DEEP LEARNING-BASED PREPROCESSING TOOLS FOR TURKISH NATURAL LANGUAGE PROCESSING

by

Buse Ak

B.S., Computer Science and Engineering, Sabanci University, 2021

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2024

ACKNOWLEDGEMENTS

Firstly, I would like to express my gratitude to my thesis advisor Tunga Güngör for invaluable guidance and support. I am sincerely thankful for the opportunities to learn and grow under his supervision.

I extend my gratitude to my family and friends for their unwavering support during my thesis. Their encouragement, understanding, and belief in me have been a constant source of strength.

Lastly, I would like to thank to my grandfather Muzaffer Taşkın for a lifetime of support, and enduring wisdom, which continue to inspire and drive my achievements, even in his absence.

ABSTRACT

DEEP LEARNING-BASED PREPROCESSING TOOLS FOR TURKISH NATURAL LANGUAGE PROCESSING

As the demand for effective natural language processing (NLP) applications in Turkish continues to rise, the need of text preprocessing tools tailored to Turkish increases. The text preprocessing tools developed play a significant role in extracting information from raw data. As an initial step for any natural language application, these tools enhance the efficiency of complex tasks such as text summarization, question-answering and machine translation.

This thesis describes the development and evaluation of deep-learning based preprocessing tools specifically designed for Turkish. The developed deep-learning based tools tackle the issues of existing rule-based approaches for Turkish. In this thesis, preprocessing tools are developed with Long Short-Term Memory (LSTM) networks and transformers. The main advantage of these architectures is that they are able to adapt to the sequential flow of the natural language effectively.

In this thesis, we investigate the performance of LSTMs and transformers architectures on Turkish preprocessing tasks including tokenization, sentence splitting, deasciification, vowelization, part-of-speech (POS) tagging, spelling correction and morphologic analysis-disambiguation as possible components of a preprocessing pipeline.

ÖZET

TÜRKÇE DOĞAL DİL İŞLEME İÇİN DERİN ÖĞRENME TABANLI ÖN İŞLEME ARAÇLARI

Türkçede etkili doğal dil işleme (NLP) uygulamalarına olan talep artmaya devam ettikçe Türkçeye özel metin ön işleme araçlarına olan ihtiyaç da artıyor. Geliştirilen metin ön işleme araçları, ham verilerden bilgi çıkarılmasında önemli bir rol oynamaktadır. Herhangi bir doğal dil uygulamasının ilk adımı olan bu araçlar, metin özetleme, soru cevaplama ve makine çevirisi gibi karmaşık görevlerin verimliliğini artırır.

Bu tezde Türkçe için özel olarak tasarlanmış derin öğrenme tabanlı ön işleme araçlarının geliştirilmesi ve değerlendirilmesi anlatılmaktadır.

Geliştirilen derin öğrenmeye dayalı araçlar, Türkçe için mevcut kural tabanlı yaklaşımların sorunlarını ele almaktadır. Bu tezde, Uzun Kısa Süreli Bellek (LSTM) ağları ve transformers ile ön işleme araçları geliştirilmiştir. Bu mimarilerin en büyük avantajı doğal dilin sıralı akışına etkili bir şekilde uyum sağlayabilmeleridir.

Bu tezde, bir ön işlemenin olası bileşenleri olarak tokenizasyon, cümle bölme, Türkçe karakterleştirme, sesli harf restorasyonu, part-of-speech(POS) etiketleme, yazım düzeltme ve morfolojik analiz-belirsizliği giderme gibi Türkçe ön işleme görevlerinde LSTM'lerin ve transformers mimarilerinin performansı detaylıca incelenmiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . ix
LIST OF TABLES . . . x
LIST OF SYMBOLS . . . xiii
LIST OF ACRONYMS/ABBREVIATIONS . . . xiv
1. INTRODUCTION . . . 1
2. BACKGROUND . . . 4
2.1. RNNs . . . 4
2.2. LSTMs . . . 7
2.3. Encoder-Decoder Models . . . 8
2.4. Transformers . . . 10
2.4.1. BERT . . . 11
2.4.2. CANINE . . . 14
2.4.3. T5 . . . 16
2.4.4. mT5 . . . 17
3. RELATED WORK . . . 19
3.1. NLP PIPELINES . . . 19
3.1.1. ITU NLP PIPELINE . . . 19
3.1.2. UDPipe . . . 21
3.2. Other Works . . . 23
4. DATASET . . . 27
4.1. BOUN Treebank . . . 28
4.2. IMST Treebank . . . 29
5. METHODOLOGY . . . 30
5.1. HuggingFace Transformers Library . . . 30
5.2. Tokenizer . . . 32

5.2.1. Dataset Preparation 32
5.2.2. Bi-LSTM Tokenizer 32
5.2.3. CANINE Tokenizer 34
5.3. Sentence Splitter 37
5.3.1. Bi-LSTM Sentence Splitter 37
5.3.2. CANINE Sentence Splitter 37
5.4. Deasciifier 38
5.4.1. Dataset Preparation 38
5.4.2. Bi-LSTM Deasciifier 38
5.4.3. CANINE Deasciifier 39
5.5. Part-of-speech Tagger 40
5.5.1. Training Dataset Preparation 40
5.5.2. BERT Pos-Tagger 41
5.5.3. mT5 POS Tagger 41
5.6. Vowelizer 42
5.6.1. Dataset Preparation 42
5.6.2. CANINE Vowelizer 44
5.7. Spell Corrector 44
5.7.1. Dataset Preparation 45
5.7.2. CANINE Spell Corrector 47
5.7.3. mT5 Spell Corrector 49
5.8. Morphologic Analyzer-Disambiguator 50
5.8.1. Dataset Preparation 51
5.8.2. mT5 Morphologic Analyzer-Disambiguator 52
6. EXPERIMENTS AND RESULTS 55
6.1. Tokenizer Results 55
6.1.1. Bi-LSTM Tokenizer 55
6.1.2. CANINE Tokenizer 56
6.2. Sentence Splitter Results 57
6.2.1. Bi-LSTM Sentence Splitter 57
6.2.2. CANINE Sentence Splitter 58

6.3. Deasciifier Results 59
6.4. Vowelizer Results 60
6.5. POS Tagger Results 61
6.6. Spell Corrector Results 62
6.7. Morphologic Analyzer-Disambiguator Results 65
7. CONCLUSION 67
REFERENCES 69
APPENDIX A: DEASCIIFIER MISTAKES 74
APPENDIX B: VOWELIZER MISTAKES 79
APPENDIX C: SPELL CORRECTOR MISTAKES 84

LIST OF FIGURES

Figure 2.1. Unfolded RNN. 5

Figure 2.2. Sequence labeling task with BERT. The *softmax* function is applied to the output of the BERT for each token. The labels are obtained by applying *argmax* function to the output of the softmax. 14

Figure 2.3. CANINE model. For the sequence labeling task, the upsampled character representations are fed to a single transformer to obtain the final character embeddings. 16

Figure 4.1. Example sentence taken from BOUN Treebank [28] test set. The Features and Relation fields are not shown since they are not used in this thesis. 28

Figure 5.1. Tokenization of the sentence "1936 yılındayız.". 33

Figure 5.2. Training of Bi-LSTM Deasciifier. 39

LIST OF TABLES

Table 5.1. Example training dataset instance. 34

Table 5.2. Model configuration parameters. 35

Table 5.3. Training hyperparameters. 36

Table 5.4. Example input and output sentences from training data. 39

Table 5.5. The vowels to be restored. 43

Table 5.6. Example modification of the sentences for training dataset preparation. 43

Table 5.7. Labeling of the word ”rüzgâr” with missing vowels. 44

Table 5.8. Common spelling mistakes. 45

Table 5.9. The possible character insertion/substitution candidates for each character. 47

Table 5.10. Different interpretations of the word ”kazı” according to the context. 51

Table 5.11. An example sentence and morphological parse of each word. 52

Table 6.1. The results of developed tokenizer models. 55

Table 6.2. Tokenizer results. 56

Table 6.3. The sentence splitter model gives best results on Wikipedia paragraphs containing sentences with punctuations at the end. 58

Table 6.4. CANINE Deasciifier results. 60

Table 6.5. Vowelizer results. 60

Table 6.6. Comparison of the vowelizer with previous studies. 61

Table 6.7. Spell corrector results. 63

Table 6.8. Token-based F1 scores of the spell correction models on different error types. 64

Table 6.9. Comparison of the spell corrector with previous studies. 65

Table 6.10. Morphological analyzer results. 66

Table 6.11. Comparison of the morphological analyzer with previous studies. . 66

Table A.1. Deasciifier mistakes on first batch of Wikipedia sentences. 74

Table A.2. Deasciifier mistakes on second batch of Wikipedia sentences. 76

Table A.3. Deasciifier mistakes on tweets. 78

Table B.1. Vowelizer mistakes on first batch of Wikipedia sentences. 79

Table B.2. Vowelizer mistakes on second batch of Wikipedia sentences. 81

Table B.2. Vowelizer mistakes on tweets. 83

Table C.1. Spell corrector mistakes on second batch of Wikipedia Sentences. . 84

Table C.2. Spell corrector mistakes on third batch of Wikipedia Sentences. . . 86

Table C.3. Spell corrector mistakes on tweets. 88

LIST OF SYMBOLS

c_i	Context vector in decoding step i
t	Current timestep
h_t	Hidden layer at timestep t
h_{t-1}	Hidden layer at timestep $t-1$
h_{t+1}	Hidden layer at timestep $t+1$
x_t	Input at timestep t
y_t	Output at timestep t

LIST OF ACRONYMS/ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
Bi-LSTM	Bidirectional Long Short-Term Memory Network
LSTM	Long Short-Term Memory Network
RNN	Recurrent Neural Networks
T5	Text-to-text Transformer
mT5	Multilingual text-to-text transformer

1. INTRODUCTION

Text pre-processing is an essential step for any natural language application. Natural Language Processing (NLP) applications are observed to produce more successful results on structured and standardized text. There exist many studies on the improvement and development of pre-processing applications in languages such as English. However, the development of NLP tools is a challenging problem in complex and low-resource languages including Turkish. The Turkish language is an agglutinative language with rich morphological properties. The rich usage of affixes increases the number of words derived from the same root which makes the NLP applications for Turkish language more difficult.

A respectable amount of the state-of-the-art methods for Turkish pre-processing such as tokenization, sentence splitting, and normalization, are rule-based approaches that require language experts to build a system. However, the rule-based systems may not perform well where the context within a sentence must be captured. The research progress in Turkish NLP applications including text pre-processing is accelerated with the improvements in deep learning methods. Therefore, this study proposes deep learning-based Turkish text pre-processing tools for NLP researchers working on Turkish. The proposed pre-processing components are tokenizer, sentence splitter, deasciifier, vowelizer, part-of-speech (POS) tagger, spell corrector, and a combined morphologic analyzer-disambiguator.

The NLP tasks such as text summarization and question-answering systems usually process documents or paragraphs. These large texts need to be split into smaller components to better extract the semantic and syntactic information of the text. The proposed deep-learning-based sentence splitter tool outputs the list of sentences contained in the large texts. The proposed tokenizer tool extracts the sub-units, usually words, of a sentence. The tokenization process is important mainly since it enables machine comprehension of natural language by breaking a sentence down into smaller,

more manageable chunks for analysis.

The ASCII is a character encoding scheme to represent text as code points where each character corresponds to a unique number. The ASCII scheme has limitations on encoding text belonging to languages such as Turkish. For instance the letter "ş" does not have a ASCII code-point correspondent. Since, the Turkish characters might not be compatible with some keyboards and or systems supporting only ASCII characters; in this thesis, a deep learning-based deasciifier tool is proposed to convert ASCII characters into their Turkish non-ASCII equivalents. For instance, when the letter "ş" is not applicable on a keyboard or a system the letter "s" is used to replace "ş". By developing a deasciifier tool, this problem is addressed.

The written communication records such as text messages, blog posts and social media entries are significant resources for NLP researchers. However, most of these texts contain misspellings or daily-life word abbreviations such as "slm" meaning "selam". Since the correct form of the word is needed to grasp the meaning accurately, a vowelizer system and a spell corrector are built in this thesis to overcome this issue. The proposed vowelizer tool is developed to restore the correct form of a word with missing vowels. The proposed spell corrector is developed to detect and fix missing letters, letter duplication, letter addition, and wrong letter order within a word.

The proposed part-of-speech tagger tool tags each word depending on the context of the sentence containing the target word. The tool categorizes the words within a sentence such as noun, verb, or adjective to indicate the part-of-speech of each word.

In this thesis study, we propose a combined morphological analyzer-disambiguator. The purpose of a usual morphological analyzer is to generate all potential morphological parses of the words in a sentence. Morphological disambiguation differs from morphological analysis in that it only selects and outputs the proper morphological parse of the word depending on the context. In this thesis, the morphologic analysis and morphologic disambiguation are applied in a single step. The proposed tool outputs the

most likely morphologic features of the words of the input sentence where the need for disambiguation is eliminated by selecting the features considering the context of the sentence.

The tokenizer, sentence splitter and deasciifier tools have been developed using both Bi-LSTM and transformers. On the other hand, vowelizer, POS tagger, spell corrector and morphologic analyzer-disambiguator are transformers-based. In this thesis, the evaluation and analysis of the trained models' performance results are discussed and compared with state-of-the-art in detail.

Lastly, it is valuable to reach preprocessing tools via an open-source website that is available for researchers. Therefore, the tools built in this study are planned to be included in the Boğazici University Text Preprocessing Pipeline developed by Text Analytics and Bioinformatics Lab (TABILAB). [1]

This thesis is structured as follows: section gives a background on related deep-learning architectures. The related works on text pre-processing tools are reviewed in section three. Section four explains the structure and properties of used datasets in the training and evaluation of the models. The training methodologies are explained in section five. The conducted experiments and their results are demonstrated in section six. Lastly, section seven provides a general overview of the study and discussion on possible further developments.

2. BACKGROUND

This chapter provides background information on related frameworks and architectures.

2.1. RNNs

The meaning of a word depends on its surroundings which builds the context of the sentence. A word may be interpreted in many ways without considering the context. The feed-forward networks process the words by making independent decisions on past and next words. However, the sequential nature of the natural language requires taking into account the related units of the text. The recurrent connections handle a word by being aware of the past words, which enables it to capture the information and the context as a whole.

The RNN is a cyclic network where each unit of the network is conditioned on the outputs of the previous units. Contrary to the feed-forward networks, in recurrent networks, each unit of the network produces a hidden layer to be the input of the next unit. Differing from the window-based approach, each unit of the RNNs processes a word at a time step which ensures the sequential approach of the network. The difference between the feed-forward and the recurrent networks lies in the hidden layer which transfers the output of the previous time step to the next time step as an input with the next word itself.

The hidden layer, as shown in the Figure 2.1, ensures the creation of a memory and transfers the memory along the network. A type of memory, or context, that encodes past processing and influences decisions to be made at later times is provided by the hidden layer from the prior time step. In the training phase, the recurrent neural network learns how to make use of the past context by adjusting the weights of the hidden layer output. The unfolded version of the RNN architecture is shown in

Figure 2.1. At each time step (t) the hidden layer (h_t) is computed depending on the previous hidden layer(h_{t-1}) and the current input (x_t).

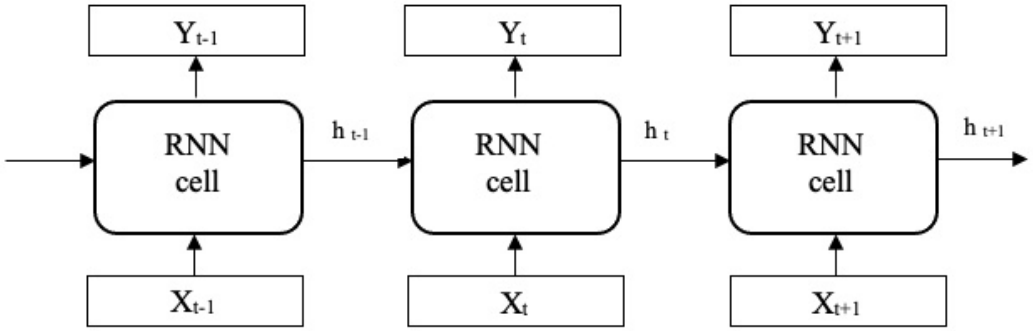


Figure 2.1. Unfolded RNN.

In the classification task, the output vector (y_t) is computed by applying the softmax function on the hidden layer (h_t) to get the probability distribution over the output classes. The obtained hidden layer (h_t) is fed into the next unit as input. The weights of the hidden layers, and inputs are the same at each time step.

In the training phase of the recurrent networks, the gradients that are required to modify the weights acquired are back-propagation, a training set, and a loss function. The loss function is applied to the output produced at time t and the hidden layer of time *t-1*. To compute the error for obtaining the hidden layer at time t (h_t), the impact of the (h_t) on the output of the current time step must be acquired. The weights are trained by the back-propagation operation. After the forward pass, the hidden layer at time t and the output at time t are calculated with loss at the current time step. To apply back-propagation, the pass starts to traverse the sentence backward. The gradients are calculated at each time step in back-propagation to obtain the error for each hidden layer.

