task are mostly some type of graph sequence model where each graph represents the state at time t and the predictions are only made for $t + 1$ [52], [53], [84].

GNNs: Limitations and Outlook

The grid search for the identification of the best GNN model is restricted by multiple limitations. There are more than the presented four options that could be analyzed. Also, a more dimensional grid search could detect special combinations of parameters that would yield good results. We selected a small number of layer types and reduced the number of parameters in the grid search in order to save computational resources. Of course this results in a restricted analysis.

As we have seen for the ARMAConv layer, the data split had an immense influence on the models performance. As we have seen in Figure 23 there exist data split where some of the ARMAConv models resulted in MAPE around 33%, just as all GATConv models.

It could be possible that the data split we chose was favourable for the ARMAConv model and that other models would have performed better on the other splits. To shed light onto this question, a comprehensive cross validation needs to be undertaken.

As mentioned in Section 4.1, surrogate models are often implemented as a physics informed NN due to the fact that surrogate models have the task to approximate a given function and therefore the information about the given function can help the PINN to solve the task. Our surrogate model is completely data driven, as it only obtains input and labels for training. In the future, we could try to adjust the model by providing more information. One example for this could be the implementation of an equation for population size maintenance into the loss function. Right now, our model has no constraint regarding the resulting population size of the prediction. This additional information could help the model to achieve better results.
As described in Section 3.2, many spatio-temporal GNNs use some kind of graph sequence model to represent the temporal characteristic of the data [52], [53], [84]. Our model differs from this approach by implementing the temporal aspect in the node features. It could be interesting to redesign our model into a graph sequence model and compare the performances. It would be interesting to analyze how the error changes over time for long time predictions compared to our GNN. Further, the analysis of prediction time should be a focus when comparing these two model types, as it would reveal differences between our direct prediction approach and the iterative approach of a sequence model.
Unfortunately, the ARMAConv GNN cannot benefit from including data about the travelling and commuting activities in the form of a non-binary adjacency matrix. However, there are models that can assign those values directly to the edges and process the data in the training process. The input for those models consists of node features, adjacency matrix, and edge weights. In Spektral, layers which include edge weights in their calculations exist, like the XENetConv layer from Spektral. The computational resources that were accessible for our work were not sufficient to employ these layers to our deeply connected 400 node graph. If more resources could be gathered for future analysis, a further comparison with these layers could be interesting. Alternatively, we could implement these layers on a smaller graph. We could reduce complexity by representing the 16 states instead of 400 counties of Germany or by focusing on the counties of one state only. We would expect a better performance compared to the presented ARMAConv GNN because more information about the underlying data is provided.

ARMAConv: Discussion

Intuitively it could be assumed that ARMAConv performs best, because the auto regressive moving average model is used for time series analysis and could fit our temporal data. But, as explained before, the ARMA-mechanisms are implemented into the filter and the temporal aspect refers to the consecutive node representations in the training process. The authors that built the ARMA-GNN [45] compare their model to some of the models we used as well. For example, they state that the GATConv model [42] only considers first order neighboring nodes per layer, which is why they need to stack multiple layer onto each other to reach farther neighborhoods. The GCN by Kipf and Welling [33] also needs to stack layer to reach higher order neighborhoods. Their presented GCN model is based on polynomial filters which transforms the input (node features) by linearly combining the inputs from the k-hop neighborhood to each vertex. The k-hop neighborhood describes all nodes that are at most k edges away from the given node. The larger the neighborhoods is that should be reached, the higher becomes the degree of the polynomial. The ARMAConv on the other hand can reach neighborhoods of higher-order than polynomial models with the same number of parameters. They achieve this by integrating the concept of auto-regressive models to the process of propagating the node features multiple times though he network.

Bianchi et al. [45] claim that their ARMAConv model resolves limitations of polynomial filters like finite impulse response or limited modelling capabilities. The strengths of the ARMAConv model are, being more robust to noise and better capturing global graph data. This could be a great advantage when the spread of a virus is modeled, since outbreaks in higher-order neighborhoods could be predicted.

ARMAConv: Limitations and Outlook

In order to save computational resources, we decided to conduct the subsequent experiments on variable prediction horizons and variable number of dampings only with the best performing ARMAConv layer. Certainly, it could be interesting to execute the experiments with multiple dampings and different prediction horizons with all remaining model types.

Dampings: Discussion

For the SimpleNN and the GroupsNN the number of days as well as the number of dampings has a clearly visible influence on the MAPE performance. The more changes in contact reduction take place, the higher the test MAPE becomes. We showed that this steady increase of error is not due to the fact that the NNs cannot handle the input we provide about the damping. Adding the damping day and damped contact matrix to the node features improves the test MAPE significantly.
The relationships as just described are not perceptible for the GNN. For the number of days, the relationship is reversed, with more days leading to lower MAPE. We could not identify the reasons for these results in this work. A deeper analysis of the losses should be undertake in the future.

