DD2417 - Group 6 - Word Predictor

Eric Banzuzi, Rosamelia Carioni, Katharina Deckenbach *

June 4, 2024

1 Abstract

Word prediction plays an essential role in improving user experience on digital devices by saving valuable time when typing and improving spelling and grammar. In this project, we implemented three distinct models -a statistical model and two neural models- to assess the impact of different architectures and training strategies. Evaluating our models in three experiments and across four datasets, we find that using artificial padding when training Transformer and GRU models improves their performance in situations when there is not much information available. Notably, the Transformer model outperformed the other models in predicting the next token or word. When integrated into our GUI, all three models successfully saved the users more than 50% of keystrokes, highlighting their practical utility in real-world applications.

2 Introduction

Making typing easier, faster and more comfortable has always been attempted ever since the first mobile phones came out. One early example is the T9 technology from the 1990s that helped users to avoid having to press the same key several times but used built-in dictionaries to suggest a word. This showed to be very efficient with an estimated accuracy of 95% [5]. Nowadays, the way of typing has changed dramatically. The use of keyboards makes a single keystroke unambiguous. Still, writing can be sped up by predicting the word that the user currently types by using the context and the first letters of the word that have already been entered. Other applications allow the user to slide over a virtual keyboard instead of pressing the keys which gives an experience that some people prefer over typing. All these tools have in common that they use methods from the field of Natural Language Processing (NLP) to allow the user to save as many keystrokes as possible. In this project, we will explore the performance of several self-implemented word prediction models.

3 Background Work

The methods underlying word prediction tools have greatly advanced over time, and while early models relied on statistical methods, recent developments of neural methods such as Recurrent Neural Networks (RNNs) or Transformer architecture [6] have driven further improvements. Almost all smartphones allow next word predictions when typing, often improving its predictions over time by including the history of the user. Examples of these tools include Swype, SwiftBoard, GBoard but also applications that help to improve the user's grammar such as Grammarly.

*`https://github.com/RosameliaCarioni/word_predictor`

4 Methodology

This section outlines the methodology used to develop our word prediction system. We begin by detailing the system design, which illustrates the overall architecture and tools used. This is followed by a description of the datasets used, including the steps taken to clean and preprocess the data. Next, we explain the sample creation process for each model, highlighting the differences between the N-Gram, GRU, and Transformer models. Finally, we provide an in-depth look at each prediction model, discussing their features and configurations.

4.1 System Design

Our solution design can be seen in Figure 2. The process begins by generating the data samples from the raw data, which are tokenized using BertTokenizer[1]. An exception is made for the N-Gram model, which skips the tokenization and data sample generation steps, instead training directly on the cleaned word-level data. Python serves as the middle layer, facilitating the training of the models with the data and connecting the user interface with the models. Users interact with the system via a Streamlit[2] web app, which provides an interface for choosing between three different models for predicting the next word. The web app also displays the number of characters saved, on the selected model, when typing and choosing to autocomplete a word.