The RNNs are among the most used architectures for the language modeling task. After inputting a word, the current unit produces a probability distribution over the entire vocabulary. Next, the word with the maximum probability is chosen as the next word after the current word. By using the memory produced by the past hidden layer

at each time step the RNN generates a sequence until the end of the sentence token is reached. The model is trained to minimize the cross-entropy loss which evaluates how the predicted distribution of the model differs from the correct distribution provided by the gold data. At each time step, the loss is calculated by the applying cross-entropy algorithm. The training adopts the teacher forcing approach where the loss of each step is independent of each other since the gold word that timestep is given as the input to the current timestep instead of what the model predicts. The weights that give the minimum average Cross Entropy loss are selected in the training phase of the RNNs.

The inputs of the RNNs are word embeddings which are created by the embedding layer or obtained from external pretrained embeddings. The word embeddings represent the words in vector space where the dimensions of the vector specify a feature of the word (Thus, similar words have similar word embeddings). The embedding matrix is in a form where each column corresponds to the embedding of each word in the vocabulary. Therefore, the output layer matrix has the same dimensions as the embedding matrix to be able to apply softmax on each column to get the most probable next word. Apart from the language modeling task, RNN architecture is used in sequence labeling, text classification, and text generation.

In the language modeling task, the output of each RNN unit was the most probable word in the vocabulary. In the sequence labeling task, the RNN aims to select the most probable class among the given classes for each unit of the given input text. This task can be predicting the Part-of-speech tag or Named Entity Recognition. The sequence labeling with RNNs, accepts word embeddings as inputs and outputs the class probabilities generated by the softmax function applied on each output layer. In sequence labeling, the cross-entropy loss function is used to be utilized with the softmax algorithm.

In the sequence classification task, instead of producing output at each timestep, the produced hidden layers at each timestep are passed until the last unit of the net-

work is processed. In the last unit, the hidden layer is output to be applied to the softmax function over the possible tags of the sequence. In sequence classification, since there are no outputs at each timestep, the loss function is only dependent on the final classification to adjust the weights.

The RNN architectures may consist of stacks or may have bi-directions. Stacked RNNs are formed by stacking RNN networks on top of each other to produce the output. The lower layer's output is used as input by the RNN network in the higher levels. The last layer of the stacked RNN produces the final output. The lower layers of the stacked RNN capture the features roughly and in each layer the information obtained from the input text increases. In a single-layer RNN network, the network may be able to extract the obvious features of the input text while missing some details. That's why stacked RNNs generally outperform RNNs with a single layer.

The RNN network processes the input text in a left-to-right fashion. Since the entire sequence is accessible usually, processing the sequence backward also helps to extract more information at each time step. The Bidirectional RNNs are structured as one RNN layer in one direction. The output is produced by concatenating the hidden layers at each time step. Therefore the output is predicted by acknowledging the left and right context, which provides much more knowledge about the context of a word while predicting.

2.2. LSTMs

Even though the RNNs transfer the hidden layers of the previous network units to the further units, the information kept in the hidden layers fails to represent the features of the whole sequence and remains relatively local. However, the knowledge gained from the beginning of the sentence may have an important role in processing the current word. The RNNs have difficulty transferring significant knowledge to the distant steps, because considering the information carried by each hidden layer is equally important for the current time step, and the vanishing gradients problem in the backward pass is

caused by the need for many multiplications while processing long sequences.

The Long Short-Term Memory(LSTM) [2] Networks are developed to tackle these issues of the RNNs while preserving the recurrent structure. LSTMs transfer the information gained from the past context by deciding the relevancy of that information for processing further inputs. In addition to RNNs, LSTMs introduce a context layer containing gates that are responsible for keeping or forgetting the knowledge. The input gate determines how much of the new knowledge is going to be kept in the context vector by looking at the current input and the previous hidden layer. The forget gate prevents the information from being transferred if the information is irrelevant to the next steps. Finally, the output gate decides how much of the new information obtained from the context vector will be used in creating the new hidden layer which will be used in the next step.

The advantages of LSTMs include capturing the information on long sequences where long-term dependencies are significant for each word. To manage gradients and information flow, LSTM makes use of gating mechanisms. In addition to helping to avoid the vanishing gradient issue, this enables the network to learn and remember data over longer sequences. The application areas of the LSTMs include language modeling, machine translation, sequence labeling, and sequence classification. LSTMs are used in tasks involving sentiment analysis, where the network is trained to identify and categorize the sentiment contained in a text. Today, LSTMs are standard architecture in NLP and sequential data processing because they can capture long-term dependencies and solve issues with conventional RNNs. Their extensive use in both research and practical implementations can be attributed to their success in a variety of applications.

2.3. Encoder-Decoder Models

The RNNs producing output at each time step require the output sequence to have the exact same amount of words as the input sequence. The encoder-decoder

model is used to eliminate the need for word-to-word correspondence between the input and the output sequence. The encoder-decoder architecture provides the flexibility for a variety of different lengths of the output sequence. In machine translation, a sentence may be expressed in 5 words in English while its translation to Turkish may have 10 words. The input word at time step t may not have a direct relationship with the output word at time step t because of the word order differences of the languages and representing information in different amounts of words. Since the encoder-decoder architecture allows the independent lengths of the input and output sequences, it is one of the most used architectures in machine translation, summarization, and question-answering tasks. Encoder-decoder networks, also known as sequence-to-sequence networks, are models that, given an input sequence, may produce output sequences of any length that are appropriate for a given context.

The sequence-to-sequence model consists of two networks: encoder and decoder. The encoder network is an RNN network that is responsible for taking the input sequence and creating the representation of the input sequence with its context. The output of the encoder network is called the context vector containing information extracted from the input sequence. The context vector is used as input by the decoder network. The decoder generates an arbitrary length of units which creates the output sequence. The RNN version of the encoder-decoder architecture operates the machine translation by concatenating the source and target sequences with a token responsible for sequence separation such as $< s >$. The decoder uses the information of the context vector which is the last hidden layer of the encoder. This context vector is utilized as a previous hidden layer in the first time step of the decoder.

The training phase of the sequence-to-sequence architectures is handled in an end-to-end fashion. The training and inference phases differ in how the output at each time step is evaluated. In sequence-to-sequence models, the decoder's outputs are used for the loss function. In the training the teacher-forcing approach is adopted for better training performance; instead of the predictions of the decoder at each time step, the gold word is fed to the next decoder unit.

2.4. Transformers

The transformer [3] architecture is one of the most common architectures for language modeling. The novel techniques provided by the transformer (self-attention and positional encodings) enable focus on how words relate to one another over large distances and help depict time. Since the RNNs are based on recurrent connections, effective parallelization of the operations may not be possible. The transformers are convenient for parallelizing the network and providing much more efficient training and inference. The transformers consist of stacked transformer blocks where the blocks are composed of linear, feedforward, and self-attention layers.

While processing an input item, the model accesses the previous input items. The self-attention approach enables capturing the relevance of the input items till the current item by assigning different weights to them. The self-attention process utilizes 3 input embeddings with different roles: query, key, and value. The query vector takes part as the current embedding on which the attention focuses. The key vector which represents the prior inputs is compared to the query vector. The value vector is utilized to calculate the attention weights of the query vector. The query, key, and value vectors have different weights applied on each input embedding to represent as query, key, and value. Since self-attention requires comparing each input as pairs, the computational sources can be overutilized. Thus, most of the state-of-the-art transformers models limit the input sequence length.

Besides the self-attention layer, the transformer involves feedforward layers, residual connections, and normalization layers. The transformer blocks can be combined as stacked RNNs since the dimensions of their inputs and outputs are the same. The training performance of the transformers is increased with the residual connections which are used to enable higher layers to access the pure information of the lower layer by passing the output of the lower layer to a higher layer without being processed by the layer in-between them. The layer normalization is another mechanism for better training efficiency by applying standardization. Furthermore, another novelty of

the transformers architecture is the multi-head attention. The different elements of a sequence may have various kinds of relationships with each other. The multi-head self-attention mechanism is utilized to overcome the problem of a single transformer block trying to learn different dependencies of the input words. Multi-head attention is represented by groups of self-attention layers each being a head in a parallel fashion. Each self-attention head focuses on capturing the distinct links among inputs at the current input level.

The word order plays a significant role in the meaning of a sentence. Unlike RNNs, the transformer architecture does not have a sequential flow of processing the inputs. Because of the transformers' non-sequential structure, to understand the word order, positional embeddings are used. The input to the transformer blocks is a combined vector of the word embeddings and the positional embeddings. The positional embeddings are randomly initialized and learned during the training phase.

The transformers select the word with the highest probability at each time step in a greedy manner which chooses the local optimal word. The greedy search may not provide the global optimum since it does not consider changing its choice in the later steps. In later steps, the chosen word may not be as meaningful as it was in the local context. This problem can be solved by applying a beam search.

The beam search does not decide on an output word directly at a time step. Instead, searches for the top k ways to produce the output word at each time step and selects the most probable sequence.

2.4.1. BERT

The transformer models proposed by Open AI are causal by addressing a word with the previously occurring words in a left-to-right manner. Devlin et al. [4] introduced the bidirectional transformer architecture called BERT in 2018. The main aim of the BERT was to overcome the major limitations of the unidirectional transformer

models. These limitations include fine-tuning and pre-training.

The pre-training step is where the model learns the meaning of the words and represents them in the vector space. The general aspects and rules of a language given large amounts of text are extracted in the pre-training step. The pre-trained model performs better in basic tasks with further training. The further training process that applies minimal adjustments to the architecture such as adding a feed-forward layer on top of the final layer, is called fine-tuning.

Even though the causal models are performing well on text generation and machine translation, they have difficulties with sequence labeling tasks such as NER because a NER tag may also depend on the right context. The BERT architecture eliminates this issue by the self-attention mechanism which considers the entire sequence. Therefore the output embeddings are based on the contextual information obtained from the whole sequence. As in classical transformer networks, blocks of the BERT contain a self-attention layer, normalization layers, and residual connections. The BERT model in the paper by Devlin et al., contains a sub-word vocabulary produced by the WordPiece algorithm proposed by Schuster and Nakajima [5] in 2012, hidden layers with a dimensions of 768, and 12 transformer blocks with 12 multi-head attention layers. The fixed input size of the BERT is 512 sub-word tokens.

The training of the BERT model applies the close task where the model tries to predict a word given its left and right context together. This training process of the BERT model is called Masked Language Modelling(MLM). Devlin et al, state that a 15% proportion of the input tokens are used for training and the remaining tokens are only used in the self-attention process. In those sampled tokens, 80% are replaced with [MASK] token, 10% are replaced with a random token and the remaining is unchanged. The goal of this training approach is to find the most possible word for the masked tokens. The cross entropy loss is applied to the predicted tokens which corresponds to the sampled tokens, in the training phase.

The BERT model requires a subword tokenization step. The input to the model is the combination of the token embeddings and positional embeddings. An output vector containing the probability distribution over the subword vocabulary is produced by BERT for the masked tokens. The cross-entropy loss is calculated for each masked token in the training step and the weights are modified according to the average loss of the selected training tokens.

Another approach BERT introduces is Next Sentence Prediction (NSP) which enables to identification of the relationship between sentence pairs. For NSP training, BERT uses [CLS] and [SEP] tokens to separate input sentence pairs. In the training, BERT learns to recognize the correct sentence pairs and mocked pairs.

By using a pre-trained BERT model, the contextual embeddings can be obtained given an input sentence. The contextual embedding of an input token contains the meaning of the word in the context of the sentence which can be used for linguistic tasks.

Most of the state-of-the-art results of the common tasks such as sequence classification, sequence labeling, and question answering are achieved using fine-tuned BERT models or the descendants of the BERT. In this thesis process, a sequence labeling study is conducted with a fine-tuned BERT model pre-trained on a Turkish corpus.

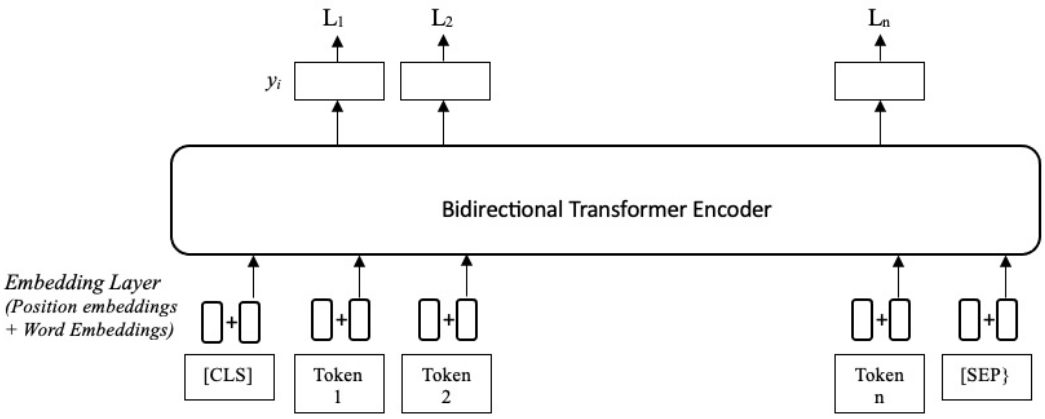


Figure 2.2. Sequence labeling task with BERT. The *softmax* function is applied to the output of the BERT for each token. The labels are obtained by applying *argmax* function to the output of the softmax.

The Figure 2.2 shows the BERT fine-tuned for the sequence labeling task. In the sequence-labeling task, the model assigns a tag for each token. The word embedding combined with the positional encoding of a token is given as the input to the BERT model. The model produces a distribution over the tag set. As in RNNs, the most probable tag can be found by applying the *argmax* function to the distribution. Since BERT uses WordPiece tokenization, the sub-words need to be tagged by the model as well. In some tasks such as named entity recognition, the tagging of sub-words may be an overwork. Some studies suggest ignoring the sub-word tagging to decrease the workload.

2.4.2. CANINE

Most of the architectures like BERT require a tokenization step. Even if the vocabulary of the model is large, any set lexicon can restrict a model's capacity for flexibility and adaptation. Clark et al. [6], proposed a novel character-based model that does not require an explicit tokenization step or a lexicon in 2022. The existing limitations of the tokenization algorithms such as Byte-Pair Encoding, WordPiece, and SentencePiece are overcome by the CANINE model's character-based architec-

ture. These limitations include the computational complexity caused by expanding the amount of the input tokens drastically as a consequence of splitting them into subwords. Furthermore, the common tokenization algorithms do not apply to each language or text since the rules of the language may not be suitable or the input text may be in informal format where they contain spelling errors and internet phrases that do not occur in formal texts. Clark et al., state that despite the common tokenization algorithms' high performance in English, these algorithms do not function well in languages with agglutinative morphology, non-concatenative morphology, etc. These aspects of the tokenization step could lead to difficulties during training on some tasks as stated in the paper.

There are three main parts of the CANINE model. First, the text embedding technique which requires no vocabulary, creates character embeddings based on the Unicode IDs of the characters combined with a hashing strategy. The character-based hashing strategy adopted by the CANINE model prevents different Unicode IDs from having the same representation. Also in fine-tuning, new character embeddings can be learned for the characters that were not present in pre-training.

The character embeddings are processed by a single local transformer to obtain contextual character embeddings. The obtained contextualized embeddings are downsampled with a strided convolutional layer with the objective of a more efficient model. The output of the downsampling is given as input to a deep-transformer stack like BERT. An upsampling stage is required to be applied to the output of the transformer stack for the sequence prediction task where the output sequence must have the same size as the input sequence. The upsampled vector is processed by a single transformer which produces the final character representations.

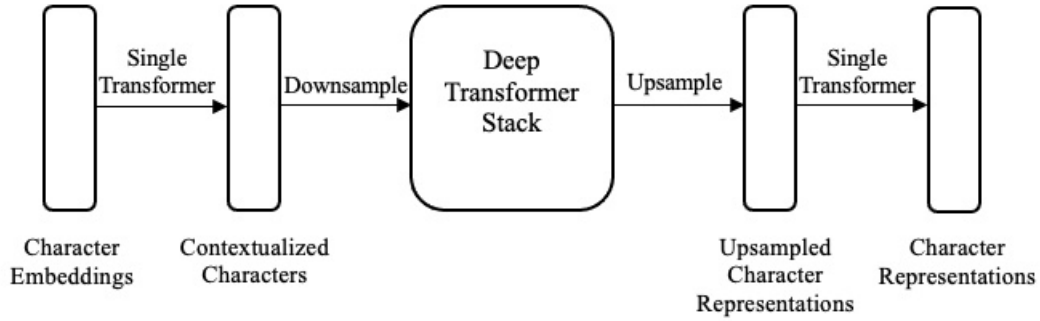


Figure 2.3. CANINE model. For the sequence labeling task, the upsampled character representations are fed to a single transformer to obtain the final character embeddings.