Concerning the dampings, only the difference between one damping and more than one dampings is detectable for the GNNs. While the model with one damping included reaches a MAPE of 2.76%, all models with more dampings have MAPE scores at least twice as high. The reason for this sharp increase is unclear and should be studied.

Dampings: Limitations and Outlook

As described before, the task becomes more difficult with increasing number of contact changes. We analyzed the performance of the same model architecture on those different levels of difficulty. When the goal is to create a best performing model, an adaption of model architecture could be tested in future experiments.

The contact reductions in our model are applied on all age groups. Since we have an age-resolved model and an age-resolved contact matrix, we could implement age specific contact reductions. This would be implemented by adjusting the values in the contact matrix depending on the age group, instead of applying the same damping factor to each entry of the contact matrix. The input dimensions will remain the same. Age-specific contact reduction could be of interest, because NPIs influence the age groups differently. School closures for example influence younger individuals more than individuals of the older age groups.

Another dimension we could add to the dampings is the spatial dimension. Local contact reductions could be applied to each node individually, by changing the contact matrices, which belong to the node input individually.

Moreover, it could be interesting to implement mobility restrictions. However, the implementation of this feature is not as intuitive as the age- or node-specific contact reductions. In the ARMAConv model, the information about mobility between populations is represented by the adjacency matrix. The adjacency matrix is one of the two elements in the input tuple that is fed into the first convolutional layer of the GNN.

A limitation of our presented surrogate models is that we summarize the contact matrices from different places like school or work into one. MEmilio considers multiple NPIs and differentiates between the places in which the contact reductions take place. This way, specific NPIs such as school closings or home office can be modeled. This level of detail cannot be implemented into our model without major changes.

MEmilio introduces the concept of dynamic dampings. Dynamic dampings automatically apply when specific infection thresholds are exceeded. This means that, e.g., if the number of infected people reaches a specific threshold, contact reductions measures will be enacted automatically. For our model, the number and time if dampings has to be specified beforehand.

Discussion: Runtime Analysis and Model Application

The results of the time analysis are significantly different for SimpleNN/GroupsNN and the GNNs. First, we measured the time each model needed to create a single simulation/prediction. After that, we analyzed the running time for multiple predictions.

In the experiments with a single prediction the ODE-based model is very fast with a simulation execution in a friction of a second. The LSTM, which is the best model architecture for both SimpleNN and GroupsNN, is also very fast with a prediction in 0.07 seconds, but remains at least three times slower as the ODE-based model. The GNN, on the other hand, is significantly faster than the graph-ODE model in every analyzed scenario.

In the experiments with multiple predictions all of our surrogate models yield great potential for fast approximation of the original model. While the ODE-based and graph-ODE model need more time the more simulations they have to execute, the NNs and GNNs remain constant regarding the running time.

Even though the GNN has much higher MAPE error that the other surrogate NNs, the 15-fold advantage in prediction time yields potential for an application for fast evaluation of a specific scenario.
We assume that some mentioned improvements due to further analysis could improve the model performance. Potentially, these models will require more time in training, but perhaps the prediction time could remain fast.

One possible application of the surrogate models could be a web application for stakeholders that can assess different NPI-strategies within seconds.

7 Conclusion

In this work we have described the potential of NNs and GNNs to serve as surrogate models in computational epidemiology.

We developed NNs and GNNs as surrogate models for the ODE- and graph-ODE based models from MEmilio. The ODE-based model from MEmilio we used in this work is an age-resolved epidemiological compartment model, designed for simulations of the COVID-19 dynamics. The graph-ODE model includes realistic travelling and commuting activities between all 400 counties of Germany, which are interconnected in a graph-structure. Both models were used for data generation of training and test data for the respective data driven surrogate models.

Regarding the surrogates for the ODE-based models we conducted a grid search comparing different architectures for the model types MLP, CNN and LSTM, where the LSTM emerged as the best performing model type for all tasks. The LSTMs were able to approximate the underlying ODE-model with great precision, resulting in MAPE scores below 1% for almost all of the examined tasks. The tasks included different prediction time spans of 30, 60, 90, 120 and 150 days as well as up to five changes of contact patterns which can result from NPIs enacted by the government.

Analogously to the NNs, we conducted a grid search to identify the best performing GNN layer for the approximation of the graph-ODE model. Among the four layers APPNPConv, GATConv, GCNConv and ARMAConv, the ARMAConv layer came out favorably. The input for the GNN is constituted of two parts, firstly the information about the distribution of individuals along the compartments, implemented as node features. The second part of the input is the adjacency matrix which provides information about the graph connectivity. The architecture of the GNN is based on the structure of the graph-ODE. We implemented a direct prediction approach, meaning that the surrogate models predict all days in one step. Likewise, the input information is provided simultaneously in one array.
The ARMAConv GNN offers an approximation of the graph-ODE while reducing the time from up to 35 seconds for the execution of the expert model to only two seconds of the execution of the surrogate model. The time advantage of the GNN enlarges with the number of predictions, as the computing time does not increase with rising number of data samples in the test dataset, in contrast to the linear growth in computing time of the graph-ODE model. The improvement in prediction time comes with a trade off between approximately 2.7-7.1% MAPE, depending on the task.

