UiA University of Agder

Master

Analyzing the performance of transformers for streamflow prediction

JONATAN HERTZBERG HINDERSLAND

SUPERVISOR
Per-Arne Andersen
Lei Jiao

University of Agder, 2023
Faculty of Engineering and Science
Department of Engineering and Sciences

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Acknowledgements

I would like to express gratitude and appreciation to my supervisors Per-Arne Andersen and Lei Jiao for their valuable guidance and insights throughout this project.

I would like to thank Bernt Viggo Matheussen and Å Energi (previously Agder Energi) for introducing the basis of this project idea.

I would like to thank the Neuralhydrology team for creating the library upon which this project is built.

Finally, I would like to thank the University of Agder for providing the resources necessary to more efficiently train the models created for this project.

Abstract

Within the field of hydrology, there is a vital need to be able to predict streamflow values from hydrological basins. This has traditionally been done through physics and mathematics-based models, where measured data are combined with physics-based formulas to estimate output values. Nowadays, machine learning has been introduced as a potential way to improve the performance of these predictions. One of the classical methods for time-series prediction has been the Long Short Term Memory (LSTM) model, but the transformer model has also shown its ability to be proficient at solving these kinds of problems. The purpose of this paper is to implement a transformer model within an existing model library for streamflow prediction and analyze its performance, both in terms of how it is affected by various hyperparameters and how it compares to other models. The findings from these tests indicate that the transformer encoder is capable of achieving comparable results to the LSTM, while the full transformer model performs noticeably worse. In addition, the paper also finds that the cumulative distribution of the full transformer is significantly different than the other models, performing worse on most basins, but significantly better on the top 20%. This indicates the model is better at specializing on certain basin groupings, at the cost of generalization. Lack of generalization is a detriment, however, if the model or the data processing could be adapted to exploit the model's ability to specialize, then it may achieve better results. This could be done by, for instance, adding an explicit categorization dimension to the model. To summarize, the transformer model does not currently outperform the state-of-the-art LSTM models, but it expresses interesting behavior that should be studied further.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This section will attempt to explain the area of research and its inherent challenges. It will also detail the stated goal of this paper.

1.1 Problem

Streamflow prediction is a very important area of study in the field of hydrology. Being able to predict the water output for a water basin is vital to predicting how much power can be generated, which in turn is vital for both electricity usage and electricity pricing. A hydropower company with a good method for streamflow prediction will likely be able to out-compete other companies by being able to provide buyers with more reliable promises. Traditionally the methods used for this purpose are physics-based prediction models. These rely on various laws of physics to interpret the input data in a way that can predict streamflow to a relatively high degree of accuracy. However, these methods are still not perfect, so the search for better prediction models is still ongoing.

1.1.1 Difficulties

Streamflow prediction is a difficult field of study due to the large amounts of variables that can impact the results. There can also be massive differences between basins. This is why physics-based models need to be calibrated to each basin individually. Doing this does of course increase the accuracy, but it also means the models cannot be effectively generalized. The result of this is that physical models are unsuited for predictions of ungauged basins. In other words, the models cannot make accurate predictions for basins where streamflow data is unavailable or limited.

1.1.2 Solution

Machine learning is a method that has a high potential for being able to outperform physics-based models. Machine learning would also potentially be able to bypass some of the issues with physical models by creating models that can be generalized, even to ungauged basins. This potential has been demonstrated by the research group Neuralhydrology by Frederik Kratzert, Daniel Klotz, Martin Gauch and Grey Nearing [13]. This research group has primarily focused on using LSTM (Long Short-Term Memory) models and have achieved good results. The team has experimented with a large variety of LSTM models, by for example introducing physics-based rules to the model. The goal of this thesis is to expand on the work done by the Neuralhydrology team by focusing more heavily on transformers. The reason for this is that transformers have shown potential for time series prediction, even within hydrology specifically [2][22] and this warrants further study.

1.2 Motivation

The basis for this thesis is a research project at a summer job for Å Energi (Previously Agder Energi). The purpose of that research project was to adapt the Neuralhydrology framework for Norwegian data to verify if such an approach would be viable. This thesis is an expansion on that project with a reduced focus on the dataset and a bigger focus on the models themselves.

1.3 Field of research

The field of hydrology is very active. Many different models have been invented for the sake of streamflow prediction. These include the Sacramento Soil Moisture Accounting model (SAC-SMA), the Variable Infiltration Capacity model (ViC), the mesoscale Hydrologic Model (mHM), and the Hydrologiska Byråns Vattenbalansavdelning model (HBV). In recent years the newest addition to the field of hydrology has been machine learning. Various machine learning models have been tested for this purpose, however, the one that seems to provide the best results is the LSTM model [11].

1.4 Thesis definition

The purpose of this project is specifically to examine the use of transformers for streamflow prediction. Some studies have indicated the potential of transformers within the field of hydrology [2][22]. The point of this paper is to try to verify the results of these studies, and more importantly, analyze the differences in behavior between the transformer model and the more traditional LSTM.

1.5 Contribution

The contributions of this paper are as follows: Firstly there is the implementation of a full transformer model within the Neuralhydrology framework [13]. In practice, this means a transformer model using the CAMELS US dataset. Yin et al (2022) [22] claims to have implemented such a model, however, the cited code repository is empty, and as such could not be the basis for this project. Secondly, the model parameters will be examined thoroughly to determine their effects on the model. Thirdly the performance of the model will be compared not just to the standard LSTM but also to other models implemented within the Neuralhydrology library, as well as physical models. Fourthly the behavior of the transformer model will be examined compared to the LSTM models.

1.6 Thesis Outline

This thesis is structured in the following manner. Chapter 2 will discuss the background theory necessary to understand this paper, including introducing the Neuralhydrology library, the transformer model and variations of it, physical hydrology models currently in use, metrics used to determine the quality of hydrological models, and finally the dataset used in this paper. Chapter 3 will talk about how the transformer model has been implemented in this paper, as well as pre-and post-processing. Chapter 4 will introduce the various experiments conducted in this paper and showcase their results. Chapter 5 will analyze the performance of the transformer model and discuss ideas for further improvement to the model. Finally, chapter 6 will summarize the findings of this paper.

Chapter 2

Theory

This chapter will focus on machine learning methods within hydrology, as well as discuss the general concepts of transformers. The first area of interest will be the Neuralhydrology library [13] and the models implemented therein. The paper will then go into further detail on transformers specifically, and then discuss transformers within the field of hydrology. Also included in this section is a basic overview of some physical models used in hydrology and a basic overview of some metrics used to examine the accuracy of hydrology models. Finally, there is a short overview of the dataset used in this paper.

2.1 Neuralhydrology

The Neuralhydrology library [13] is a state-of-the-art model repository focused on hydrology predictions. The main focus is on Long Short-Term Memory (LSMT) models, as they are indicated to be the machine learning models best designed for streamflow prediction. A large focus of the research group is designed to modify the traditional LSTM models in an attempt to create better results. The reason this section is important is twofold. Firstly we need to establish what the Neuralhydrology framework is and how it works. Secondly, it is important to recognize which methods have been attempted, specifically on the CAMELS dataset.

2.1.1 CudaLSTM

The CudaLSTM (see figure 2.1) is the standard implementation of the LSTM model. It relies on the base implementation present in PyTorch. The model contains a cell state and a hidden state. Input data is divided into sequences where each step of the input sequence is used to modify the cell state which is in turn used to modify the hidden state. The model can then output a prediction at each time step as well as recognize continuous patterns over time. This model implementation was first described in "Rainfall–runoff modeling using Long Short-Term Memory (LSTM) networks" (Kratzert et al, 2018) [11]. The study demonstrates that the LSTM model is able to provide comparable results to the widely used Sacramento Soil Moisture Accounting Model (SAC-SMA), or specifically SAC-SMA + Snow-17.

2.1.2 ARLSTM

The AutoRegressive LSTM is an LSTM model where one of the inputs is a time-lagged version of the output, in other words, the streamflow. The standard LSTM is in a way able to do this as well, as the hidden state is supposed to represent the previous streamflow value. However this is an estimated value, and so the AR-LSTM instead provides a measured value. The model is described in the paper "Technical note: Data assimilation and autoregression for using near-real-time streamflow observations in long short-term memory

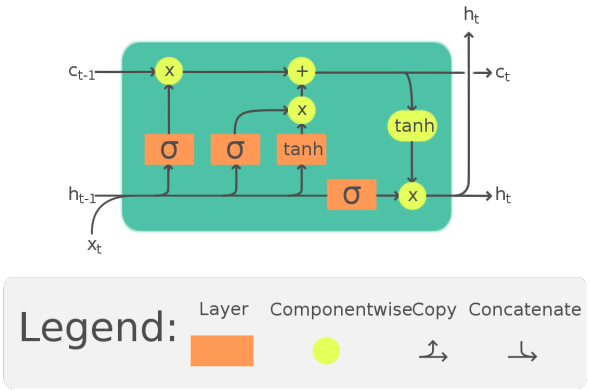

Figure 2.1: Standard Long Short-Term Memory (LSTM) model. Consists of a cell state and a hidden state, as well as three gates, the input gate, the output gate and the forget gate. The hidden state and the input are together used to update the cell which is in turn used to determine the output and the next hidden state [11][5].

networks" (Nearing et al, 2022), and the results of the study indicate that the AR-LSTM performs better than the standard LSTM model [18].

2.1.3 EA-LSTM

The Entity-Aware LSTM model (see figure 2.2) is an LSTM model where the static inputs are separated from the dynamic inputs. The static inputs are used for the input gate activation while the dynamic inputs are used for the other gates. This is an attempt to adapt the standard LSTM model to be more in line with hydrological theory by separating the catchment attributes into a separate feature layer. This model is described in the paper "Towards learning universal, regional, and local hydrological behaviors via machine learning applied to large-sample datasets" (Kratzert et al, 2019) and the results of the study indicate that the model performs slightly worse than the standard LSTM with static inputs, though it still outperforms the physical models Variable Infiltration Capacity (VIC) and mesoscale Hydrologic model (mHm) [12].

