LovelyLlamas

Common Paper

Philipp Donn, Niels Hannemann, Franko Jolic

March 16, 2025

1 Introduction

With the huge growth of machine learning in recent years, a lot of advances have been made in almost all fields of computer science, including computer vision, speech recognition, and natural language processing (NLP). Especially, NLP has seen impressive progress through large-scale transformer models such as GPT-4 and LLaMA-2. These state-of-the-art models have striking capabilities in the fields of text generation, reasoning, and multilingual understanding [15, 9].

However, real-world applications often come with computational constraints. Large models require a lot of processing power and thus have a high energy consumption, making them unfeasible in environments with limited resources. Optimization in machine learning is not just about building bigger models, but also about designing efficient architectures that still achieve a strong performance.

Our project addresses this challenge by using small, lightweight transformer models and a custom LSTM architecture to generate natural language weather reports in English. The goal of this project was to provide a web interface to view the generated weather reports. Additionally, the models should be able to run on a laptop and ideally also be trained on laptops. By applying optimizations specific for the domain and the architectures, we tried to find two models that balance accuracy and efficiency to fit this constraint. This is especially interesting because it demonstrates how NLP techniques can be used for resource-limited scenarios.

1.1 Research Questions

Before starting our project, we stated three research questions to guide our approach:

1. **Is it possible to generate weather reports with a small LLM?**
   Given the clear structure of weather data and weather reports, we hypothesized that a lightweight language model should be able to generate accurate weather reports without requiring a lot of parameters.

2. **How much data is needed, and how large should the model be?**
   The focus of this project should be the trade-off between dataset size, model size and model capacity. We aim to find the minimal dataset and model size required to achieve well-written weather reports.

3. **Can the model produce creative and informal reports with minimal effort?**
   While traditional weather reports are often standardized and follow a certain structure, we also wanted to explore whether a lightweight model could generate more engaging outputs, focusing on a more natural language.

1.2 Outline

This paper will first describe the data acquisition process and the obtained datasets. The two datasets will be analyzed and compared to each other. In the following, the general pipeline of the experiments will be explained. This includes the two main architectures, the tokenizers, the training routine and the metrics used to evaluate the models. At last, the results of the conducted experiments are presented. This section focuses on what changes had the biggest influence and how the two architectures compare to each other performance-wise.

2 Dataset

2.1 Data Acquisition

For the data acquisition, we decided to use the website wetter.com [4]. It provides weather reports (shorter and longer versions) as well as matching weather data (temperature, rainfall, etc.), which will be explained in the next section. Our first challenge was gathering a good dataset. We did not know back then that this was a challenge that would follow us until the end of the project.

At the start, we had difficulties cleaning the data from wetter.com from nested structures, repetitive data and all kinds of weird separators and symbols. The amount of variance between the declarations of two table objects was astonishing. So, to combat this, we developed a scraping class that handled the discrepancies through regex matching and HTML-element iteration. After managing to scrape a few data points over a week, we noticed that the amount would not suffice and that our data did not have a good global spread. Instead of handpicking locations for our dataset, we found a way to automatically scrape through the link tree of wetter.com and gather data from all locations available on wetter.com. After sifting through some data and noticing that most of it was useless, we managed to reduce our location count to roughly 28 000. We also implemented a multithreaded worker queue with a termination rule (timetable not starting at midnight) and cross-checking against existent files to overcome the issue of different time zones and locations.

2.2 What does the data look like?

For each location, the collected data consists of two parts: a textual weather report in German and a set of attributes describing the weather throughout the entire day. The weather reports are very structured, i. e. they consist of up to four sentences, and each sentence describes a certain time of day. The first sentence is usually concerned with the morning, the second sentence with the midday / afternoon, the third sentence with the evening, and the fourth sentence with the night. The number of sentences is not fixed, though, as it greatly depends on how unstable the prevailing weather is. For example, a weather report from a desertlike location where it is hot and sunny all day will likely deviate from the described pattern and summarize certain times of day in just one sentence. The sentences themselves are highly structured as well. They provide some general weather information, such as whether it is going to be be sunny, rainy, or snowy, whether there will be clouds, fog, etc., together with a temperature value or range. Figure 1 provides an example for a four-sentence long weather report.

> In Carroll ist es morgens bedeckt bei Werten von -13°C. Mittags wird die Sonne von einzelnen Wolken verdeckt bei Höchstwerten von -8°C. Am Abend ist es in Carroll wolkenlos bei Temperaturen von -11 bis -9°C. Nachts gibt es keine Wolken und die Sterne sind klar zu erkennen bei einer Temperatur von -12°C.

Figure 1: Example for a weather report, here for the city of Carroll, USA. The sentences are color-coded: Morning | Midday / Afternoon | Evening | Night

In total, there are seven attributes that describe the weather of a location (see Table 1). For each attribute and location, there are 24 values that span from midnight to midnight, i. e. each attribute value describes a period of one hour.

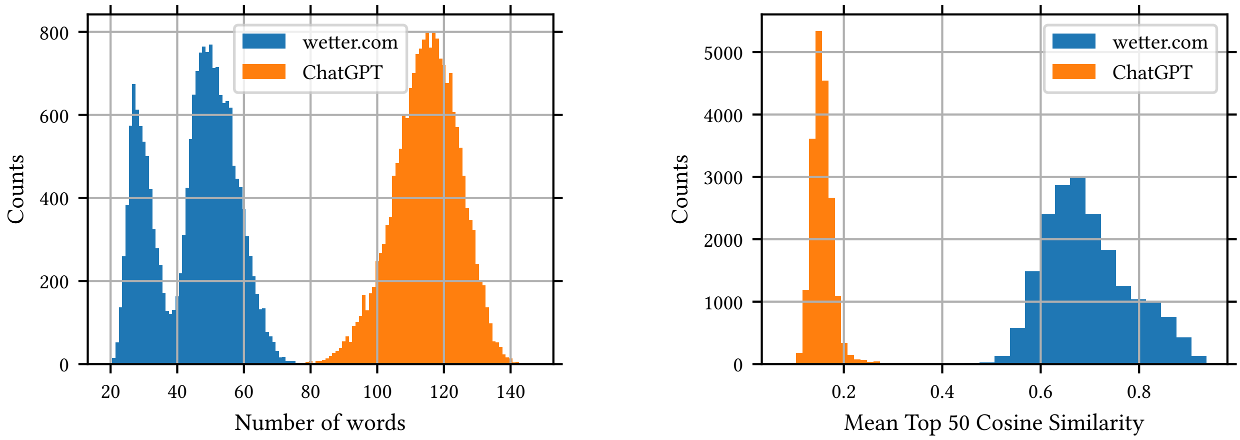
Table 1: Weather attributes available for each location and collected from wetter.com.

Attribute Name	Type	Description
Clearness	String	A string describing the weather in a few words, such as “covered”, “fog”, “stormy”, “light drizzle”, etc.
Temperature	Number	Temperature in °C
Risk of Precipitation	Number	Risk of precipitation in %
Amount of Precipitation	Number	Amount of precipitation in l/m^2
Wind Direction	String	The wind direction, such as “NW” for wind from the northwest
Wind Speed	Number	Wind speed in km/h
Cloudiness	Number	A number from 0 to 8 encoding the cloudiness in terms of eights, such as 8/8 for a fully closed cloud layer

2.3 The Dataset

The latest dataset was collected on December 12, 2024. It contains 19 362 data points that are distributed throughout the world, as shown in Figure 7 in Appendix A. There is an obvious bias towards Europe, West Africa and the Americas. However, since both the northern and southern hemispheres are sufficiently covered, there should be enough variety in the dataset with respect to warm and cold places, as well as with respect to places with little and much precipitation. Nevertheless, since the dataset was collected on a single day in December, it obviously does not contain all seasons and thus cannot be considered exhaustive. For example, when we compared this dataset to another smaller dataset collected in November 2024, we noticed that the set of *Clearness* attribute values is neither identical in both datasets nor is the set of *Clearness* attribute values of the smaller dataset fully contained in the set of *Clearness* attribute values of the December dataset, and vice versa.

On average, the weather reports are roughly three sentences long and contain 44.9 words. The language is rather narrow, with only 205 unique words (neglecting all numbers), hinting that the weather reports are fairly repetitive and that there is little variety in the dataset. In order to quantify this impression, Figure 2b shows the distribution of the mean top 50 cosine similarity of the dataset, i. e. for each weather report, the average cosine similarity of the 50 most similar weather reports (excluding the report itself) is calculated based on TF-IDF vectors. With an average mean top 50 cosine similarity of 0.69, it becomes obvious that the weather reports are highly repetitive.



(a) Distribution of the length of the reports

(b) Distribution of the similarity of the reports

Figure 2: Some statistics of the wetter.com and the ChatGPT datasets. The cosine similarity was computed using TF-IDF vectors. The weather reports in the wetter.com dataset are much shorter and considerably more repetitive than the reports in the ChatGPT dataset.

The weather data are almost complete; less than 0.3 % of the data is missing. This very low number of missing values does not really justify a sophisticated treatment of these values. Hence, they are accepted for now and will be dealt with later in the tokenization step. Overall, the data are more or less clean, with only minor, easy-to-fix issues such as misspelled city names or random numbers added to the wind directions.