In conclusion, we reached the goal to replace the ODE-based model and the graph-ODE model with NNs and a GNN that can make single or multiple predictions very fast with only a little trade off in accuracy. Moreover, we selected the ARMAConv model architecture in a way that is easy to extend for future experiments which could include space or age resolved changes in contact patterns, by only changing the input data. This yields great potential for fast approximation of more complex simulations. One possible utilization of our surrogate models could be the implementation as a public web-application that offers user friendly and time efficient simulations for decision-makers.

8 Appendix

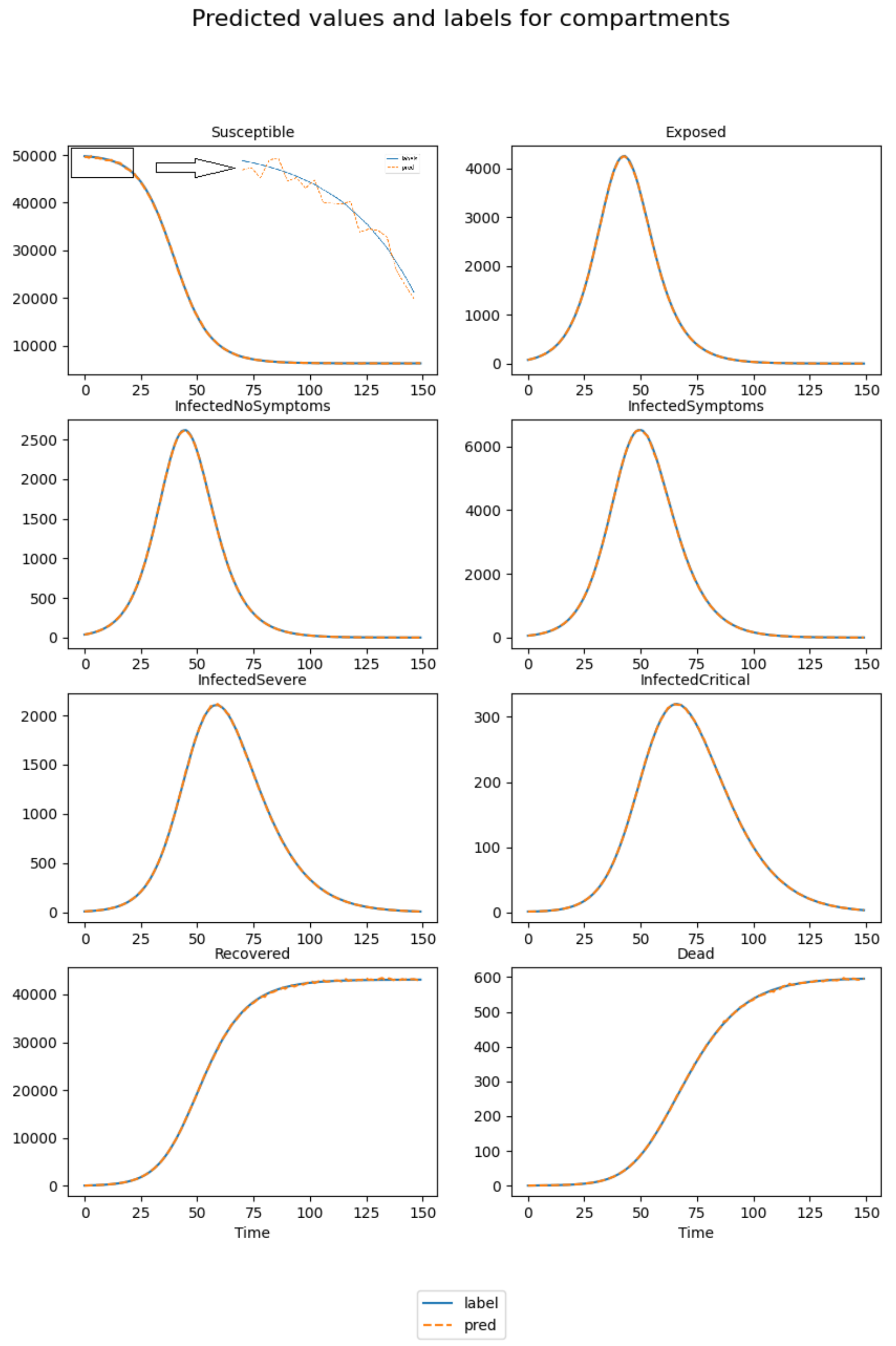


Figure 53: 150-days-SimpleNN: Number of individuals per compartment - dataset 1

References

[1] W. H. O. (WHO), *Coronavirus disease (COVID-19)*, en. [Online]. Available: `https://www.who.int/news-room/fact-sheets/detail/coronavirus-disease-(covid-19)` (visited on 05/16/2024).

