MACHINE TRANSLATION USING TRANSFORMERS FOR TURKISH TO RUSSIAN

2024
MASTER THESIS
COMPUTER ENGINEERING

Nurzhan AMANTAY

Thesis Advisor
Assist. Prof. Dr. Yasin ORTAKCI

MACHINE TRANSLATION USING TRANSFORMERS FOR TURKISH TO RUSSIAN

Nurzhan AMANTAY

Thesis Advisor
Assist. Prof. Dr. Yasin ORTAKCI

T.C.
Karabuk University
Institude of Graduate Programs
Department of Computer Engineering
Prepared as
Master Thesis

KARABUK
December 2024

I certify that, in my opinion, the thesis submitted by Nurzhan AMANTAY titled "MACHINE TRANSLATION USING TRANSFORMERS FOR TURKISH TO RUSSIAN" is fully adequate in scope and quality as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Yasin ORTAKCI .....
Thesis Advisor, Department of Computer Engineering

This thesis is accepted by the examining committee with a unanimous vote in the Department of Computer Engineering as a Master of Science thesis. December 10, 2024

Examining Committee Members (Institutions)	Signature
Chairman : Assist. Prof. Dr. Yasin ORTAKCI (KBU)	.....
Member : Assist. Prof. Dr. Ahmet KARAOĞLU (SNU)	.....
Member : Assist. Prof. Dr. Oğuzhan MENEMENCİOĞLU (KBU)	.....

The degree of Master of Science by the thesis submitted is approved by the Administrative Board of the Institute of Graduate Programs, Karabuk University.

Assoc. Prof. Dr. Zeynep ÖZCAN .....
Director of the Institute of Graduate Programs

"I hereby state that all the information incorporated in this thesis has been collected and presented in accordance with academic regulations and ethical principles. Moreover, I have conscientiously adhered to the demands specified by these regulations and principles, duly acknowledging all sources referenced in this work that are not original to it."

Nurzhan AMANTAY

ABSTRACT

M. Sc. Thesis

MACHINE TRANSLATION USING TRANSFORMERS FOR TURKISH TO RUSSIAN

Nurzhan AMANTAY

Karabuk University
Institute of Graduate Programs
Department of Computer Engineering

Thesis Advisor:
Assist. Prof. Dr. Yasin ORTAKCI
December 2024, 56 pages

Machine translation (MT) represents a pivotal area in the field of natural language processing, continuously striving for advancements. The transformer architecture, which employs the attention mechanism, has surpassed popular neural models such as Recurrent Neural Networks (RNNs) and Bidirectional Long Short-Term Memory (BiLSTM) in terms of memory usage and efficiency. This research investigates the application of the transformer in a neural MT model for Turkish to Russian. The neural model was trained using the AuroraDataset, which comprising 476K Turkish-to-Russian parallel sentence pairs. Our transformer model consists of six encoders and six decoders, each with eight multi-head self-attention mechanisms. After training over 30 epochs, the model achieved a BLEU score of 0.4903, a WER of 0.2915, and a CER of 0.3935. These results underscore the potential of the transformer architecture in addressing the complexities of Turkish-to-Russian translation, particularly

considering the linguistic disparities and grammatical intricacies between the two languages.

This paper presents a comprehensive analysis of the model's architecture and implementation. The findings contribute to the understanding of transformer-based MT systems and their applicability to languages with limited text corpora, such as Turkish.

Keywords : Neural Machine Translation, Transformer, Self-Attention, Encoder & Decoder, Turkish To Russian Machine Translation.

Science Code : 92432

ÖZET

Yüksek Lisans Tezi

TRANSFORMERS KULLANARAK
TÜRKÇE'DEN RUSÇA'YA MAKİNE ÇEVİRİSİ

Nurzhan AMANTAY

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı:
Dr. Öğr. Üyesi Yasin ORTAKCI
Aralık 2024, 56 sayfa

Makine çevirisi (MT), sürekli gelişim gösteren, doğal dil işleme alanında kilit bir konu olarak öne çıkmaktadır. Attention mekanizmasını kullanan transformer mimarisi, verimlilik ve bellek kullanımı açısından Tekrarlayan Sinir Ağları (RNN'ler) ve Çift Yönlü Uzun Kısa Süreli Bellek (BiLSTM) gibi popüler sinirsel modelleri geride bırakmıştır. Bu araştırma, Türkçe-Rusça için bir sinirsel makine çeviri modelinde Transformer mimarisinin uygulanmasını incelemektedir. Sinirsel model 476 bin Türkçe-Rusça paralel cümle çifti içeren AuroraDataset verisetini kullanılarak eğitilmiştir. Transformer modelimiz, attention mekanizmasına sahip her biri sekiz multi-head ve altı encoder ve altı decoder'den oluşmaktadır. Model, 30 epoch eğitimden sonra 0.4903 BLEU puanı, 0.2915 WER ve 0.3935 CER sonuçlarına ulaşmıştır. Bu sonuçlar transformer mimarisinin Türkçe ile Rusça arasındaki dil farklılıkları ve dilbilgisel karmaşıklıkları göz önüne alındığında, Türkçe'den Rusça'ya çevirideki potansiyelini ortaya koymaktadır.