2.4 Rewriting the Weather Reports with ChatGPT

Because the original weather reports collected from wetter.com are very repetitive and thus uncreative, we tried to spice them up. Initially, our plan was to create funny weather reports that included jokes using GPT-4o mini. We did not impose a lot of bounds on the to-be-generated reports because we did not want to restrict ChatGPT's creativity too much. Although the initial results looked very promising (see Appendix G.1), an EDA revealed many problems. These problems ranged from encoding issues, over ChatGPT trying to interpret the name of the place and thus including country information (it often failed in doing so, though), to really weird model behavior. For example, ChatGPT showed some affection for the Russian and some Arabic languages. From time to time, it switched alphabets for single words and also produced curiosities like these:

[...], also ziehen Sie Ihre зимние (ja, das ist Russisch für "Winter-") Socken an, [...]

This was not the biggest problem, though. ChatGPT also showed a strong tendency towards creating a lot of compound words. It made use of this characteristic of the German language much more than humans would normally do. Needless to say, this behavior eventually blew up the vocabulary of the dataset, with more than 60 000 words and almost two out of three words appearing only once in the entire dataset.

In a second attempt, we set ChatGPT far tighter bounds by deriving restrictions from the results and problems of our first attempt. These restrictions were intended to help with two things, namely reducing the effort required to clean the rewritten reports and preventing ChatGPT from using a too vast vocabulary. The full list of restrictions and an example can be found in Appendix G.2. Before creating the second dataset, we ran small-scale tests with varying moods as a target to assess the influence of the restrictions on ChatGPT's creativity. We tested three different moods and asked ChatGPT to create funny, informal and apocalyptic-sounding weather reports given the restrictions. The tests revealed that our restrictions take away a huge amount of creativity from the generated weather reports. In general, all three target moods produced well-sounding weather reports with the respective mood, but in particular the funny weather reports lacked that certain indefinable something that made the reports from our first attempt stand out in comparison to the original weather reports from wetter.com. We therefore changed our goal to creating gloomy and apocalyptic-sounding variants of the original weather reports.

By running the weather reports through ChatGPT, the statistics of the weather report dataset changed significantly. The reports are now on average 114.7 words long with a mean of 10.9 sentences; no report is shorter than eight sentences, and one even reaches 25 sentences at 131 words. Figure 2a illustrates this shift to longer reports graphically. It is also interesting to note that the sentences got a little shorter: in the original weather reports, there are on average 15.2 words per sentence, whereas this number decreased to 11.0 words per sentence in the ChatGPT dataset. The vocabulary grew noticeably to 16 262 unique words, albeit this number is still significantly smaller than in the first attempt. Last but not least, the average mean top 50 cosine similarity plummeted from 0.69 to merely 0.16 (see Figure 2b). Although the increased length of the weather reports and the bigger vocabulary most likely both play a role in this drastic change, it shows that we somewhat went from one extreme to the other, i. e. the weather reports in the ChatGPT dataset now only show very little commonality and repetitiveness. In hindsight, the restrictions therefore proved to be effective with respect to reducing the post-processing effort, but they only partly helped with the vocabulary problem.

3 Models and Training

Throughout the semester, we mainly experimented with two different types of models: a transformer and an LSTM.

3.1 Transformer

For the transformer model, we decided to start with the original transformer [16] because it is the by far best documented transformer and none of us had ever worked with transformers. However, we quickly ran into the problem of vanishing gradients. This manifested itself by the fact that, depending on the model configuration, the models either did not learn to generate weather reports at all or learned to generate weather reports but completely ignored the context provided by the encoder. The solution to this problem was simple, though. Switching from the post-LN configuration to the pre-LN configuration solved all our problems. Perhaps this problem would have never occurred if we had known about the fact that post-LN transformers need a dedicated warm-up phase, whereas pre-LN transformers do not need such a phase [17]. In the end, the transformer was producing very good results, so we stuck to the model and rather experimented with different positional embeddings (Appendix C), tokenizers, etc. instead of trying a more recent transformer model.

3.2 LSTM

Initially, the LSTM was implemented with a completely different training routine. However, this approach showed problems during training, and the model failed to generate any reasonable outputs. To combat this and keep the code more consistent, we adjusted and built the LSTM architecture to fit the existing transformer training routine. This change allowed the model to finally generate reasonable prompts. It also allowed us to test out further optimizations as well as parameter configurations. Unfortunately, we were not able to find out why the old training routine ran without errors but did not produce any useful outputs. To improve the performance of the architecture, the LSTMs also utilize Bahdanau attention [1]. This allows the models to focus on the most important parts of the input.

3.3 GRU

In comparison to the transformer and LSTM, the GRU was written for the sole purpose of generating long reports scraped from wetter.com. Its initial pipeline differed as it needed to incorporate features irrelevant to short report generation, and named entity filtering. Later on, an effort was made to generate the stylized ChatGPT short reports. In total, this made for the creation of three GRU models (basic, attention, advanced) for two task types (long report and stylized short report generation) with individual pipelines. This endeavor, while being short, created good results that are not comparable to the other two models in most metrics, due to time constraints in development as well as key differences in the utilized pipelines. Therefore, results for the GRU will only be discussed in Section 4.8 and its pipelines will not be part of this paper.

3.4 Network Inputs

Both the transformer and the LSTM use an encoder-decoder architecture. While the decoder is fed with snippets taken from the weather reports (each snippet is 20 tokens long), the encoder is fed with the weather data. For that reason, the weather data are written in a sequence of the following form:

00 - 01 Uhr;<Clearness>;<Temperature>;<Risk of Precipitation>;<Amount of Precipitation>; <Wind Direction>;<Wind Speed>;<Cloudiness>;01 - 02 Uhr;<Clearness>;<Temperature>;<Risk of Precipitation>;<Amount of Precipitation>;<Wind Direction>;<Wind Speed>;<Cloudiness>;02 - 03 Uhr;...

Here, each <...> indicates the attribute value in the respective period of time. Subsequently, the sequence is tokenized and fed into the encoder. Missing values are taken care of by inserting a special `<missing>` value into the sequence at the respective positions.

3.5 Tokenizers

We tested and compared two types of tokenizers. First, we implemented a custom word-level tokenizer – in the following referred to as “Set of Words” tokenizer (SoW) – and second, we used a prebuild subword-level BERT tokenizer [3] – in the following referred to as “Bert” tokenizer. While we used the Bert tokenizer only to encode the weather reports, the SoW tokenizer was used for both encoding the weather data and the weather reports. We exclusively tokenized the encoder input with the SoW tokenizer because it has the advantage of having an encoder input of constant length. Moreover, there is no real benefit from using a subword-level tokenizer for the weather data because the sets of *Clearness* attribute values and *Wind Direction* attribute values are small enough to tokenize them directly on the value level.

For both the encoder and the decoder, we calculated the token sets based on the entire dataset. This yields the smallest possible token set while still being large enough to tokenize all data. For the encoder, the token set comprises 1260 tokens. The size of this set is driven by the amount of numbers contained in the weather data since there are only 71 non-numeric values encoded in the token set and 3 special tokens. For the decoder, the SoW token set comprises 282 tokens and the Bert token set comprises 302 tokens on the wetter.com dataset (neglecting special tokens, such as the start-of-sequence token). It is interesting to note that while the set of decoder tokens for both tokenizers is almost the same in size, they differ significantly in terms of what tokens are contained: their Intersection over Union (IoU) is only 0.49.

As a side note, we did not tokenize the names of the places in the weather reports individually. Instead, we replaced the names with a common `<city>` tag before tokenizing the weather report. Otherwise, the token sets would be significantly larger, and the models might depend on the name of the place when generating weather reports.

3.6 Training Setup

The entire pipeline was implemented in PyTorch [11]. All models were trained for up to 100 epochs, using early stopping to terminate the training early if the evaluation loss did not improve over a period of ten epochs. The models were optimized using AdamW [8] and the learning rate was halved if the evaluation loss did not improve over a period of five epochs. Despite the “train on a laptop” constraint, all models were trained on a GPU. The transformers were trained on an Nvidia GTX Titan X and the LSTMs on an Nvidia GTX 1660 Super. These GPUs are either old (in terms of the Titan X) or entry-level cards (in terms of the 1660 Super). Hence, the “train on a laptop” constraint was not strictly met, but also no top-of-the-line hardware was utilized to get as close as possible to the “train on a laptop” constraint without actually having to train the models on a laptop CPU.

3.7 Metrics

We experimented with well-established metrics like Rouge [7], Bleu [10] and BERTScore [18] during the test phase. However, none of the three metrics produced convincing results. While Rouge and Bleu, given their nature as n-gram matching metrics, are unable to capture the context of words, BERTScore proved to be too insensitive. In addition, none of the three metrics can assess the correctness of the generated weather reports, even though BERTScore uses contextualized word embeddings to compute the score.