[2] D. Koh, "COVID-19 lockdowns throughout the world," *Occupational Medicine*, vol. 70, no. 5, p. 322, Jul. 2020, ISSN: 0962-7480. DOI: `10.1093/occmed/kqaa073`. [Online]. Available: `https://doi.org/10.1093/occmed/kqaa073` (visited on 04/19/2024).

[3] M. J. Kühn, D. Abele, D. Kerkmann, *et al.*, *MEmilio v1.0.0 - A high performance Modular EpideMIcs simuLatIOn software*, Dec. 2023. [Online]. Available: `https://elib.dlr.de/201660/`.

[4] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019, ISSN: 0021-9991. DOI: `10.1016/j.jcp.2018.10.045`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0021999118307125` (visited on 04/28/2024).

[5] L. Sun, H. Gao, S. Pan, and J.-X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, p. 112 732, Apr. 2020, ISSN: 0045-7825. DOI: `10.1016/j.cma.2019.112732`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S004578251930622X` (visited on 04/28/2024).

[6] M. J. Kühn, D. Abele, T. Mitra, *et al.*, "Assessment of effective mitigation and prediction of the spread of SARS-CoV-2 in Germany using demographic information and spatial resolution," *Mathematical Biosciences*, p. 108 648, 2021, ISSN: 0025-5564. DOI: `https://doi.org/10.1016/j.mbs.2021.108648`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0025556421000845`.

[7] W. O. Kermack, A. G. McKendrick, and G. T. Walker, "A contribution to the mathematical theory of epidemics," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 115, no. 772, pp. 700–721, Aug. 1927. DOI: `10.1098/rspa.1927.0118`.

[8] M. Y. Li, *An Introduction to Mathematical Modeling of Infectious Diseases*, en. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-72121-7 978-3-319-72122-4. DOI: `10.1007/978-3-319-72122-4`. [Online]. Available: `http://link.springer.com/10.1007/978-3-319-72122-4` (visited on 04/28/2024).

[9] M. Osman, I. Kwasi Adu, and C. Yang, "A Simple SEIR Mathematical Model of Malaria Transmission," *Asian Research Journal of Mathematics*, vol. 7, pp. 1–22, Jan. 2017. DOI: `10.9734/ARJOM/2017/37471`.

[10] P. E. Lekone and B. F. Finkenstädt, "Statistical inference in a stochastic epidemic SEIR model with control intervention: Ebola as a case study," eng, *Biometrics*, vol. 62, no. 4, pp. 1170–1177, Dec. 2006, ISSN: 0006-341X. DOI: 10.1111/j.1541-0420.2006.00609.x.

[11] X. Tan, L. Yuan, J. Zhou, Y. Zheng, and F. Yang, "Modeling the initial transmission dynamics of influenza A H1N1 in Guangdong Province, China," eng, *International journal of infectious diseases: IJID: official publication of the International Society for Infectious Diseases*, vol. 17, no. 7, e479–484, Jul. 2013, ISSN: 1878-3511. DOI: 10.1016/j.ijid.2012.11.018.

[12] W. H. Organization, "WHO-convened global study of origins of SARS-CoV-2: China part," en, World Health Organization, Report, Mar. 2021, Journal Abbreviation: Joint WHO-China Study 14 January-10 February 2021: joint report. [Online]. Available: https://apo.org.au/node/311637 (visited on 04/19/2024).

[13] B. Tang, X. Wang, Q. Li, *et al.*, "Estimation of the Transmission Risk of the 2019-nCoV and Its Implication for Public Health Interventions," *Journal of Clinical Medicine*, vol. 9, no. 2, p. 462, Feb. 2020, ISSN: 2077-0383. DOI: 10.3390/jcm9020462. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7074281/ (visited on 04/28/2024).

[14] F. Ndaïrou, I. Area, J. J. Nieto, and D. F. M. Torres, "Mathematical modeling of COVID-19 transmission dynamics with a case study of Wuhan," *Chaos, Solitons & Fractals*, vol. 135, p. 109 846, Jun. 2020, ISSN: 0960-0779. DOI: 10.1016/j.chaos.2020.109846. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960077920302460 (visited on 04/28/2024).

[15] Z.-Y. Zhao, Y.-Z. Zhu, J.-W. Xu, *et al.*, "A five-compartment model of age-specific transmissibility of SARS-CoV-2," en, *Infectious Diseases of Poverty*, vol. 9, no. 1, p. 117, Dec. 2020, ISSN: 2049-9957. DOI: 10.1186/s40249-020-00735-x. [Online]. Available: https://idpjournal.biomedcentral.com/articles/10.1186/s40249-020-00735-x (visited on 01/16/2023).