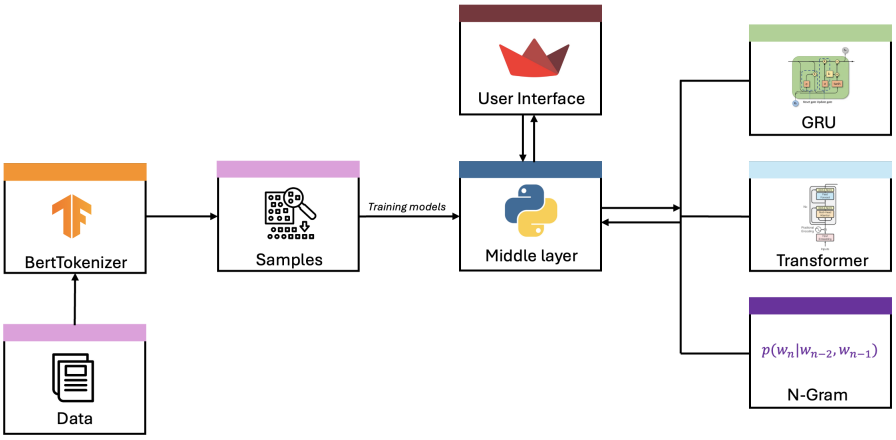

Figure 1: System Architecture

4.2 Data

This project utilizes a total of 819.5 MB for training the models. For some experiments we additionally used parts of a dataset containing mobile messages to evaluate the models. These datasets encompass a wide range of text sources, including news summaries, articles, mobile messages, and tweets. See Appendix A for a detailed description of each dataset used as well as their references.

4.2.1 Cleaning

All datasets underwent a preprocessing phase to clean and standardize them. Although the preprocessing was specific to each dataset, several common steps were involved. Character replacement was performed to standardize characters, such as replacing ”” with their standard equivalents. Punctuation handling

[1] https://www.tensorflow.org/text/api_docs/python/text/BertTokenizer
[2] https://streamlit.io/

was conducted to preserve certain punctuation characters, such as "#", "-" and "", while ensuring that periods in abbreviations like "U.S." and "P.M." were retained. All other punctuation characters were removed. Lowercasing was applied to ensure uniformity across the text data.

4.2.2 Samples Creation

The creation of samples varies based on the model being used. For the N-Gram model, the text is scanned, and a word is considered as any sequence of characters between white spaces. During this process, unigram, bigram, and trigram counts are generated and stored.

The process of creating samples for the GRU and Transformer models is more complex and involves several steps:

- Concurrent file reading: a thread pool is created to read multiple data files simultaneously, improving the efficiency of data loading.
- Chunk processing: the text is read in chunks of $1,000,000$ characters to manage large datasets efficiently. This chunk size ensures that the system handles data efficiently without overwhelming memory. See Appendix B for the values tested in the memory and time optimization when reading the data.
- Segment splitting: each chunk is split into smaller segments with a maximum size of 512 characters. This size fits within the capacity of the tokenizer library.
- Tokenization: the BertTokenizer is used to convert the text into sequences of tokens. BERT, a state-of-the-art tokenizer, has a vocabulary of $30,522$ tokens and handles uncommon words by splitting them into subwords (e.g., "blared" is split into "b", "##lar", "##ed").
- Sequence creation: two types of sequences can be created, as seen in Figure 2. The main purpose of this was to evaluate the effectiveness of artificial padding, which can be beneficial when the context is smaller than n, which in our case is set to 5.