To address the problem of assessing the correctness of the generated weather reports, and inspired by the GAN setting [5], we trained a custom classifier on deciding whether the reports are good or bad. The classifier is based on the transformer architecture, which we turned into a classifier network by adjusting the output dimension to one and adding a sigmoid activation function in the end, i. e. it outputs a score between 0 (bad) and 1 (good) for each input sequence. To train the classifier, we paired each weather report with a random weather report from the dataset every epoch. Thus, given a set of weather data attributes, the original weather report is assigned a class label of 1, whereas the randomly selected weather report is assigned a class label of 0. Similar to the generative models that operate on sequences of a finite block size, the classifier does the same and the mean of the resulting set of scores is then taken as the final score for the entire weather report. Table 2 gives an example for the performance of the custom classifier. In fact, we actually trained two classifiers $\mathrm{CC}_{\mathrm{Full}}$ and $\mathrm{CC}_{\mathrm{CT}}$ (both are based on the Medium transformer, see Appendix B) that take different sets of weather data attributes into account. $\mathrm{CC}_{\mathrm{Full}}$ has access to all weather data attributes, whereas $\mathrm{CC}_{\mathrm{CT}}$ can only access the *Clearness* and the *Temperature* attributes. One can see that the classifiers consider the correctness of the weather report both in terms of the clearness and the temperature, while being slightly more susceptible to the temperature attribute.

Table 2: Example for the performance of the custom classifiers. The top row represents the original weather report, and rows two and three are changed, incorrect versions of it. Rouge, Bleu and BERTScore still yield relatively high scores for the two incorrect versions, and their reduced scores mainly stem from the fact that the words have changed. The two custom classifiers $\mathrm{CC}_{\mathrm{Full}}$ and $\mathrm{CC}_{\mathrm{CT}}$, on the other hand, are more sensitive to the correctness of the report and their scores decrease significantly.

Sample	Rouge-L	Bleu	BERTScore F1	$\mathrm{CC}_{\mathrm{Full}}$	$\mathrm{CC}_{\mathrm{CT}}$
In <city> gibt es den ganzen Tag einen wolkenlosen Himmel bei Temperaturen von 5 bis 21°C	1.0	1.0	1.0	0.82	1.0
In <city> gibt es den ganzen Tag einen wolkenlosen Himmel bei Temperaturen von **0** bis **4**°C	0.88	0.77	0.98	0.00	0.00
In <city> ist es den ganzen Tag **bewölkt und regnerisch** bei Temperaturen von 5 bis 21°C	0.74	0.62	0.89	0.01	0.29

During EDA, we observed that some weather reports state temperature values that are slightly different from the values contained in the respective weather data (such values are called "ghost values" in the following). For example, the report in Figure 1 states a nighttime low of $-12\,^{\circ}$C; the corresponding weather data however only goes as low as $-11\,^{\circ}$C for that night. In order to quantify to what extent the trained models have learned to put ghost values in their generated weather reports, we came up with two metrics specifically targeting this problem:

- $\mathrm{Temp}_{\mathrm{Ghost}}$: This metric quantifies the fraction of non-ghost values. In other words, the number of stated and in the weather data contained temperature values is divided by the total number of stated temperature values for a given dataset. It is 1 if no ghost values appear and 0 if all stated temperature values are ghost values. This metric does not address the situation in which the stated temperature values are contained in the weather data but at a different time of day.

- $\mathrm{Temp}_{\mathrm{Overlap}}$: This metric measures the overlap between the stated temperature range of a weather report and the temperature range according to the weather data. It is similar to the IoU score, but since weather reports are not required to state the minimum and maximum temperatures of a day, the intersection is compared to the temperature range of the weather report instead of the union of both ranges. Hence, the score will be 1 if the stated temperature range is fully contained in the range given by the weather data, 0 if both intervals do not intersect, and between 0 and 1 if they partially overlap. Hence, this metric measures "the position" of the stated temperature range with respect to the temperature range given by the weather data, i. e. whether it is fully contained (the optimum) or whether the interval bounds are somewhat shifted to lower or higher temperatures.

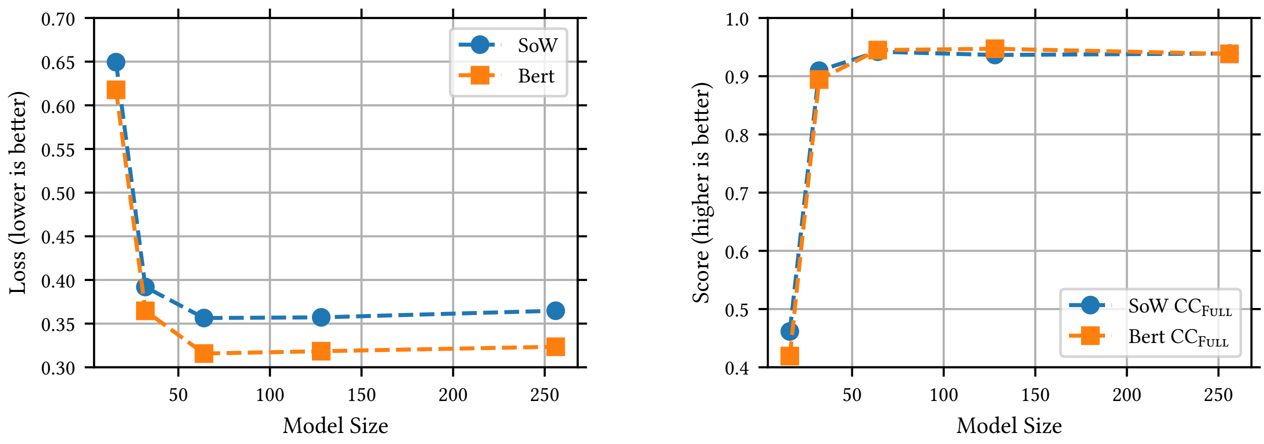
As weather reports are always bound to a specific location, we figured that a weather report not mentioning the name of the place it is created for does not make much sense. Moreover, all weather reports in the wetter.com dataset mention their location at least once, usually right at the beginning of the report. Hence, the City metric quantifies the fraction of generated weather reports that contain the `<city>` tag at least once for a given dataset.

4 Experiments

We carried out several experiments, first with the transformer model, and later repeated these experiments with the LSTM once its architecture was finalized. This is the reason why the intermediate and the final talk focused on the transformer results. To be in harmony with the talks, we keep the focus on the transformer results for this report and add the LSTM results / plots to the appendix. Except for Section 4.7, all experiments were carried out on the wetter.com dataset.

4.1 Tokenizers

The first of our experiments dealt with the question of which tokenizer to use. We tried both the SoW and the Bert tokenizer on transformers of differing sizes. The smallest tested transformer has a model dimension of 16, and the largest has a model dimension of 256. A full overview over the relevant model parameters can be obtained from Table 6 in Appendix B (Table 7 in Appendix D for the LSTM). Figure 3 and Figure 10 show the results of this experiment. With respect to all our metrics, both tokenizers performed almost equally well on both models, although the Bert tokenizer led to slightly better results in the case of the larger transformers (Figure 3b). Moreover, the evaluation loss is lower when the Bert tokenizer is used (Figure 3a), indicating that these models are on average a bit more certain about the next token. Therefore, we decided to use Bert for the decoder and SoW for the encoder.



(a) Evaluation loss over model size (d_model) (b) CC_{FULL} over model size (d_model)

Figure 3: Model performance with respect to model size and decoder tokenizer. Only CC_{FULL} is displayed in Figure 3b for clarity. The utilized tokenizer has little impact on the metrics, but Bert leads to models with a lower evaluation loss, i. e. models that are on average slightly more certain about the next token.

4.2 Model Size

By looking at Figure 3 (Figure 11 in Appendix F for the LSTM), it becomes apparent that the Medium, Big and Huge transformers show a very similar performance. Only the two smallest transformers Tiny and Small show a noticeably worse performance both in terms of the loss and the metrics. This indicates that the Medium transformer is in principle large enough for this task, whilst the additional parameters in the two bigger transformers seem to be an unnecessary overhead. Nevertheless, the number of parameters is only one aspect of efficiency. It is also worth considering the time it takes to train these models. Obviously, the bigger the number of parameters in a model, the longer one epoch takes during training. However, due to the increased model capacity in the larger models, they required fewer epochs to reach their peak performance and to trigger the early stopping criterion. We therefore also considered the "time to best model" T_{best}.

Table 3 shows T_{best} for the five transformer sizes tested. It is more than obvious that the Big and Huge transformers achieved their peak performance significantly faster than the other three models. In fact, the Big transformer achieved the smallest T_{best}; it triggered the early stopping criterion almost twice as fast as the Medium transformer. Hence, deciding which model to use for the final two experiments was a trade-off between the Medium transformer (the smallest model with good performance) and the Big transformer (the fastest model to train). We eventually chose the smaller of the two for the subsequent experiments because the number of parameters impacts not only space but also the inference time. Moreover, a more aggressive early stopping would have reduced T_{best} for the smaller models. For example, the evaluation loss barely changed during the last 3 h of training of the Medium transformer.

Table 3: Time to best model T_{best} for all tested transformer sizes. The Tiny and the Small transformers did not trigger early stopping within the 100 epochs. Moreover, we cannot fully guarantee that the GPU was not used by other processes during the training.