Bu makale, modelin mimarisi ve uygulamasına dair ayrıntılı bir analiz sunmaktadır. Bulgular, sınırlı metin derlemine sahip diller, örneğin Türkçe için Transformer tabanlı MT sistemlerinin anlaşılmasına katkıda bulunmaktadır.

Anahtar Kelimeler : Sinirsel Makine Çevirisi, Transformer, Self-Attention, Encoder & Decoder, Türkçe'den Rusça'ya Makine Çevirisi.

Bilim Kodu : 92432

ACKNOWLEDGEMENT

I am grateful to my advisor, Assist. Prof. Dr. Yasin ORTAKCI, for guiding me through this thesis. His advice helped me shape my research and complete it successfully.

I am also grateful to Fatih BALKI, Enver YILDIRIM, Salih SARP, Enes GÜMÜŞKAYNAK, and Vladislav KULEIKIN for their feedback, suggestions, contributions, and support during this research.

Thanks also to Aurora Bilişim Research Lab. Organization for their research funding and computational resources.

Finally, I would like to thank my wife, Izet, for her support and love.

CONTENTS

Page

Page

Page

LIST OF FIGURES

Page

Page

LIST OF TABLES

Page

SYMBOLS AND ABBREVIATIONS INDEX

NLP : Natural Language Processing
MT : Machine Translation
RNNs : Recurrent Neural Networks
NLP : Natural Language Processing
BiLSTM : Bidirectional Long Short-Term Memory
SVO : Subject-Object-Verb
WER : Word Error Rate
CER : Character Error Rate
SMT : Statistical Machine Translation
NMT : Neural Machine Translation
SL : Source Language
TL : Target Language
RBMT : Rule-based Machine Translation
CBMT : Corpus-Based Machine Translation

PART 1

INTRODUCTION

The issue of linguistic barriers among individuals has been a long-standing concern for humanity. The act of communication between individuals who speak different languages is made possible through the intermediary of human translation. Nevertheless, the availability of high-quality human translation services remains a persistent challenge, largely due to the limited number and high cost of such skilled individuals. The unavailability and expense of human translation, coupled with the advances in Natural Language Processing (NLP) and computer science, have given rise to the concept of Machine Translation (MT), which aims to automate the process of translating languages.

MT is a significant task focusing that uses computer technology to facilitate the translation text from one language to another. Early studies in this field include rule-based and statistical approaches. However, these approaches face difficulties in translating long sentences and capturing the context effectively. Subsequently, deep learning-based methods, including Recurrent Neural Networks (RNNs) and Bidirectional Long Short-Term Memory (BiLSTM), began to gain attention in MT research. Nevertheless, these techniques also struggle with handling long text translations. A state-of-the-art neural model, transformers supported with the attention mechanism, has emerged as a highly effective alternative to the existing MT architecture [1].

On the other hand, an examination of the existing literature on MT of the Turkish language revealed a research gap in translations from Turkish to Russian. Translating Turkish to Russian presents significant challenges due to cultural and linguistic differences [2]. Another challenge arises from the structural differences between Turkish and Russian. Turkish is an agglutinative language that employs a Subject-

Object-Verb (SVO) sentence structure, whereas Russian is a fusional language with SVO structure. These differences can complicate the translation process, especially when dealing with complex sentences [3]. These linguistic disparities increase the complexity of capturing accurate translations. Furthermore, the complete absence of Turkish-Russian text pair corpora represents another critical challenge.

Although there are translation studies in the literature addressing Turkish to different languages are generally focused on English as the primary target language (TL) within the broader context of MT research. In contrast, there is a no studies have been found in research addressing translations from Turkish to Russian, which this study aims to address. This study presents a new approach to Turkish to Russian MT by developing a transformer-based neural network model that utilizes attention mechanisms to overcome the linguistic and structural challenges unique to this language pair.