[16] D. Acemoglu, V. Chernozhukov, I. Werning, and M. D. Whinston, "Optimal Targeted Lockdowns in a Multigroup SIR Model," en, *American Economic Review: Insights*, vol. 3, no. 4, pp. 487–502, Dec. 2021, ISSN: 2640-2068. DOI: 10.1257/aeri.20200590. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/aeri.20200590 (visited on 04/28/2024).

[17] F. Balabdaoui and D. Mohr, "Age-stratified discrete compartment model of the COVID-19 epidemic with application to Switzerland," eng, *Scientific Reports*, vol. 10, no. 1, p. 21 306, Dec. 2020, ISSN: 2045-2322. DOI: 10.1038/s41598-020-77420-4.

[18] C. M. Batistela, D. P. F. Correa, Á. M. Bueno, and J. R. C. Piqueira, "SIRSi compartmental model for COVID-19 pandemic with immunity loss," *Chaos, Solitons & Fractals*, vol. 142, p. 110 388, Jan. 2021, ISSN: 0960-0779. DOI: 10.1016/

j.chaos.2020.110388. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960077920307827 (visited on 04/28/2024).

[19] M. V. Barbarossa, J. Fuhrmann, J. H. Meinke, *et al.*, "Modeling the spread of COVID-19 in Germany: Early assessment and possible scenarios," en, *PLOS ONE*, vol. 15, no. 9, L. Pujo-Menjouet, Ed., e0238559, Sep. 2020, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0238559. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0238559 (visited on 01/16/2023).

[20] M. Dashtbali and M. Mirzaie, "A compartmental model that predicts the effect of social distancing and vaccination on controlling COVID-19," en, *Scientific Reports*, vol. 11, no. 1, p. 8191, Apr. 2021, Publisher: Nature Publishing Group, ISSN: 2045-2322. DOI: 10.1038/s41598-021-86873-0. [Online]. Available: https://www.nature.com/articles/s41598-021-86873-0 (visited on 04/28/2024).

[21] S. Walczak, "Artificial Neural Networks," *Encyclopedia of Physical Science and Technology*, Jan. 2003. [Online]. Available: https://www.academia.edu/15726358/Artificial_Neural_Networks (visited on 05/18/2024).

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[23] S. Ren, G. Chen, T. Li, Q. Chen, and S. Li, "A Deep Learning-Based Computational Algorithm for Identifying Damage Load Condition: An Artificial Intelligence Inverse Problem Solution for Failure Analysis," *Computer Modeling in Engineering & Sciences*, vol. 117, pp. 287–307, Dec. 2018. DOI: 10.31614/cmes.2018.04697.

[24] A. Azawii, S. Al-Janabi, and B. Al-Khateeb, "Survey on Intrusion Detection Systems based on Deep Learning," *Periodicals of Engineering and Natural Sciences (PEN)*, vol. 7, pp. 1074–1095, Sep. 2019. DOI: 10.21533/pen.v7i3.635.

[25] P. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990, Conference Name: Proceedings of the IEEE, ISSN: 1558-2256. DOI: 10.1109/5.58337.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, Conference Name: Proceedings of the IEEE, ISSN: 1558-2256. DOI: 10.1109/5.726791. [Online]. Available: https://ieeexplore.ieee.org/document/726791 (visited on 05/14/2024).

[27] C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, en. Cham: Springer International Publishing, 2024, ISBN: 978-3-031-45467-7 978-3-031-45468-4. DOI: 10.1007/978-3-031-45468-4. [Online]. Available: https://link.springer.com/10.1007/978-3-031-45468-4 (visited on 05/14/2024).

[28] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, Dec. 2022, Conference Name: IEEE Transactions on Neural Networks and Learning Systems, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3084827.

[29] A. Borovykh, S. Bohte, and C. W. Oosterlee, *Conditional Time Series Forecasting with Convolutional Neural Networks*, en, arXiv:1703.04691 [stat], Sep. 2018. [Online]. Available: `http://arxiv.org/abs/1703.04691` (visited on 08/23/2023).

[30] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, Conference Name: Neural Computation, ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`.

[31] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, Mar. 2020, ISSN: 0167-2789. DOI: `10.1016/j.physd.2019.132306`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167278919305974` (visited on 04/28/2024).

[32] X. H. Le, H. Ho, G. Lee, and S. Jung, "Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting," *Water*, vol. 11, p. 1387, Jul. 2019. DOI: `10.3390/w11071387`.

[33] T. N. Kipf and M. Welling, *Semi-Supervised Classification with Graph Convolutional Networks*, arXiv:1609.02907 [cs, stat], Feb. 2017. DOI: `10.48550/arXiv.1609.02907`. [Online]. Available: `http://arxiv.org/abs/1609.02907` (visited on 04/28/2024).

[34] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical Graph Representation Learning with Differentiable Pooling," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. [Online]. Available: `https://proceedings.neurips.cc/paper/2018/hash/e77dbaf6759253c7c6d0efc5690369c7-Abstract.html` (visited on 04/28/2024).