Transformer	Tiny	Small	Medium	Big	Huge
Epoch	100	97	71	27	20
T_{best} in s	17095	19614	20889	**11342**	<u>13624</u>

Since the transformers and the LSTMs were trained on different hardware, the training times of the LSTMs cannot be directly compared to the ones of the transformers. However, the LSTMs show a similar trend, where a bigger model does not automatically yield better results, as can be seen in Figure 11 in Appendix F. Thus, for the same reasons as for the transformer and to keep the models rather comparable, the Medium LSTM was also used for further experiments.

4.3 Context Size

The attributes in Table 1 convey the impression that at least some of them are redundant. For example, the weather reports never state anything about the wind or specify the amount and risk of precipitation. Hence, we trained a series of transformers that use different sets of weather attributes as their contexts in order to investigate which attributes are vital for good weather reports. The results are displayed in Table 4 (Table 8 in Appendix F for the LSTM).

Table 4: Results of the ablation study regarding the weather data attributes. Apart from the *Temperature* attribute, the *Clearness* attribute is the most important attribute. Including the *Cloudiness* has a very small positive effect on the performance of the transformer, whereas the four precipitation and wind attributes are apparently negligible. The time to best model improves significantly for smaller contexts (again, we cannot fully guarantee the accuracy of the measured times). The last row contains the values for $\text{Temp}_{\text{Ghost}}$, $\text{Temp}_{\text{Range}}$ and City on the original weather reports from wetter.com as a reference.

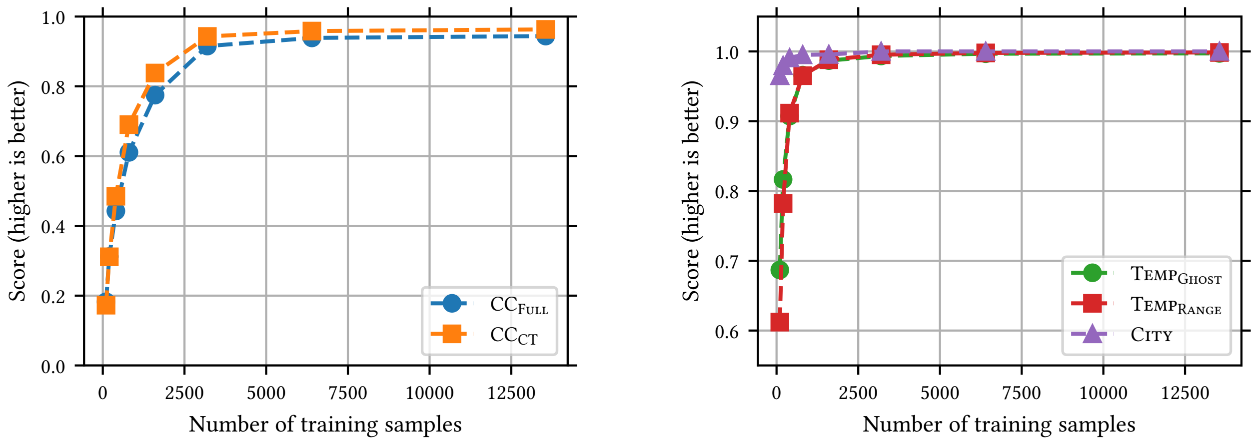
Context	CC_{Full}	CC_{CT}	$\text{Temp}_{\text{Ghost}}$	$\text{Temp}_{\text{Range}}$	City	T_{best} in s
Full: All attributes	**0.945**	0.962	<u>0.997</u>	**0.999**	**1.0**	20889
CTPC: Clearness, Temperature, Precipitation, Cloudiness	0.941	0.961	<u>0.997</u>	<u>0.998</u>	**1.0**	13330
CTC: Clearness, Temperature, Cloudiness	**0.945**	**0.964**	**0.998**	**0.999**	**1.0**	**11358**
CT: Clearness, Temperature	<u>0.944</u>	<u>0.963</u>	<u>0.997</u>	**0.999**	**1.0**	<u>11399</u>
TPWC: Temperature, Precipitation, Wind, Cloudiness	0.799	0.847	0.996	<u>0.998</u>	**1.0**	15340
wetter.com Dataset	-	-	1.0	0.999	1.0	-

Clearly, the two wind attributes *Wind Direction* and *Wind Speed* as well as the two precipitation attributes *Risk of Precipitation* and *Amount of Precipitation* hardly influence the performance of the transformers. The *Cloudiness* attribute helps slightly as the CTC transformer outperforms the CT transformer by a very thin margin. Omitting the *Clearness* attribute (TPWC) leads to a massive drop in the model's performance in terms of CC_{Full} and CC_{CT}. Leaving out the *Temperature* attribute was not tested because this makes very little sense. A weather report without temperature values is more or less worthless, and, given the sentence structure (see Section 2.2), it is almost guaranteed that omitting this attribute will have a detrimental effect on the quality of the generated weather reports.

It is also worth noting that a smaller context also massively improves the time to best model. This is most likely the result of the fact that the complexity of standard self-attention scales quadratically with the sequence length [16].

4.4 Dataset Size

One of our research questions was how large a dataset is needed to train a model that is supposed to create weather reports. To answer this question, we trained a series of transformers (with CT context) in which the training set was roughly cut in half for each model, i. e. starting from a model trained on the entire training set (13 553 samples), the next model was trained on roughly half of the training set (6400 samples), the next model on half of the previous set (3200 samples), and so on until only 100 samples were left in the training set. Figure 4 (Figure 12 in Appendix F for the LSTM) show the results for all our five metrics.



(a) (b)

Figure 4: Model performance with respect to the size of the training set. Transformers were trained on training sets with 13 553, 6400, 3200, 1600, 800, 400, 200 and 100 samples. One can clearly see that the smallest yet sufficient training set size is somewhere between 3200 and 6400 sample.

The first halving did not impact the performance of the model at all, whereas a minor decrease in all metrics can be observed for the model that was trained on 3200 samples. From 1600 samples and onward, the performance of the model degrades rapidly. It becomes so bad in the end that the transformer trained on 100 samples not only fails to generate correct weather reports, it also fails to generate correct German sentences. The sweet spot should thus be somewhere between 3000 and 6400 samples.

4.5 Transformer vs. LSTM

As seen in the previous sections, the transformer and the LSTM show very similar trends in almost all the performed experiments. It is important to keep in mind that the two architectures cannot be compared directly due to different parameter counts and the fact that they were trained on different machines. Yet, the transformers consistently outperform LSTMs of similar size, which can be seen in Figure 5. This discrepancy becomes most telling for the custom classifiers. As shown in Section 3.7, the custom classifier is very sensitive to the temperature values. This indicates that even though the LSTM models seem to produce weather reports with a similar quality than the transformers, they lack correctness in the temperature values. This is probably explained, on the one hand, by the transformer attending to all relevant tokens of the input at once, allowing the

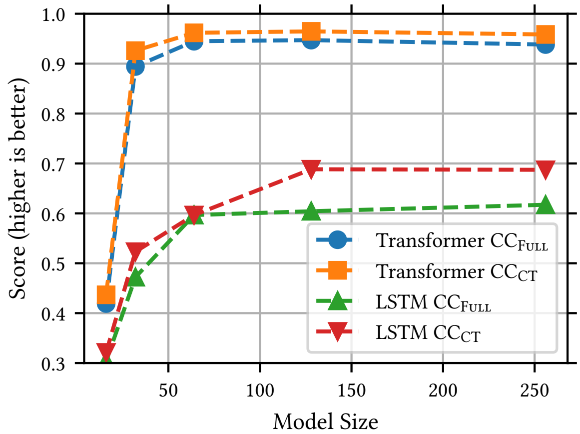


Figure 5: Values of the custom classifiers for the transformer and the LSTM with respect to model size. The transformer outperforms the LSTM across all sizes.

model to achieve high accuracy with specific numerical values. On the other hand, LSTMs tend to accumulate errors due to their recurrent structure, which could explain why these specific numerical values proved to be difficult for the model.

4.6 Example

Figure 6 shows an example weather report that was generated by the MEDIUM CTC transformer. All relevant parts are correct. The only difference is the nighttime temperature, where wetter.com states it is going to be $-12\,^{\circ}\mathrm{C}$ and our transformer states it is going to be $-11\,^{\circ}\mathrm{C}$. As described in Section 3.7, this is an example of a ghost temperature value in the report from wetter.com, and our transformer is actually correct. $-12\,^{\circ}\mathrm{C}$ are reached that day, but in the morning and not during the night, where the weather data states it is going to be $-11\,^{\circ}\mathrm{C}$.

> In Carroll ist es morgens bedeckt bei Werten von -13°C. Mittags wird die Sonne von einzelnen Wolken verdeckt bei Höchstwerten von -8°C. Am Abend ist es in Carroll wolkenlos bei Temperaturen von -11 bis -9°C. Nachts gibt es keine Wolken und die Sterne sind klar zu erkennen bei einer Temperatur von -12°C.