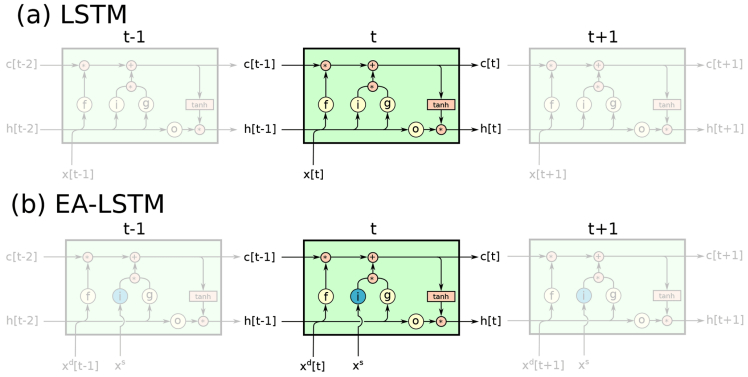

Figure 2.2:
a) Standard LSTM, same as Fig. 2.1.
b) Entity Aware LSTM (EA-LSTM) is an LSTM model where the static attributes are separated from the dynamic attributes. The static inputs are used as activation for the input gate, whereas the dynamic inputs are used as activation for the forget and output gates [12].

2.1.4 MC-LSTM

The Mass-Conserving LSTM (see figure 2.3) is a model designed to account for the physical law of conservation of mass. The idea is that mass inputs are separated from the auxiliary inputs so that the model can better account for mass conservation, where the mass inputs are fed directly into the input gate, whereas the auxiliary inputs are used to determine the activation for the three gates, forget, input and output. Unlike standard LSTM models, this model does not contain a hidden state. When using the CAMELS data set the mass inputs would be the precipitation whereas the auxiliary inputs would be the rest of the inputs. The MC-LSTM model was described in the paper "MC-LSTM: Mass-Conserving LSTM" (Hoedt et al, 2021) [9] and the results of the study indicate that the MC-LSTM model is not necessarily better than the standard LSTM for streamflow prediction, but it still showed better results than other mass conserving models. The MC-LSTM is also able to solve mathematical problems that the standard LSTM cannot, and thus while it may not be directly better for streamflow prediction it can still prove superior in certain use cases.

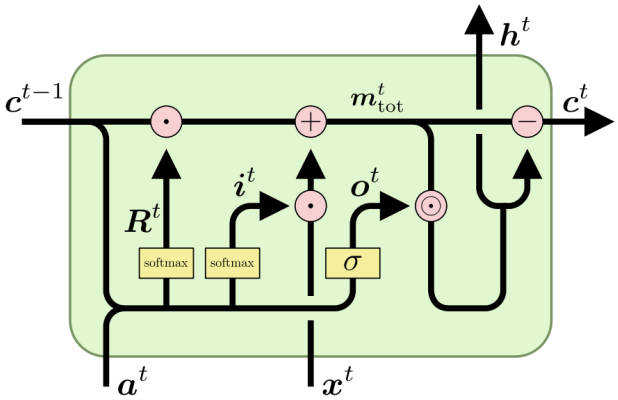

Figure 2.3: Mass Conserving LSTM (MC-LSTM): An LSTM model for the mass inputs (ie precipitation) are separated from the auxiliary inputs. The mass inputs are fed directly through the input gate where the other inputs are used as activation for the three gates. Unlike other LSTM models, this model does not contain a hidden state [9].

2.1.5 MTS-LSTM

The Multi-Timescale LSTM is a model designed to account for time-series data of multiple temporal resolutions, for instance daily and hourly. It is of course possible to simply train an LSTM model on an hourly timescale, but this can be computationally inefficient and so the MTS-LSTM model was suggested. This model allows long-past inputs to be processed at a different temporal resolution than more recent inputs. The model can also calculate different variables at different timescales. The model was described in the paper "Rainfall–runoff prediction at multiple timescales with a single Long Short-Term Memory network" (Gauch et al, 2021) [7] and shows that the MTS-LSTM performs very similarly to the naive LSTM, where the MTS-LSTM performs slightly worse and the shared Multi-TimeScale LSTM (sMTS-LSTM) performs slightly better. However, both models outperform the National Water Model (NWM).

2.1.6 ODE-LSTM

The Ordinary Differential Equation LSTM is a model designed to handle irregularly sampled time-series data [14]. The model uses Ordinary Differential Equations (ODE) to represent the hidden state of the model. The way this model is implemented in the Neuralhydrology

framework is that it simulates the data irregularity by "aggregating parts of the input sequence to random frequencies" [13]. No study describing the results of this implementation could be found for this paper. One reason for this could be that the concepts implemented by the ODE-LSTM became the basis for the MTS-LSTM, though this is speculation.

2.1.7 GRU

The Gated Recurrent Unit (GRU) [4] (see fig 2.4) is a recurrent machine learning model quite similar to the LSTM model. Both models are based heavily on the concepts of gates, but while the LSTM has three gates, the forget gate, the input gate, and the output gate, the GRU model only has two, the reset gate and the update gate. Additionally, the GRU model does not use the hidden and cell states present in the LSTM model and instead relies only on the hidden state. The reset gate determines which part of the previous hidden state should be forgotten for the input. The update gate determines both which inputs should be remembered and which past information should be forgotten for the output. To compare with the LSTM model the update gate is a combination of the forget and input gates for the next state, whereas the reset gate is in a sense the forget gate for the previous state. The GRU model in the Neuralhydrology framework uses the standard PyTorch implementation and seems to exist as an example of how to add new models, and as such there does not seem to be any paper describing the results of the model.

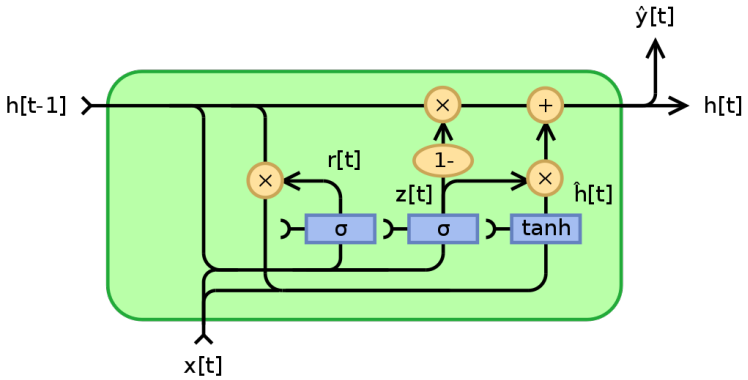

Figure 2.4: Gated Recurrent Unit (GRU) is a recurrent machine learning model that uses two gates, the reset gate and the update gate. The reset gate removes data from the previous hidden state input, and the update gate both determines what input should be added and what data in the hidden state should be kept for the output [4].

2.1.8 Transformer

The transformer implemented in Neuralhydrology uses the default PyTorch transformer encoder implementation. A more detailed look at this model will be included in the transformer section.

2.2 Transformers

2.2.1 Model types

Basic transformer

The original transformer model was described in the 2017 paper "Attention is all you need" [21]. The central principle of the model is that it relies entirely on the concept of attention.
To properly describe the transformer model we must first discuss the pre-processing required by the model. Most machine learning models use the model input to predict the output

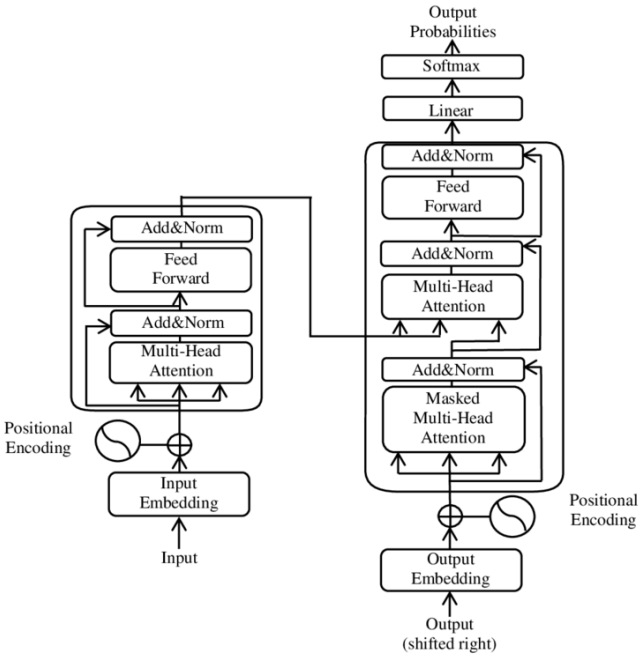

Figure 2.5: Transformer model. The output is shifted right to remove the value to be predicted. The input and shifted outputs are then embedded and given positional encoding. The input is passed through the encoder layer and then both the shifted output and the output from the encoder are passed into the decoder. The decoder output is then passed through the model head to get the result [21].

whereas the true output is only used to calculate loss for back-propagation. The transformer model is based on attention and is thus also able to take the true output as an input in order to better understand the relationship between the input and the output. Since the last element of the true output is never supposed to be visible to the model the entire output is shifted to the right, effectively removing the last element. The input and output values are then embedded to the right format and dimensions for the model. After this both values are assigned positional encoding to give them an explicit time dimension. The positional encoding in the original paper [21] is described as such:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}), \tag{2.1}$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right). \tag{2.2}$$

After this the source input is fed into the transformer encoder and the target input is fed into the transformer decoder. The first part of the transformer model itself is the multi-head attention layer. This layer consists of multiple parallel scaled dot-product attention layers, depending on the number of transformer heads (see figure 2.6). For each scaled dot-product attention the input is duplicated three times and passed through a linear layer. This then creates the matrices Queries, Keys, and Values (Q, K, and V). The Q and K matrices are multiplied together and scaled. The result can then be masked depending on the requirement of the model. For the decoder this is generally recommended, but for the encoder it can vary. The masked result is then passed through a softmax and is multiplied by the V matrix. This essentially means that the Q and K matrices create a gate for the V matrix. These operations are done for all transformer heads in the multi-head attention, after which the outputs are concatenated and passed through a linear layer to return to the original dimensions.
After the multi-head attention, or masked multi-head attention, we then have a layer that adds together the output of the attention layer with the original input and normalizes the value. This adding and normalizing happens after every layer in the transformer. For the

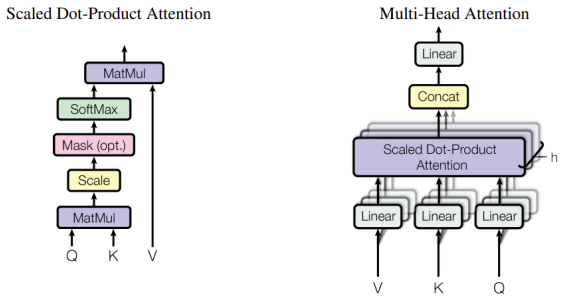

Figure 2.6: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. The inputs are Query, Key and Value [21].

encoder, we then move on to the feed-forward layer. This layer consists of two fully connected layers with a ReLu activation between them.