The Deep Transformer Stack has 3 transformers and 1 deep transformer which has the same architecture as BERT as shown in Figure 2.3. The maximum sequence length CANINE accepts is 2048 characters. Since the upsampled sequence can have a maximum of 2048 characters, the input sequence must be adjusted to be compatible with this restriction. CANINE has two pre-trained versions based on their loss function as follows: Canine-c and Canine-s. CANINE-c has 12 layers and 12 heads and is pre-trained with autoregressive character loss. CANINE-s has the same number of layers and heads and is pre-trained with sub-word loss. The character-based architecture makes CANINE successful in many languages that do not require pre-training. The evaluation results show that CANINE outperforms another multilingual state-of-the-art model, mBERT on various tasks in different languages.

2.4.3. T5

Google AI [7] developed the transformer-based language model known as T5 or Text-To-Text Transfer Transformer. T5 is a flexible model that frames all NLP jobs as text-to-text challenges, in contrast to standard models created for particular NLP tasks (such as translation or summarization). This implies that every task's input and output are considered as text strings. The model is optimized for particular tasks after being pre-trained on a big corpus of varied data.

The transformer model, on which T5's architecture is built, processes input sequences in parallel by use of self-attention mechanisms. T5 can execute a wide range of activities, including question answering, translation, summarizing, and more, thanks to the "text-to-text" approach, which turns each task into a standard format that maps input text to target text. T5 shows the possibility for a single, highly capable model to cover a wide range of language comprehension difficulties by exhibiting the efficacy of a unified model architecture for numerous natural language processing tasks.

2.4.4. mT5

MT5, or mT5 (Multilingual T5) [8], is an extension of the T5 paradigm designed to handle multilingual activities. MT5 is an extension of the Text-To-Text Transfer Transformer (T5) architecture and is intended to handle various language-related issues with natural language processing. MT5, as a development tool, aims to improve language generation and understanding in contexts with multiple languages.

Both input and output are treated as text sequences by MT5, which uses the same text-to-text approach as T5. As a result, the model can perform a wide range of multilingual tasks, including as translation, summarization, and question-answering. MT5 can identify and infer patterns from a variety of linguistic structures because of its comprehensive pre-training on a broad range of language text content.

The primary benefit of MT5 is its multilingualism, which adds value to the program for tasks involving the production and interpretation of text in multilingual settings. It illustrates how effective solutions for natural language processing across a variety of linguistic situations could be provided by a single, integrated approach.

The mT5 model is only pre-trained on the mC4 dataset which contains multilingual texts extracted from the Common Crawl dataset. Unlike T5, mT5 is trained unsupervised. Therefore for fine-tuning a single task, the model does not demonstrate a significant advantage of using a prefix as T5 needs. The pre-trained tokenizer uses

a SentencePiece tokenization like T5. As stated by Xue et al. [8], the mT5 model outperforms mBERT in question-answering and named entity recognition tasks.

3. RELATED WORK

In this chapter, related works on Turkish pre-processing tools are described in detail.

3.1. NLP PIPELINES

3.1.1. ITU NLP PIPELINE

The Turkish NLP Pipeline of Istanbul Technical University [9] is a web service providing the NLP components are used as a whole pipeline or atomically. The components are grouped under 4 layers: pre-processing, morphological processing, multiword expression handling, and syntactic processing. The pre-processing layer is focused especially on raw social media texts and consists of a tokenizer, a diacritic restorer, a spelling corrector, and a normalizer. The morphological layer consists of a rule-based morphological analyzer, a hybrid morphological disambiguator, and the Turkish component that validates a word. The multiword expression handling layer is a named entity recognizer. The last layer of the pipeline is the syntactic layer which includes a dependency parser.

A two-stage hybrid model consisting of a discriminative sequence classifier and a language validator is suggested for solving the vowelization and diacritic restoration problems [10]. The proposed model is language-independent and provides state-of-the-art results for Turkish. Due to the complexity and rich inflectional morphology of Turkish, the search space of the model gets larger so quickly that the machine learning models perform weakly. Therefore, Eryiğit and Adalı, suggest a Conditional Random Field (CRF) based discriminative sequence classifier that considers vowelization and diacritization as a character-level sequence labeling task. The open-source CRF++ model is trained with sequences containing the information of features related to current and neighboring words and characters, and a class label. The n best sequences of

the discriminative classifier are used in the model's second stage, which is a language validator. The language validator uses a two-stage morphological analyzer instead of a single lexicon-lookup step due to the complexity of Turkish. The language validator chooses the most probable word among the best sequences produced by the discriminative classifier. Such a scenario can exemplify the model: the input Word is "kd", the n best sequence produced by the discriminative sequence classifier is "kedi", "kadı", "kado", "kada", and lastly the language validator outputs "kedi" as the most likely valid sequence.

The spelling corrector of the pipeline uses a unigram language model and an error model jointly for candidate generation [11]. The unigram language model is trained on surface forms of words from the Turkish corpus. The error model is the Aho-Corasick tree structure formed by the rules extracted from the misspelled words and their corrected forms using the Levenshtein edit distance algorithm. The tree consists of string operations and their likelihoods. For the unknown words, Laplace smoothing is used. The candidate word is the word with minimum rule cost and maximum unigram probability. The morphological layer is a two-stage morphological analyzer that is an extension of Xerox and finite-state transducer implementation [12]. A custom morphotactic scheme and lexicon, containing flag diacritics, are used in the morphological analyzer. The evaluation results on METU-Sabancı Turkish Treebank [5] demonstrate that the analyzer is 99.8% accurate. The proposed named entity recognizer is a CRF-based model [6]. The tokenized words are processed by the morphological analyzer component. The morphological features of the word are used as features for the CRF model. The discriminative CRF model splits and tags the input sequence. The tags in the NER task are named entity labels namely person, location, organization, date, time, money, and percentage.

The normalization architecture consists of two stages that are named ill-formed word detection and candidate generation [13]. Before the normalization, input tweets are tokenized. The ill-formed word detection model uses a morphological analyzer and an abbreviation list to obtain the out-of-vocabulary(OOV) words. The filtered OOV

words are sent to the candidate generation stage. Firstly, the words are converted to lowercase according to the classification of proper nouns or abbreviations. The letter casing of proper nouns and abbreviations are kept unchanged. Secondly, slang words, words with character repetitions, twitter specific words such as words containing hashtags and emotion symbols are replaced with their normal forms using lexicon-lookup. Thirdly, the proper nouns written in lowercase are detected and converted to the correct form where the first letter is uppercase and the rest is lowercase, and apostrophes are added if needed. Furthermore, deasciification and spelling correction functions are applied using a replacement rules dictionary. For accent normalization, words are morphologically parsed to look up the normal form from the replacement rules dictionary. The model is evaluated on a test set split from 1200 Turkish tweets. To measure accuracy f1-score and accuracy is utilized. The results obtained for the ill-formed word detection reach an f1-score of 0.78 and an accuracy of 0.80. This depicts that the proposed model provides better almost four times more than the state-of-the-art models with Turkish tweets. In addition to that, the accuracy of the candidate generation model is 0.71 which is claimed as an improvement of almost 40 percent, compared to previous methods.

3.1.2. UDPipe

The UDPipe [14] is a pipeline, developed by Charles University, for various NLP applications such as tokenization, sentence segmentation, part-of-speech tagging, and lemmatization in all 50 languages present in the Universal Dependencies library including Turkish.

The tokenization and sentence segmentation operations are trained together. The model learns sentence and word boundaries by deciding for each character whether that character is an end of a sentence or a word/token. The model assigns a label for each character indicating that the character belongs to whitespace, sentence ending, or a word ending.

The UDPipe tokenizer and sentence segmenter is a bidirectional Long-Short-Term Memory (LSTM) artificial neural network that processes characters of an input text one by one to capture the non-linear and local relationships in a sequence. The input to the segmenter is a predefined fixed-length sequence where the segmenter aims to divide the fixed-length sequence into tokens and sentences. The division operations are done by classifying each character into three categories as no boundary, token boundary, and sentence boundary.

They classify each character by applying softmax on the hidden vectors of the Gated Recurrent Unit (GRU) network.

The fixed-length input text is tokenized by first removing the determined sentence boundaries and going on processing with the remaining characters. If no sentence boundary is found, then the sequence is examined by processing the first and second half of the sequence separately to find and capture the word boundaries. The training dataset consists of concatenating the randomly shuffled sentences into a single long sequence. The experiments of tokenization are evaluated in many languages. The UDPipe tokenizer outperforms the rule-based tokenizer in most languages. The authors highlight that tokenizer works better on the treebanks with space-after property. par The POS tagging model is a multi-layer bidirectional LSTM that processes embedded words working with all three kinds of tagging schemes present in the Universal Dependencies. A softmax classifier is applied to the Outputs of the LSTM layer to obtain the most probable tag for a word. Moreover, to improve the success, a dense layer and a residual connection layer are appended before the softmax classifier.

The lemmatization operation classifies the input words by lemma generation rules. The lemma generation rules are constructed based on the sequence of overlapping probabilities. In addition, the dependency parser is a graph-based biaffine attention parser. The outputs of the LSTM layer, mapped into arc-head and arc-dep representation to be used in biaffine attention which provides the probability distribution over all words for each word, representing the probability of being a dependency head.

The UDPipe tokenizer [15] and sentence segmenter reaches 0.99 and 0.97 F1-scores for, respectively, token-level and sentence-level on Turkish IMST Treebank [16]. The POS tagging model is a multi-layer bidirectional LSTM that processes words with all three tagging schemes in UD. The POS tagging model reaches 0.93 F1-score on the IMST Turkish Treebank for the UPOS field. The joint model obtains 0.89 F1-score for the AllTags metric in ConLL 2018 UD Shared Task [17].

3.2. Other Works

Another normalization study proposed by Göker et al. [18], suggests a sequence-to-sequence architecture with LSTM networks in order to normalize the Turkish social media texts. The proposed model consists of two methods: capturing contextual relationships and deep learning architecture. The sequence-to-sequence model consists of consecutive components: an encoder and a decoder. The encoder is a bidirectional LSTM network. The sentences are fed into the encoder in a form of token lists. The token/word embeddings are processed by the Bi- LSTM encoder character by character. The output context vector of the encoder is used as an input for the decoder. The decoder is also a Bi-LSTM network that learns to convert the context vector to normalized tokens/words.

The model is trained on 1200 manually collected Turkish tweets that contain spelling mistakes, informal structured text, and out-of-vocabulary words. The authors use character embeddings of size 256. The success rate of the model is measured using the F1 score on the predictions of the model on Turkish social media text. An approximate accuracy of 74% is reached. The limitations of the task are stated as that the sequence-to-sequence model is only concerning the word-level morphological and orthological relations. Thus, while normalizing an input sentence, the model only considers relations in a word instead of word-level relations in a sentence. Because of that, the sentence level normalization is an area open to further improvements.

Moreover, Yuret's Turkish deasciifier [19] is a rule-based system where the English characters are converted to their Turkish correspondent by applying pre-defined rules. The rules are embedded into the source code directly. The model processes each input sentence character by character. Firstly, the model decides whether the given sentence character needs correction. The characters which need correction are mapped into the correct Turkish characters in the pre-defined table. The pre-defined table consists of Turkish correspondents of both uppercase and lowercase letters.

Stankevicius et al. [20], states the problems of the state-of-the-art models on spell correction tasks as poor diacritics restoration of spell checkers. In their work, they developed a ByT5-based sequence-to-sequence model which does not depend on a language-specific model structure because of the byte-based nature of the ByT5 model. The authors address the issue of the rule-based checkers: checkers can not detect an error when the misspelled word is also present in the vocabulary. They underline that depending on a vocabulary may not result in efficient spell correction. Therefore, they fine-tuned the ByT5 model which eliminates the need for an additional tokenizer which results in a vocabulary-free model. They experimented on many languages including Turkish written with a QWERTY keyboard. The model they proposed achieved 98.86% accuracy in the Turkish diacritics restoration task on a synthetically generated typo-inserted dataset containing sentences. They interpreted their results as an increase of 0.37% on the state-of-the-art.

Kuznetsow et al. [21], presented a transformer-based model applying spelling correction on short input text such as queries and words in 2021. The model is trained on an artificially generated dataset where the texts are synthetically polluted with realistic typos and misspelling errors caused by keyboard writing. They have defined 4 types of misspelling errors as insertion, substitution, deletion, and transposition. Kuznetsow et al, produced a statistical map showing the location of where spelling errors generally occur in a word by length. Their distribution confirms that the misspellings tend to happen near the end of the text. The model is 4 layer transformer architecture with 2 attention heads and a hidden layer size of 256. They compare the results of their

model on texts with realistic and uniformly generated spelling errors. They stated that the model performs better on realistic errors.

The Hierarchical Character tagger for spelling correction model is developed aimed at converting short text into error-free forms. Gao et al. [22] state the success of the models involving a token vocabulary such as BERT remains limited. Therefore they proposed a LSTM-based character-level language encoder and Bi-LSTM decoder. The decoder predicts a tag for each character for the input text as an edit operation as [KEEP], [DELETE], [APPEND] and [REPLACE]. The [APPEND] and [REPLACE] tags are populated with the characters to state which character to append or replace with. The Hierarchical Character Tagger outperforms Neuspells BERT-based spell corrector on the Webis dataset which contains 54,772 queries obtained from search logs. Do et al. [23] reaches 81% accuracy of corrected errors with transformers-based sequence-to-sequence model in Vietnamese.

The complex morphology of a language causes ambiguity which prevents it from producing only a single morphological analysis of a sentence. The morphological disambiguator of Sak et al. [24], utilizes a perceptron algorithm to rank the morphological analysis of a sentence. The model uses the trigram-based model of Hakkani-Tür [25] for Assigning probabilities to trigram parse sequences. They used the Viterbi algorithm to decode the most probable candidate parses of a sentence. The candidates are ranked with the Perceptron algorithm. This model is utilized as our baseline model.

The Morse model of Yuret et al. [26] (2019), is a recurrent encoder-decoder model that produces morphologic analyses of the words in a sentence. The encoder is composed of three LSTM encoders with different responsibilities. The word encoder processes a character of the word at a time. The output encoder processes the morphological tags of the preceding words before the target word. The context encoder is a Bi-LSTM network that captures the information from the left and right of the target word. The decoder is a 2-layer LSTM network that produces the lemmas and tag of the target word. The first layer of the decoder is initialized with the output of the context

encoder. The outputs of the word encoder and output encoder are concatenated and used to initialize the second layer of the decoder. At each time step decoder produces an output corresponding to a character or a morphological tag by applying softmax on the probability distribution over the character set and morphological features lexicon.

4. DATASET

In this thesis study, BOUN and IMST treebanks under Universal Dependencies [27] are used. The Universal Dependencies (UD) project aims to provide consistent treebank annotation across for a wide range of languages. It facilitates the comparison and analysis of sentence syntactic structures across languages by offering a set of rules and a common annotation mechanism for syntactic relationships. The Universal Dependencies project serves the linguistic community by providing cross-lingual guidelines and categories as well as language-specific extensions. The cross-linguistic consistency environment of UD, makes it possible for NLP researchers to work on syntactic data coherently. The Universal Dependencies serves as an inventory consisting of a collection of treebanks of various languages.

The treebanks under UD consist of datasets that are annotated using UD guidelines and rules. The common annotation style of the different treebanks allows cross-lingual comparison of syntactic structures. Research in the fields of machine translation, and natural language processing has advanced significantly since the introduction of universal dependencies. The task of creating and training multilingual models is made simpler by the Universal Dependencies.

The treebanks are accessible in CoNLL-U format. The annotations are stored in plain text files encoded in the CoNLL-U format. The file contains the sentence with its tokens.The fields present in the file for each word in a sentence are declared as ID, FORM, LEMMA, UPOS, XPOS, FEATS;HEAD, DEPREL, DEPS and MISC. In this thesis study, FORM and XPOS fields are extracted for the proposed preprocessing tools. The FORM field corresponds to the text of the token and XPOS field contains the list of part-of-speech tags for the related word.

The Turkish UD inventory under UD, contains four Turkish treebanks and more to be added. The treebanks under Turkish UD are annotated with a similar approach

according to the specified guidelines and rules of the Turkish UD. The sentences of Turkish treebanks are generally separated into tokens by white-spaces or punctuations. The sentences are annotated using the same tag set provided under Turkish UD. The tags available under Turkish UD, are the combination of Turkish part-of-speech tags and Turkish morphological features. The morphological features are divided into lexical and inflectional features.

4.1. BOUN Treebank

BOUN Treebank v2.11 [28] consists of 9761 annotated sentences the raw forms are taken from the Turkish National Corpus. The sentences are taken from several text resources varying from essays to biographical texts. A semi-automatic approach is adopted in the annotation process of the treebank. First, the automatic annotation phase converted the sentences to the CoNLL-U format. Then, manual annotation was applied to the ConLL-U version. The annotation is performed parallel to the UD guidelines. Figure 4.1 shows an example format for a sentence taken from BOUN Treebank Test set.