(a) Weather report as it was collected from wetter.com

> In Carroll ist am Morgen der Himmel bedeckt, die Sonne ist nicht zu sehen bei Werten von -13°C. Am Mittag verdecken einzelne Wolken die Sonne und die Höchstwerte liegen bei -8°C. Abends gibt es in Carroll einen wolkenlosen Himmel bei Werten von -11 bis zu -9°C. In der Nacht gibt es keine Wolken und die Sterne sind klar zu erkennen bei Werten von -11°C.

(b) Generated weather report using the MEDIUM CTC transformer.

Figure 6: Example weather report generated with the MEDIUM CTC transformer. The relevant parts are underlined and color-coded.

4.7 ChatGPT

Based on our experiments on the wetter.com dataset, we first tried to train a MEDIUM CT transformer on the second ChatGPT dataset. However, this was not crowned with success. Despite the model reaching an evaluation loss of about 2.0, which is noticeably smaller than anything we had reached on the first ChatGPT dataset and yet still far away from the evaluation losses achieved on the wetter.com dataset (Figure 3a), the generated weather reports seem to be somewhat presentable from a linguistic point of view. However, they often appear to be bloated with meaningless sentences and barely describe the weather itself (Figure 13c in Appendix H). Furthermore, they sometimes do not terminate within our set maximum of 200 tokens.

Realizing that this behavior is not surprising because the MEDIUM transformer represents the smallest tested transformer that yields good weather reports on the dramatically easier wetter.com dataset, we tried again with a HUGE CT transformer. The HUGE transformer did not reach a considerably lower evaluation loss (about 1.83), but its generated weather reports improved immensely (Figure 13d). Hence, we concluded that the MEDIUM transformer is in fact too small for the more complex ChatGPT dataset, and a model close to or even larger than the HUGE transformer would be needed to achieve very good results on this dataset.

Obviously, we cannot use our custom classifiers trained on the wetter.com dataset to assess the correctness of the more sophisticated apocalyptic weather reports. In addition, given how good / bad the generated reports still are, it is unlikely that we would be able to train an equally good classifier on the ChatGPT dataset within a reasonable timeframe. Hence, Table 5 only states values for $\text{TEMP}_{\text{GHOST}}$, $\text{TEMP}_{\text{RANGE}}$ and CITY. Nevertheless, the metric values confirm the impression that the HUGE transformer yields far better results than the MEDIUM transformer since the temperature metrics improved significantly.

Table 5: Values for $\text{TEMP}_{\text{GHOST}}$, $\text{TEMP}_{\text{RANGE}}$ and CITY of the two transformers trained on the ChatGPT dataset. Clearly, the HUGE transformer is a big improvement over the MEDIUM transformer. The last row contains the metric values on the original weather reports from the ChatGPT dataset as a reference.

Transformer	$\text{TEMP}_{\text{GHOST}}$	$\text{TEMP}_{\text{RANGE}}$	CITY
MEDIUM CT	0.967	0.856	0.991
HUGE CT	**0.990**	**0.957**	**0.999**
CHATGPT Dataset	0.999	0.998	0.999

4.8 Evaluation using ChatGPT

Within this project, we managed to establish metrics suited for evaluating the raw performance of our main models. To further increase the validation of our results accross different models, we decided to use ChatGPT. Our goal was to automatically evaluate the difference between the report that the model was trained on and its prediction. For this we have built a prompt that will return a CASQO (Creativity, Accuracy, Similarity, Quality, Overall) score (Appendix G.3). The prompt is built using detailed constraints on evaluation methods, scoring ranges (0-10) and formatting. The first step is to select cities from which the generated weather reports will be used for evaluation. The models do not need to be trained on the same pipeline for this comparison to work. After generating multiple scores for each model-city combination, we calculate the average and variance of a single run. In addition, we then calculate the mean and variance across all runs (Appendix I). From this analysis, we can tell that the transformer trained on the short reports from wetter.com had the best scores of any model (8.29) tied together with the lowest variance (0.11). The LSTM trained on wetter.com is not far behind (6.63), while the models trained on reports from the ChatGPT dataset share similar statistics (Appendix J). To fully understand the evaluation and prevent misinterpretation, we need to keep the following aspects in mind:

- The creativity metric is evaluated between target and prediction without taking into account the overall creativity of the text. This in turn means that the wetter.com transformer manages to keep more of the creativity from the original data than the one trained on ChatGPT reports. However, this does not mean that the transformer trained on wetter.com data is writing more creative texts.

- When comparing the creativity of the wetter.com reports and the ChatGPT reports through a slightly changed prompt, we get a creativity score of 2 for wetter.com and 9 for ChatGPT (on average; evaluated by the language model Claude to not induce a model bias). This entails that the models trained on ChatGPT are working with a tougher data set.

- When comparing the GRU and transformer (both trained on ChatGPT), it is noticeable that the best models have similar statistics, but we need to keep in mind that the GRU can only achieve this level of similarity with three to four times as many parameters. A transformer with more parameters would most likely achieve better scores, but in the case of the GRU, the parameter count is on the verge of being sufficient to generate qualitative texts without noticeable tradeoffs (repetition, losing context over long sequences).

5 Summary

In this project, we investigated if small and efficient NLP models can generate accurate and creative weather reports while being small enough to run on a laptop. We trained both a transformer and an LSTM using weather data and reports collected from wetter.com. Since the original reports from wetter.com were very repetitive and rather short, we also created a second dataset by rewriting every individual weather report with ChatGPT to enhance their variety and creativity. In order to simulate the "train on a laptop" constraint without actually having to train the models on a laptop, we used old and entry-level GPUs.

Our results confirmed that a small LLM can generate structured weather reports, answering our first research question. While both the transformer and LSTM were able to generate readable reports, the transformer consistently outperformed the LSTM, especially regarding precise values like temperatures. The LSTM often struggled with precision, whereas the transformer created reports with higher accuracy.

Regarding the second research question, we found that a medium-sized transformer was sufficient to achieve good results. Training on around 3,200 to 6,400 samples proved to be a good balance between dataset size and model performance, showing that a relatively small dataset can be enough for the generation of highly structured weather reports.

For the third research question, we observed that generating creative and informal weather reports required larger models. While ChatGPT-generated reports introduced creativity, our trained models tended to get lost in the stylistic variation. The transformer could produce engaging outputs, but the correctness and usefulness of the generated reports was lagging behind the original weather reports.

To evaluate the models, we tried standard NLP metrics, such as Bleu and Rouge, but eventually settled on custom metrics and a custom classifier to measure the correctness of the reports. It is important to mention that the custom metrics are not the answer to everything. In particular, the temperature metrics are only able to address the problem of ghost values partially. Additionally, we used ChatGPT to rate the quality and creativity of the generated reports. The results confirm that lightweight models can work well in environments with limited resources.

Overall, this project and the results demonstrate that small NLP models work well in real-world applications. Future work could explore further optimizations and further experiments on the more creative data set.

References

[1] Dzmitry Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate." In: Proc. of the International Conference on Learning Representations (ICLR). San Diego, CA (USA), May 2015, pp. 1–15.

[2] Torch Contributors. *PyTorch Documentation. torch.nn.Transformer.* Version 2.5.1. 2023. URL: `https://pytorch.org/docs/2.5/generated/torch.nn.Transformer.html#transformer` (visited on 03/07/2025).

[3] Hugging Face. *BERT Documentation. BertTokenizer.* URL: `https://huggingface.co/docs/transformers/v4.49.0/en/model_doc/bert#transformers.BertTokenizer` (visited on 03/06/2025).

[4] wetter.com GmbH. *wetter.com.* URL: `https://www.wetter.com` (visited on 03/13/2025).

[5] Ian J. Goodfellow et al. "Generative Adversarial Networks." In: arXiv, (June 2014), pp. 1–9. arXiv: `1406.2661`.

[6] Alex Graves et al. "Speech recognition with deep recurrent neural networks." In: Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing. Vancouver, BC (Canada), May 2013, pp. 6645–6649.

[7] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries." In: Proc. of the Text Summarization Branches Out. Barcelona (Spain), July 2004, pp. 74–81.

[8] Ilya Loshchilov et al. "Decoupled Weight Decay Regularization." In: Proc. of the International Conference on Learning Representations (ICLR). New Orleans, LA (USA), May 2019, pp. 1–8.

[9] OpenAI. "GPT-4 Technical Report." In: arXiv, (Mar. 2024), pp. 1–100. arXiv: `2303.08774`.

[10] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation." In: Proc. of the 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia, PA (USA), July 2002, pp. 311–318.

[11] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: Proc. of the Conference on Neural Information Processing Systems (NIPS). Vancouver, BC (Canada), Dec. 2019, pp. 1–12.

[12] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks From Overfitting." In: Journal of Machine Learning Research, vol. 15 (2014), pp. 1929–1958.

[13] Jianlin Su et al. "RoFormer: Enhanced Transformer with Rotary Position Embedding." In: arXiv, (Nov. 2023), pp. 1–14. arXiv: `2104.09864`.

[14] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision." In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV (USA), June 2016, pp. 2818–2826.