Using the Transformer in our MT model with the Aurora Dataset, I achieved a Bilingual Evaluation Understudy (BLEU) score of 0.4903, a Word Error Rate (WER) of 0.2915, and a Character Error Rate (CER) of 0.3935. These high results underscore the effectiveness of our model. The findings demonstrate that the self-attention mechanism in transformers is highly effective at capturing contextual information, leading to significant improvements in translation quality compared to existing methodologies.

The following is a summary of the contributions of this study:

1. A transformer-based Turkish-to-Russian MT model with high-quality translation capabilities has been developed
2. Deep insights into the encoder-decoder structures of the transformers used for MT from Turkish to Russian, including their configurations and the hyperparameters utilized during training, are provided
3. Creation of a 476K Turkish-to-Russian parallel corpus.

This thesis is structured as follows: Part 2 provides a related studies regarding MT for Turkish. In Part 3, I provide an overview of MT by outlining its historical

development, approaches to MT and challenges of MT systems. Part 4 presents the core components of the Transformer architecture and highlights its applications in MT systems. Part 5 describes the methodologies of proposed model. Part 6 presents the details regarding the hyperparameter tuning, datasets, evaluation metrics and score results. Part 7 provides a conclusion of the study's results, along with suggestions for future work.

PART 2

LITERATURE REVIEW

The Turkish language is a member of the Turkic language family, which also includes Kyrgyz, Kazakh, Crimean Tatar, and Tatar, shares structural and linguistic features such as agglutinative grammar, vowel harmony, and shared vocabulary [4]. Bayatli et al. built a rule-based MT system for Kazakh to Turkish using the Apertium platform, which performed comparably to Google Translate despite higher out-of-vocabulary rates which underscores the potential of rule-based approaches in shallow-transfer translation scenarios [5]. Gökırmak et al. created the first publicly available MT system for Crimean Tatar to Turkish using Apertium. Their rule-based system outperformed neural methods in low-resource contexts, reinforcing the utility of such approaches for related language pairs [6].

The initial methodology utilized in MT primarily relies on manually developed translation rules and linguistic knowledge. Nevertheless, the intrinsic complexity of natural languages renders it difficult to account for all linguistic irregularities through the establishment of manual translation rules. With the advent of large-scale parallel corpora, an alternative approach, Statistical Machine Translation (SMT), which is based on the direct learning of underlying structural elements such as word alignments and phrase sequences from parallel corpora, has been proposed [7-8]. Ferhan Türe's hybrid system combining transfer-based and statistical methodologies for Turkish-English translation achieved competitive results, showcasing the value of integrating multiple techniques to address structural and morphological complexities [9]. Similarly, Ataman et al. proposed vocabulary reduction strategies tailored to morphologically rich languages like Turkish, achieving significant BLEU score improvements [10]. Such strategies could inform the development of MT systems for Turkish-Russian translations, where similar morphological challenges exist. Yıldırım and Tantug explored an SMT-based method, re-ranking N-best lists in English-to-

Turkish MT using Google Translate Research API, achieving an 11.81% relative improvement in BLEU scores by reordering candidate translations [11]. Similarly, El-Kahlout and Oflazer improved English-to-Turkish SMT by using morphological information and local word reordering, achieving a 62% relative improvement in BLEU scores with a score of 25.17 [12]. Yeniterzi and Oflazer put forth a methodology for translating from English to Turkish that incorporates syntactic analysis at the source text level. This approach yielded a 39% relative improvement in BLEU scores compared to a word-based baseline [13]. Yılmaz and Uguz developed a Turkish-English MT system using Moses and the BOUN-ITU Turkish-English Parallel Corpus, achieving a BLEU score of 17.02 and highlighting the benefits of shuffling corpus data and using a 4-gram language model [14]. Oflazer et al. improved English-to-Turkish SMT by representing sub-lexical units, achieving a BLEU score of 25.08 using Moses, SRILM, and GIZA++ on a corpus from international relations and legal documents [15]. The findings of these studies have led to the conclusion that SMT is unable to model dependencies between words over long distances, which results in a suboptimal translation quality.

The deep learning paradigm has led to the emergence of Neural Machine Translation (NMT) which is replacing SMT as the dominant approach to MT [16-19]. The system offers a more straightforward end-to-end training process and showcases state-of-the-art performance on a multitude of language pairs [20]. The initial NMT architectures, including RCTM [17], RNNEncDec [19], and Seq2Seq [16], employ a fixed-length approach, wherein the size of the source representation is fixed, irrespective of the length of the source sentences. These models typically employ RNNs as the decoder network for generating variable-length translations. However, it has been demonstrated that the efficacy of this methodology declines as the length of the input sentence increases [19]. Gülçehre et al. improved NMT by integrating language models trained on monolingual data using shallow and deep fusion, achieving up to 2 BLEU points improvement for low-resource Turkish-English and Chinese-English and 0.39-0.47 BLEU points for high-resource German-English and Czech-English [21]. Firat et al. developed a multi-way, multilingual NMT system that efficiently translates multiple language pairs, achieving better quality than conventional SMT for

low-resource Turkish-English and Uzbek-English translations, with notable improvement for Uzbek-English due to linguistic similarities [22].