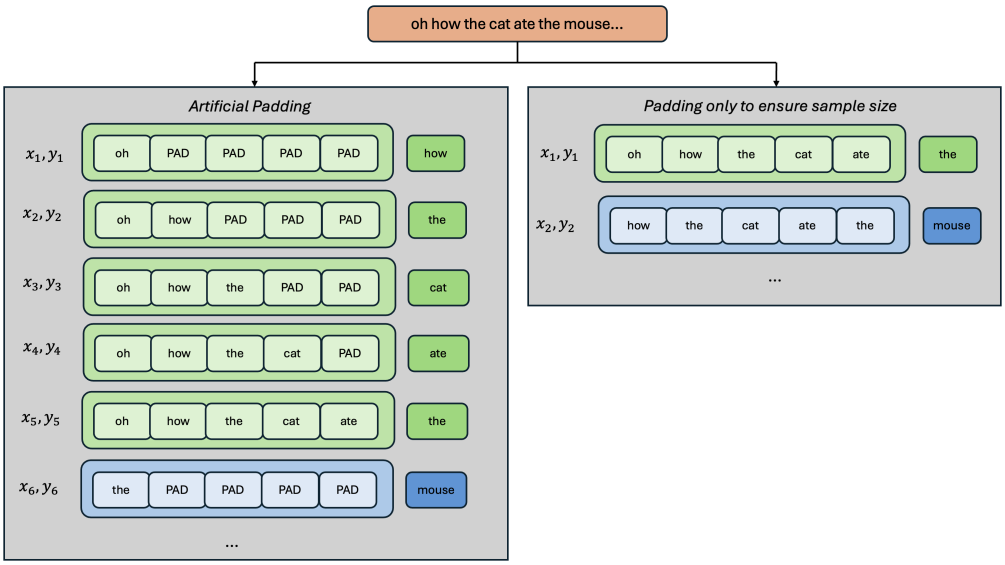

Figure 2: Comparison between samples creation techniques used in GRU and Transformer models

4.3 Prediction Models

This section looks in detail into the three models implemented in this work.

4.3.1 N-Gram

The N-Gram model is a statistical language model used to predict the probability of the next word in a sequence based on the previous $n-1$ words [2]. In this project, we implement a trigram model ($N = 3$) to predict the next word using the previous two words. To address the issue of zero probabilities for unseen word combinations, we employ linear interpolation. This technique combines the probabilities of unigrams, bigrams, and trigrams, weighted by predefined λ values. The general formula used for our trigram model is:

$$\hat{P}(w_n \mid w_{n-2}w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n \mid w_{n-2}w_{n-1}) + \lambda_4$$

where λ_1, λ_2, λ_3, λ_4 are the interpolation weights, set, as recommended on the course material, as follows:

$$\lambda_1 = 0.9, \lambda_2 = 0.09, \lambda_3 = 0.01 - 10e-6, \lambda_4 = 10e-6$$

To predict the next word w_i, if the user has not finished typing a word, the model considers all words in the vocabulary that match the partially typed input and calculates their probabilities based on the context. If the user has finished typing a word, the model calculates the probabilities for all possible subsequent words in the vocabulary. The words are then ranked and presented in order of their predicted likelihood.

The N-Gram model is a straightforward and computationally efficient approach to word prediction, providing a solid baseline against which more complex models can be compared.

4.3.2 Recurrent Neural Network: Gated Recurrent Unit

RNNs are well suited for sequential data like texts because they make use of a hidden state that is updated after every step when processing the sequence. If the sequence consists of n words the last hidden state will contain information of all words. However, the vanilla RNN cell does not perform too well if the sequence is very long. This is because training them via Backpropagation Through Time leads to problems with vanishing and exploding gradients.

There are several variants of RNNs and more sophisitcated RNN-cell architectures like the Gated Recurrent Unit (GRU). It uses different gates to control how the hidden states gets updated and has an increased ability in learning long-range dependencies compared to vanilla RNNs. Another more sophisticated RNN-cell architectures is Long Short-Term Memory (LSTM) which was developed before GRU, uses more gates to update the hidden state. With that, it can deal better with longer sequences than the GRU, but having more parameters, its training is less efficient. Because of that, we decided to use a GRU architecture in our project. The model design was kept rather simple with one encoding layer, and two GRU layers with a dropout layer between them to see how good it already performs on the word prediction task. For further details on our model architecture, including a schematic illustration of the GRU cell and the overall model setup, please refer to Appendix C and Figure 3.