For the next layer of the decoder, we have another attention layer. This functions mechanically the same as the first, except for the inputs themselves. Instead of Q, K, and V matrices derived from the previous decoder attention layer, we take the Q matrix from the decoder attention and the V and K matrices from the encoder output. The output from this decoder attention layer is then added and normalized with the output from the previous decoder attention layer. Finally, there is a feed-forward layer similar to the encoder section. Multiple encoder and decoder layers can be used in conjunction.

After the transformer model itself, there is the model head. This can vary wildly based on the particular use case, but a typical solution is a linear layer. For the transformer model shown in figure 2.5 the head consists of a linear layer and a softmax activation function.

BERT

The Bidirectional Encoder Representation from Transformers (BERT) model is a type of transformer model focused specifically on the encoder section [6]. Instead of containing an encoder and a decoder section, the BERT model is simply a stack of transformer encoder layers. As the name implies BERT is a bidirectional model, as opposed to the unidirectional approach of the standard masked transformer model. This means that instead of only having access to previous inputs the model has access to both previous and future inputs. The methods to train BERT are twofold. The first is Masked Language Modeling (MLM) which means that random tokens of the inputs are masked and the model is tasked with predicting the masked tokens, using both earlier and later tokens to learn the context. The second method is Next Sentence Prediction (NSP) where the model is given two sequences and is tasked with predicting whether they are consecutive. These two methods together help the model get a good understanding of context and correlation in time series data.

GPT

The Generative Pre-trained Transformer (GPT) model is a transformer model based specifically on the decoder architecture. Instead of containing an encoder and a decoder section the GPT model is simply a stack of transformer decoder layers. This means that each decoder layer contains a masked self-attention layer and a feed forward layer. The attention layer in the standard transformer model that takes input from the encoder layer is not present in GPT as it does not contain an encoder.

2.2.2 Hydrological implementation

This section will detail the transformers implemented in the field of hydrology.

Neuralhydrology

The transformer model implemented in the Neuralhydrology framework [13] is most closely based on the GPT architecture. It utilizes a transformer encoder from the PyTorch library with an upper triangular (sequential) mask. This means that, like GPT, only previous values are visible while future values are masked. The reason the model is classified as an encoder is that while GPT receives the target sequence as input, the input in the Neuralhydrology transformer is the source sequence, whereas the target sequence is never visible to the model. When it comes to results for this model there does not seem to be any published papers discussing such findings, at least not that could be acquired for this paper. An attempt was made to contact the development team to see if any such data was available, but no response has been received at the time of writing.

Apalachicola River

The paper "Hydrological Drought Forecasting Using a Deep Transformer Model" (Amanambu et al) [2] is a study conducted on the Apalachicola River in Florida, where data from two gauging stations were used to test the comparative performance of transformer versus LSTM models. The findings are that the transformer model performs better than the LSTM model on all timescales. The findings of this study are very valuable for this paper as it suggests the potential for the transformer model to outperform the LSTM, however, since the dataset is rather limited it is valuable to compare performance over a wider number of basins/gauging stations.

RR-Former

The Rainfall-Runoff Transformer (RR-Former) is a transformer model described in the paper "RR-Former: Rainfall-runoff modeling based on Transformer" (Yin et al, 2022) [22]. The paper describes implementing a basic transformer model on the CAMELS dataset. This model differs somewhat from the implementation that will be described later in this paper. For one the model does not employ a mask for the encoder section, only for the decoder. This is more accurate to the original transformer model described by Vaswani et al [21] however such an implementation is not necessarily realistic in the field of hydrology. The RR-Former allows the prediction of streamflow values based on all dynamic attributes, both past and future, but for practical implementation, any future values would be estimated and thus less accurate. The mask employed for the decoder is also different. The decoder for the RR-Former simply masks the final elements of the sequence, the ones that should be predicted, whereas the other elements are unmasked. This is different from the transformer described in this paper which uses an upper triangular mask for both the encoder and the decoder. Additionally, it seems that for the RR-Former this decoder mask is used instead of the right shift described by Vaswani et al [21], meaning instead of the values to be predicted being removed from the sequence they are simply masked.

The conclusion by the authors is that the RR-Former outperforms the LSTM models used for bench-marking. The paper goes into detail about the effects of pretraining and finetuning and, as expected, indicate both are important. The paper also compares these adjusted models with the LSTMs, however it does not explain whether the LSTM models are also pretrained or finetuned, and as such the comparison is slightly vague. Listed in the paper is a link to a code repository, however this repository is empty and as such the results of this study could not be independantly verified.

2.3 Physical hydrology models

This section will give a basic overview of some physical models used in hydrology. These are models that rely on the laws of physics and mathematics to estimate streamflow values, rather than machine learning. This section is included because values from these models are used for comparison in chapter 4.

2.3.1 SAC-SMA

The Sacramento Soil Moisture Accounting (SAC-SMA) model is a hydrological model first suggested by Burnash et al, 1973 [3]. The model is focused on calculating the moisture condition of the soil.

2.3.2 VIC

The Variable Infiltration Capacity (VIC) is a hydrological model developed by Liang et al, 1994 [15] at the University of Washington. The model separates the land surface into a grid and calculates the capacity for water to enter the soil.

2.3.3 mHM

The Mesoscale Hydrologic Model (mHM) is a hydrological model developed by Samaniego et al, 2010 [20]. The main principle of the model is to divide regions into grids on a mesoscale.

2.4 Metrics

This section will detail the different metrics that will be used in chapter 4

2.4.1 NSE

The Nash-Sutcliffe model Efficiency coefficient (NSE) [17] is an equation used to calculate the efficiency of hydrological models. NSE has a value range from -∞ to 1, where values close to 1 are desirable and 0 represents the average for the sequence. The metric is described in Eq. (2.3):

$$\text{NSE} = 1 - \frac{\Sigma_{t=1}^{T}(Q_o^t - Q_m^t)^2}{\Sigma_{t=1}^{T}(Q_o^t - \overline{Q}_o)^2}, \tag{2.3}$$

where Q is is discharge, o is observed, m is modelled and t is time.

2.4.2 Alpha NSE decomposition

The alpha NSE decomposition [8] is a metric that calculates the fraction between standard deviations between simulated and observed data. The range of possible values is from 0 to ∞ where values closer to 1 are desirable. The metric is described in Eq (2.4):

$$\alpha = \frac{\sigma_s}{\sigma_o}, \tag{2.4}$$

where σ is the standard deviation, s is simulated and o is observed.

2.4.3 Beta NSE decomposition

The beta NSE decomposition [8] is a metric that calculates the difference in means between the simulated and observed values over the standard deviation. The value range of the equation is from -∞ to ∞ where values closer to 0 are desirable. The metric is described in Eq. (2.5)

$$\beta = \frac{\mu_s - \mu_o}{\sigma_o}, \tag{2.5}$$

where μ is the mean, σ is the standard deviation, s is simulated and o is observed.

2.4.4 KGE

The Kling-Gupta Efficiency [8] is a model metric that attempts to improve upon NSE by combining attributes in a more balanced way. Similarly to NSE, KGE has a value range from -∞ to 1, where values closer to 1 are desirable. The metric is described in Eq. (2.6):

$$\text{KGE} = 1 - \sqrt{[s_r(r-1)]^2 + [s_\sigma(\sigma-1)]^2 + [S_\beta(\beta_{KGE}-1)^2]}, \tag{2.6}$$

where r is the correlation coefficient, α is the alpha NSE (Eq. (2.4)), β_{KGE} is the beta KGE (Eq. (2.7)) and s_r, s_α and s_β are the corresponing weights.

2.4.5 Beta KGE

The beta KGE [8] is a metric that calculates the fraction of the means between the simulated and observed values. The value range is between $-\infty$ and ∞, where values closer to 1 are desirable. The metric is described in Eq. (2.7)

$$\beta_{KGE} = \frac{\mu_s}{\mu_o}, \tag{2.7}$$

where μ is the mean, s is simulated and o is observed

2.5 CAMELS

The Catchment Attributes and MEteorology for Large-sample Studies (CAMELS) dataset is a dataset created by Newman et al, 2015 [19], and Addor et al, 2017 [1]. The dataset contains forcing and hydrologic response data for 671 hydrologic basins in the USA. The data is gathered from a period between 1980 and 2010.

Chapter 3

Practical implementation

The focus of this paper will be the implementation of a full transformer model. The preexisting Neuralhydrology framework does contain a transformer model, however, this model only contains the encoder section. This paper will therefore try to implement a full transformer model, with both an encoder and a decoder. The theoretical basis for this is that a full transformer model will have access to more data than the transformer encoder, and could thus potentially find patterns that the encoder could not. Both this full transformer model and the transformer encoder will be used for testing, however, since the encoder model is already implemented this chapter will focus on the full transformer.