[15] Hugo Touvron et al. "LLaMA: Open and Efficient Foundation Language Models." In: arXiv, (Feb. 2023), pp. 1–27. arXiv: `2302.13971`.

[16] Ashish Vaswani et al. "Attention is All you Need." In: Proc. of the Conference on Neural Information Processing Systems (NIPS). Long Beach, CA (USA), Dec. 2017, pp. 1–11.

[17] Ruibin Xiong et al. "On Layer Normalization in the Transformer Architecture." In: Proc. of the International Conference on Machine Learning (ICML). Virtual Conference, July 2020, pp. 10524–10533.

[18] Tianyi Zhang et al. "BERTScore: Evaluating Text Generation with BERT." In: Proc. of the International Conference on Learning Representations (ICLR). Addis Ababa (Ethiopia), Apr. 2020, pp. 1–14.

A Distribution of data points

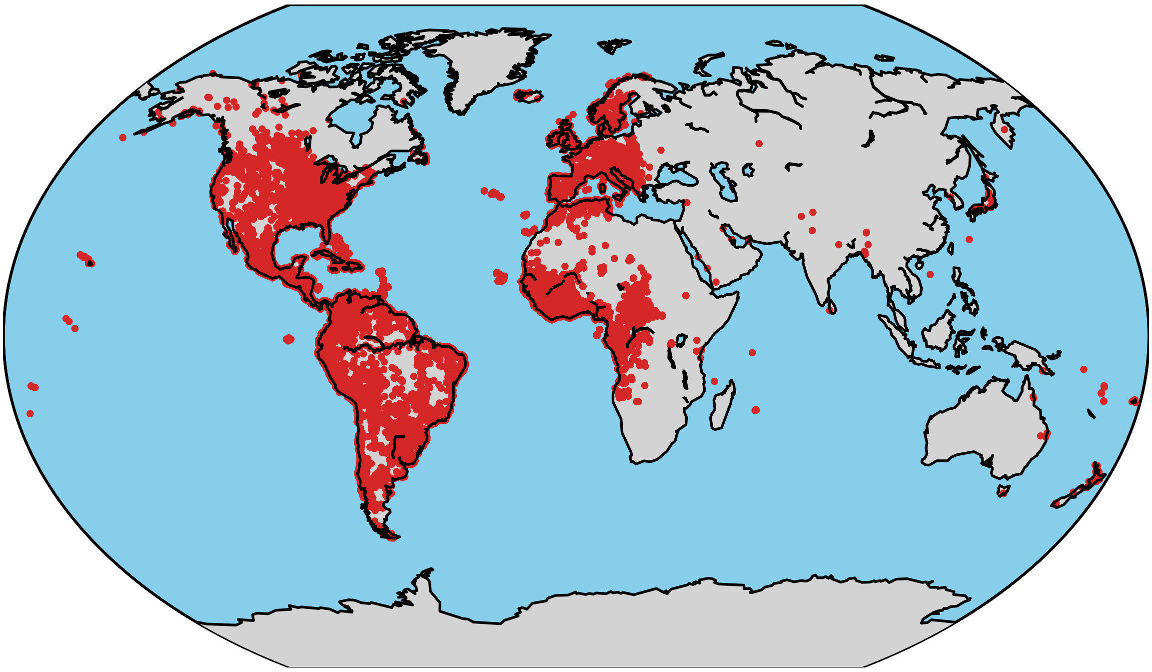


Figure 7: Distribution of places contained in the dataset.

B Transformer Configurations

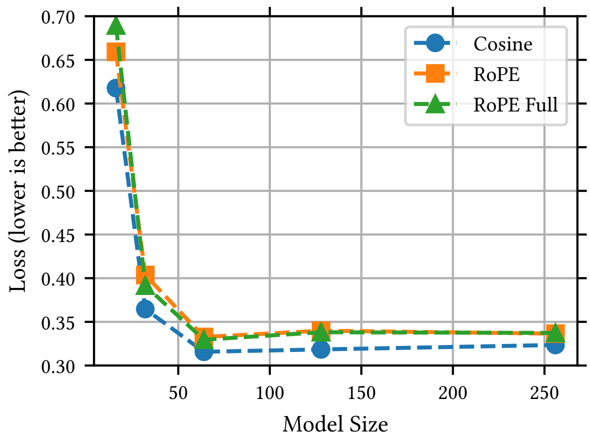
Table 6: Relevant parameters of the utilized transformer models. For all other parameters, the default values of the PyTorch implementation are used (see [2]). num_encoder_layers = num_decoder_layers = nlayers. *# Parameters* is with respect to the wetter.com dataset, these values increase noticeably for the ChatGPT dataset due to the much larger token set.

Name	d_model	dim_feedforward	nhead	nlayers	# Parameters (Bert, Full context)
Tiny	16	64	4	2	40 818
Small	32	128	4	2	110 002
Medium	64	256	4	2	334 386
Big	128	512	4	2	1 127 218
Huge	256	512	4	2	3 038 514

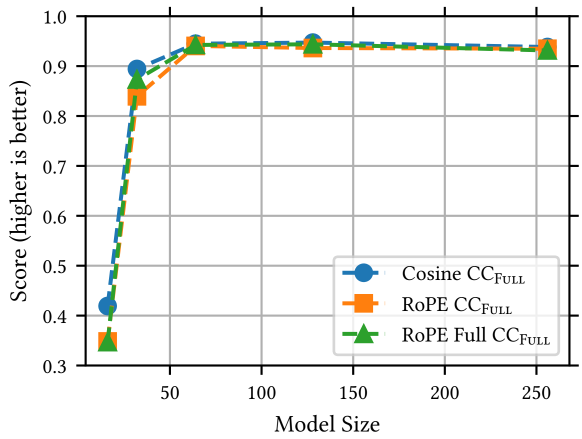
C Positional Embeddings (Transformer)

For the transformer, we tested the default absolute cosine positional embeddings against Rotary Positional Embeddings (RoPE) [13]. We implemented two variants for RoPE: the standard RoPE where the positional embeddings are only applied to the key and query vectors, and a second variant where the positional embeddings are applied to the key, query and value vectors (on the following referred to as "RoPE Full"). We decided to test both variants because the latter variant is significantly easier to implement. Unlike RoPE Full, implementing standard RoPE required us to reimplement most of the PyTorch Transformer classes in order to have different inputs for key, query and value at the first self-attention block.

The results of this experiment are displayed in Figure 8. Similarly to the tokenizer experiment, all three variants are almost on par when looking at the metrics (Figure 8b) and larger differences only come into the picture when looking at the loss (Figure 8a). Based on the loss, absolute cosine positional embeddings lead to models that are on average slightly more certain about the next token. Hence, we decided to stick to the default absolute cosine positional embeddings for the rest of the experiments.



(a) Evaluation loss over model size (d_model)



(b) CC_{FULL} over model size (d_model)

Figure 8: Model performance with respect to model size and positional embeddings. Only CC_{FULL} is displayed in Figure 8b for clarity. The utilized positional embeddings have little impact on the metrics, but absolute cosine leads to models with a lower evaluation loss.

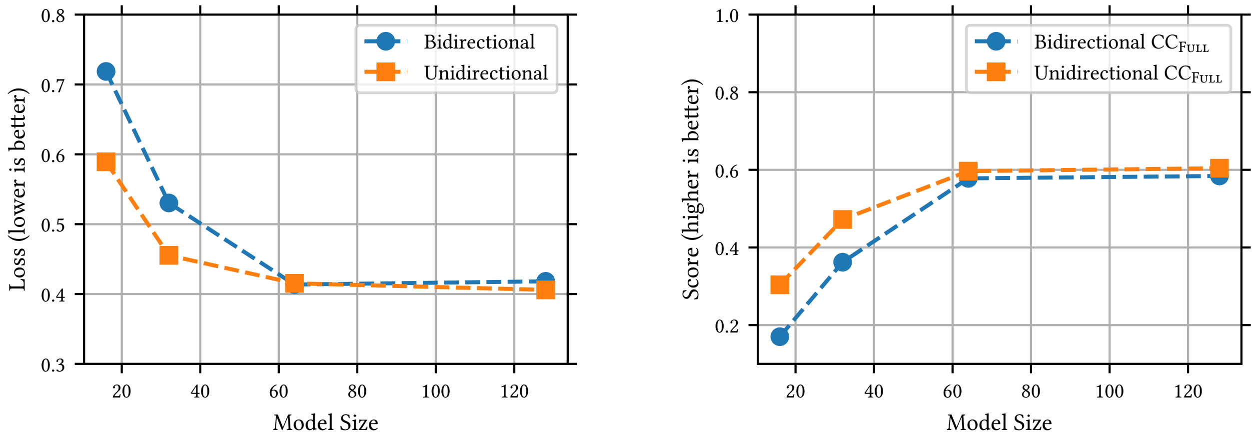
D LSTM Configurations

Table 7: Relevant parameters of the utilized LSTM models. For all other parameters. Further optimizations include the use of label smoothing [14] and dropout [12] of 0.1 as well as Bahdanau attention [1]. *# Parameters* is with respect to the wetter.com dataset, these values increase noticeably for the ChatGPT dataset due to the much larger token set.