Herkes öyle ya da böyle içinde bir şekilde bir şeylere dair mutlaka bir umut taşır.

Token id	Token text	Lemma	Upos	Xpos
1	Herkes	herkes	NOUN	-
2	öyle	öyle	ADV	-
3	ya	ya	CCONJ	Conj
4	da	da	CCONJ	Emph
5	böyle	böyle	ADV	-
6	içinde	iç	NOUN	-
7	bir	bir	DET	Indef
8	şekilde	şekil	NOUN	-
9	bir	bir	DET	Indef
10	şeylere	şey	NOUN	-
11	dair	dair	ADP	PCDat
12	mutlaka	mutlaka	ADV	-
13	bir	bir	DET	Indef
14	umut	umut	NOUN	-
15	taşır	taşı	VERB	-
16	.	.	PUNCT	Stop

Figure 4.1. Example sentence taken from BOUN Treebank [28] test set. The Features and Relation fields are not shown since they are not used in this thesis.

BOUN Treebank is randomly divided into three sets as training, development, and test having 80%, 10%, and 10% proportions respectively. In this thesis study, the training set is selected as the training set. The development set is used for validation. The evaluation set of the models includes the test set.

4.2. IMST Treebank

The IMST Treebank [16] is the reannotated version of METU-Sabancı Treebank. The treebank consists of 5635 sentences gathered from Turkish daily news reports and novels.

5. METHODOLOGY

In this chapter, the implementation details of tokenizer,sentence splitter, deasciifier, vowelizer, POS tagger, spell corrector and morphologic analyzer tools are explained in detail. The preprocessing tools are divided into two where the tokenizer, sentence splitter, and deasciifier are built using both Bi-LSTM and transformers architectures (CANINE), and the remaining are built using transformers.

The Bi-LSTM architecture is specialized for each task mentioned above. The Bi-LSTM model is applied to character-level representation of the sequence where a character is processed at a time step. The tag set for the tokenizer, sentence splitter, and deasciifier is synthetically created for the training phase. The Bi-LSTM architecture produces a tag for each character in these models.

The character-based models above are compared with those built using the pre-trained CANINE model provided by the Huggingface Transformers [29] library. CANINE's tokenization-free structure was suitable for the character-level operations needed in tokenizer, sentence splitter, deasciifier, vowelizer, and spell corrector. The CANINE models are separately trained on T4 and A100 GPUs provided by Google Colab. The Part-of-speech tagger tool is developed by fine-tuning the pre-trained BERTurk's uncased model using Hugginface Transformers Library. The morphologic analyzer tool is developed using the mT5 model of Google Research Team available in the Huggingface Transformers Library. In this study, the mT5-small checkpoint provided by Google Research is utilized instead of the mT5-base or mT5-large checkpoints because of resource insufficiency.

5.1. HuggingFace Transformers Library

Hugging Face Transformers Library [29] offers a selection of state-of-the-art natural language processing (NLP) models that have already been trained, including well-

known models like BERT, GPT, RoBERTa, and many more. Users may quickly apply, optimize, and implement pre-trained models for a range of natural language processing (NLP) applications, including question answering, named entity recognition, text categorization, and more with the help of the Transformers library.

Huggingface Transformers Library facilitates the usage of pre-trained transformers models. The library supports integration between machine learning frameworks like Pytorch. In this thesis study, Huggingface Transformers API is used for loading and fine-tuning the models and tokenizers of the CANINE, BERT, and mT5 models.

The library provides built-in heads for downstream tasks for fine-tuning the pre-trained models. These heads vary from sequence classifier to text generator. The configuration parameters of these heads can be changed by the user in a fine-tuning process. For instance, for token classification, Hugging Face's built-in class adds a linear layer on top of the [CLS] token.

HuggingFace provides a Data Collator class for language modeling enabling it to handle large sizes of data and batch operations. The class can also be configured to apply padding to the sequences, data augmentation, and random masking. The tokenizers of most of the popular pre-trained models can be loaded from the Huggingface library.

In this thesis study, the Huggingface Transformers library is utilized for loading and fine-tuning the BERT, CANINE, and mT5 models. The evaluation metrics such as SacreBleu and SeqEval contained within the PyTorch are used to calculate training and validation loss through the training parameters HuggingFace provides.

5.2. Tokenizer

5.2.1. Dataset Preparation

The sentences extracted from the BOUN treebank are transformed to be processed by the model. The sentences are converted to a character list as the input to the model. Each character is tagged according to its location in a token. If the character is the starting element of the token, it has a B tag corresponding to the beginning of the token. The characters inside a token belong to the I tag set corresponding to the inside of the token. The other characters are tagged with N meaning that they are not a part of a token in the sentence. The tags of the characters of a sentence are stored as a tag list, where each tag is represented by a character. The ith character of the sentence is matched to the ith tag in the tag list. The process of assigning tags to characters is not dependent on any external tokenizer or a rule. The characters are tagged by identifying the boundaries of each token present in the sentences of BOUN treebank. The tokens that are members of a compound token in the treebank are eliminated and treated as single words for simplicity.

5.2.2. Bi-LSTM Tokenizer

The tokenization operation is one of the first steps in text-analysis applications. The tokenization model is a Recurrent Neural Network consisting of Bidirectional Long Short Term Memory cells. The input to the model is a sentence split by its characters. Each character in a sentence is an input. The goal of the model is to classify each character in an input sentence.

The model is trained on the training set of the BOUN-Treebank dataset. Firstly from the characters in training sentences, a character-index dictionary is built where each character corresponds to a number in the dictionary. The input character sequence is enumerated according to the word-index dictionary. Therefore, the input to the model is suitable to be processed. Each LSTM cell accepts a number corresponding

to the character in the input sentence. For each character, a character embedding is built. The outputs of the Bi-LSTM cells are fed into a linear layer with the dimensions of hidden vector size and target class count. The linear layer results are fed into the log softmax layer in order to obtain categorical probability distribution over target classes. The logarithmic softmax layer computes a distribution on three target classes (i.e. N,B, and I) for each character. The output of the log softmax layer is a probability vector with a size of 1x3. The index of the highest probability in the probability vector corresponds to the index of the target class in the tag-index dictionary. The model has four parameters as character embedding dimension, hidden vector size, character dictionary, and tag dictionary. The loss function is a negative log-likelihood function that is compatible with the logarithmic softmax function and multiple categorical classifications.

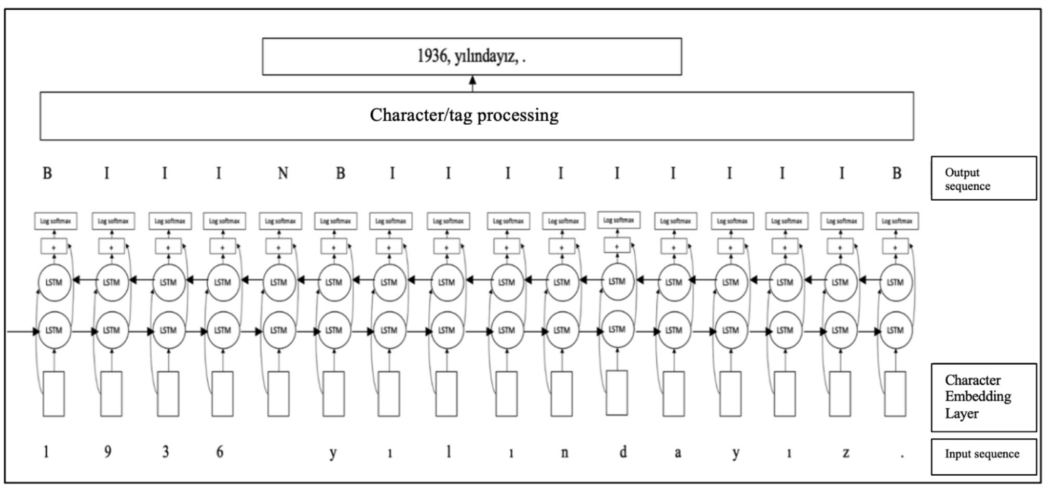


Figure 5.1. Tokenization of the sentence "1936 yılındayız.".

The model is trained with 10 epochs. To obtain the best-resulting model, the experiments include measurements with character embeddings in dimensions of 100,200 and 300 and hidden vector sizes of 128 and 256.

In Figure 5.1, the training process of the model with input sentence "1936 yılındayız." is demonstrated. The input sentence is split into its characters and the character embeddings are fed into the Bi-LSTM cells. For each character, the expected output of

the model is one of the target classes N,B or I. Considering the tags of each character, the sample sentence is tokenized as “1936”, “yılındayız”, and “.”.

5.2.3. CANINE Tokenizer

The second version of the tokenizer is built by fine-tuning the pre-trained CANINE-s model. The same dataset built for the LSTM model is utilized in the development of the CANINE-based tokenizer. The only difference with the previously created dataset is that the sentences are not transformed into a character list since the tokenizer of the CANINE model is utilized. The pre-trained tokenizer of CANINE applies a splitting from each character so that the need of additional implementation on input texts is not needed. A character-index lookup table is created where each character corresponds to a unique number. The tags in the tag set are enumerated. This conversion is done since the CANINE-s model supports working on Unicode numbers. The characters in the input sentences are converted to their numerical identity and tagged by the related numbers by the CANINE model.

The created dataset is converted into a list of dictionaries using the HuggingFace Datasets library which facilitates the process of tokenization and data loading. An instance of the created dataset with the library is shown in Table 5.1.

Table 5.1. Example training dataset instance.

Sentence Text	1	9	3	6		y	ı	l	ı	n	d	a	y	ı	z	.
Labels	B	I	I	I	N	B	I	I	I	I	I	I	I	I	I	B

As mentioned above, the tag set is also enumerated for categorization. Figure 5.1, shows an instance of training data with the gold labels of each character. The elements of the dataset are tokenized with the pre-trained CANINE tokenizer. The sentence texts are directly tokenized by the tokenizer. The labels need to be aligned

in terms of adding a new label as -100 to the beginning and the end of the label list. As a result, the special tokens [CLS] and [SEP] are ignored by the cross entropy loss of the Pytorch during training. The tokenized dataset includes "input ids", "token type", "attention mask" by default, and "labels" fields for each item in the dataset. The field "input ids" is a list where each ith item of the list corresponds to the Unicode Id of the original input sentence's ith character. The Unicode IDs are contained in the vocabulary of the CANINE model. The "token type ids" field is a list storing which input text each input id belongs to.

Since the tokenization task does not require multiple input sentences divided by [SEP] token, the "token type" field does not carry a role in the proposed tokenization task. Finally, the last field the tokenizer creates is attention mask which is a list of binary values indicating if a mask is needed for each token. The labels field is added by our implementation which stores the tag of each token including the special tokens. This field is used by the model in the training phase to learn to classify each character.

The number of hidden layers and their size, number of attention heads, downsampling rate, and upsampling kernel size are configurable for the CANINE model with the CanineConfig class Huggingface provides. We have reached the best accuracy with the default parameters of the pre-trained CANINE model. The model configuration parameters are shown in Table 5.2.

Table 5.2. Model configuration parameters.

Hidden layer size	768
Number of hidden layers	12
Number of attention heads	12
Downsampling rate	4
Upsampling Kernel size	4

The deep-transformer stack of the CANINE has 12 hidden layers with size 768 and 12 attention heads. The downsampling rate is the rate applied to the original character sequence length before applying a deep transformer encoder. The upsampling kernel size stands for the number of characters in each window of the projection layer after the transformer encoder. In this convolutional projection layer the output of the transformer encoder is projected back from the hidden layers.

The token classification is done by placing a linear layer on top of the hidden states output for each token. The linear layer produces a distribution over the tag set. In the training phase, two metrics are used to understand the training performance. First is the cross-entropy loss. The second metric used is *seqeval*. A Python framework called seqeval is used to evaluate sequence labeling. Seqeval is capable of assessing chunking tasks such as semantic role labeling, named-entity recognition, part-of-speech tagging, and more. The predictions of the model and the ground truth labels are compared during the training. *Seqeval* metric produces averaged accuracy, precision, recall, and F1 score as output for each epoch.

Table 5.3. Training hyperparameters.

Learning rate	2e-05
Training batch size	32
Weight decay	20

The training parameters of the model are shown in Table 5.3. The model is trained on the A100 GPU the Google Colab environment provides. Adam optimizer is used for the regularization of the training weights. The weight decay parameter is 0.01. Weight decay is a regularization technique used by the Adam optimizer that penalizes high model weights. To apply this, a term proportionate to the sum of the squares of the weights is added to the loss function.

5.3. Sentence Splitter

5.3.1. Bi-LSTM Sentence Splitter

The sentence-splitting operation is a process of dividing a text into sentences. In most natural language applications, texts are processed based on words or sentences. For instance, in named entity recognition systems the inputs are the words whereas the inputs are words and sentences in the machine translation systems. Therefore, tokenization and sentence splitting are the most used preprocessing steps.

Similarly to the tokenization system, the sentence splitting system is a character-based Bi LSTM where each Character is classified into a category. The target classes are represented by N(not end-of-sentence) and S(end-of-sentence) tags. The parameters of the model are vector dimensions, character dictionary, and tag dictionary. The model is trained with the sentences extracted from the Boun Treebank dataset. The sentences in the training set of the treebank are concatenated together to obtain one long sequence. The sequence is divided into 100 characters long 6074 sequences for training. The model is trained with 20 epochs.

5.3.2. CANINE Sentence Splitter

The second version of the sentence splitter tools is built by fine-tuning the pre-trained CANINE model. The training of the model is similar to the training of the tokenizer tool except for the labels. The same labels as the Bi-LSTM (“B” and “N”) sentence splitter are used.

Since a sentence splitter works on text including many sentences such as paragraphs, the sentences in the training set are grouped to form paragraphs. The sentences are joined by size and white space between them so that 25% of the sentences are added after each other without whitespace in between to form 5 sentence-long paragraphs and 25% of the sentences are added after each other with whitespace in between to form 5

sentence-long paragraphs. The remaining half of the sentences are added together in groups of 10 with/without whitespace in between them. The labels for each sentence in each paragraph group are concatenated to obtain the labels of the characters present in the paragraphs.

5.4. Deasciifier

5.4.1. Dataset Preparation

The BOUN-Treebank consists of proper sentences without any problem with Turkish characters. To build the training and test set for the proposed model, the sentences are disrupted intentionally. According to this, each Turkish character in each sentence is converted into its corresponding English character. The word in the sentences that should have been written with Turkish characters but instead written with ASCII characters (c, g, i, o, s, u, C, G, I, O, S, U; and a, A respectively) are converted to special Turkish characters (Ç, ğ, ı,o, ş, u, Ç, Ğ, İ, Ö, Ş, Ü; and â, respectively).

The Turkish characters are enumerated and stored in a character-to-index dictionary. And the ASCII characters that should be converted to Turkish are labeled as its Turkish correspondent. Additionaly, each character is labeled as "NaN" to indicate no change is needed or the Unicode of the replacement character. The 10% of the dataset was kept as error-free for the model to learn what should be the correct spelling of some of the words.

5.4.2. Bi-LSTM Deasciifier

Especially in online platforms, English characters are used in Turkish Words to avoid encoding problems in different systems. For this purpose, similarly to tokenization and sentence splitting a character-level Bi-LSTM model is proposed.

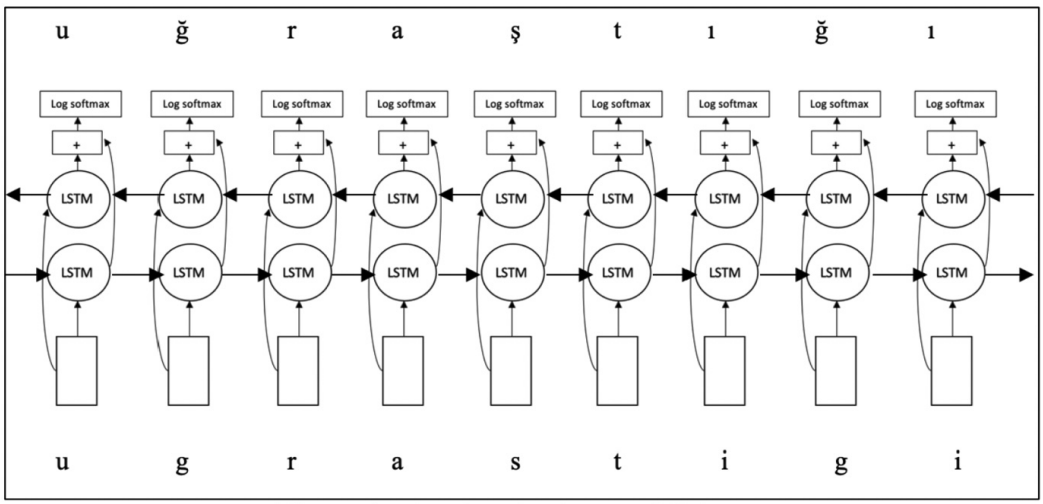


Figure 5.2. Training of Bi-LSTM Deasciifier.