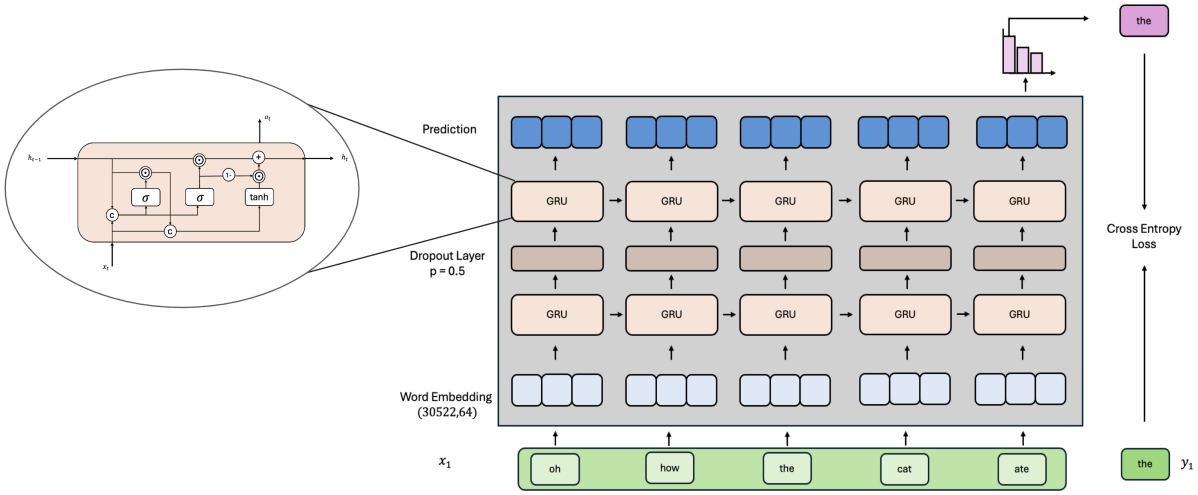

Figure 3: Componenets of the GRU model

4.3.3 Transformer

The Transformer model is a state-of-the-art neural network architecture designed for handling sequential data, making it ideal for tasks such as text prediction. Unlike traditional RNNs, the Transformer model uses self-attention mechanisms [6] to process and encode input data, effectively capturing long-range dependencies. This capability has made it the backbone of large language models like GPT-4, Gemini, and BERT, which is why we are implementing it in this project.

Our Transformer model employs only the encoder part and is configured with a vocabulary size of $30,522$ (using BertTokenizer), and its architecture based on model C from "Attention is All You Need" [6], with modifications to fit our computational constraints. See Appendix C for further details on the model configuration, and Figure 4 for the overall components and architecture of the model.

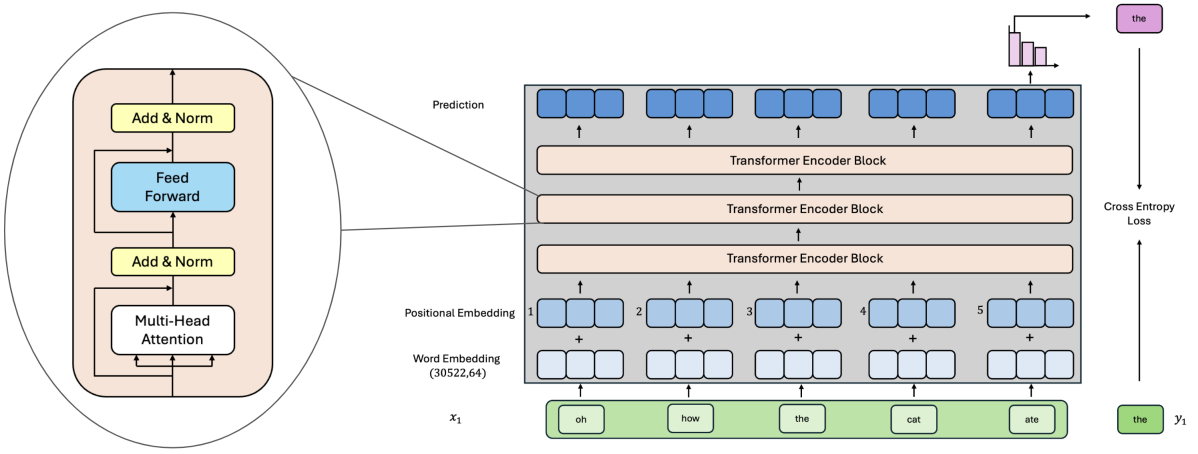

Figure 4: Components of the Transformer model. The process begins with word embeddings and positional embeddings, which are added together to form the input representations. These inputs pass through three layers of Transformer encoder blocks, each consisting of multi-head self-attention and feedforward neural networks. The output of the encoder blocks is used to predict the next word by linearly mapping the encoded representation to the vocabulary size. Lastly, the prediction is compared to the actual next word using cross-entropy loss.