Name	d_embedding	dim_hidden	# Parameters (Bert, Full context)
Tiny	16	16	35 235
Small	32	32	78 209
Medium	64	64	195 315
Big	128	128	537 779
Huge	256	256	1 665 075

E Bidirectional LSTMs

A common approach for working with LSTMs is bidirectionality [6]. By processing the input forward and backward, the model is given future context, which sounded promising for better accuracy especially regarding the temperature accuracy. As the bidirectional model has a hidden dimension twice as large as the unidirectional model, the hidden dimension was halved for better comparability. Otherwise, the bidirectional network would just get more context and thus perform better. As visible in Figure 9 the results did not improve. For lower model sizes, it shows that having more parameters for unidirectional processing outperforms half the parameters but bidirectional processing. But even for the bigger models, where more parameters improve the model's performance, a bidirectional approach did not improve the capability of the model. This shows that even with bidirectional processing, the model still failed to precisely generate the temperatures.



(a) Comparison of model loss of unidirectional and bidirectional LSTM.

(b) Comparison of custom classifier on unidirectional and bidirectional LSTM.

Figure 9: Model loss and custom classifier with respect to the size of the training set. The bidirectional LSTM did not improve the results in comparison to the unidirectional LSTM.

F LSTM Results

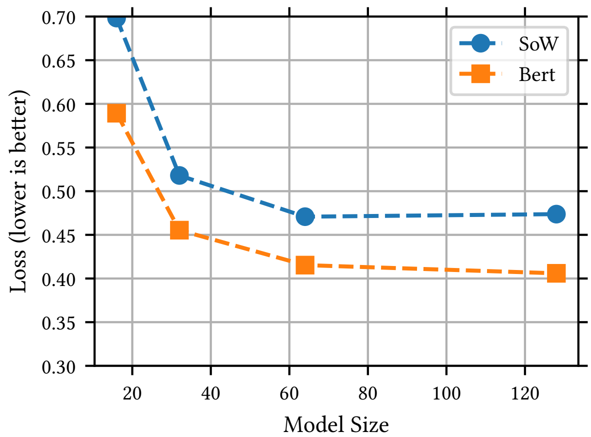
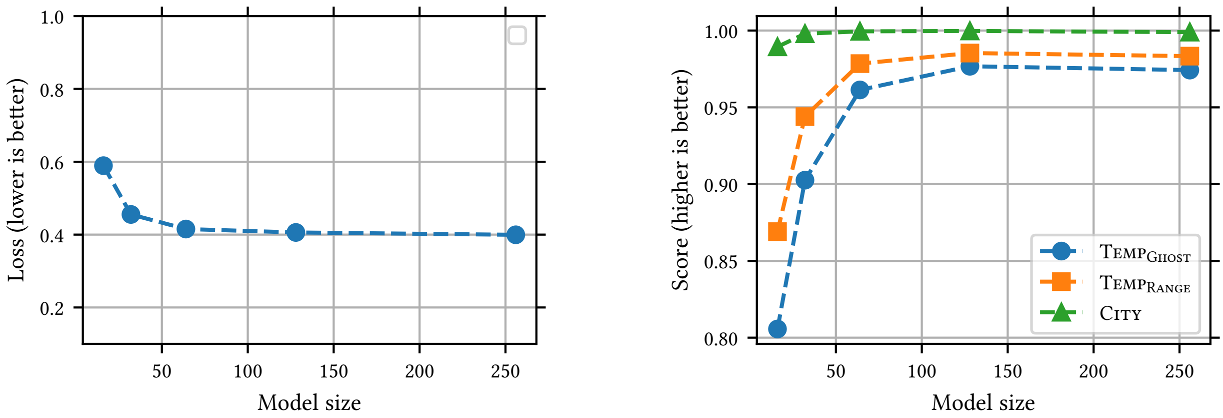


Figure 10: Model loss with respect to model size and decoder tokenizer. Just like for the transformers, Bert leads to models with a lower loss, i. e. models that are on average slightly more certain about the next token.



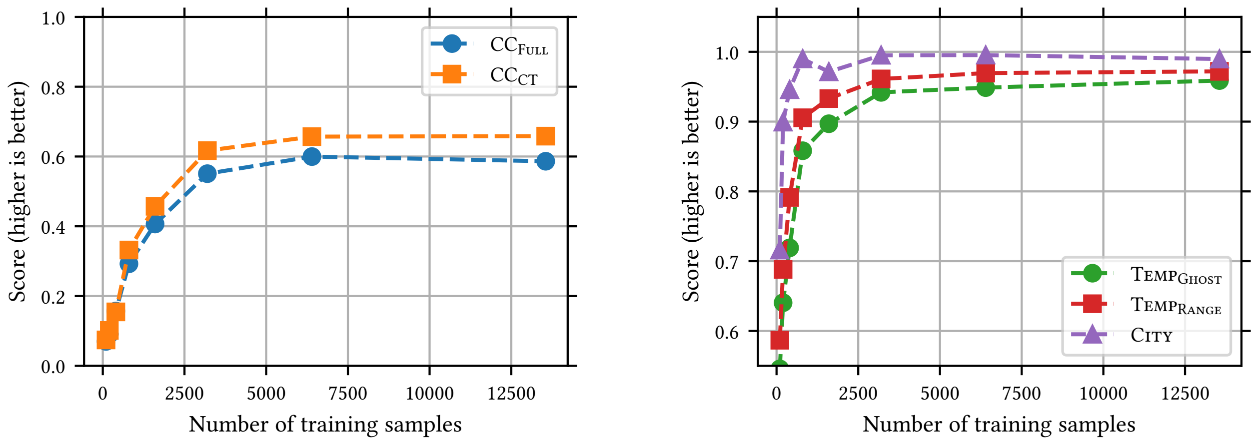
(a) Loss over model size.

(b) Custom metrics over model size.

Figure 11: Model loss and custom metrics on the LSTM architecture. As can be seen, there is no real improvement after reaching a size of 128.

Table 8: Results of the ablation study regarding the weather data attributes on the LSTM architecture. Apart from the Temperature attribute, the *Clearness* attribute is the most important attribute. Including the *Cloudiness* has a very small positive effect on the performance just like for the transformer, whereas the four precipitation and wind attributes are apparently negligible. The time to best model does not improve significantly for smaller contexts, but it is important to mention that we cannot guarantee for the accuracy of the measured times. Also, since the LSTMs were trained on a different machine, these values can not be directly compared to the values of the transformer).

Context	$\mathrm{CC}_{\mathrm{Full}}$	$\mathrm{CC}_{\mathrm{CT}}$	$\mathrm{Temp}_{\mathrm{Ghost}}$	$\mathrm{Temp}_{\mathrm{Range}}$	City	T_{best} in s
Full: All attributes	0.596	0.658	0.961	0.978	0.999	13825
CTPC: Clearness, Temperature, Precipitation, Cloudiness	0.596	0.668	0.970	0.980	0.999	13959
CTC: Clearness, Temperature, Cloudiness	0.560	0.666	0.968	0.980	0.998	11850
CT: Clearness, Temperature	0.586	0.658	0.958	0.972	0.990	13959
TPWC: Temperature, Precipitation, Wind, Cloudiness	0.544	0.604	0.956	0.975	0.997	15031



(a) Custom classifier over training samples.

(b) Custom metrics over training samples.

Figure 12: Model performance with respect to the size of the training set. LSTMs were trained on training sets with 13 553, 6400, 3200, 1600, 800, 400, 200 and 100 samples. One can clearly see that the smallest, yet sufficient training set size is somewhere between 3200 and 6400 sample.

G Example Prompts for ChatGPT

G.1 1st Dataset

```
Du bist ein professioneller Autor, welcher Wetterberichte und Flachwitze schreibt. Nun möchtest du beides kombinieren und
deine Wetterberichte in lustige, und durchaus etwas unseriös klingende Wetterberichte verwandeln. Schreibe bitte folgenden
Wetterbericht um: <bericht>
```

Example output:

```
In Ab Tchetchenia gibt die Sonne heute ihr Bestes, wird aber von ein paar Wolken sabotiert – nennen wir es ein freundliches
Duell am Himmel. Die Temperaturen schwanken dabei sportlich zwischen 19 und 33 <degC>. Nachts übernehmen dann wieder ein paar
Wolken das Kommando, aber sie sind eher von der entspannten Sorte. Mit 19 <degC> bleibt es angenehm – also keine Panik, die
Winterjacke kann im Schrank bleiben!
```

G.2 2nd Dataset

```
background = '''
Du bist ein professioneller Autor, welcher Wetterberichte schreibt.
Nun möchtest du deine Wetterberichte in < attribut > klingende
Wetterberichte verwandeln. '''

task = '''
Schreibe bitte folgenden Wetterbericht um:'''

requirements = '''
< restrictions >
1. Verwende einfache Sprache.
2. Keine langen Verkettungen von Wörtern oder adjektiven.
3. Das <city> Token darf nicht verändert werden.
4. Beschränke dich auf die deutsche Sprache.
5. Nur deutsche Alphabet (A–z, ö,ü,ä, ß) und latin-1 Zeichensatz.
6. Nutze nur " als Anführungszeichen.
7. Einfaches Vokabular ohne seltene Wörter.
8. Beschränke dich auf 150 Wörter.
9. Nur den Wetterbericht ohne Einleitung/Ausleitung.
10. Keine wörtliche Rede.
</ restrictions >
'''
```