[35] M. Zhang and Y. Chen, "Link Prediction Based on Graph Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2018/hash/53f0d7c537d99b3824f0f99d62ea2428-Abstract.html` (visited on 04/28/2024).

[36] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "MGAE: Marginalized Graph Autoencoder for Graph Clustering," en, in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, Singapore Singapore: ACM, Nov. 2017, pp. 889–898, ISBN: 978-1-4503-4918-5. DOI: `10.1145/3132847.3132967`. [Online]. Available: `https://dl.acm.org/doi/10.1145/3132847.3132967` (visited on 04/28/2024).

[37] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 714–735, May 1997, Conference Name: IEEE Transactions on Neural Networks, ISSN: 1941-0093. DOI: `10.1109/72.572108`.

[38] M. Gori, G. Monfardini, and F. Scarselli, “A new model for learning in graph domains,” in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, ISSN: 2161-4407, vol. 2, Jul. 2005, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942.

[39] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The Graph Neural Network Model,” *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009, Conference Name: IEEE Transactions on Neural Networks, ISSN: 1941-0093. DOI: 10.1109/TNN.2008.2005605.

[40] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” en, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan. 2021, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2020.2978386. [Online]. Available: https://ieeexplore.ieee.org/document/9046288/ (visited on 08/17/2023).

[41] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural Message Passing for Quantum Chemistry,” en, in *Proceedings of the 34th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Jul. 2017, pp. 1263–1272. [Online]. Available: https://proceedings.mlr.press/v70/gilmer17a.html (visited on 04/28/2024).

[42] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph Attention Networks*, arXiv:1710.10903 [cs, stat], Feb. 2018. DOI: 10.48550/arXiv.1710.10903. [Online]. Available: http://arxiv.org/abs/1710.10903 (visited on 04/28/2024).

[43] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, “Spectral Networks and Locally Connected Networks on Graphs,” Dec. 2013.

[44] D. Grattarola and C. Alippi, “Graph Neural Networks in TensorFlow and Keras with Spektral [Application Notes],” *IEEE Computational Intelligence Magazine*, vol. 16, pp. 99–106, Feb. 2021. DOI: 10.1109/MCI.2020.3039072.

[45] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, “Graph Neural Networks with Convolutional ARMA Filters,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1–1, Jan. 2021. DOI: 10.1109/TPAMI.2021.3054830.

[46] J. Gasteiger, A. Bojchevski, and S. Günnemann, *Predict then Propagate: Graph Neural Networks meet Personalized PageRank*, arXiv:1810.05997 [cs, stat], Apr. 2022. DOI: 10.48550/arXiv.1810.05997. [Online]. Available: http://arxiv.org/abs/1810.05997 (visited on 04/29/2024).

[47] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank Citation Ranking : Bringing Order to the Web,” Nov. 1999. [Online]. Available: https://www.semanticscholar.org/paper/The-PageRank-Citation-Ranking-%3A-Bringing-Order-to-Page-Brin/eb82d3035849cd23578096462ba419b53198a556 (visited on 05/15/2024).

[48] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, “Graph WaveNet for Deep Spatial-Temporal Graph Modeling,” in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 1907–1913. DOI: 10.24963/ijcai.2019/264. [Online]. Available: https://doi.org/10.24963/ijcai.2019/264.

[49] A. Jain, A. Zamir, S. Savarese, and A. Saxena, *Structural-RNN: Deep Learning on Spatio-Temporal Graphs*. Jun. 2016, Pages: 5317. DOI: 10.1109/CVPR.2016.573.

[50] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, *GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs*, arXiv:1803.07294 [cs], Mar. 2018. DOI: 10.48550/arXiv.1803.07294. [Online]. Available: http://arxiv.org/abs/1803.07294 (visited on 04/29/2024).

[51] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, “Structured Sequence Modeling with Graph Convolutional Recurrent Networks,” in *Neural Information Processing*, L. Cheng, A. C. S. Leung, and S. Ozawa, Eds., Cham: Springer International Publishing, 2018, pp. 362–373, ISBN: 978-3-030-04167-0.

[52] S. Yan, Y. Xiong, and D. Lin, “Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition,” en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018, ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v32i1.12328. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/12328 (visited on 04/28/2024).

[53] B. Yu, H. Yin, and Z. Zhu, “Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting,” en, in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 3634–3640, ISBN: 978-0-9992411-2-7. DOI: 10.24963/ijcai.2018/505. [Online]. Available: https://www.ijcai.org/proceedings/2018/505 (visited on 04/28/2024).

[54] M.-H. Tayarani N., “Applications of artificial intelligence in battling against covid-19: A literature review,” *Chaos, Solitons, and Fractals*, vol. 142, p. 110 338, Jan. 2021, ISSN: 0960-0779. DOI: 10.1016/j.chaos.2020.110338. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7532790/ (visited on 04/28/2024).