This model will be implemented within the Neuralhydrology library. This is because the library has implemented useful tools for data loading, training, testing, and evaluation. Some of the elements in this section, such as the dataset and the optimizer, are unchanged from the Neuralhydrology studies this paper is based on. There are two main reasons for this. The first is that changing too many variables at once invalidates comparisons to other papers. The second is that changing these values is simply outside the scope of this paper. There is a near-infinite number of changes that could potentially be made to this framework, but this paper will focus primarily on the model itself and the preprocessing required for the model to run.

3.1 Data

3.1.1 Dataset

The dataset used for this project is the CAMELS US hydrology dataset. This is a large dataset well suited for training hydrological models, and it has been used in many other studies. This dataset is split into basins that each contain static and dynamic measurements. The static variables are data points for each basin that provide an overview of the basin itself, such as topography, vegetation, soil, geology, and climate. The dynamic variables are the regularly measured values. These exist both as daily and hourly values, but for this project, only the daily values will be used. Finally, there is the target variable, which is streamflow. Like the dynamic variables, these values are also daily measurements. The dataset will be split into training, validation, and testing sets, where the training set contains data from 1999 to 2008, the validation set contains data from 1980 to 1989, and the testing set contains data from 1989 to 1999. The attributes used in this paper are as follows [1]:

Dynamic attributes

- Precipitation
- Solar radiation
- Max temperature

- Min temperature
- Vapor pressure

Static attributes

- Mean precipitation
- Mean Potential EvapoTranspiration (PET)
- Aridity
- Precipitation seasonality
- Snow fraction
- Frequency of high precipitation
- Duration of high precipitation
- Frequence of low precipitation
- Duration of low precipitation
- Mean elevation
- Mean slope
- Catchment area
- Forest fraction
- Maximum Leaf Area Index (LAI)
- Difference between maximum and minimum Leaf Area Index
- Maximum Green Vegetation Fraction (GVF)
- Difference between maximum and minimum Green Vegetation Fraction
- Depth to bedrock
- Soil depth
- Soil porosity
- Soil hydraulic conductivity
- Max water content
- Sand fraction
- Silt fraction
- Clay fraction
- Carbonate rocks fraction
- Surface permeability

3.1.2 Dataloading

When loading the data, the static, dynamic, and target values are selected from a configuration file. The configuration file also determines the period from which the data should be loaded. The specific basins used are determined by a text file listing all basin ids. The three variable categories; static, dynamic, and target variables, are all loaded as separate tensors and kept as a dictionary. The static and dynamic variables remain separate until the preprocessing. This is because some of the models in the Neuralhydrology library use dynamic and static variables as separate inputs, but for the transformer model they will be combined, similarly to the standard LSTM.

3.2 Preprocessing

The main parts of the preprocessing pipeline are the embeddings, positional encoding, and masks.

3.2.1 Input shifting

For the prediction to be accurate, we need to shift the inputs provided to the model. First, we perform a right shift on the target (streamflow) input. This is done by removing the final element from the input tensor. The purpose of this is to indicate that the last element is the target of the prediction, and thus should not be included as part of the input. For example, if we use 99 days of streamflow data to predict the 100th day, then there is no reason for that final day to ever be sent into the model.

3.2.2 Embedding

The next section of preprocessing is the embedding. This embedding serves two purposes. The first is to combine the dynamic and static variables. The dynamic inputs are in the shape [batch size, sequence length, number of variables], whereas the static inputs are in the shape [batch size, number of variables]. This means that to combine the two tensors the dynamic inputs are transposed to contain the sequence length first and the static inputs are repeated to the sequence length. When the tensors are the same shape they are concatenated. The reason why the two tensors are combined is that the transformer encoder only accepts a single input tensor.

The second purpose of the embedding is to increase the size of the inputs to the correct shape for the final model. This is especially important for the transformer model as the input value needs to be divisible by the number of transformer heads. This is done through a fully connected layer. The method is applied to both the source and target tensors, and ensures that both have the same size. The reason they need to have the same size is that the scaled dot-product attention requires them to be multiplied.

3.2.3 Positional Encoding

The positional encoding is similar for both the source and target inputs, and is based on the formula described in the paper "Attention is all you need" (Vaswani et al, 2017) [21], shown in equations 2.1 and 2.2

The positional encoding serves the purpose of adding an explicit time dimension to the input values useful for attention. This means the model will be better able to determine where in the sequence a value belongs. The positional encoding allows the model to better understand the relationships between elements.

3.2.4 Masking

Masking is an important part of this transformer model. To ensure the model is not able to view future values in the time series those values need to be masked. This is achieved with a PyTorch triu mask where all future values are masked with '-inf' and all unmasked entries are indicated with 0. This mask is applied to both the source and target inputs, however, for the source inputs, we also shift the mask one position to the right. If we did not perform this right shift it would mean that if the model was trying to predict target values for day 2 it would use target and source inputs from day 1. This is a perfectly functional model and may be the most realistic for real-world applications because it does not need to rely

on estimated values. However, the LSTM models use source inputs to estimate same-day target outputs, so for comparison it is useful for the transformer to also have access to those values. With the mask shifted to the right it means that for predicting day 2 target outputs the model will have access to day 1 target inputs as well as source inputs for both day 1 and day 2. The masks can be seen in 3.1

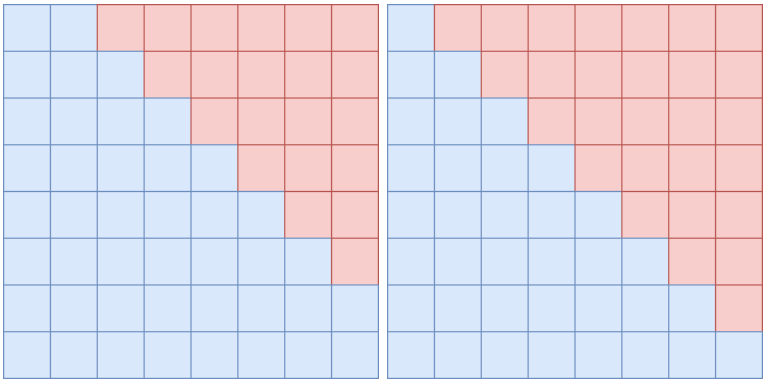

Figure 3.1: (left) Encoder mask. (right) Decoder mask.

3.3 Model

3.3.1 Transformer

The model itself utilizes the standard PyTorch transformer implementation. All of the model parameters, such as model dimensions, number of transformer heads, number of layers, feedforward dimensions, and dropout, are defined in the configuration file. Testing with all these parameters will be detailed in the chapter 4. The main difference to the transformer implementation described by Vashwani et al, 2017 [21], is that both the encoder and decoder use Masked Multi-Head attention. This is because for both the source and target inputs it does not make sense to have access to future values. For instance, it would not make sense to predict the streamflow values for one day using the precipitation from the following day. The implemented transformer model can be seen in figure 3.2

3.3.2 Model head

The Neuralhydrology framework supports several model heads, but for this project, only the standard regression head will be used. This is a simple linear layer that converts the transformer model output back into the target format.

3.4 Loss

The loss metric for this project is the Nash–Sutcliffe model efficiency coefficient (NSE), shown in Eq. (2.3). Using NSE for loss calculation seems to be the standard for Neuralhydrology studies. The Loss calculation in this paper has been attempted in two different ways. The first method attempted was based on a tutorial by Samuel Lynn-Evans [16] which said that the loss should be calculated over the entire sequence length. For instance, with a sequence length of 100, the loss would be calculated over all 100 elements, not just the elements being predicted. This method proved to make the model very unstable, and as such a different loss calculation was implemented, calculating loss over only the predicted parts of the sequence. The hypothesis of why did result occurred will be detailed in chapter 4.

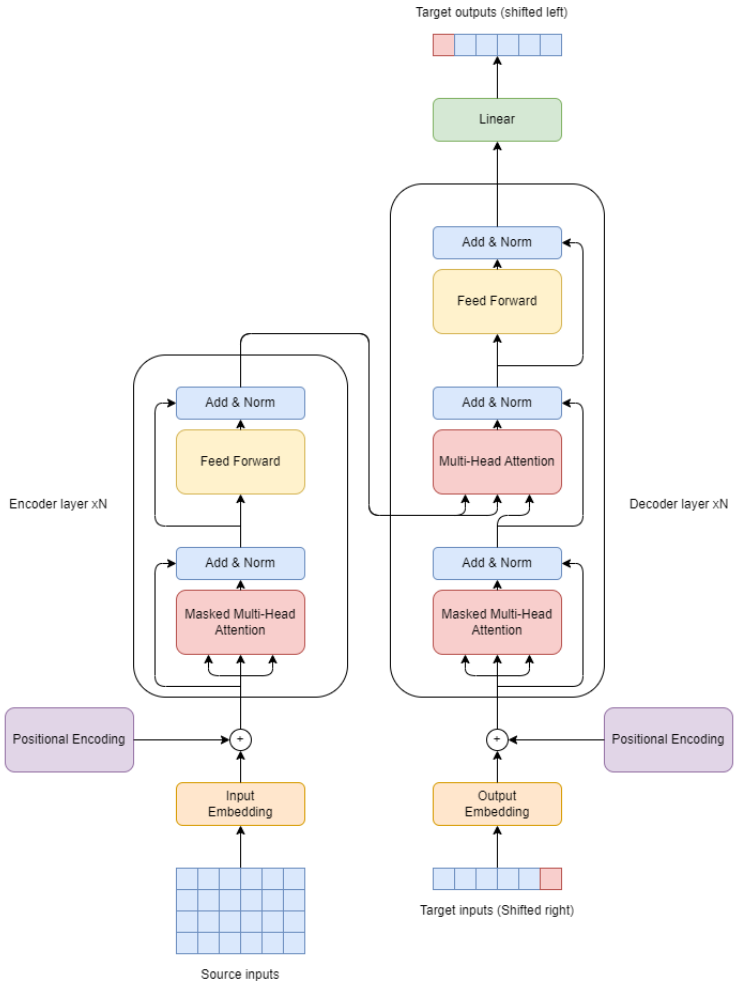

Figure 3.2: Slightly modified transformer model

3.5 Optimizer

The optimizer part of the Neuralhydrology framework has not been changed for this project. As such the used optimizer is Adam, which is currently the only optimizer implemented in the Neuralhydrology library.