In the training of the model, the sentences with English characters are fed into the model, and the model is expected to learn to convert the input sentence into a sentence with the correct Turkish characters. The model is trained with 20 epochs. A section of the training process of the model is represented in Figure 5.2 where a word is converted into the correct Turkish form.In the training input word "ugrastigi" is converted into "uğraştığı" in a given text shown in Table 5.4.

Table 5.4. Example input and output sentences from training data.

Input Sentence	O nasil derse desin ugrastigi sanatin kendisine emanet oldugunu soyleyen ustadlari vardi.
Ouput Sentence	O nasıl derse desin uğraştığı sanatın kendisine emanet olduğunu söyleyen üstadları vardı.

5.4.3. CANINE Deasciifier

The CANINE-based deasciifier is fine-tuned on the same dataset used for the Bi-LSTM deasciifier.

The model is trained by 20 epochs with the parameters shown in Table5.3. A linear layer is added on top of the model with 768 input features and 13 output features so that one of the labels can be selected to replace the target character in the word including the "NaN" label. The number of trainable parameters is 4880. As for the

previously explained models, the loss function is the combination of the cross entropy loss and SeqEval metric which ignores the special tokens during evaluation.

5.5. Part-of-speech Tagger

The part-of-speech tagger aims to label each token as the part-of-speech of the sentence.

5.5.1. Training Dataset Preparation

The training of the model is done on Boun Treebank. The sentences in ConLL-U format, store the POS for each token. There are two different Pos tagging schemes in CONLLU format as UPos and XPos. In this study, the UPOS field is considered as the gold data where the tokens are a member of the Universal POS tag set, while the XPos field represents the language-specific features.

The sentences in Boun Treebank are annotated in a way that the multi-words and their components are both shown in sentence tokens. The dataset does not contain a pos tag for these multiwords. Instead, the components of the multiword have pos tags. To be able to label the original word present in the sentence, the pos tag of the multiword is taken as the pos tag of its first component. While preparing the dataset for training, the components of the multiword are not considered as members of sentence tokens.

5.5.2. BERT Pos-Tagger

The BERTurk model, which is the pre-trained BERT on Turkish, is fine-tuned for the part-of-speech tagging task. The sentences are tokenized by the WordPiece tokenizer pre-trained in Turkish. Since each word has a pos tag in the dataset, the pieces of a word and special tokens are labeled with -100 to be ignored by the loss function. So only the first token of a word is labeled as the pos tag corresponding to the id in the label to id dictionary.

The BERT model is loaded from the BERTurk checkpoint. The model consists of the BERT Embeddings and the deep-transformer stack. The BERT embeddings layer involves 3 embedding layers for word embeddings, position embeddings, and token type embeddings. These embeddings are concatenated and used as input by the BERT encoder. The encoder consists of 12 BERT blocks. Each block contains a self-attention layer with a hidden size of 768. A linear layer is placed on top of the output of the model to assign a pos tag to each token. The linear layer takes the hidden state so the input size is 768. Then outputs a probability distribution over the possible pos tags where the gold set has 16 labels. This classifier head involving a linear layer is provided by the AutoModelForTokenClassification class of HuggingFace Transformers API.

In the training, both training and validation losses affect the learning parameters of the model. The validation dataset is the tokenized and adjusted version of the BOUN Treebank Development set. The loss function for the training is calculated with the combination of the SeqEval metric and cross-entropy loss. The model is trained with 50 epochs with the hyperparameters as in Table 5.3. In training, a validation loss of 0.77 is reached.

5.5.3. mT5 POS Tagger

The other version of the Pos tagger is based on the pre-trained mT5 model. In the training of mT5-based POS tagger, we tackled the POS tagging problem as a machine

translation task where the inputs are translated into their POS tags. The model is fine-tuned on the training set created above. The only difference is that the labels are not turned into indices and kept as original so that the mT5 model treats the pos-tagging task as a translation problem where the sentences are translated into pos tags. A sequence-to-sequence mT5 model is obtained with a linear on top of. The encoder is a basic mT5 stack and the decoder is in the same shape except the cross-attention layer. The model is trained on V100 GPU with 50 epochs with the combination of SeqEval and cross-entropy loss calculated at each step to adjust the weights of the model.

5.6. Vowelizer

The texts with missing vowels may affect the performance of the NLP tasks since a word with missing vowels lacks syntactic and semantic features. The vowels can be restored in a rule-based fashion by checking the similarity between the input and the words in the vocabulary. However, this method would be error-prone if the context is not considered. For this purpose, in this thesis study, a transformers-based vowelization tool to reconstruct the words with missing diacritics is proposed. The vowelization tool tries to assign the vowels to the correct place of the word by taking into account the characters of the words as well as the surrounding words. In this thesis study, the pre-trained CANINE model is fine-tuned for the vowelization task. As in the other preprocessing tools, the Huggingface Transformers library is utilized for training and inference.

5.6.1. Dataset Preparation

For the training and evaluation of the vowelization tool, BOUN Treebank's training, development, and test set are selected. The vowels to be restored are case-sensitive and suitable for the hat presentation of Turkish as shown in Table 5.5. The training sentences are modified in a way that the possibility of a vowel occurring in a position remains.

The training dataset is created synthetically by removing the vowels of the sentences randomly. The vowel removal algorithm is applied to each word independent of the other words in the sentence. First, the algorithm identifies the positions of each vowel in the list shown in Table 5.5. Not all of the vowels are deleted since it may not be a realistic case even for the tweet inputs. Therefore, the vowels to be removed are randomly selected. The vowel removal number is determined depending on the word length dynamically so that the long words have more missing vowels than the shorter words. The algorithm randomly selects the vowels with the amount of half size of the word length and removes the vowels by replacing a dummy character in the vowels' position. As the dummy character, we chose a letter that is not usually seen in Turkish sentences which is ß. The reason for substituting a dummy character into the removed vowel positions is to maintain the input and output length consistency required by the CANINE model. Another place the dummy character is inserted in the word is between the consonant letters in the word or the last position of the word. Because, generally the vowels appear between the consonants. Table 5.6 shows an example output of the vowel removal and substitution algorithm.

Table 5.5. The vowels to be restored.

Vowels to be restored
a,â,e,ı,i,o,ö,u,ü,A,Â,E,I,İ,O,Ö,U,Ü

As shown in Table 5.6, the dummy characters take place in the positions where a vowel may appear. The proposed model is expected to determine which dummy character is going to be replaced by a vowel and by which vowel.

Table 5.6. Example modification of the sentences for training dataset preparation.

Input sentence	Rüzgâr yine güçlü esiyordu.
Sentence after vowel removal	Rüzßgßrß yinß güçßlß esiyßrßdu.

5.6.2. CANINE Vowelizer

With a training set of sentences like in the Table **??**, the model is supposed to learn to assign the same character to the consonants and the vowels that are not removed. This would increase the model's training time. To decrease the learning surface of the model, a tagset consisting of vowels and an additional 'NaN' label is created. Therefore the model would only learn to not change characters that are not relevant to the vowelization task. The labels are converted to a list as in the previously explained tools. An example labeled version of the word "rüzgâr" in the table is shown in Table 5.7.

Table 5.7. Labeling of the word "rüzgâr" with missing vowels.

R	ü	z	ß	g	ß	r	ß
NaN	ü	NaN	ß	g	â	NaN	ß

As shown in Table 5.7, the data is prepared for the model to learn not to change the correct characters including not replacing the dummy character which does not require vowel restoration. From this word model learns to substitute the dummy character between "g" and "r" letters with "â". The word is reconstructed again after the prediction of the tags, by simply replacing "NaN" labels with the original letter of the word. The dummy character "ß" is removed afterward.

5.7. Spell Corrector

The main goal of the spelling correction task is fixing the misspelled words in a text. The spell corrector identifies the non-words which are the corrupted versions of a real word then converts the found word to its correct form. To be able to build a spell corrector tool, we have focused on the mostly occurring misspelling types while writing using a QWERTY keyboard.

Turkish is a morphologically rich language containing many forms of a root word changed with derivation. The rich morphological structure of Turkish makes it infeasible to keep all the words in a lexicon to find if the target word is similar to any word in the lexicon. In addition to that, even if the lexicon lookup is applied, the word in the sentence is dependent on the context where the misspelled word may be reconstructed in many different ways. Therefore, in this study, a context-dependent model without a vocabulary is developed.

In the online world, most of the text resources are obtained from the web. These text resources are prone to spelling corrections due to keyboard usage. The spelling errors Most are caused by missing a character, character duplication/insertion, character order change or pressing another character instead of the correct one. These errors can be simplified as shown in Table 5.8.

Table 5.8. Common spelling mistakes.

Spelling error	**Example**
Missing character	Slm =>Selam
Character duplication	Yaarın =>Yarın
Character insertion	Gidiyorum =>Gidiüyorum
Character order change	Mehraba =>Merhaba
Character substitution	Dsnışma =>Danışma

5.7.1. Dataset Preparation

In this thesis study, errors are created synthetically for training and evaluation of the spelling correction tool. The algorithm creates a misspelled version of the target word in a randomized fashion in terms of the error positioning and error type. After the error type is randomly chosen for a word, the error is randomly positioned. The words with numeric values and short words with lengths less than 3 are excluded by the error-creation algorithm as well as the punctuation. We have excluded the words

containing numeric values since it is not possible to identify if a number is correctly written. And the words with lengths 1 or 2 such as “o”, and ”bu” are kept original to be able to capture syntactic and semantic information from these words. The error positioning method can select all possible indices in a word except the first character of a word because we thought it may be confusing for the model.

The application of the character insertion and character substitution methods is based on a QWERTY keyboard. For each character in the QWERTY keyboard, at most top 3 closest characters are identified to be used in the character insertion and substitution error creation. Table 5.9 shows the mistaken characters for each character we have determined. The possible insertion/deletion candidates are also randomly selected.

The character deletion error-creation is considered as two different types of errors as vowel deletion and consonant deletion while choosing which error to be applied to a word since the missing vowels and consonants impact a word’s meaning on different levels.

Table 5.9. The possible character insertion/substitution candidates for each character.

Mispelled Character	Possible Character insertion/substitution candidates
a	w, s
s	a, d
e	w, r
r	e, t
o	u, ı, i
ı	ı, o, k
i	ü
ü	i
l	k
k	l, m
m	k, m, n
g	f, h
h	j, g
b	v, m
z	a, s
y	u
t	r
n	b, m
d	a, s

5.7.2. CANINE Spell Corrector

In this thesis study, the spell corrector model is built to correct the misspelled words by taking into account the context of the sentence. For example in English, the word “form” may be the misspelled version of the word “from”. However, since “form” is a likely word to appear in a sentence, the context of the sentence plays an important role in selecting a word as misspelled and correcting it. For instance

in Turkish, the possible correct candidates for the misspelled word “güdiyordu” can be “gidiyordu”, or “güdüyordu” if the word has a single spelling error otherwise the candidate space increases dramatically. To be able to determine what is the correct form of the misspelled word “güdiyordu”, the candidate space is narrowed down by checking the surrounding words and characters in a sentence by fine-tuning the pre-trained CANINE model.

The sentences with errors are modified by inserting a dummy character, which is not likely to be seen in Turkish sentences, after each character in the sentence as the word may have a missing character in any position. The characters that do not need any change are labeled as “NaN”. The characters that need change due to character misorder or substitution are labeled as the correct character to be replaced. If a missing character should be in the position of a dummy character, the dummy character is labeled as the correct character, otherwise labeled as “NaN”. In the error case where a character duplication or an insertion occurs, the duplicated/inserted character is labeled as the dummy character to be able to indicate the erroneous character is required to be discarded afterward.

In order to obtain realistic training data, the count of erroneous words is proportioned to the word count in the sentence. As a result, half of the words in each sentence of the dataset are converted to their misspelled version.

The second separate approach on creating a dataset containing misspelled sentences diverges in the placement of the dummy character. In the second approach, a possible missing character error amount is predicted and dummy characters are added as the predicted amount. Therefore, if the word has a character missing error, the correct character is inserted into the position of the word as expected, so the other characters coming after the newly added are shifted by one character to the end of the sentence.

Each unique character seen in the dataset is enumerated to create a character-to-index dictionary including the labels “NaN” and the dummy character. In total 106 unique characters are obtained.

As the pre-trained CANINE model, the Canine-s model is selected. The model is trained with 20 epochs with the parameters demonstrated in Table 5.3 except the batch size of 4. The batch size is smaller compared to the training of the other tools proposed in this thesis since the addition of dummy characters to the sentences increases the input sequence length which leads to higher computation done by each unit of the GPU.

As in the other NLP tools explained previously, the combination of cross entropy loss and SeqEval metrics is used to calculate the training and validation loss. The hyperparameters are shown in the table. The batches are decreased compared to other proposed models since the higher batch size dramatically affects the amount of used GPU resources due to the high number of trainable parameters for a single epoch. The results of different hyperparameter values as shown in the Table, and the loss calculation on training or validation datasets are compared in the Experiments and Results chapter.

5.7.3. mT5 Spell Corrector

While the CANINE-based spell corrector processes the sentence character by character, the mT5 model can treat the spell correction task as machine translation. For instance, the sentences with spelling errors are translated into their corrected versions. Since the mT5 is a sequence-to-sequence model, the dataset is suited for sequence-to-sequence architecture using HuggingFace Data Collator. Both of the sentences with mistakes and their correct correspondents are tokenized using the pre-trained tokenizer of mT5 with the configuration of truncating each dataset item to the max length of 12. For the training and validation losses are calculated with the combination of the cross entropy loss and SacreBLEU metric which is language-independent

and a lightweight metric in terms of computation. The SacreBLEU metric produces a BLUE score at each epoch. Even the BLEU score is mostly used for evaluation of the machine translations, it can be used to evaluate any text generated by a model. The BLEU score is mostly used for the translation tasks which is calculated by comparing candidate translation to multiple gold instances. In detail, the BLEU score is proportioned to the occurred number of n-grams in the gold data. The special tokens labeled with -100 are ignored in the calculation.

The pre-trained mT5-small model is loaded using the Seq2SeqLM class of the HuggingFace Transformers API. The Seq2SeqLM class provides a head consisting of a linear layer on top of the model for the sequence-to-sequence language modeling task. The model is trained with 20 epochs and with the same hyperparameters in Table 5.3 except the batch size of 8.

The number of trainable parameters is 19520, where it is 976 for each training epoch. The generated average sentence length for each sentence starts from 13.55 and ends with 15.78 at the last epoch. At each epoch, the BLEU score increases by 0.92 on average. In the mT5 model, the generated output must not be at the same length as the input sentence. Therefore, the generated text length is measured as another aspect of the evaluation besides the spell correction performance.

5.8. Morphologic Analyzer-Disambiguator

A linguistic analysis called morphological analysis divides words into their smallest components of meaning, or morphemes. The smallest meaningful elements of language that cannot be further split are called morphemes. The structure and construction of words are the focus of morphological analysis, which examines how words are constructed from morphemes. The morphological analyzer is responsible for producing the possible morphemes and tags of the words given an input text. The tags that morphological analyzers produce, represent the morphological features of words. These tags include part-of-speech tags and the role of a subword in the meaning or derivation

of a word. Furthermore, another output is the lemma that the word is derived from. Morphologic analyzers identify the lemma and the inflectional or derivational affixes of the target word. The output of the morphologic analyzer for a word may depend on the context of the sentence since a word can be interpreted in many ways without context. In the Table Table 5.10, the different interpretations of the word “kazı”.

Table 5.10. Different interpretations of the word ”kazı” according to the context.

Kazı çalışması yürütülüyor.	**Construction** is ongoing.
Kazı kümesten kaçtı.	Her **goose** escaped.
Beni kalbine **kazı**.	**Engrave** me in your heart.

Therefore, in this study, we have built a model depending on context while deciding the morphologic analysis of a word. Usually, a morphological disambiguator tool is needed to select the most possible analysis of the sentence. In this study, we propose a combined model for morphological analysis and disambiguation. Tackling the issue of disambiguation, the proposed model handles the problem as a machine translation task. For this purpose, the mT5 architecture is adapted for sequence-to-sequence language modeling with the Seq2SeqLM class of the Hugginface. The morphological analysis of the sentence is approached as the translation of the sentence.

The proposed model is a fine-tuned version of the pre-trained mT5 model. Unlike the BERT model, the mT5 model does not require a language-specific pre-trained model to perform on a language. In this study, a pre-trained mT5-small model is loaded from the HuggingFace library.

5.8.1. Dataset Preparation

The training sentences are taken from the training set of BOUN and IMST Treebanks. The sentences are tagged by Haşim Sak’s morphological parser to compose the gold data. BOUN TABILAB provides an API for the parser. The docker image of the

parser needs to be built to be able to be used. The sentence is sent as a request to the parser and a JSON result containing the analysis is returned.