[55] H. S. Maghdid, K. Z. Ghafoor, A. S. Sadiq, K. Curran, D. B. Rawat, and K. Rabie, *A Novel AI-enabled Framework to Diagnose Coronavirus COVID 19 using Smartphone Embedded Sensors: Design Study*, arXiv:2003.07434 [cs, q-bio], May 2020. DOI: 10.48550/arXiv.2003.07434. [Online]. Available: http://arxiv.org/abs/2003.07434 (visited on 05/17/2024).

[56] M. Jamshidi, A. Lalbakhsh, J. Talla, *et al.*, "Artificial Intelligence and COVID-19: Deep Learning Approaches for Diagnosis and Treatment," *IEEE Access*, vol. 8, pp. 109 581–109 595, 2020, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3001973. [Online]. Available: https://ieeexplore.ieee.org/document/9115663 (visited on 05/17/2024).

[57] M. Ahishali, A. Degerli, M. Yamac, *et al.*, "Advance Warning Methodologies for COVID-19 using Chest X-Ray Images," en, *IEEE Access*, vol. 9, pp. 41 052–41 065, 2021, arXiv:2006.05332 [cs, eess], ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3064927. [Online]. Available: http://arxiv.org/abs/2006.05332 (visited on 05/17/2024).

[58] T. K. Torku, A. Q. M. Khaliq, and K. M. Furati, "Deep-Data-Driven Neural Networks for COVID-19 Vaccine Efficacy," en, *Epidemiologia*, vol. 2, no. 4, pp. 564–586, Dec. 2021, Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2673-3986. DOI: 10.3390/epidemiologia2040039. [Online]. Available: https://www.mdpi.com/2673-3986/2/4/39 (visited on 05/17/2024).

[59] K. Avchaciov, O. Burmistrova, and P. Fedichev, *AI for the repurposing of approved or investigational drugs against COVID-19*. Mar. 2020. DOI: 10.13140/RG.2.2.20588.10886.

[60] Y. Wu, Y. Yang, H. Nishiura, and M. Saitoh, "Deep Learning for Epidemiological Predictions," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, ser. SIGIR '18, New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 1085–1088, ISBN: 978-1-4503-5657-2. DOI: 10.1145/3209978.3210077. [Online]. Available: https://dl.acm.org/doi/10.1145/3209978.3210077 (visited on 04/28/2024).

[61] S. Deng, S. Wang, H. Rangwala, L. Wang, and Y. Ning, *Graph Message Passing with Cross-location Attentions for Long-term ILI Prediction*. Dec. 2019.

[62] A. Kapoor, X. Ben, L. Liu, *et al.*, *Examining COVID-19 Forecasting using Spatio-Temporal Graph Neural Networks*. Jul. 2020.

[63] J. Gao, R. Sharma, C. Qian, *et al.*, "STAN: Spatio-temporal attention network for pandemic prediction using real-world evidence," *Journal of the American Medical Informatics Association*, vol. 28, no. 4, pp. 733–743, Apr. 2021, ISSN: 1527-974X. DOI: 10.1093/jamia/ocaa322. [Online]. Available: https://doi.org/10.1093/jamia/ocaa322 (visited on 04/28/2024).

[64] G. Panagopoulos, G. Nikolentzos, and M. Vazirgiannis, "Transfer Graph Neural Networks for Pandemic Forecasting," en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 6, pp. 4838–4845, May 2021, Number: 6, ISSN: 2374-3468. DOI: 10.1609/aaai.v35i6.16616. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16616 (visited on 04/28/2024).

[65] R. Alizadeh, J. K. Allen, and F. Mistree, "Managing computational complexity using surrogate models: A critical review," *Research in Engineering Design*, vol. 31, no. 3, pp. 275–298, Jul. 2020, ISSN: 1435-6066. DOI: `10.1007/s00163-020-00336-7`. [Online]. Available: `https://doi.org/10.1007/s00163-020-00336-7`.

[66] Z. Liu, Y. Yang, and Q. Cai, "Neural network as a function approximator and its application in solving differential equations," *Applied Mathematics and Mechanics*, vol. 40, pp. 237–248, Feb. 2019. DOI: `10.1007/s10483-019-2429-8`.

[67] G. Calzolari and W. Liu, "Deep learning to replace, improve, or aid CFD analysis in built environment applications: A review," *Building and Environment*, vol. 206, p. 108 315, Dec. 2021, ISSN: 0360-1323. DOI: `10.1016/j.buildenv.2021.108315`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0360132321007137` (visited on 04/28/2024).

[68] R. Shi, Z. Mo, K. Huang, X. Di, and Q. Du, "A Physics-Informed Deep Learning Paradigm for Traffic State and Fundamental Diagram Estimation," en, *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 11 688–11 698, Aug. 2022, ISSN: 1524-9050, 1558-0016. DOI: `10.1109/TITS.2021.3106259`. [Online]. Available: `https://ieeexplore.ieee.org/document/9531557/` (visited on 04/15/2024).