Tukeyev et al. introduced a novel morphological segmentation method, which utilises a Complete Set of Endings to improve NMT for Turkic languages. Their method reduced vocabulary size significantly and improved BLEU scores, demonstrating the effectiveness of linguistically informed approaches [23]. The limitations of classical NMT prompted the introduction of the attention mechanism by Bahdanau et al. [19], which signifies a substantial leap forward in the field of RNN architectures. This mechanism enables the implementation of variable-length representations, a crucial aspect in the development of effective translation models. The attention mechanism guarantees that the distances between any given source and target words remain constant. As a result, the attention mechanism has facilitated the optimization process. The transformer employ multi-layered neural networks and rely entirely on self-attention networks which include encoder-decoder networks. Yirmibeşoğlu and Güngör improved Turkish-English translation using data augmentation and morphologically motivated input variations, achieving a BLEU score of 26.38 with a hybrid BiDeep and Transformer model trained on the SETimes parallel corpora and augmented monolingual data [24].

When considering Turkish-to-Russian translation, the linguistic and cultural differences between the two languages pose unique challenges. Russian's inflectional grammar and Turkish's agglutinative structure demand sophisticated MT strategies. Abduvalieva examined these linguistic divergences in simultaneous interpretation, emphasizing the importance of context-sensitive methods and advanced linguistic proficiency to bridge structural and cultural gaps effectively [25]. Broader initiatives like the Turkic Interlingua project, as discussed by Mirzakhalov, highlight the potential of large-scale MT systems for Turkic languages. The project trained bilingual MT models across 22 Turkic languages, addressing low-resource challenges through participatory research and expansive datasets [26]. While the focus was not explicitly on Turkish-Russian translation, such projects lay a foundation for future advancements in Turkic-Russian language pair translation systems.

While specific MT systems for Turkish-to-Russian translation remain underexplored, the insights gained from related Turkic language studies and advances in morphological and cultural adaptation highlight the potential for future developments. Efforts to enhance MT for Turkish and Turkic languages, including projects addressing agglutinative grammar and context-sensitive translation, provide a robust framework for tackling Turkish-Russian translation challenges.

In view of the findings of this literature review, it is apparent that there is a notable absence of academic research dedicated to the field of Turkish-to-Russian translation, and there is no Transformer-based MT study to date. This gap is largely attributed to the limited availability of parallel corpora and linguistic resources for these languages, compounded by their complex morphological structures and syntactic differences. By undertaking this translation study from Turkish to Russian, this research aims to address a critical gap in the field of MT, advancing understanding and improving methodologies for this challenging language pair.

PART 3

MACHINE TRANSLATION

3.1. OVERVIEW OF MT

MT refers to the process of translating text from a Source Language (SL) to a TL using computer-based systems, either independently or with human assistance [27, 28].

With a history spanning 65 years, this field represents one of the earliest applications of computers. During this period MT has been a subject of study for psychologists, philosophers, linguists, engineers and computer scientists. It is fair to say that the initial advancements in MT played a crucial role in shaping fields like computational linguistics, artificial intelligence, and practical NLP [29].

3.2. HISTORY OF MT

The concept of employing computers for translation emerged around 1945, marking the beginning of early research efforts in MT. During the 1950s, the United States government sought to develop automated methods to translate Russian texts into English, primarily to decode Russian communications during the Cold War between the United States and the USSR. Multiple projects received funding up until the mid-1960s, but the outcomes were largely disappointing. Both scientists and the government anticipated the rapid development of a functional translation system, but research revealed that the complexities of language and translation posed greater challenges than initially anticipated [30]. The Automatic Language Processing Advisory Committee issued a report in 1966, highlighting that automatic translation systems were less efficient and more costly compared to human translators. The report determined that there was no necessity for continued research in MT, suggesting that such systems were only useful as tools to aid human translators. Following the release

of the Automatic Language Processing Advisory Committee report, the majority of financial support for MT research was withdrawn [31].