The parser deconstructs the target word as a lemma, tags, affixes, and tags of the affixes as shown in Table 5.11.The parser outputs morphological tags for the inflectional and derivational affixes. Since we handle the problem as a machine translation task, the affixes are considered also as the tags of a word in this study. The training data creation includes removing the brackets indicating the tags. Since the affixes set is known, the sequential lemma, tag, and affix combination produced by the model can be reconstructed as the output of the parser.

Table 5.11. An example sentence and morphological parse of each word.

Fakülteyi bitirenler en uçtan göreve başlıyorlarmış. (Those who graduate from the faculty start working at the bottom.)	
Word	**Morphological Analysis**
Fakülteyi	fakülte[Noun] [A3sg] [Pnon] YH[Acc]
bitirenler	bitir[Verb] [Pos] YAn[Adj PresPart] [Noun] lAr[A3pl] [Pnon] [Nom]
en	en[Adv]
uçtan	uç[Noun] [A3sg] [Pnon] DAn[Abl]
göreve	görev[Noun] [A3sg] [Pnon] YA[Dat]
başlıyorlarmış	başla[Verb] [Pos] Hyor[Prog1] lAr[A3pl] YmHş[Narr]
.	.[Punc]

5.8.2. mT5 Morphologic Analyzer-Disambiguator

For the training, the correct morphologic features of the sentence are interpreted as the translated version of the input sentence. The models learn to extract the lemma,

tags of the lemma, affixes, and tags of the affixes in the correct order. So the order of the tags is an additional result to be calculated in the evaluation step. Since the model involves the analysis and disambiguation together, the beam search is used to handle disambiguation by selecting the most likely sequence that provides the disambiguation depending on the context.

The model consists of two parts: an encoder and a decoder. The encoder is the basic mT5 model. The decoder consists of mT5 blocks with a cross-attention layer. The decoder attends the encoder output at each timestep to generate the morphological output. The encoder first creates embeddings. The embeddings are fed into the mT5 stack which is built up of 7 mT5 blocks. Inside the mT5 block, the embeddings are sent to the Self-attention layer where query, key and value vectors are linear layers with input length of 512 and output length of 384. The feed-forward layer is applied on the output of the self-attention layer. As the activation function, GELU which weights inputs by their percentile in the Gaussian cumulative distribution is used. After the layer norm is applied to the output of the feed-forward layer, the resulting hidden state which contains contextualized information, is provided to the decoder.

The decoder of the model includes a stack of mT5 blocks where each block contains a self-attention layer followed by a cross-attention layer. The output of the cross-attention layer is then fed into the feed-forward layer and layer norm is applied afterward. The cross-attention layer uses the Hidden state provided by the encoder to calculate the decoder key and value vectors. Therefore, the decoder attends to the entire encoder output in contrast with the self-attention layer which only takes into account the preceding words.

The model involves a Seq2Seq head provided by the Huggingface Transformers API. The Seq2Seq Head is a linear layer applied to the decoder's hidden states.

The trained mT5-small is fine-tuned on the half of the BOUN Treebank training dataset in 15 epochs, and 15 epochs on the other half. The resulting model is trained

on the IMST Treebank training dataset by 15 epochs. The training hyperparameters are shown in the table. There are 14625 trainable parameters in total for 15 epochs.

The training method of the model is based on the teacher-forcing method which prevents the decoder from generating output for a possibly wrong decoder prediction for the previous decoder step.

6. EXPERIMENTS AND RESULTS

In this chapter, the results of each tool are discussed and compared in detail.

6.1. Tokenizer Results

As mentioned in the methodology section, two models for the tokenizer are built and evaluated in this study. The results are shown in Table 6.2.

Table 6.1. The results of developed tokenizer models.

Bi-LSTM 300 Emb. Character-level	CANINE Character-level	CANINE Token level F1 Score	CANINE Sentence-level
0.99	0.99	0.99	0.97

6.1.1. Bi-LSTM Tokenizer

The BiLSTM-based tokenizer was evaluated on BOUN Treebank's test set. The sentences in the test set are directly fed into the trained tokenizer to label each character in the input sentence considering their positioning inside a token. The expected output for each sentence was a tag sequence list containing "B", "I" and "N" labels.

The trained BiLSTM tokenizers differ in hidden layer size and character embedding size. The models with 100, 200, and 300 embedding layer sizes are compared as well as the models with hidden layer sizes of 128 and 256. The best resulting model has reached 99% accuracy on the character level and 99 % F1 score on the token level. The character level accuracy is calculated by comparing the model's output label sequence with the expected output.

To analyze the token-based performance of the model, the output label sequence of the model for each sentence was used to create a token list. The label sequence was started to be processed from the beginning and when a "B" is encountered a new token string is created and the adjacent "I" labeled characters are added to the token string with the beginning character of the token.

To calculate the token-based performance, the gold tokens of the BOUN Treebank's test set are compared with the extracted tokens. In some cases, the amount of predicted tokens by the model differed from the number of gold tokens. That's why precision and recall measurements are applied.

The best-performing model was the tokenizer model with an embedding size of 300 dimensions and a hidden size of 256 with the F1 score of 0.99.

6.1.2. CANINE Tokenizer

We evaluated our model on sentences from the BOUN Treebank test set. The test set contains challenging sentences for the tokenization operation, such as sentences including decimals, percentages, and quotations. The performance of the tokenizer is measured in terms of success rate based on characters, tokens, and sentences. During the test phase, we process the labels of each character to extract the tokens. A token is formed starting from the character with the beginning-of-token label. The token ends if another beginning-of-token or not-token-related label is encountered.

Table 6.2. Tokenizer results.

Character-level	Token level F1-score	Sentence-level
0.99	0.99	0.97

Table 6.2 shows the results. The model mislabeled only 34 characters out of 75011 characters, resulting in 0.99 accuracy. These mislabeled characters caused the actual number of tokens to change during prediction. For instance, if a character is labeled as inside-of-token where it should be labeled as beginning-of-token, the character is included in the previous token while it should be the starting character of another token. The model predicted 12023 tokens, whereas the gold set had 12016 tokens. 11965 of the predicted tokens were the same as gold ones. The tokenizer correctly predicted all tokens of 951 sentences out of 979 sentences.

The performance of the tokenizer is similar to the state-of-the-art results such as the 0.99 token-based F1-score of UDPIPE [14] on UD Turkish IMST dataset [16].

6.2. Sentence Splitter Results

In this section, the results of BiLSTM-based and CANINE-based sentence splitter models are discussed. We have focused on CANINE-based sentence splitter results in more since the inference time of BiLSTM tokenizer is higher due to lack of parallelity in LSTMs and high input length.

Sentence splitter models were evaluated on the paragraphs obtained from BOUN Treebank's test set. The sentences in the set are grouped by 5 or 10 to form paragraphs.

6.2.1. Bi-LSTM Sentence Splitter

The Bi-LSTM-based sentence splitter's performance on dividing the paragraphs into sentences is observed in terms of character-based. The output labels "B" and "N" are used as references to convert paragraphs into sentence lists by splitting from the "B" label. The BiLSTM-based model has reached 0.97 character-based.

6.2.2. CANINE Sentence Splitter

BOUN Treebank's test set contains various sentences taken from different text resources. There are various sentence types extracted from books, articles, and recipes within the BOUN Treebank's test set. Some of the sentences are ill-formed or in the form of daily conversation while some of them are well-structured formal texts. Therefore, the evaluation results on the BOUN Treebank's test set give insights about the flexibility of the Canine Sentence Splitter. Also, some of the instances of the test set contain two or three sentences together which mislead the performance measurement. Therefore, the sentence splitter is experimented on both BOUN Treebank's test set and the wiki.conllu [30] which contains 2278 sentences extracted from Wikipedia articles.

Table 6.3. The sentence splitter model gives best results on Wikipedia paragraphs containing sentences with punctuations at the end.

	Precision	Recall	F1	Paragraph accuracy
BOUN1	0.89	0.88	0.89	0.73
BOUN2	0.92	0.93	0.92	0.78
WIKI1	0.95	0.94	0.95	0.79
WIKI2	0.96	0.97	0.97	0.86

Two versions of each evaluation set are obtained. In the first versions of the evaluation sets the sentences are kept as original and used for forming paragraphs. In the second version of evaluation sets, the sentences with no clear ending such as punctuation are kept outside when creating paragraphs so that the performance of the sentence splitter can be observed for two versions.

The first BOUN set contains 98 paragraphs whereas the second one contains 92 paragraphs. The first Wiki set contains 359 paragraphs and the second one contains 342 paragraphs.

For each evaluation set, the sentences are grouped in paragraphs so that the sentences present in the paragraphs can be extracted. The labels predicted for each character in the paragraph, are used to obtain the sentences. The sentences are obtained by processing each label-character pair where the label is predicted by the sentence splitter and the character corresponds to the target position in the paragraph. The characters labeled as "B" is the starting letter of the sentence and the others tagged as "N" are concatenated to this character to get the sentence.

The performance of the sentence splitter is measured by evaluating the sentence-based, and paragraph-based accuracies. The results for each dataset in as shown in Table 6.3.

The predicted sentence amount differs from the gold data since in some cases the sentence splitter did not label the beginning of the sentence as "B" so the splitting did not perform.

Some of the challenging points for the sentence splitter were splitting from the quotation marks, punctuations other than ".", "!", "?" indicate the end of sentence and three dots which are adjacent to the first character of the next sentence.

The performance of the sentence splitter is slightly behind the performance of the sentence segmentation model of UDPIPE [14] which has an accuracy of 0.97.

6.3. Deasciifier Results

The BOUN Treebank's test set is used for the evaluation of the Deasciifier tool. The 90% sentences were converted to their version without having any Turkish characters. The deasciifier either labeled a character as "NaN" or characters in the Turkish set. The labels produced by the deasciifier are then used for recreating the predicted sentence by the Deasciifier. To create the predicted sentence, the characters are processed one by one. The "NaN" labeled characters are taken as original, and the char-

acters labeled with a tag in the Turkish character set are converted to their Turkish corresponding.The character-based accuracy is calculated by comparing each predicted label to the gold labels.To calculate the token-level accuracy, the predicted words are compared with the gold words. The results of the deasciifier model is demonstrated on Table 6.4.

Table 6.4. CANINE Deasciifier results.

Character-level	Token-level
0.97	0.96

The deasciifier model competes with the state-of-the-art results, ITU NLP Pipeline's [31] 0.97 and Yuret's [19] 0.95 accuracies on words.

Mistakes of the Deasciifier on Wikipedia [30] texts and tweets [32] are shown in Table A.1, Table A.2, and Table A.3.

6.4. Vowelizer Results

The performance of the vowelizer is evaluated on the sentences taken from the BOUN Treebank's test set, where we randomly removed the vowels in random positions from these sentences. After the vowel removal, we inserted dummy characters after each letter in a word as in the training phase.

Table 6.5. Vowelizer results.

Dataset	Char-based F1 score	Token-based F1 score
100% of vowels missing	0.8164	0.5757
50% of vowels missing	0.9571	0.8303
20% of vowels missing	0.9022	0.8457
Random proportion missing	0.8480	0.6863

We evaluated the vowelizer on different versions of the test set, which vary in the ratio of the missing vowels in each word of the sentences. As shown in Table 6.5, the vowelizer model performs slightly better on words with 20 percent missing vowels. We think that 20 percent is a realistic ratio for missing vowels in practice. We compare the results of the vowelizer with previous studies in Table 6.6. Although the performance of the vowelizer developed in this work is behind the performance of others, we should note that the test sets are different and the comparison should be approached with care.Mistakes of the Vowelizer on Wikipedia [30] texts and tweets [32] are shown in Table B.1, Table B.2, and Table B.2.

Table 6.6. Comparison of the vowelizer with previous studies.

	Accuracy
Yuret(2006)	95.93
Zemberek(2007)	87.71
ITU	97.06
Vowelizer	84.57

6.5. POS Tagger Results

The POS tagger is also evaluated on BOUN Treebank's test set. For each sentence in the test set, the UPOS field value for each token is added to the label list in order for gold data. The results include only BERT-based model evaluation since the mT5-based model was observed to be unsuitable for this task.

The input sentences are tokenized with the pre-trained BERTurk tokenizer and ged into the trained POS tagger. The output of the POS tagger is a label id from the label list containing POS tags created during training. These label ids are converted to their label correspondent to compare with the gold data. Since BERT tokenizer is a WordPiece tokenizer, in the tokenization step the model processes more tokens than the original token count present in the sentence. Therefore the labels for the first

workpiece token of an original account are considered in evaluation. The sub-tokens produced by the BERT tokenizer are ignored so their labels are also ignored.

On the other hand, the token count and therefore label count differences are observed between the predicted tags and the gold tags because of the tokenization style of the pre-trained BERTurk tokenizer. The tokenizer tends to identify the punctuations such as apostrophes as distinct tokens. However, the gold set treats these punctuations as part of its adjacent word. For instance, the word "Büyükada'ya" is a single token in gold data whereas the model's tokenizer splits this word into 5 tokens including the apostrophe. Therefore the model predicts a label for the apostrophe even if it does not have a label in gold data.

Out of 979 sentences, 719 of the sentences contain the same amount of predicted and gold labels. In these sentences, the model predicted 7352 of the tags correctly out of 7901 total tags reaching 0.93 accuracy. And, all POS tags are predicted correctly for 378 of 719 sentences.

The POS Tagger performance is similar to the state-of-the-art results as the 0.93 accuracy in UDPIPE [14] on IMST Turkish Treebank [16].

6.6. Spell Corrector Results

To evaluate the spell corrector tool, we trained the model on different versions of the training set. We built four models and each model performs better in different cases.

We trained our first model for 100 epochs on the training set containing duplicate sentences that differ in error positioning and type. For 80 epochs, we trained the model on sentences containing errors in half of their words. In the last 20 epochs, we trained the model on a training set version with sentences without errors to see the model's performance on the sentences without spelling mistakes. Our second model differs from

the first model by the last 20 epochs. We trained the model with a set of sentences with different ratios of word errors as 0.10, 0.20, 0.50, and 0.75, and all of the words containing errors. Our third model differs from the second model in terms of the ratios of errors. We trained the last model for 20 epochs on sentences containing one word with an error in 10 words. Our last model is the ensemble of the first and the last model. During the prediction phase, we processed the predictions of the first and the second models and proceeded with the tag with the maximum probability between the combined predictions.

We evaluated the models on BOUN Treebank's test set. To see the performance of the models on different error ratios in the given sentences, we synthetically created four test sets containing the same amount of sentences differing in error ratio and positions.

Table 6.7. Spell corrector results.

Model / Error proportion by sentence	%10	%20	%50	%100
First model	0.4771	0.6217	0.8606	0.8414
Second model	0.7387	**0.7642**	0.8450	**0.8574**
Third model	**0.7414**	0.7310	0.6596	0.6344
Ensemble model	0.6179	0.7256	**0.8668**	0.8558

Table 6.7 shows the results. The ensemble model performs better on sentences with half of the words misspelled. We see that training the third model on the sentences where 10 percent of the words contain spelling errors is better on sentences containing a few erroneous words. Training the second model with different ratios increased the performance on sentences containing 20 percent misspelled words.

We also evaluated the performance of each model on each spelling error type. We compare the models on the sentences they are most successful in terms of erroneous

word ratio by sentence as shown in Table 6.7. Table 6.8 shows the results. The models perform worst in correcting character deletion mistakes, where all the models struggle to insert the deleted consonant to form the right word. Since there are many consonants, it is natural that there is a difference between the performances on correcting the vowel deletion and the consonant deletion. Moreover, we see that all models perform better when removing redundantly inserted letters in words. This result satisfies our expectations since Turkish words usually do not accept the same type of letters as consonants and vowels being adjacent.

Table 6.8. Token-based F1 scores of the spell correction models on different error types.

Model/Error type	**Duplication**	**Insertion**	**Substitution**	**Order Change**	**Consonant Deletion**	**Vowel Deletion**
First model	0.9516	0.9556	0.8533	0.8629	0.4586	0.7004
Second model	0.9608	0.9657	0.8667	0.8796	0.4750	0.7130
Third model	0.7941	0.8550	0.7536	0.7101	0.2898	0.5507
Ensemble model	0.9469	0.9610	0.8736	0.8746	0.4789	0.7269

We compare our ensemble model with the spell corrector results in the literature in Table 6.9. Since the test sets of the previous studies differ from our test set, we can only show an indirect comparison. We should note that the models in the literature were tested using single errors or multiple errors. We evaluated our model with only single errors in test sentences.

Table 6.9. Comparison of the spell corrector with previous studies.

	Accuracy
Torunoğlu-Selamet	80.7%
Burcu Can	57.50%
Zemberek	61.4%
MSWord	66.3%
Ensemble Model	86.6%

It is seen that the long sentences and long words are a challenging problem for the model since they contain many possibilities of the correct form of a word which makes it harder for the model to decide. However, in 2-4 sentences the model is seen to be successful especially if the words are not that long. Mistakes of the Spell Corrector on Wikipedia [30] texts and tweets [32] are shown in Table C.1, Table C.2, and Table C.3.