Chapter 4

Results

The results chapter will detail two main sections. The first is hyperparameter testing, where models were trained with different hyperparameters, and their results will be presented and compared. The second section is a comparison of the transformer model with other existing models, both those included in the Neuralhydrology library and physical models like SAC and ViC. This also includes attempting to make changes to the model to see if performance can be improved. All model code, configuration files, and results are available through appendix A.

4.1 Parameter testing

4.1.1 Loss calculation

The initial attempt to test parameters was done on a dataset of 100 basins. The model implementation was based on a tutorial by Samuel Lynn-Evans [16] which claimed that the correct way to determine loss during training is to calculate the loss over the entire sequence, rather than just the elements to be predicted. This proved to make the results wildly unstable, causing massive swings in accuracy. This unpredictability was found to be significantly reduced when the loss was only calculated for the predicted values. However, since this issue was discovered rather late in the project, time limitations meant that repeating the experiments with the new loss calculation would be impractical. As such the experiment was redone but reduced to 10 basins.

For a demonstration of the difference between the two methods table 4.1 shows their relative NSE scores. Included in the table are results from a standard LSTM model with the same two loss calculation methods. This indicates that the reduction in accuracy happens for both methods, though the results seem less severe for the LSTM model. Calculating loss over the entire output sequence is a method that is based on the assumption that the input sequence could be of different lengths. This is reasonable to assume for text prediction, but in this study, the sequence length is fixed. As such it makes sense that the method of calculating loss only over the predicted elements would be the better alternative. All models are trained at 100 basins.

Model	Loss over all elements	Loss over only predicted elements
Transformer	0.165	**0.543**
LSTM	0.659	**0.723**

Table 4.1: NSE Scores with different loss calculations over 100 basins.

4.1.2 Hyperparameters

The hyperparameters examined in these tests are:

- Batch size: Size of batches passed through the model
- Epochs: Number of training iterations
- Hiddens: Size of the embedding layer for static and dynamic variables
- Seq len: The sequence length used to predict future values, includes both known and unknown values, ie a sequence length of 100 when predicting 1 value means 99 known values.
- Transformer dim feedforward: The internal dimension inside the transformer model used during the feedforward section.
- Transformer nheads: The number of transformer heads used during the attention mechanism
- Transformer nlayers: The number of transformer layers, ie the number of encoders and decoders stacked on top of each other
- Output dropout: Dropout value applied after the main model
- Transformer dropout: Dropout applied inside the transformer model
- Transformer positional dropout: Dropout applied to the positional encoding for the transformer

This section will contain tables for each variable, containing data from both the experiments on 100 basins and the ones on 10 basins. However, as mentioned in section 4.1.1, due to variations in loss calculations, the experiments on 100 basins provide significantly less reliable results, and this should be kept in mind. Some values from the 100 basin experiments are also missing, this is because they were deprioritized when it was seen that the method was faulty. The tables will also show what values caused the model to become too large to run, indicated by OOM (Out of Memory).

Batch size

As seen in table 4.2 higher batch sizes generally provide better results, but this does come at the cost of performance and it seems the advantage somewhat plateaus around 128 - 256. This is logical as smaller batch sizes mean that each model weight needs to be able to account for more basin values.

Basins	4	8	16	32	64	128	256	512	+512
10	-0.247	-0.257	-0.245	-0.076	0.335	0.612	0.640	**0.647**	OOM
100	0.450		-0.448	0.355	0.340	0.377	0.329	0.285	OOM

Table 4.2: NSE scores with different batch sizes.

Epochs

Table 4.3 perfectly demonstrates the unpredictability of the 100 basin experiments, seeing as how longer training does not seem to provide better results. However the 10 basin experiments do seem to align with known machine learning theory, that the model increases in accuracy but eventually starts to over-fit.

Basins	10	20	30	40	50	60	70	80	90	100
10	0.606	0.642	**0.643**	0.627	0.612	0.632	**0.643**	0.597	0.611	0.620
100	0.441	0.368	0.373	0.229	-1.149	0.118	-0.077	0.184		-0.319

Table 4.3: NSE scores with different epoch numbers.

Hiddens

The results from table 4.4 indicate that the size of the hiddens for the input embeddings does not make too much of a difference unless it becomes too large, at which point accuracy drops significantly. This result makes sense because of the size of the input variables, shown in section 3.1.1. Expanding the input variables too much, creating too much of an increase in the dimension, would logically muddle the values.

Basins	8	16	32	64	128	256	+256
10	0.642	**0.648**	0.642	0.646	-0.128	0.115	OOM
100	0.383	0.293	0.382	-0.867	-0.447	-0.029	OOM

Table 4.4: NSE scores with different hiddens.

Sequence Length

Table 4.5 indicates that shorter sequence lengths generally perform better. This result is interesting since some papers from the Neuralhydrology team [12] indicate a slightly higher sequence length, such as 270, is ideal. The logic behind a longer sequence length is that the streamflow values can fluctuate quite a bit depending on the season, and thus including a significant part of the year can help the model determine which part it is looking at. However, the results of this test seem to indicate that such a long sequence may not be necessary.

Basins	50	100	150	200	250	300	350	400	450	500	550	600
10	**0.661**	0.660	0.642	0.645	0.643	0.636	0.635	0.639	0.648	0.642	0.637	0.633
100	0.542	0.437	0.358	0.070	0.359	0.327	0.006	0.289	0.291	0.115	-0.096	0.185

Table 4.5: NSE scores with different sequence lengths.

Feedforward

As seen in table 4.6 it seems the dimension of the transformer feedforward does not play a massive part in the overall accuracy. The main focus of the transformer model is the attention mechanism. It is this mechanism that contributes the most to the model accuracy, not the following feedforward section. The results seem to validate this theory.

Transformer heads

The results of table 4.7 show that the number of transformer heads does not have a large impact on accuracy, however, it does have a decent impact on performance. This result may be because the basin set is limited, but it does indicate that increased transformer heads are more of insurance in case one or more model heads behave unpredictably, rather than a direct benefit to accuracy.

Transformer layers

Table 4.8 indicates that between 1 and 5 layers show negligible differences, but higher values reduce accuracy significantly. The logic behind this might be that too many layers cause the

Basins	32	64	128	256	512	1024
10	0.644	0.638	0.628	0.641	**0.651**	0.637
100	0.152	-1.841	-0.387	0.340	0.237	0.199

Table 4.6: NSE scores with different transformer feedforward dimensions.

Basins	1	2	4	8	+8
10	0.645	0. 643	0.645	**0.647**	OOM
100	0.378	0.323	0.146	0.259	OOM

Table 4.7: NSE scores with different numbers of transformer heads

results to become muddled

Basins	1	2	3	4	5	6	7	8	9	10
10	**0.649**	0.647	0.640	0.635	0.644	0.157	-0.231	0.056	-0.231	-0.239
100	0.339	-1.904	-0.052	0.234	0.313	-0.447		-0.447	-0.100	-0.447

Table 4.8: NSE scores with different numbers of transformer layers.

Dropout

Table 4.9 details the effect of changing the dropout values. As seen from the output dropout and positional encoding dropout we can see that the values are mostly within the natural fluctuations of the model, although they do drop slightly when the dropout becomes exceptionally high. However, the most interesting results are from the transformer dropout, where it seems the accuracy drops quickly and significantly when the dropout value is increased. One reason the transformer dropout has such an increased impact might be because it is applied in both sections of the encoder, as well as all three sections of the decoder. This means that in practice the transformer dropout values are significantly higher than the others. This underlines the idea that stacking multiple dropout values on top of each other is bad for accuracy, and as such it would likely be best to either use output, transformer, or positional encoding dropout, not multiple of them.

4.2 Large dataset testing

This section will detail the tests conducted on a larger section of the CAMELS dataset, 531 basins as opposed to 10 and 100 from the hyperparameter testing. The hyperparameters for all models are identical if not otherwise stated. This is to better analyze the reason for changing results.

Hyperparameters

- Batch size: 256
- Epochs: 100
- Hiddens: 32
- Sequence length: 100
- Learning rate: Epoch 0 - 0.001, Epoch 30 - 0.0005, Epoch 60 - 0.0001
- Transformer feedforward dimension: 256
- Number of transformer heads: 4

Dropout	Basins	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Output	10	0.638	0.635	0.638	0.615	0.637	0.632	**0.645**	0.640	0.634	0.590
	100			0.411			-0.012		0.380	0.357	
Transformer	10	**0.638**	0.619	0.604	0.530	0.404	0.325	0.080	-1.577	-0.244	-0.239
	100	0.130		0.448			-1.492		-3.998		-0.447
Positional Encoding	10	0.639	0.644	**0.645**	0.638	0.639	0.626	0.636	0.622	0.583	0.481
	100	0.337	0.285	0.241	0.381		0.007	0.416	0.361		-0.114

Table 4.9: NSE scores with different dropout values.

- Number of transformer layers: 4
- Output dropout: 0.4
- Transformer dropout: 0.0
- Transformer positional dropout: 0.0

4.2.1 Model comparisons

We start with a basic test of the performance of both the full transformer model, as well as the transformer encoder included in the Neuralhydrology library, against other hydrological models. The LSTM was trained manually as part of this paper, however, the results from the other models listed are taken from reports from the Neuralhydrology team. Worth noting is that these reports achieve different results for the base LSTM model. The LSTM results most comparable to those achieved in this report are those from the EA-LSTM [12]. This table also includes values from physically based models as a comparison with the non-machine learning methods currently employed in the field of hydrology.