5 Experiments

Our experiments are designed to evaluate the different solutions presented to the problem under consideration, based on three key questions:

1. How do different sample generation techniques impact the performance of GRU and Transformer models in terms of accuracy and perplexity?
2. Which of the three models achieves the highest accuracy, perplexity, and the best balance between efficiency and computational complexity?
3. Which of the three models provides the most optimal user experience by maximizing the number of characters saved per keystroke?

5.1 Evaluation Framework

To systematically assess the performance of the three implemented models and address our research questions, we established detailed training procedures, parameters, and evaluation metrics.

Data Preparation The data was split into training, validation, and test sets with a ratio of $80 : 5 : 15$, ensuring that the models were evaluated on unseen data. To ensure replicability, a random seed (numpy 5719) was set.

Training Environment The training procedure differed for the N-Gram and the neural models because of their differing underlying methods as described in Section 4.3. More specifically, the N-Gram model was trained by updating trigram, bigram and unigram counts while going through the training dataset. We used a single 16GB M1 MacBook Pro to process the data line by line. The neural network models were trained and evaluated on the same hardware configuration within the Jupyter environment provided by this course (using a NVIDIA H100 80GB GPU), ensuring a fair comparison. For both the Transformer and GRU based models, we used 4 processes, a batch size of 64, and trained each model for the same number of epochs.

Training Procedure For each model, the training process involved:

1. Initializing the model with the specified hyperparameters.
2. Splitting the data into training, validation, and test sets.
3. Training the model on the training set while monitoring the accuracy performance on the validation set to avoid overfitting.
4. Evaluating the model's final performance on the test set.

This consistent training procedure ensures that all models are evaluated under the same conditions, allowing for a fair comparison of their performance.

We used the described procedure to train a total of five models: one N-Gram, two GRU and two Transformer models. The neural network models were either trained or not with artificial padding for 2 epochs on the News Summaries and the Articles dataset. Additionally, the GRU models were trained for one epoch on the Twitter dataset and we intended to do the same with the Transformer models, however, due to limited time we had to interrupt the training on them after half an epoch.

5.2 Experiment 1: Impact of Sample Generation Techniques

We started by investigating whether neural models trained with artificial padding had an advantage over the same models that had not been trained with artificial padding. In theory, the models that are trained with artificial padding should be better at dealing with situations in which only little information is available i.e. at the beginning of a sentence. To test our hypothesis, we randomly selected 200 sentences that had a length between two and six words from the dataset with mobile messages and let our models predict the last word of each of the sentences. We wrote a function that, given a prompt, predicts the next tokens and builds words out of them. For this experiment, we used a top-k accuracy, with $k = 1$ and $k = 5$, and checked whether the last word of the sentences was included in the top k suggestions that our models made.

Model	Artificial Padding	Top-5 word-level accuracy	Top-1 word-level accuracy
GRU	Yes	25%	9%
GRU	No	22%	10%
Transformer	**Yes**	**31%**	**13.5%**
Transformer	No	28.5%	12.5%
N-Gram	Not applicable	9%	2%

Table 1: Top-5 and top-1 accuracy of models when predicting the last word of a short sequence. Neural models were either trained on a dataset with or without artificial padding.

These findings illustrate that using artificial padding during training helps the neural models to better predict the next words at test time even when the sequences are short and the model has less information available. For the following experiments, we decided to continue analyzing only the best performing models that were trained with artificial padding.