6.7. Morphologic Analyzer-Disambiguator Results

The mT5-based model was evaluated on the morphologically tagged version of the sentences in BOUN Treebank's test set. The model has two parameters, maximum new token number and beam search, which affect the results of the model. We evaluated the model with different parameter values.

Table 6.10 shows the results. We generated the morphological sequences with 500 and 1024 maximum new tokens and compared with sequence length-dependent maximum new token count (100, 200, 300). When the maximum new token count depends on the sequence length, we observed that some of the generated sequences cannot represent the input sentence fully because of the token count limit. Our model performs best on generating the correct morphological tags and lemmas of the words with 1024 as the max new token count, and top-20 filtering with 0.92 likelihood.

Table 6.10. Morphological analyzer results.

Hyperparameters			**Tag-based**
Max new token	**Top-k filtering**	**Probability**	**F1 score**
500	40	0.92	0.9243
1024	40	0.92	0.9245
100,200,300	40	0.92	0.9164
1024	20	0.92	**0.9260**
1024	40	0.95	0.9234

Table 6.11. Comparison of the morphological analyzer with previous studies.

Model/Dataset	TrMor2018
Yuret et al (Morse)	97.67 %
Yuret et al (Morse-Disambiguator)	98.59 %
Morphological Analyzer-Disambiguator	89.11 %

We compare our model with two other models in the literature using the TrMor2018 test dataset. Table 6.11 shows the comparison results. Our model reached 0.89 F1 score. We should note that the other models were trained on the TRMor2018 training dataset, while our model was trained on the BOUN Treebank training set.

7. CONCLUSION

The text pre-processing step plays a crucial role in the success of NLP applications. The text pre-processing tools are utilized for structuring the raw data and extracting information from the data such as the part-of-speech information of a word.

Since Turkish is a morphologically rich and complex language, the overall successful multilingual text-preprocessing tools may fail at working on Turkish. Also, the rule-based text pre-processing methods for the Turkish language, cannot fully capture the context of a sentence.

In this thesis study, seven different deep-learning-based Turkish text pre-processing tools are proposed where each of them is trained on Turkish texts obtained from the BOUN Treebank [28] dataset.

During the training of the proposed tools, transformers-based pre-trained models were provided by the HuggingFace Transformers Library [29]. The tokenizer tokenizes a sentence into its words. The sentence splitter extracts the sentences in long texts. The deasciifier is helpful for converting foreign letters into their equivalent form. The vowelizer and spell corrector tools play an important role in fixing spelling mistakes in a text. Lastly, the part-of-speech tagger and morphological analyzer provide information about a word in a sentence.

The proposed tools can be useful for NLP researchers to adjust and structure their raw data. Therefore, it is planned to add the proposed tools to The Turkish Language Processing Platform (TULAP) created by the members of TABILAB [1].

In further studies, to enhance the performance of Turkish text-preprocessing tools, the proposed deep-learning methods can be utilized in hybrid text-preprocessing tools where the rule-based approaches are applied. Also, the models can be trained on

long epochs and more learning parameters or bigger models such as mT5-XXL can be utilized to increase the performance of the models.

REFERENCES

1. Boğaziçi University Computer Engineering Department, "Text Analytics and BIoInformatics Lab", 2020, https://tabilab.cmpe.bogazici.edu.tr/, accessed on July 21, 2024.

2. Hochreiter S. and J. Schmidhuber, "Long Short-Term Memory" *Neural Computation*, Vol. 9, No. 8, pp. 1735—1780, 1997.

3. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez , L. Kaiser, and I. Polosukhin, Attention is All You Need, *Advances in Neural Information Processing Systems*, edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, USA, Vol.30, pp. 5998–6008, 2017.

4. Devlin, J., M. W. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1, pp. 4171–4186, Association for Computational Linguistics, Minneapolis, Minnesota, 2019.

5. Schuster M. and K. Nakajima, "Japanese and Korean voice search", *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Kyoto, Japan, pp. 5149–5152, 2012.

6. Clark, J. H., D. Garrette, I. Turc and J. Wieting, "Canine: Pre-training an Efficient Tokenization-Free Encoder for Language Representation", *Transactions of the Association for Computational Linguistics*, Vol.10, pp. 73–91, 2022.

7. Raffel C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and J. L. Peter, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer", *The Journal of Machine Learning Research*, Vol. 21, No. 1.40, pp. 5485–5551, 2020.

8. Xue, L., N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua and C. Raffel, "mT5: A massively multilingual pre-trained text-to-text transformer",*Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Online, pp. 483–498, Association for Computational Linguistics, 2021.

9. Eryiğit, G., "ITU Turkish NLP Web Service", *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, edited by S. Wintner, M. Tadi c and B. Babych, , Gothenburg, Sweden, pp. 1—4, 2014.

10. Adali, K. and Eryiğit G., "Vowel and Diacritic Restoration for Social Media Texts", *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)*, edited by Farzindar A., D. Inkpen, M. Gamon and M. Nagarajan, Gothenburg, Sweden, pp. 53—61, 2014.

11. Torunoğlu–Selamet, D., E. Bekar, T. Ilbay and G. Eryiğit, "Exploring Spelling Correction Approaches for Turkish", *Proceedings of The First International Conference on Turkic Computational Linguistics TurCLing 2016*, Konya, Turkey, pp. 7–11, 2016.

12. Şahin M., U. Sulubacak, and G. Eryiğit, "Redefinition Of Turkish Morphology Using Flag Diacritics", *Proceedings of The Tenth Symposium on Natural Language Processing (SNLP-2013)*, Phuket, Thailand, 2013.

13. Eryiğit, G. and D. Torunpğlu-Selamet, "Social Media Text Normalization for Turkish", *Natural Language Engineering*, Vol. 23, No. 6, pp.835–875, 2017.

14. Straka M., J. Straková and J. Hajic, "Evaluating Contextualized Embeddings on 54 Languages in POS Tagging, Lemmatization and Dependency Parsing", *CoRR*, 2019.

15. Straka M., "UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task", *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, Brussels, Belgium, pp. 197-207, 2018.

16. Sulubacak, U., M. Gokirmak, F. Tyers, Ç. Çöltekin, J. Nivre and G. Eryiğit, "Universal Dependencies for Turkish", *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, edited by Matsumoto Y. and R. Prasad, Osaka, Japan, pp. 3444–3454, 2016.

17. Zeman D. and J. Hajič, "CoNLL 2018:Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies", 2018, https://universaldependencies.org/conll18/, accessed on May 10, 2024.

18. Göker, S. and B. Can, "Neural Text Normalization for Turkish Social Media", *3rd International Conference on Computer Science and Engineering (UBMK)*, Sarajevo, Bosnia and Herzegovina, pp.161–166, 2018.

19. Yuret, D., M. de la Maza. "The Greedy Prepend Algorithm for Decision List Induction", *Computer and Information Sciences – ISCIS 2006*, edited by Levi, A., E. Savaş, H. Yenigün, S. Balcısoy, Y. Saygın, Lecture Notes in Computer Science, Vol 4263. Berlin, Heidelberg, 2006.

20. Stankevicius, L., M. Lukosevicius, J. Kapociute-Dzikiene, M. Briediene and T. Krilavicius, "Correcting Diacritics and Typos with ByT5 Transformer Model", Applied Sciences, Vol. 12, No. 5, 2636, 2022.

21. Kuznetsov, A. and H. Urdiales, "Spelling Correction with Denoising Transformer", *CoRR*, 2021.

22. Mengyi G., X. Canran, and S. Peng, "Hierarchical Character Tagger for Short Text Spelling Error Correction", *Proceedings of the 2021 EMNLP Workshop W-NUT: The Seventh Workshop on Noisy User-generated Text*, Punta Cana, Dominican Republic, pp. 106–113, 2021.

23. Do, DT., H.T. Nguyen, T.N. Bui, and H.D. Vo, "VSEC: Transformer-Based Model for Vietnamese Spelling Correction", *PRICAI 2021: Trends in Artificial Intelligence*, edited by Pham, D.N., T. Theeramunkong, G. Governatori and F. Liu, Hanoi, Vietnam, Vol. 13032, pp. 259–272, 2021.

24. Sak, H., T. Gungor and M. Saraclar, "Morphological Disambiguation of Turkish Text with Perceptron Algorithm", *Computational Linguistics and Intelligent Text Processing*, pp. 107–118, 2007.

25. Hakkani-T ur, D. Z., K. Oflazer and G. Tür, "Statistical Morphologi- cal Disambiguation for Agglutinative Languages", *Computers and the Humanities*, Vol.36, No.4, pp. 381–410, 2002.

26. Akyürek E., E .Dayanık, ad D. Yuret, "Morphological Analysis Using a Sequence Decoder", *Transactions of the Association for Computational Linguistics* Vol. 7, No.4, pp. 567—579, 2019.

27. Universal Dependencies, "Universal Dependencies", 2024, https://universaldependencies.org/, accessed on May 10, 2024.

28. Marşan, B., S. F. Akkurt, M. Şen, M. Gürbüz, O. Güngör, Ş. B. Özateş, S. Üsküdarlı, A. Özgür, T. Güngör and B. Öztürk, "Enhancements to the BOUN Treebank Reflecting the Agglutinative Nature of Turkish", *Proceedings of the ALTNLP The International Conference and workshop on Agglutinative Language Technologies as a challenge of Natural Language Processing*, Koper, Slovenia, Vol. 3315, pp. 71–80, 2022.

29. Hugging Face, "Hugging Face Transformers Library", 2024, https://huggingface.co/docs/transformers/index, accessed on May 10, 2024.

30. Kayadelen, T., A. Öztürel and B. Bohnet, "A Gold Standard Dependency Treebank for Turkish", *Proceedings of The 12th Language Resources and Evaluation Conference*, Marseille, France, pp. 5158–5165, 2020.

31. Adali, K. and G. Eryi˘git, "Vowel and Diacritic Restoration for Social Media Texts", *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)*, edited by Farzindar A., D. Inkpen, M. Gamon and M. Nagarajan, Gothenburg, Sweden, pp. 53—61, 2014.

32. Z. A. Güven Et Al. , "Comparison of N-stage Latent Dirichlet Allocation Versus Other Topic Modeling Methods For Emotion Analysis," *JornalL of The Faculty of Engineering And Architecture of Gazi University*, Vol.35, No.4, pp.2135-2145, 2020.

APPENDIX A: DEASCIIFIER MISTAKES

This section demonstrates the errors of the deasciifier on different types of texts.

Table A.1. Deasciifier mistakes on first batch of Wikipedia sentences.

	Input sentence	Predicted sentence
1	Cumhuriyet, siyasi gucun halk ve temsilcileri tarafindan paylasildigi bir devlet yonetim seklidir ve yapisi geregi monarsinin yoklugu uzerine kuruludur.	Cumhuriyet, siyasi gücün halk ve temsilcileri tarafından paylaşıldığı bir devlet yönetim şeklidir ve yapısı gereği **monarsının** yokluğu üzerine kuruludur.
2	Bir cumhuriyette temsil, genel vatandaslar tarafindan serbestce secilebilir veya secimle belirlenebilir.	Bir cumhuriyette temsil, genel vatandaşlar tarafından serbestçe seçilebilir veya seçimle belirlenebilir.
3	Bircok tarihi cumhuriyette, temsil kisisel statuye dayanmis ve secimlerin rolu sinirli olmustur. Bu durum gunumuzde de gecerlidir; resmi adlarinda "cumhuriyet" kelimesini kullanan 159 devlet (2017 itibariyla) ve diger cumhuriyet olarak kurulmus devletler, temsil hakkini ve secim surecini dar bir sekilde sinirlayan devletler arasinda yer almaktadir.	Birçok tarihi cumhuriyette, temsil kişisel statüye dayanmış ve seçimlerin rolü sınırlı olmuştur. Bu durum günümüzde de geçerlidir; resmi adlarında "cumhuriyet" kelimesini kullanan 159 devlet (2017 itibarıyla) ve diğer cumhuriyet olarak kurulmuş devletler, temsil hakkını ve seçim sürecini dar bir şekilde sınırlayan devletler arasında yer almaktadır.

Table A.1. Deasciifier mistakes on first batch of Wikipedia sentences (cont.)

	Input sentence	Predicted sentence
4	Bu terim, MO 509'da kralların devrilmesinden MS 27'de İmparatorluğun kurulmasına kadar süren Antik Roma Cumhuriyeti'nin anayasasına atıfta bulunarak modern anlamını geliştirmiştir.	Bu terim, **MO** 509'da kralların devrilmesinden MS 27'de İmparatorluğun kurulmasına kadar süren Antik Roma Cumhuriyeti'nin anayasasına **atıfta** bulunarak modern anlamını geliştirmiştir.
5	Bu anayasa, etkili bir şekilde güç sahibi olan zengin **şöylulardan** oluşan bir Senato; magistraları seçme ve yasaları kabul etme yetkisine sahip tüm özgür vatandaşların katıldığı birkaç halk meclisi; ve çeşitli türde sivil ve siyasi yetkilere sahip bir dizi **mağistratlıktan** oluşuyordu.	Bu anayasa, etkili bir şekilde güç sahibi olan zengin **şöylulardan** oluşan bir Senato; magistraları seçme ve yasaları kabul etme yetkisine sahip tüm özgür vatandaşların katıldığı birkaç halk meclisi; ve çeşitli türde sivil ve siyasi yetkilere sahip bir dizi **mağistratlıktan** oluşuyordu.

Table A.2. Deasciifier mistakes on second batch of Wikipedia sentences.

	Input sentence	Predicted sentence
1	Yapay zekâ ya da kısaca YZ, (İngilizce: Artificial intelligence ya da kısaca **AI**), insanlar da dahil olmak üzere hayvanlar tarafından, doğal zekânın aksine makineler tarafından görüntülenen zekâ çeşididir.	Yapay zekâ ya da kısaca YZ, (İngilizce: Artificial intelligence ya da kısaca **AI**), insanlar da dahil olmak üzere hayvanlar tarafından, doğal zekânın aksine makineler tarafından görüntülenen zekâ çeşididir.
2	İlk ve ikinci kategoriler arasındaki ayrım genellikle seçilen kısaltmayla ortaya çıkar.	İlk ve ikinci kategoriler arasındaki ayrım genellikle seçilen kısaltmayla ortaya çıkar.
3	Güçlü yapay zeka genellikle Yapay genel zekâ (İngilizce: Artificial General **Intelligence** kelimelerinin kısaltımı olarak: **AGI**) olarak etiketlenirken, doğal zekayı taklit etme girişimleri yapay biyolojik zeka (İngilizce: Artificial Biological **Intelligence**: **ABI**) olarak adlandırılır.	Güçlü yapay zeka genellikle Yapay genel zekâ (İngilizce: Artificial General **Intelligence** kelimelerinin kısaltımı olarak: **AGI**) olarak etiketlenirken, doğal zekayı taklit etme girişimleri yapay biyolojik zeka (İngilizce: Artificial Biological **Intelligence**: **ABI**) olarak adlandırılır.

Table A.2. Deasciifier mistakes on second batch of Wikipedia sentences (cont.)

	Input sentence	Predicted sentence
4	Önde gelen yapay zeka ders kitapları, alanı zeki etmenlerin çalışması olarak tanımlar: Çevresini algılayan ve hedeflerine başarıyla ulaşma şansını en üst düzeye çıkaran eylemleri gerçekleştiren herhangi bir **cihaz**.	Önde gelen yapay zeka ders kitapları, alanı zeki etmenlerin çalışması olarak tanımlar: Çevresini algılayan ve hedeflerine başarıyla ulaşma şansını en üst düzeye çıkaran eylemleri gerçekleştiren herhangi bir **cihaz**.
5	Halk arasında, yapay zeka kavramı genellikle insanların insan zihni ile ilişkilendirdiği öğrenme ve problem çözme gibi **bilissel** eylemleri taklit eden makineleri tanımlamak için kullanılır.	Halk arasında, yapay zeka kavramı genellikle insanların insan zihni ile ilişkilendirdiği öğrenme ve problem çözme gibi **bilissel** eylemleri taklit eden makineleri tanımlamak için kullanılır.

Table A.3. Deasciifier mistakes on tweets.