	Mean NSE	Median NSE	KGE	Beta NSE	No. Basins with NSE ≤ 0
Full transformer	0.555	0.558	0.619	-0.027	**1**
Transformer Encoder	0.665	0.701	0.734	-0.013	4
LSTM	0.661	0.743	0.739	-0.037	2
$EA-LSTM^a$	**0.67**	0.71			3
$MC-LSTM^b$		0.744		-0.020	
$AR-LSTM^c$		**0.879**	**0.896**	**−0.007**	
$SAC-SMA^b$		0.603		-0.066	
VIC^b		0.551		-0.018	
mHM^b		0.666		-0.040	

Table 4.10: Comparison between different hydrology models.
[a]: Taken from Kratzert et al, 2019 [12]. Of note is that the LSTM model described in the cited paper performs roughly the same as the model described in this paper
[b]: Taken from Hoedt et al, 2021 [9]. Of note is that the LSTM model described in the cited paper performs slightly better than the model described in this paper.
[c]: Taken from Nearing et al, 2022 [18]. Of note is that the LSTM model described in the cited paper performs significantly better than the model described in this paper.

As seen in table 4.10 and figure 4.1, the full transformer performs a bit worse than both the LSTM and the transformer encoder. The encoder however does achieve very similar results to the LSTM, which is exemplified well in the cumulative density graph. This graph also showcases very interesting results for the full transformer, in that its distribution is very different from other models. Other models showcased by the Neuralhydrology team also seem to follow a similar curve to the LSTM and transformer encoder [10]. The fact that the full transformer performs better than the other models for the last 20% seems to indicate that the model becomes specialized in a smaller group of basins, and thus less generalized.

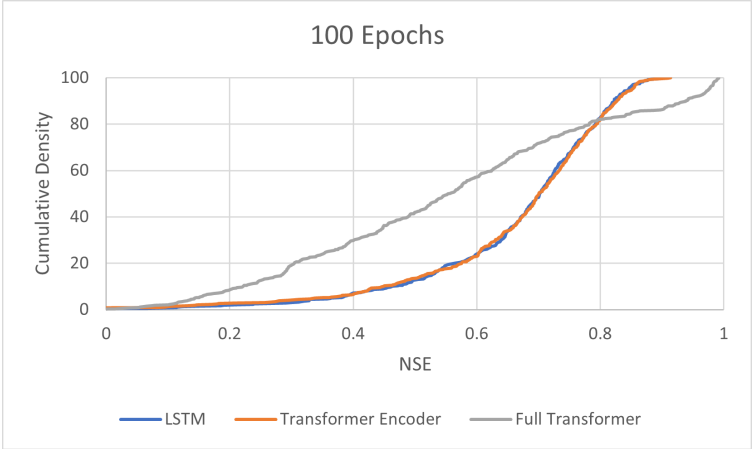

Figure 4.1: Comparison between different hydrology models.

Also worth noting is that the full transformer performs slightly better at the lower end of the curve, having fewer basins with NSE ≤ 0.

4.2.2 Change over time

This section will look at how the three models, the full transformer, transformer encoder, and LSTM, develop over time with different numbers of training epochs. The reason for this is to look for differences in the way the models learn over time.

	25	50	75	100
Full Transformer	0.527	0.548	0.560	0.555
Transformer Encoder	0.640	0.606	0.649	**0.665**
LSTM	**0.656**	**0.644**	**0.659**	0.661

Table 4.11: NSE scores for models trained different numbers of epochs.

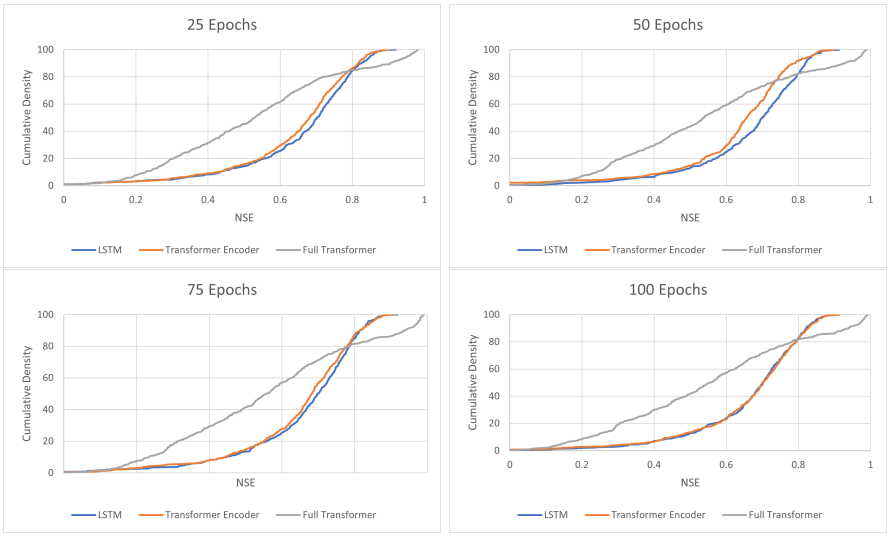

Figure 4.2: NSE scores for models trained different numbers of epochs.

As seen in table 4.11 and figure 4.2, the difference between 25 epochs and 100 is not massive, but there are some interesting findings. The LSTM seems to settle on certain values relatively quickly, whereas the encoder takes some time to settle on top of the LSTM. The difference for the full transformer seems minimal, although it does increase accuracy slightly.

4.2.3 Longer training

This test is to see if a longer training period can increase the performance of the full transformer. The reasoning behind this is that this transformer receives more data than the other two models and may thus require more training.

	Mean NSE	Median NSE	KGE	Beta NSE	Basins with NSE ≤ 0
Transformer 100 Epochs	0.568	0.565	0.616	-0.028	**1**
Transformer 200 Epochs	0.565	0.566	0.633	-0.015	**1**
Transformer 300 Epochs	0.540	0.548	0.627	**−0.012**	7
Encoder 100 Epochs	**0.665**	0.701	0.734	-0.013	4
LSTM	0.661	**0.743**	**0.739**	-0.037	2

Table 4.12: Comparison of transformer models trained for longer.

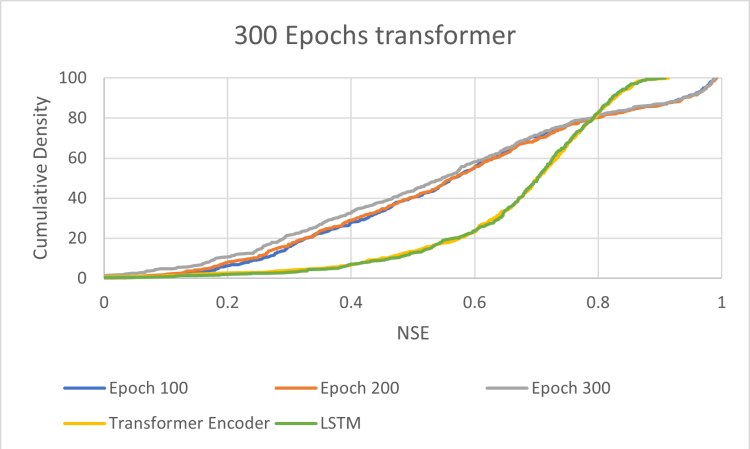

Figure 4.3: Comparison of transformer models trained for longer.

The results of table 4.12 and figure 4.3 show that longer training is not particularly helpful. The model becomes even more specialized and thus performs worse on most basins, without a particularly noticeable improvement on the basins it specializes on. This is useful for future research because it shows that the key to improving accuracy lies in improving the model itself, not training it for longer.

4.2.4 Longer input sequence

This test is to see if a longer input sequence results in better performance for the full transformer model. The reasoning behind this is that reports from the Neuralhydrology team [12] indicate that a longer input sequence, around 250, results in better performance. The hyperparameter testing in section 4.1 indicates that a shorter sequence, around 100, is better, but this result might be affected by the fact the test was not conducted on the full 531 basin set.

	Mean NSE	Median NSE	KGE	Beta NSE	Basins with NSE ≤ 0
Transformer 100 seq length	0.555	0.558	0.619	-0.027	**1**
Transformer 250 seq length	0.553	0.548	0.590	-0.031	**1**
Encoder 100 seq length	**0.665**	0.701	0.734	**−0.013**	4
LSTM 100 seq length	0.661	**0.743**	**0.739**	-0.037	2

Table 4.13: Comparison of transformer models trained different sequence lengths.

Table 4.13 and figure 4.4 confirms the hyperparameter testing, that the longer sequence length does not increase accuracy, and in fact, makes it marginally worse. The longer

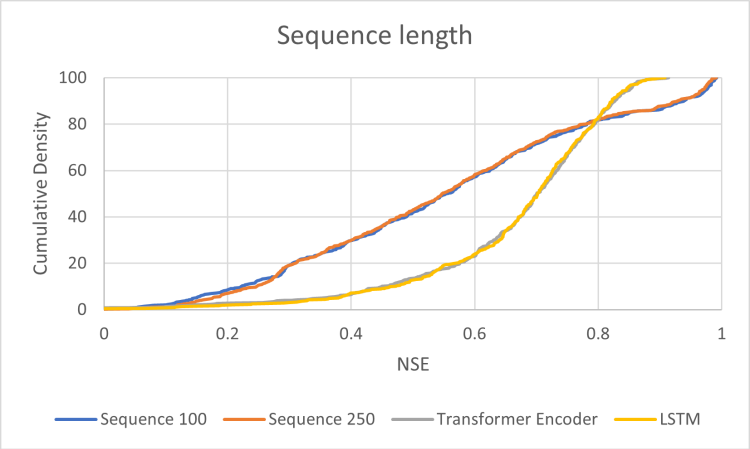

Figure 4.4: Comparison of transformer models trained different sequence lengths.

sequence does slightly increase performance on the lower part of the density graph, but the reduced performance on the rest of the basins makes the overall performance worse. The reason for this finding conflicting with previous research may be that the full transformer model is, unlike the LSTM, more suited for shorter sequences. This may be because the LSTM receives all inputs sequentially whereas the transformer has access to all previous inputs at the same time.