5.3 Experiment 2: Model Performance Comparison

Our second experiment consisted of comparing the model performances using the metrics of accuracy and perplexity on the test datasets. Given how our models work, those metrics reflect how good the neural models are at predicting the next token and how accurately the N-Gram model can predict the next word. After having trained the models on three different datasets, we used the test split of each of those datasets to evaluate our final neural models on it. As the datasets varied in size, we report the mean accuracy and perplexity weighted by the number of datapoints in the test datasets. More details about the test datasets can be found in Appendix A - Table 5. For the N-Gram, the test set consisted of the final 15% of the sentences on each dataset.

Model	Training Strategy	Training Time (hour)	Accuracy	Perplexity
N-Gram	Word Counts	1	7.936%	1384.1467
GRU	Artificial Padding	2	**12.982%**	741.0349
Transformer	Artificial Padding	41	12.959%	**639.1329**

Table 2: Accuracy and perplexity on the test datasets. As the datasets varied in size, the reported values are weighted averages.

The results obtained from this experiment demonstrate the strengths and weaknesses of the different models. The N-Gram, with its approach relying on word counts, obtained the highest perplexity among all models suggesting that while it is a simple and quick model to train, it struggles to capture complex dependencies in language. On the other hand, the GRU and Transformer models, which are designed to capture long-term dependencies within text, performed significantly better in terms of perplexity. In particular, the Transformer model achieved the lowest score, indicating its superior ability to anticipate words given their context. This could be attributed to the model's attention mechanisms.

The accuracy results further reinforce the differences in model efficacy. Although the GRU performed slightly better than the Transformer, this could be due to the fact that it was trained with all of the Twitter train data, whereas the Transformer was not. We believe that further training on the Transformer would improve it's performance and allow it to outperform the GRU model, as that dataset was also the largest used in this work.

5.4 Experiment 3: User Experience and Character Savings

Finally, we tested how many keystrokes a user can save when using our models. We randomly selected 15 sentences from the dataset with mobile messages and manually typed them in using our GUI. As soon as the word appeared among five of the predictions, we selected it. The length of the sentences varied between 3 and 24 words. We report the number of keystrokes that could be saved through the suggestions of our models, as well as the percentage that they represent out of the total number of characters.

Model	Saved Keystrokes	Percentage of Saved Keystrokes
N-Gram	367	53.035%
GRU	371	53.613%
Transformer	**391**	**56.503%**
iPhone	456	65.896%

Table 3: Number of saved keystrokes per model.

As can be seen in Table 3, among our implemented models, the Transformer performed best with almost three percentage points difference to the N-Gram and GRU model which performed at a similar level. Even though the word predictor of the iPhone is more powerful, our results show that also more basic, statistical models can help users to save a fair amount of keystrokes when typing. However, the fact that a more sophisticated model as the one included in the iPhone do not reach more than 70% of saved keystrokes reflects the complexity of natural languages.
It is also worth mentioning that while typing we had the feeling that the models we implemented were not able to keep track of long-term dependencies. For example, when typing a sentence about bikes or cars, words related to those topics did not appear under the suggestions more frequently. However, the word predictor on a modern iPhone seemed to be able to adapt its suggestions depending on the current topic of the sentence. This, of course, was only a subjective impression, which we did not have the time to verify through further empirical experiments.

6 Conclusion

This study provides insights into the practical application of word prediction models, showing significant variance in performance and complexity across the three models tested. Overall our results fit well with what we have learned about the models during the course. The statistical approach of training an N-Gram model was outperformed by the neural models. However, considering the short training time, its performance was still decent and at test time could save the user more than half of the characters. As expected, the Transformer model performed best, showcasing the power of the self-attention mechanism underlying it. However, a look at the training times shows that the price one has to pay for that is mostly computationally cost, making the GRU the most efficient of our three models. These findings underscore the importance of model selection based on specific application needs and available resources. For further work we would have liked to further train the Transformer model and to look into ways to optimize the training time.