	Input sentence	Predicted sentence
0	Sabah sabah Cok Guldum Ellerine saglik emegi gecenlerin	Sabah sabah Çok Güldüm Ellerine sağlık emeği geçenlerin
1	Guldum, eglendim keyfim yerine geldi ya	Güldüm, eğlendim keyfim yerine geldi ya
2	hersey icin tesekkur ederim yine harikasiniz ve iyi ki varsiniz	herşey için teşekkür ederim yine harikasınız ve iyi ki varsınız
3	Selim Ileriye samimi mektubu ve nezaketi icin tesekkur ederim	Selim İleriye samimi mektubu ve nezaketi için teşekkür ederim
4	Hazir misinizz sergiye geldik ve bunlaru asmislar cok mutluyum	Hazır mısınızz sergiye geldik ve bunlaru aşmışlar çok mutluyum
5	kardes tip bayramin mubarek olsun	kardeş tıp bayramın mübarek ölsün
6	o kadar mutluyum ki bazen twet atmayi unutuyorum	o kadar mutluyum ki bazen twet atmayı unutuyorum
7	Asiri heyecanliyim sanki 20 gun degil de aylar olmus gibi hesabi kapatali, sizleri cok ozledim	Aşırı heyecanlıyım sanki 20 gün değil de aylar olmuş gibi hesabı kapatalı, sizleri çok özledim
8	cok heyecanliyim 15 gun boyunca uyu- mayi planliyorum	çok heyecanlıyım 15 gün boyunca uyu- mayı planlıyorum
9	Kurucumuz Gazanfer Ozcan'a ozlemle Dogum gunu kutlu olsun !	Kurucumuz Gazanfer Özcan'a özlemle Doğum günü kutlu olsun !
10	Elektrikler yok delirecegim sarjim ol- mus %10	Elektrikler yok delireceğim **sarjım** olmuş %10

APPENDIX B: VOWELIZER MISTAKES

This section demonstrates the errors of the vowelizer on different types of texts.

Table B.1. Vowelizer mistakes on first batch of Wikipedia sentences.

	Input sentence	Predicted sentence	Correct word
1	Cmhuryet, siysi gücn halk ve tmsilcleri tarfındn paylşldığı bir devlt yönetm şklidir ve ypısı greği monarşnn yokluğ üzrne kruludr.	Cumhuriyet, siyasi gücün halk ve temsilcileri tarafından paylaşıldığı bir devlet yönetim şeklidir ve yapısı gereği **monarşınınyokluğu** üzerine kuruludur.	monarşinin
2	Bir cumhuriytt tmsil, genl vtandaşlr tarfndan srbestçe sçilbilir vya seçmle belrlenblir.	Bir cumhuriyette temsil, genel vatandaşlar tarafından serbestçe seçilebilir veya seçimle belirlenebilir.	-
3	Brçok tarhi cumhriytte, tmsil kşisel statüy dyanmış ve seçmlern rlü sınırl olmştur. Bu durm günmüzd de geçrlidr; rsmi adlrnda ”cumhuryt” klimsini kullnan 159 dvlet (2017 itibryla) ve diğr cumhryet olrak kurlmuş devletlr, tmsil hakknı ve sçim srecin dar bir şeklde snırlyan dvletler arsınd yer almaktdr.	Birçok tarihi cumhuriyette, temsil kişisel statüye dayanmış ve seçimlerin rolü sınırlı olmuştur. Bu durum günümüzde de **geçerlidir**; resmi adlarında ”cumhuriyet” kelimesini kullanan 159 devlet (2017 itibarıyla) ve diğer cumhuriyet olarak kurulmuş devletler, temsil hakkını ve seçim **sürecini** dar bir şekilde sınırlayan devletler arasında yer almaktadır.	geçerlidir sürecini

Table B.1. Vowelizer mistakes on first batch of Wikipedia sentences (cont.)

	Input sentence	Predicted sentence	Correct word
4	Bu trim, MÖ 509'da krallrın devrlmesindn MS 27'de İmpartorlğn krulmsına kdar sren Antk Rom Cumhriyt'nin anyasasn atıft bulnrak modrn anlmnı geliştrmiştr.	Bu terim, MÖ 509'da **kuralların** devrilmesinden MS 27'de İmparatorluğun kurulmasına kadar süren Antik Roma Cumhuriyet'nin anayasasına atıfta bulunarak modern anlamını geliştirmiştir.	kralların

Table B.2. Vowelizer mistakes on second batch of Wikipedia sentences.

	Input sentence	Predicted sentence	Correct sentence
1	Ypay zkâ ya da ksaca YZ, (İnglzce: Artificl intlligenc ya da ksaca A), insanlr da dahl olmk üzre hyvanlar tarafndn, doğl zekânn aksin mkneler tarafndn görntlenen zek çeşddir.	Yapay zekâ ya da kısaca YZ, (İngilizce: Artifical **intelligenci**ya da kısaca **A**), insanlar da **dahili** olmak üzere hayvanlar tarafından, doğal zekânın **aksini makineleri** tarafından görüntülenen zeka **çeşidedir**.	intelligence AI dahil aksine makineler çeşididir
2	İlk ve ikinc ktegorler arsındak ayrım genllkle seçiln ksaltmayl ortay çkar.	İlk ve ikinci **kitegorler** arasındaki ayrım genellikle seçilen kısaltmayla ortaya çıkar.	kategoriler
3	Güçl yapy zek gnellikl Yapy gnel zkâ (nglizce: Artficil Generl Intlligenc klmelrinin ksltılmışı olrak: AG) olrak etktlenirkn, dğal zekyı tklit etm girişmler yapy biylojk zek (nglizce: Artficil Bolgical Intlligenc: AB) olrak adl- ndrılır.	Güçlü yapay zeka genellikle Yapay genel zekâ (**engelizce**: **Artificil** General **Intilligence** kelimelerinin **kısıltılmışı** olarak: **AG**) olarak **etketlenirken**, doğal zekâyı **takalit** etme girişimleri yapay biyolojik zeka (**engelizce**: **Artficil Bolgical** **Intilligence**: **AB**) olarak ad- landırılır.	İngilizce Artificial Intelligence AGI etiketlenirken taklit Biological ABI

Table B.2. Vowelizer mistakes on second batch of Wikipedia sentences (cont.)

	Input sentence	Predicted sentence	Correct sentence
4	Önd glen ypay zek ders kitaplr, alnı zki etmnlrin çlşması olark tnımlar: Çevrsin algılyn ve hedeflrne başrıyl ulaşm şansnı en üst dzeye çıkarn eylmlri grçekleştren herhngi bir cihz.	Önde gelen yapay **zeki** ders **kitaplar**, **alını** zeki etmenlerin çalışması olarak tanımlar: Çevresini algılayan ve hedeflerine başarıyla ulaşma şansını en üst düzeye çıkaran eylemleri gerçekleştiren herhangi bir cihaz.	zeka kitapları alanı
5	Halk arsınd, yapy zka kavrmı gnellikl insanlrn nsn zhni ile ilşkilndrdiği öğrenm ve problm çzme gib bilişsl eylemlr taklt edn makinler tanmlmak içn kllanlır.	Halk arasında, yapay zeka kavramı genellikle insanların insan zihni ile ilişkilendirdiği **öğrenime** ve problem çözme gibi bilişsel eylemleri taklit eden makineleri **tanımlamak** için kullanılır.	öğrenme tanımlamak

Table B.2. Vowelizer mistakes on tweets.

	Input sentence	Predicted Sentence	Correct word
0	Sabh sbah Çok Güldm Ellrne sğlık emği geçnlern	Sabah sabah Çok Güldüm Ellerine sağlık emeği geçenlerin	-
1	Gldüm, eğlndim kyfim yerne gldi ya	Güldüm, eğlendim **kıyfim** yerine geldi ya	keyfim
2	herşy için tşekkür edrim yne hrikasınz ve iyi ki varsnız	**herşeyi** için teşekkür ederim yine harikasınız ve iyi ki varsınız	herşey
3	Slim İlery smimi mektbu ve nzaket içn teşkkür edrim	Selim **İleriyi** samimi mektubu ve nezaket için teşekkür ederim	İleriye
4	Hazr msınızz sergiy gldik ve bnlaru asmşlar çok mtluyum	Hazır **mısınızız** sergiye geldik ve bunlaru **asmışları** çok mutluyum	mısınızz asmışlar
5	kardş tıp bayramn mübrek olsn	kardeş tıp bayramın **müebrek** olsun	mübarek
6	o kdar mutluym ki bzen twet atmyı untuyorm	o kadar mutluyum ki bazen **tawet** atmayı **unutuyorma**	twet unutuyorum
7	Aşır heyecanlym snki 20 gün değl de aylr olmş gib hsabı kptalı, sizlri çok özldim	Aşırı heyecanlıyım sanki 20 gün değil de aylar olmuş gibi hesabı **kaptalı**, sizleri çok **özeldim**	kapatalı özledim
8	çok heyecnlıym 15 gün byunca uyumy planlyrum	çok heyecanlıyım 15 gün boyunca **uyumaya** planlıyorum	uyumayı
9	Kurcmuz Gazanfr Özcan' özlmle Dğum gün ktlu olsn !	**Kurcumuz** Gazianfer **Özcan**' özlemle Doğum **gün** kutlu olsun !	Kurucumuz Özcan'a günü
10	Elktriklr yok delirecğm şarjm olmş %10	Elektrikler yok delireceğim şarjım olmuş %10	-

APPENDIX C: SPELL CORRECTOR MISTAKES

This section demonstrates the errors of the spell corrector on different types of texts.

Table C.1. Spell corrector mistakes on second batch of Wikipedia Sentences.

	Input sentence	Predicted sentence	Correct word
1	Yapya zkeâ ya da kısacqa YZ, (İngilizce: Artificialk intelligencd ya da kısaca AI), insalar da dahil olmak üzere hayvnlar tarafından, doğaal zekânın aksien makineelr tarafındabn görüntülenenb zelâ çeşidiir.	**Yaypa** zekâ ya da kısaca YZ, (İngilizce: Artificial intelligence ya da kısaca AI), insanlar da dahil olmak üzere hayvanlar tarafından, doğal zekânın aksine makineler tarafından görüntülenen **zelâ çeşidilir**.	Yapay zekâ çeşididir
2	İlk ve iknci kategoriler arasındaki ayrım genellikel seçilen kısaltmayla ortyaa çıkatr.	İlk ve ikinci kategoriler arasındaki ayrım genellikle seçilen kısaltmayla ortaya çıkar.	-

Table C.1. Spell corrector mistakes on second batch of Wikipedia Sentences (cont.)

	Input sentence	Predicted sentence	Correct word
3	Güçlü yapayu zekq genelikle Yapy geenel zkeâ (İngliizce: Artificiakl General Intelligence kelimelerinin kısaltılmışı olarak: AGI) olarak etiketlenirken, doğalk zekayı taklit etme girişimlerü yaay biyolojik zka (İngilizce: Artificcial Biologicla Intelilgence: ABI) olaralk adlandırılır.	Güçlü yapay zeka genellikle **Yapıy** genel zekâ (İngilizce: **Artificil** General Intelligence kelimelerinin kısaltılmışı olarak: AGI) olarak etiketlenirken, doğal zekayı taklit etme girişimleri **yasay** biyolojik **zaka** (İngilizce: Artificial **Biologilla Intelilgence**: ABI) olarak adlandırılır.	Yapay Artificial zeka Biological Intelligence
4	Önde gelenm yapau zea ders kitapları, alnı zeeki etmelnerin açlışması olarak tanımlatr: Çevresiniü algılayan ve hedeflerind başaıyla ulaşma şansını en üst düzeye çıkaran eylemleri gerçekleştirn herhhangi bir cihaz.	Önde gelen **yapau zeya** ders kitapları, **alını** zeki etmenlerin çalışması olarak tanımlar: Çevresini algılayan ve hedeflerine başarıyla ulaşma şansını en üst düzeye çıkaran eylemleri gerçekleştiren herhangi bir cihaz.	yapay zeka alanı
5	Halk aarsında, yaapy zekqa kavramı genellikle insanlarınb insam zihni ile ilişkilendirdiği öğenme ve problm çözme gibi billişsel eyelmleri taklit deen makinelerüi tanımlamak içinm kullanılır.	Halk arasında, yapay zeka kavramı genellikle insanların insan zihni ile ilişkilendirdiği öğrenme ve problem çözme gibi bilişsel eylemleri taklit eden makineleri tanımlamak için kullanılır.	-

Table C.2. Spell corrector mistakes on third batch of Wikipedia Sentences.

	Input sentence	Predicted sentence	Correct word
1	”uFtbol” keliemsi Türkçeye, İngilizcedekşi footr (”ayak”) ve ball (”tıp”) kelimelerinin birletirilmesiyle oluşturulan football kelimesindn geçmiştir.	”Futbol” kelimesi Türkçeye, İngilizcedeki **footr** (”ayak”) ve **balk** (”**tıp**”) kelimelerinin **birletirilmesiyle** oluşturulan football kelimesinden geçmiştir.	foot ball top birleştiril- mesiyle
2	İngilizcedde football adını tşaıyan dieğr futbol sporlarındabn ayırmakm amacıyls, moden futbolun ilk kurallarını belrleyen kurrumun adı oaln Futbol Birliğinden (İngilizce: The Football Association) yola çıkılaralk association footballo (”birlil futbolu”) ifadesi kullanılmatadır.	**İngilizcedde** footbal adını taşıyan diğer futbol sporlarından ayırmak amacıyla, **moden** futbolun ilk kurallarını belirleyen kurumun adı olan Futbol Birliğinden (İngilizce: The **Fotoball** Association) yola çıkılarak association **footballo** (”birlik futbolu”) ifadesi **kullanılmatadır**.	İngilizcede modern Football kullanılmak- tadır
3	Football association ifdesindeki soc hecessine -er eki getirielrek oluşturulan ve nİgilizce konuşamn bazı ülkelerded futboku tanımamak için kullanılan soccr kellimesinin çıkşı 1880’lere denk gelir.	Football **associatiyon fedesindeki** soc hecesine -er eki getirilerek oluşturulan ve İngilizce konuşan bazı ülkelerde futbolu **tanımamak** için kullanılan **soccur** kelimesinin çıkışı 1880’lere denk gelir.	association ifadesindeki tanımlamak soccer

Table C.2. Spell corrector mistakes on third batch of Wikipedia Sentences (cont.)

	Input sentence	Predicted sentence	Correct word
4	Günümüzde, İngilizce konuşamn ülkelerin bazılarık futbolu tanımlaak içn yaalnızca football ifaedsini kullanırkebn bu kelimer diğer bazo dillere değişeek girmş ve bu şekilde kullanılmaya başlaanmıştır. Bazı dillered fubtolu tanımlamak içibn özgünm kelimeler kullanırken bazılarınds ise oot ve ball kelimelerinin o dillerdki karşılıklarrı kullanılarak bu kelmieler birleştirilmiş ve futbol iafdesi bu şekildwe yer edinmiştirt.	Günümüzde, İngilizce konuşan ülkelerin bazıları futbolu **tanımlarak** için yalnızca **fotoball** ifadesini kullanırken bu kelime diğer bazı dillere değişerek girmiş ve bu şekilde kullanılmaya başlanmıştır. Bazı dillerde futbolu tanımlamak için özgün kelimeler kullanırken bazılarında ise **odot** ve ball kelimelerinin o dillerdeki karşılıkları kullanılarak bu kelimeler birleştirilmiş ve futbol ifadesi bu şekilde yer edinmiştir.	tanımlamak football foot

Table C.3. Spell corrector mistakes on tweets.

	Input sentence	Predicted sentence	Correct word
0	Sabah sabah Çok Gülüdm Elelrine sağlılk emeğiü geçenlerin	Sabah sabah Çok Güldüm Ellerine sağlık emeği geçenlerin	-
1	Güldüm, eğlendim keyfim yeine gldi ya	Güldüm, eğlendim keyfim yerine geldi ya	-
2	herşey iççin teşekükr deerim yine harikasınıaz ve iyi ki varsınızs	herşey için teşekkür ederim yine harikasınız ve iyi ki varsınız	-
3	Selim İleriye samimü mektubu ve nezaketi iin teşekkr ederrim	Selim İleriye samimi mektubu ve nezaketi için **teşekker** ederim	teşekkür
4	Hazrı mısınızz segriye geldilk ve bunlaru asmışlart çok mutluyum	**Hazrı** mısınız sergiye geldik ve **bunlaru** asmışlar çok mutluyum	Hazır bunları
5	kardrş tıp bayramın mübarek olun	kardeş tıp bayramın mübarek **olun**	olsun
6	o kadar mutluym ki bazen twwet atmaıy nuutuyorum	o kadar **mutluym** ki bazen **taet** atmayı unutuyorum	mutluyum tweet
7	Aşırı heyecanlıyıkm sankiş 20 gün değio de aylar oluş gbi hesabı kapatalı, sizlerri çok özldeim	Aşırı heyecanlıyım sanki 20 gün değil de aylar olmuş gibi hesabı kapatalı, sizleri çok özledim	-

Table C.3. Spell corrector mistakes on tweets (cont.)

	Input sentence	Predicted sentence	Correct word
8	çok heyecanlııym 15 gün boyuncwa uyumayıı planlıyorum	çok heyecanlıyım 15 gün boyunca uyumayı planlıyorum	
9	Kurucumua Gazafer Özcn'a özlemlle Doğum günü kutlu olsun !	Kurucumuz **Gazaffer Özcun'a** özlemle Doğum günü kutlu olsun !	Gazanfer Özcan'a
10	Elektirkler yok delireecğim şarjım olnmuş %10	**Elektirkler** yok delireceğim şarjım olmuş %10	Elektrikler