[69] C. Angione, E. Silverman, and E. Yaneske, "Using machine learning as a surrogate model for agent-based simulations," eng, *PloS One*, vol. 17, no. 2, e0263150, 2022, ISSN: 1932-6203. DOI: `10.1371/journal.pone.0263150`.

[70] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987–1000, Sep. 1998. DOI: `10.1109/72.712178`.

[71] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural Ordinary Differential Equations," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf`.

[72] R. G. Nascimento, K. Fricke, and F. A. C. Viana, "A tutorial on solving ordinary differential equations using Python and hybrid physics-informed neural network," *Engineering Applications of Artificial Intelligence*, vol. 96, p. 103 996, Nov. 2020, ISSN: 0952-1976. DOI: `10.1016/j.engappai.2020.103996`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S095219762030292X` (visited on 04/28/2024).

[73] S. Mall and S. Chakraverty, "Comparison of Artificial Neural Network Architecture in Solving Ordinary Differential Equations," *Advances in Artificial Neural Systems*, vol. 2013, pp. 1–12, Jan. 2013. DOI: `10.1155/2013/181895`.

[74] M. J. Kühn, D. Abele, D. Kerkmann, *et al.*, *MEmilio - a high performance Modular EpideMIcs simuLatIOn software*, 2022. [Online]. Available: https://github.com/DLR-SC/memilio.

[75] W. Koslow, M. J. Kühn, S. Binder, *et al.*, "Appropriate relaxation of non-pharmaceutical interventions minimizes the risk of a resurgence in SARS-CoV-2 infections in spite of the Delta variant," *medRxiv*, 2021. DOI: 10.1101/2021.07.09.21260257.

[76] H. Zunker, R. Schmieding, D. Kerkmann, *et al.*, "Novel travel time aware metapopulation models: A combination with multi-layer waning immunity to assess late-phase epidemic and endemic scenarios," en, Epidemiology, preprint, Mar. 2024. DOI: 10.1101/2024.03.01.24303602. [Online]. Available: http://medrxiv.org/lookup/doi/10.1101/2024.03.01.24303602 (visited on 03/20/2024).

[77] M. J. Kühn, D. Abele, S. Binder, *et al.*, "Regional opening strategies with commuter testing and containment of new SARS-CoV-2 variants in Germany," en, *BMC Infectious Diseases*, vol. 22, no. 1, p. 333, Dec. 2022, ISSN: 1471-2334. DOI: 10.1186/s12879-022-07302-9. [Online]. Available: https://bmcinfectdis.biomedcentral.com/articles/10.1186/s12879-022-07302-9 (visited on 01/09/2023).

[78] F. a. o. Chollet, *Keras*, en, 2015. [Online]. Available: https://keras.io (visited on 05/19/2024).

[79] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," en, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2016.2582924. [Online]. Available: http://ieeexplore.ieee.org/document/7508408/ (visited on 04/28/2024).

[80] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, *Graph Neural Networks: A Review of Methods and Applications.* Dec. 2018.

[81] BMAS, *Pendlerverflechtungen der sozialversicherungspflichtig Beschäftigten nach Kreisen - Deutschland (Jahreszahlen)*, de, Publication Title: www.bmas.de, 2020. [Online]. Available: https://statistik.arbeitsagentur.de/SiteGlobals/Forms/Suche/Einzelheftsuche_Formular.html?topic_f=beschaeftigung-sozbe-krpendd (visited on 05/11/2021).

[82] M. Fey and J. E. Lenssen, *Fast Graph Representation Learning with PyTorch Geometric*, original-date: 2017-10-06T16:03:03Z, May 2019. [Online]. Available: https://github.com/pyg-team/pytorch_geometric (visited on 05/18/2024).

[83] M. Wang, D. Zheng, Z. Ye, *et al.*, *Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks*, arXiv:1909.01315 [cs, stat], Aug. 2020. DOI: 10.48550/arXiv.1909.01315. [Online]. Available: http://arxiv.org/abs/1909.01315 (visited on 05/18/2024).

[84] Y. Li, R. Yu, C. Shahabi, and Y. Liu, *Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting*, arXiv:1707.01926 [cs, stat], Feb. 2018. DOI: 10.48550/arXiv.1707.01926. [Online]. Available: http://arxiv.org/abs/1707.01926 (visited on 04/28/2024).

[85] M. E. J. Newman, "Analysis of weighted networks," en, *Physical Review E*, vol. 70, no. 5, p. 056 131, Nov. 2004, ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.70.056131. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.70.056131 (visited on 04/04/2024).

[86] L. Tang, Y. Zhou, L. Wang, *et al.*, "A Review of Multi-Compartment Infectious Disease Models," eng, *International Statistical Review = Revue Internationale De Statistique*, vol. 88, no. 2, pp. 462–513, Aug. 2020, ISSN: 0306-7734. DOI: 10.1111/insr.12402.