A Appendix: Datasets

Name	Information	Size	Reference
News Summaries	Text data for news	264 MB	[4]
Articles	Medium articles	3.9 MB	[1]
Mobile Text	Corpus of mobile messages	932.2 MB	[7]
Twitter	Tweets data	583.1 MB	[3]

Table 4: Description of datasets used in the project

Dataset	Train Size	Validation Size	Test Size	Proportion
Articles	559893	34993	104979	0.00437402
News Summaries	39850470	2490654	7471963	0.3113221
Twitter	87593636	5474664	16423744	0.6843039

Table 5: Distribution and proportion of training, validation, and test data across the datasets used in the second experiment

B Appendix: Data Reading

Data used	Chunk Size	Artificial Padding	Thread Active	Time (seconds)	Memory (GB)
News Summarization	1,000,000	No	No	528.20	6.44
News Summarization	1,000,000	Yes	No	571.33	6.93
News Summarization	2,000,000	Yes	No	561.67	6.95
News Summarization	500,000	Yes	No	562.89	6.95
News Summarization	1,000,000	Yes	Yes	546.93	6.86
News Summarization and Twitter	1,000,000	Yes	Yes	1,102.74	13.61
News Summarization and Twitter	1,000,000	Yes	No	1,674.06	20.01

Table 6: Performance results for different chunk sizes, padding options, and threading configurations. The dataset sizes are 264 MB for News Summarization and 551.9 MB for Twitter

C Appendix: Model Architectures

```
==========================================================================
Layer (type:depth-idx)                        Param #
==========================================================================
├─PositionalEncoding: 1-1                     --
|    └─Dropout: 2-1                           --
├─TransformerEncoder: 1-2                     --
|    └─ModuleList: 2-2                        --
|    |    └─TransformerEncoderLayer: 3-1      83,008
|    |    └─TransformerEncoderLayer: 3-2      83,008
|    |    └─TransformerEncoderLayer: 3-3      83,008
├─Embedding: 1-3                              1,953,408
├─Linear: 1-4                                 1,983,930
==========================================================================
Total params: 4,186,362
Trainable params: 4,186,362
Non-trainable params: 0
==========================================================================
```

Figure 5: Architecture of Transformer model

Parameter	Value
Batch Size	64
Hidden Size	512
Embedding Size	50
Number of Layers	3
Number of Attention Heads	8
Learning Rate	0.001
Dropout Rate	0.1
Sequence Length	5

Table 7: Hyperparameters of Transformer model

```
==========================================================================
Layer (type:depth-idx)                        Param #
==========================================================================
├─Embedding: 1-1                              1,526,100
├─GRU: 1-2                                    47,232
├─Linear: 1-3                                 1,983,930
==========================================================================
Total params: 3,557,262
Trainable params: 3,557,262
Non-trainable params: 0
==========================================================================
```

Figure 6: Architecture of GRU model

Parameter	Value
Batch Size	64
Hidden Size	64
Embedding Size	50
Number of Layers	2
Learning Rate	0.001
Drouput Rate	0.5
Sequence Length	5

Table 8: Hyperparameters of GRU model

References

[1] Hsankesara. Medium articles. [Internet] 2018 [cited 2024 Jun 2]. Available from: `https://www.kaggle.com/datasets/hsankesara/medium-articles`.

[2] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.

[3] Ayako Nagao. Predicting next words using n-gram. [Internet] 2023 [cited 2024 Jun 2]. Available from: `https://rpubs.com/NAyako/1036093`.

[4] Sbhatti. News summarization. [Internet] 2018 [cited 2024 Jun 2]. Available from: `https://www.kaggle.com/datasets/sbhatti/news-summarization`.

[5] Miika Silfverberg, I Scott MacKenzie, and Panu Korhonen. Predicting text entry speed on mobile phones. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 9–16, 2000.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[7] Keith Vertanen and Per Ola Kristensson. Mining, analyzing, and modeling text written on mobile devices. *Natural Language Engineering*, 27(1):1–33, 2021.