Example output:

```
"In <city> ziehen dunkle Schatten über den Himmel. Den ganzen Tag kämpfen Wolken mit der Sonne. Die Temperaturen steigen und
erreichen 33 <degC>. Die Hitze erdrückt die Stadt, während die Wolken nur teilweise Licht durchlassen. In der Nacht kehrt
Dunkelheit zurück. Nur einzelne Wolken schweben am Himmel und verdecken die Sterne. Es wird kühl. Die Temperaturen fallen auf
19 <degC>, während eine unheimliche Stille einsetzt. Ein Gefühl von Unruhe liegt in der Luft. Die Schatten der Wolken bilden
gefährliche Formen. Es scheint, als würde die Nacht die Stadt in ihren eisigen Griff nehmen. Die Welt kommt zur Ruhe, doch die
Wolken behalten ihre Geheimnisse. Die Kälte bringt eine unheilvolle Stille mit sich. Der Atem der Nacht ist spürbar, während
die Dunkelheit sich über <city> legt. Die Ungewissheit bleibt, was der neue Tag bringen wird."
```

G.3 Evaluation

Du bist ein objektiver Textanalyse−Experte, der spezialisiert ist auf die Bewertung von Wetterberichten. Deine Aufgabe ist es,
eine systematische und reproduzierbare Bewertung von generierten Wetterberichten abzugeben.

BEWERTUNGSRICHTLINIEN:
1. Bewerte auf einer Skala von 1−10 in jeder Kategorie, wobei 10 die höchste Punktzahl ist.
2. Gib immer genaue numerische Werte an, keine Bereiche oder Dezimalstellen.
3. Achte auf strenge Konsistenz in deiner Bewertungsmethodik.
4. Berü cksichtige keine persönlichen Präferenzen oder subjektiven Eindrücke.
5. Die Bewertung soll deterministisch sein − gleiche Texte müssen immer zu gleichen Bewertungen führen.

KATEGORIEN:
− GENAUIGKEIT (1−10): Wie genau werden die gleichen Wetterdaten wie im Zieltext wiedergegeben?
− KREATIVITÄT (1−10): Wie kreativ ist der Ausdruck, die Wortwahl und die Struktur im Vergleich zum Zieltext?
− QUALITÄT (1−10): Wie gut ist die allgemeine sprachliche Qualität und Lesbarkeit?
− ÄHNLICHKEIT (1−10): Wie ähnlich ist der Text zum Zieltext in Bezug auf Stil und Inhalt?
− GESAMTBEWERTUNG (1−10): Die zusammenfassende Bewertung der Vorhersage.

FORMAT:
Nach deiner Analyse musst du deine Bewertung in folgendem exakten Format zurückgeben:
GENAUIGKEIT: [Punktzahl]
KREATIVITÄT: [Punktzahl]
QUALITÄT: [Punktzahl]
ÄHNLICHKEIT: [Punktzahl]
GESAMTBEWERTUNG: [Punktzahl]

Deine Bewertung muss diesem Format folgen, ohne zusä tzliche Erklärungen, Einführungen oder Nachbemerkungen. Die
Gesamtbewertung ist kein Durchschnitt, sondern deine Einschätzung der Gesamtqualität unter Berücksichtigung aller Faktoren.

WICHTIG: Du musst für jede Kategorie genau einen ganzzahligen Wert zwischen 1 und 10 angeben. Keine Erklärungen, kein zusä
tzlicher Text.

Target:
In Paracin ist am Morgen der Himmel bedeckt, die Sonne ist nicht zu sehen bei Werten von 2 <degC>. Später ist es wolkig bei
Höchsttemperaturen bis zu 4 <degC>. Abends ist es in Paracin bedeckt bei Werten von 2 bis zu 3 <degC>. Nachts bleibt die
Wolkendecke geschlossen bei Tiefsttemperaturen von 2 <degC>.
Prediction:
In Paracin kann sich am Morgen die Sonne nicht durchsetzen und es bleibt bedeckt bei Werten von 2 <degC>. Am Mittag ist es
wolkig, die Sonne zeigt sich nur vereinzelt und die Temperatur erreicht 4 <degC>. Abends ist in Paracin der Himmel bedeckt
und die Temperaturen liegen zwischen 2 und 3 Grad. In der Nacht überwiegt dichte Bewölkung, aber es bleibt trocken bei einer
Temperatur von 2 <degC>.

H Example Outputs Using the ChatGPT Dataset

In Carroll ist es morgens bedeckt bei Werten von -13°C. Mittags wird die Sonne von einzelnen Wolken verdeckt bei Höchstwerten von -8°C. Am Abend ist es in Carroll wolkenlos bei Temperaturen von -11 bis -9°C. Nachts gibt es keine Wolken und die Sterne sind klar zu erkennen bei einer Temperatur von -12°C.

(a) wetter.com

In Carroll ist der Himmel am Morgen schwarz und kalt. Die Temperatur sinkt bis auf -13°C. Zur Mittagszeit kämpft die Sonne gegen die Wolken. Die Luft bleibt eisig bei -8°C. Am Abend wird der Himmel klar und die Dunkelheit umhüllt die Stadt. Die Temperaturen schwanken zwischen -11°C und -9°C. In der Nacht herrscht eisige Stille. Der Himmel ist wolkenlos. Die Sterne leuchten wie kleine Augen im Dunkeln, während die Kälte auf -12°C absinkt. Ein kalter Hauch weht durch die Straßen von Carroll. Die Dunkelheit hält die Stadt fest in ihrem Griff. Hoffnung auf Wärme bleibt verborgen.

(b) Above weather report from wetter.com rewritten in an apocalyptic mood using ChatGPT

In Carroll erhebt sich der Tag gegen dunkle Schleier der Wolken. Die Sonne versteckt sich mit aller Kraft und die Temperatur steigt auf bedrohliche -8°C. Doch der Abend naht, und der Himmel bleibt klar, die Sterne blitzen wie ferne Augen fern. Es ist gefrorener Windes der Kälte. Es wird bitter und friedlich, doch die Kälte umgibt die Stadt. Nichts ist verschwunden. In der Nacht bleibt die letzte Hülle von Wolken, es wird gefroren und die Stille ist fast trügerisch. Die Sicht wird weiterhin frisst durch die Dunkelheit, und die Temperaturen fallen auf gefrorene Sicht schwanken zwischen -11 und -8 Grad. Diese Kälte beißt näher und die Sonne verliert die Sonne ihre Strahlen aus. Am Abendsten klaren Himmel. Die Sonne brennt hell und die Temperatur sinkt auf -11°C. Doch in der Nacht sind die einzigen Wolken da, während die Sonne schwach weiterhin unerbittlich ist. Der Frost kämpft

(c) Generated apocalyptic weather report using a Medium CT Transformer

In Carroll ist der Himmel grau und düster. Die Temperaturen liegen bei frostigen -13°C. Im Laufe des Tages wird die Sonne durch die Wolken dringen, bleiben sie fest im Kampf gegen die Kälte, der Himmel strahlt blau. Die Temperaturen steigen leicht auf -8°C. Mit dem Abend kommt die Kälte. Der Himmel bleibt überwiegend klar, doch die Kälte bleibt, während die Temperaturen erneut auf -12°C fallen. Die Stille wird erdrückend, während die Frostgeister über Carroll ziehen.

(d) Generated apocalyptic weather report using a Huge CT Transformer

Figure 13: Examples for weather reports generated with transformers that were trained on the second, apocalyptic-sounding ChatGPT dataset

I GPT Metrics Calculation

Single model-city average

$$\mu_{m,c,M} = \frac{1}{n_{c,M}} \sum_{i=1}^{n_{c,M}} s_{m,i,c,M}$$

Variance for model-city combination

$$\sigma^2_{m,c,M} = \frac{1}{n_{c,M}} \sum_{i=1}^{n_{c,M}} (s_{m,i,c,M} - \mu_{m,c,M})^2$$

Overall model score across all cities:

$$\mu_{m,M} = \frac{1}{|C_M|} \sum_{c \in C_M} \mu_{m,c,M}$$

Variance of model performance across cities:

$$\sigma^2_{m,M} = \frac{1}{|C_M|} \sum_{c \in C_M} (\mu_{m,c,M} - \mu_{m,M})^2$$

$$\begin{aligned}
\text{where } s_{m,i,c,M} &\text{is the i-th score for metric m for city c and model M} && (1)\\
n_{c,M} &\text{is the number of evaluations for city c and model M} && (2)\\
C_M &\text{is the set of all cities evaluated for model M} && (3)\\
|C_M| &\text{ is the number of cities in that set, and} && (4)\\
m &\in \{\text{creativity, accuracy, similarity, quality, overall}\}. && (5)
\end{aligned}$$

J GPT Evaluation Metrics Plotted