4.2.5 Changed target preprocessing

The target input sequence for the transformer decoder has to be scaled up to the same dimensions as the source input sequence for the two to be multiplied in the transformer model. There are two main ways to do this, the first is to use a linear machine learning layer, and the second is to manually duplicate the elements of the sequence to the correct dimensions. The former is the default implementation of the full transformer model, and so this test is based on the hypothesis that perhaps this linear layer adds unnecessary noise to the model.

	Mean NSE	Median NSE	KGE	Beta NSE	Basins with NSE ≤ 0
Transformer Linear Exp	0.555	0.558	0.619	-0.027	**1**
Transformer Manual Exp	0.562	0.561	0.621	-0.032	2
Transformer Encoder	**0.665**	0.701	0.734	**−0.013**	4
LSTM	0.661	**0.743**	**0.739**	-0.037	2

Table 4.14: Comparison of transformer models trained different methods for expanding the target sequence.

The results of table 4.14 and figure 4.5 show that the performance of the model with a manual expansion is slightly better than the model with a linear layer, however, the difference is not large. This indicates that to some degree the hypothesis that the linear layer makes the data more noisy may be true, but it does not account for the difference between the transformer and the LSTM.

4.2.6 Mask removal

The performance of the full transformer model has thus far been shown to be noticeably poorer than that of both the LSTM and the transformer encoder. This is in stark contrast to the RR-Former by Yin et al, 2022 [22], which boasted a better performance than the LSTM used for comparison, also on the CAMELS dataset. The main difference between the RR-Former and the one used in this paper is in terms of masking, where the transformer encoder

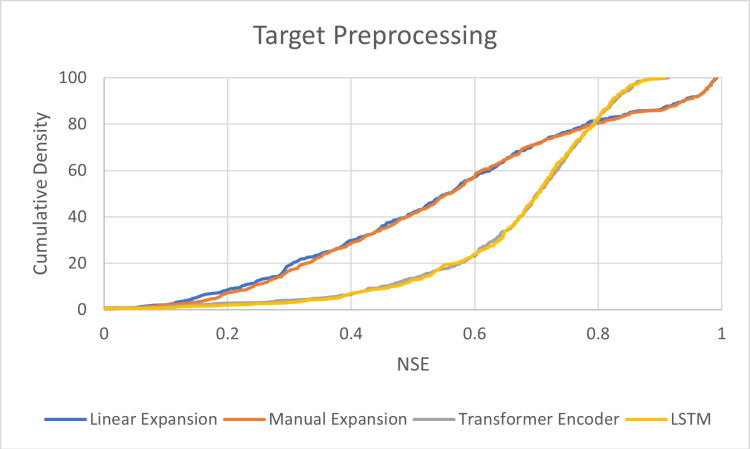

Figure 4.5: Comparison of transformer models trained different methods for expanding the target sequence.

used no masks and the mask for the transformer decoder was only applied to the predicted elements. This test is to see if this masking method makes a difference in performance by removing the mask for both the encoder and decoder. The reason why the mask is removed from the decoder is because unlike the RR-Former the target input sequence is right-shifted to remove the predicted elements entirely.

	Mean NSE	Median NSE	KGE	Beta NSE	Basins with NSE ≤ 0
Masked Transformer	0.562	0.561	0.621	-0.032	**2**
No Mask Transformer	0.534	0.511	0.573	-0.014	**2**
Transformer Encoder	**0.665**	0.701	0.734	**−0.013**	4
LSTM	0.661	**0.743**	**0.739**	-0.037	**2**

Table 4.15: Comparison of transformer models trained different masking options.

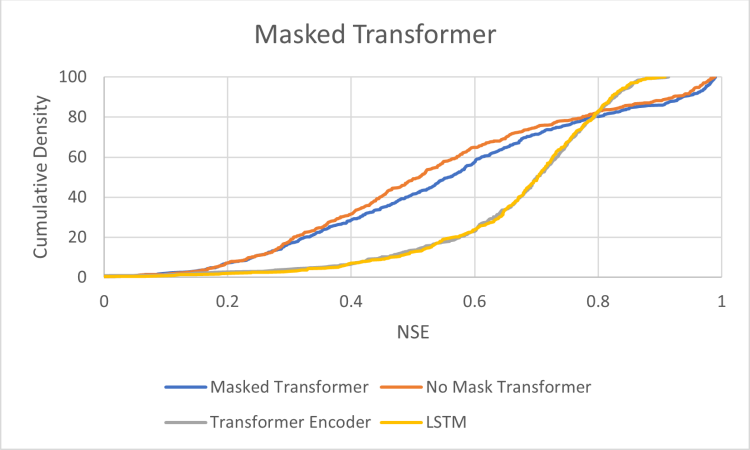

Figure 4.6: Comparison of transformer models trained different masking options.

The results of table 4.15 and figure 4.6 once again contradict Yin et al, 2022 [22] since the performance of the unmasked transformer is noticeably worse than the masked transformer throughout the entire distribution. This is backed up theoretically by the fact that the model having access to future values when predicting streamflow may simply add unnecessary noise rather than useful data. Further testing on this topic would require access to the RR-Former code which is not currently available.

4.2.7 Reset parameters

The transformer encoder created by the Neuralhydrology team contained a function to reset the model parameters to initialize the weights and biases. When creating the full transformer model this feature was accidentally omitted. This test is to see if resetting the parameters when initializing the model makes a difference. Since this was the final test all models were trained twice to verify that their relationships remain the same.

	Mean NSE	Median NSE	KGE	Beta NSE	Basins with NSE ≤ 0
Transformer No Reset	0.555	0.558	0.619	-0.027	**1**
Transformer Reset	0.550	0.552	0.582	-0.024	2
Transformer Encoder	**0.665**	0.701	0.734	**−0.013**	4
LSTM	0.661	**0.743**	**0.739**	-0.037	2

Table 4.16: Comparison of transformer models that reset or don't reset starting parameters.

	Mean NSE	Median NSE	KGE	Beta NSE	Basins with NSE ≤ 0
Transformer No Reset	0.565	0.569	0.627	**−0.020**	**2**
Transformer Reset	0.551	0.550	0.613	-0.023	**2**
Transformer Encoder	0.632	0.699	**0.740**	-0.022	7
LSTM	**0.665**	**0.702**	0.720	-0.046	3

Table 4.17: 2nd comparison of transformer models that reset or don't reset starting parameters.

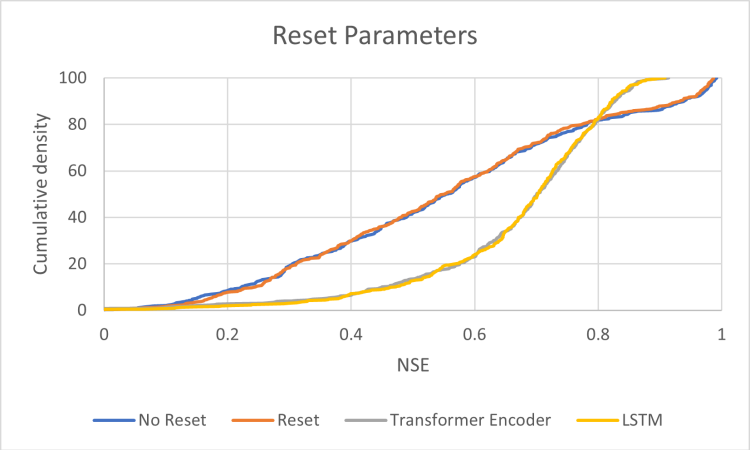

Figure 4.7: Comparison of transformer models that reset or don't reset starting parameters.

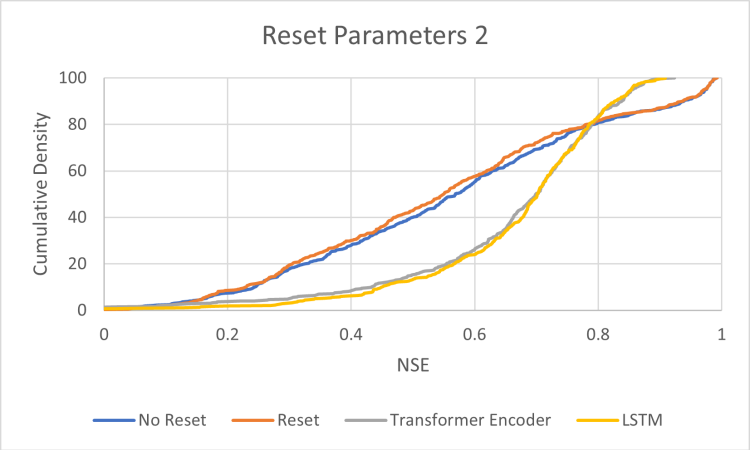

Figure 4.8: 2nd comparison of transformer models that reset or don't reset starting parameters.

From table 4.16 and figure 4.7, we can see that resetting parameters does not improve performance, in fact, the performance is slightly worse. Table 4.17 and figure 4.8 make this even more clear with an even bigger difference between the two. This does not necessarily prove that resetting the weights is a bad idea, but it does disprove the hypothesis that resetting the weights is vital to model accuracy. The second model set also showcases a slight shift in the relationship between the transformer encoder and the LSTM, with the LSTM performing slightly better and the encoder performing slightly worse. This indicates that the two models are close enough in accuracy that it may be a toss-up which one is more accurate.

Chapter 5

Discussion and Future work

The results of this study are very interesting, and it is clear more testing is certainly required. The transformer encoder performs very similarly to the LSTM model, even being able to sometimes exceed it. The LSTM does perform better on a more consistent basis, but the transformer encoder has the potential to be able to outperform the LSTM with the correct model parameters.

The full transformer is a lot more complicated. The model is not able to beat the LSTM model, or even really come within the margin of error, but the large differences shown in the cumulative density graphs nonetheless show that this model can provide an interesting field of study. The model seems to be significantly better than other models at specialization while suffering when it comes to generalization. This does mean however that if there was a way to adapt the model and/or data to play to this strength then the model may have the potential to outperform the LSTM

5.1 Basin grouping

As discussed the full transformer model seems to be better at specialization. This could make sense as there are noticeable different patterns within the streamflow data, as indicated by figure 5.1.

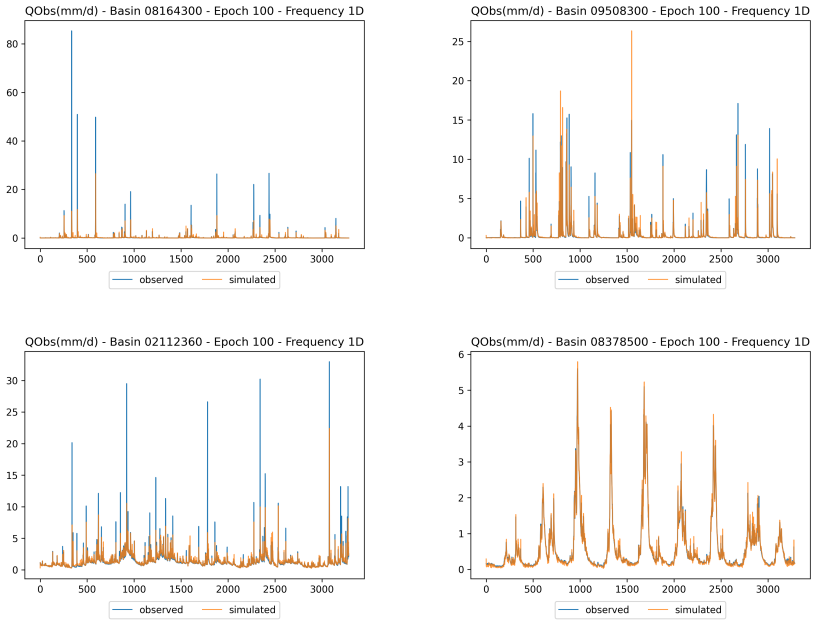

Figure 5.1: Streamflow for different basins

All streamflow values are from the same period, and from figure 5.1 we can see that the variations between basin streamflow patterns are significant. The models should in theory be able to separate these different basin categories using the static attributes provided as input, however, adding an explicit categorization dimension could benefit the full transformer model, or maybe even the other models as well. This could be done either as part of the preprocessing or as a part of the model itself.

5.2 EA-Transformer

One of the models created by the Neuralhydrology team is the Entity Aware LSTM (EA-LSTM) [12]. The way this model works is by separating the static attributes of the input from the dynamic attributes, and explicitly using the static attributes as the activation for the input gate. This method may be replicated for the transformer scaled dot-product attention mechanism by using the static attributes as the key matrix and the dynamic attributes as the query and value matrices, creating an Entity Aware Transformer (EA-Transformer). Worth noting is that the EA-LSTM did not outperform the standard LSTM model, however attempting to create an EA-Transformer might still be a direction worth pursuing.

Chapter 6

Conclusion

The purpose of this paper was to implement a full transformer model inside the Neuralhydrology library. The study would then try to determine the effect of various hyperparameters as well as evaluate the performance of both the full transformer model and the existing transformer encoder compared to other hydrology models.

The transformer model was implemented using PyTorch and the preprocessing was adapted to suit this model. This involves right-shifting the target inputs, embedding the inputs, and adding positional encoding. The hyperparameters were tested on a limited basin set, and finally, the models were trained on a much larger dataset. This also included testing various implementations of the models.

The results show that when it comes to hyperparameter testing, there are certain parameters, such as the size of the embedding hiddens and the number of transformer layers, have a large impact on accuracy, whereas others, such as the feedforward dimension and the sequence length, have only marginal effects. The results on the larger basin set with multiple models show that the full transformer model performs worse than the LSTM and transformer encoder, but the cumulative density graphs show that the full transformer has a very different distribution than other models. It seems to perform better at the top 20% of basins but worse at the rest. This indicates it becomes better at specialization at the cost of generalization. The transformer encoder, however, seems to achieve very comparable results to the LSTM.

These results show potential for future work within this field of research. The transformer encoder is already very comparable to the LSTM, so further study should be carried out to examine if different model configurations can bring the results to outperform the LSTM. When it comes to the full transformer, the distance is larger for it to beat the state-of-the-art, however, the seeming ability of the model to specialize might be an advantage. Attempting to take advantage of this feature by adding an explicit categorization dimension might be a meaningful direction of research.

Appendix A

Models and results

The model code, configuration files and results for this paper are available at the following github location: https://github.com/JonatanHindersland/neuralhydrology

Bibliography

[1] N. Addor et al. "The CAMELS data set: catchment attributes and meteorology for large-sample studies." In: *Hydrology and Earth System Sciences* 21.10 (2017), pp. 5293–5313. DOI: 10.5194/hess-21-5293-2017. URL: https://hess.copernicus.org/articles/21/5293/2017/.

[2] Amobichukwu C. Amanambu, Joann Mossa, and Yin-Hsuen Chen. "Hydrological Drought Forecasting Using a Deep Transformer Model." In: *Water* 14.22 (2022). ISSN: 2073-4441. DOI: 10.3390/w14223611. URL: https://www.mdpi.com/2073-4441/14/22/3611.

[3] Robert JC Burnash and R Larry Ferral. *A generalized streamflow simulation system: Conceptual modeling for digital computers*. US Department of Commerce, National Weather Service, and State of California ..., 1973.

[4] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[5] Wikimedia Commons. *LSTM*. File:LSTM Cell.svg. 2021. URL: https://commons.wikimedia.org/wiki/File:LSTM_Cell.svg.

[6] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[7] M. Gauch et al. "Rainfall–runoff prediction at multiple timescales with a single Long Short-Term Memory network." In: *Hydrology and Earth System Sciences* 25.4 (2021), pp. 2045–2062. DOI: 10.5194/hess-25-2045-2021. URL: https://hess.copernicus.org/articles/25/2045/2021/.

[8] Hoshin V. Gupta et al. "Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling." In: *Journal of Hydrology* 377.1 (2009), pp. 80–91. ISSN: 0022-1694. DOI: https://doi.org/10.1016/j.jhydrol.2009.08.003. URL: https://www.sciencedirect.com/science/article/pii/S0022169409004843.

[9] Pieter-Jan Hoedt et al. "MC-LSTM: Mass-Conserving LSTM." In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 4275–4286. URL: https://proceedings.mlr.press/v139/hoedt21a.html.

[10] F. Kratzert et al. "A note on leveraging synergy in multiple meteorological data sets with deep learning for rainfall–runoff modeling." In: *Hydrology and Earth System Sciences* 25.5 (2021), pp. 2685–2703. DOI: 10.5194/hess-25-2685-2021.

[11] F. Kratzert et al. "Rainfall–runoff modelling using Long Short-Term Memory (LSTM) networks." In: *Hydrology and Earth System Sciences* 22.11 (2018), pp. 6005–6022. DOI: 10.5194/hess-22-6005-2018. URL: https://hess.copernicus.org/articles/22/6005/2018/.

[12] F. Kratzert et al. "Towards learning universal, regional, and local hydrological behaviors via machine learning applied to large-sample datasets." In: *Hydrology and Earth System Sciences* 23.12 (2019), pp. 5089–5110. DOI: 10.5194/hess-23-5089-2019. URL: https://www.hydrol-earth-syst-sci.net/23/5089/2019/.

[13] Frederik Kratzert et al. "NeuralHydrology — A Python library for Deep Learning research in hydrology." In: *Journal of Open Source Software* 7.71 (2022), p. 4050. DOI: 10.21105/joss.04050. URL: https://doi.org/10.21105/joss.04050.

[14] Mathias Lechner and Ramin M. Hasani. "Learning Long-Term Dependencies in Irregularly-Sampled Time Series." In: *CoRR* abs/2006.04418 (2020). arXiv: 2006.04418. URL: https://arxiv.org/abs/2006.04418.

[15] Xu Liang et al. "A simple hydrologically based model of land surface water and energy fluxes for general circulation models." In: *Journal of Geophysical Research: Atmospheres* 99.D7 (1994), pp. 14415–14428. DOI: https://doi.org/10.1029/94JD00483. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/94JD00483. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/94JD00483.

[16] Samuel Lynn-Evans. *How to code the Transformer in Pytorch*. Oct. 2018. URL: https://towardsdatascience.com/how-to-code-the-transformer-in-pytorch-24db27c8f9ec.

[17] J.E. Nash and J.V. Sutcliffe. "River flow forecasting through conceptual models part I — A discussion of principles." In: *Journal of Hydrology* 10.3 (1970), pp. 282–290. ISSN: 0022-1694. DOI: https://doi.org/10.1016/0022-1694(70)90255-6. URL: https://www.sciencedirect.com/science/article/pii/0022169470902556.

[18] G. S. Nearing et al. "Technical note: Data assimilation and autoregression for using near-real-time streamflow observations in long short-term memory networks." In: *Hydrology and Earth System Sciences* 26.21 (2022), pp. 5493–5513. DOI: 10.5194/hess-26-5493-2022. URL: https://hess.copernicus.org/articles/26/5493/2022/.

[19] A. J. Newman et al. "Development of a large-sample watershed-scale hydrometeorological data set for the contiguous USA: data set characteristics and assessment of regional variability in hydrologic model performance." In: *Hydrology and Earth System Sciences* 19.1 (2015), pp. 209–223. DOI: 10.5194/hess-19-209-2015. URL: https://hess.copernicus.org/articles/19/209/2015/.

[20] Luis Samaniego, Rohini Kumar, and Sabine Attinger. "Multiscale parameter regionalization of a grid-based hydrologic model at the mesoscale." In: *Water Resources Research* 46.5 (2010). DOI: https://doi.org/10.1029/2008WR007327. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2008WR007327. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2008WR007327.

[21] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[22] Hanlin Yin et al. "RR-Former: Rainfall-runoff modeling based on Transformer." In: *Journal of Hydrology* 609 (2022), p. 127781. ISSN: 0022-1694. DOI: https://doi.org/10.1016/j.jhydrol.2022.127781. URL: https://www.sciencedirect.com/science/article/pii/S0022169422003560.