Beginning in the 1970s, research in MT accelerated across various countries, driven by diverse motivations. In Canada, efforts were focused on addressing challenges stemming from the country's multilingual framework. In 1976, a system named Meteo, designed to translate weather reports between English and French in Montreal, was successfully demonstrated [32]. In Europe, the Commission of European Communities developed an English-to-French MT system built upon the earlier Systran project. This initiative was later expanded to create systems for additional language pairs, including English-Italian and English-German [31]. Another project, designed to create a multilingual system encompassing all European languages, was initiated in the late 1970s [33]. In Japan, after overcoming the challenge of processing Chinese characters in 1980, numerous scientists began engaging in MT research. Examples of Japanese systems from this period include the TITRAN translation system, the MU project at Kyoto University [34], and a project at the University of Osaka Prefecture [31].

In the early 1990s, with the expansion of the Internet, extensive bilingual corpora became publicly available. A bilingual corpus consists of aligned sentences, where each sentence in the SL corresponds to a sentence in the TL. This development encouraged researchers to utilize statistical methods on bilingual corpora to automatically construct a model of the translation process. In SMT from a SL F to a TL E, the goal is to determine the most probable translation of a sentence f in F. This involves creating a language model for the TL to represent the likelihood of a sentence being naturally expressed in that language, and developing a statistical translation model to represent the probability of a sentence in the TL being translated back into f. The most successful SMT systems have been described by Koehn et al. [35], Brown et al. [36], and Chiang [37].

3.3. CHALLENGES IN MT

Translating between languages requires considering the vocabulary, morphological characteristics, and grammatical structures of both the SL and TL individually.

Additionally, the morphological, syntactic, and semantic differences arising from these aspects must be addressed with care.

Different morphological properties pose significant challenges in MT. In agglutinative languages, words can consist of multiple morphemes distinctly separated by boundaries. In contrast, inflectional languages like Russian often have morphemes that represent multiple morphological features, leading to ambiguity. Isolating languages, such as Vietnamese, have a one-to-one correspondence between words and morphemes, whereas polysynthetic languages pack numerous morphemes into a single word, which can correspond to an entire sentence in languages like English [38].

Beyond morphological differences, syntactic differences present another significant challenge in MT, with word order being one of the most common issues. Many major languages, including English, Spanish, German, French, Italian, and Mandarin, follow an SVO structure, where the verb typically appears immediately after the subject. In contrast, languages like Japanese and Turkish follow an Subject-Object-Verb structure, while languages such as Arabic, Hebrew, and Irish use a Verb-Subject-Object order. Word order plays a crucial role in determining the syntactic structure of a language [38].

3.4. ARCHITECTURES OF MT SYSTEMS

The widely recognized Vauquois triangle (Figure 3.1), illustrates the relationship between the three primary stages of traditional MT: analysis, transfer, and generation. Initially, the source sentence undergoes analysis to produce an intermediate representation (Analysis). This representation is then converted into the TL (Transfer) and subsequently used to generate a target sentence (Generation). The overarching concept is to represent a sentence in the source SL in a manner that facilitates its transfer and regeneration into a sentence in the TL. In practical MT systems, however, some of these steps may be omitted or the approach may emphasise specific steps over others.

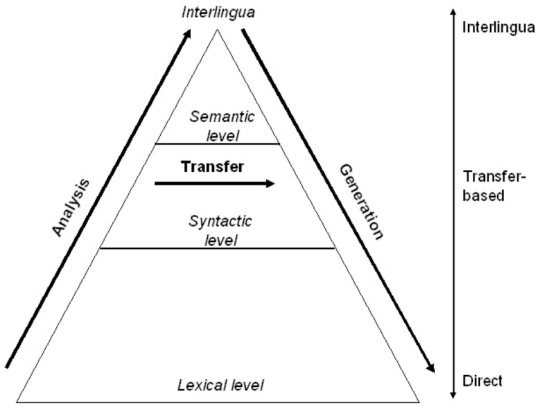

Figure 3.1. Vauquois triangle [39].

Throughout the history of MT, researchers have adopted various strategies at different times. These choices not only highlight the linguistic depth and diversity of the field but also reflect the scale of ambition involved.

3.4.1. Linguistic Architecture

The linguistic architecture employs three fundamental approaches for developing MT systems, each varying in complexity and sophistication [40]. These approaches are direct, transfer based and interlingua.

3.4.1.1. Direct Approach

In direct translation (Figure 3.2), the process involves a straightforward conversion from the source text to the target text. The vocabulary of the SL text is analyzed to resolve ambiguities, accurately identify TL expressions, and establish the correct word order in the TL. This method entails taking a sequence of words from the SL, stripping away morphological inflections to obtain base forms, and referencing them in a bilingual dictionary between the SL and TL. Key components of this system include a comprehensive bilingual dictionary and a program for lexical and morphological analysis and text generation [41].

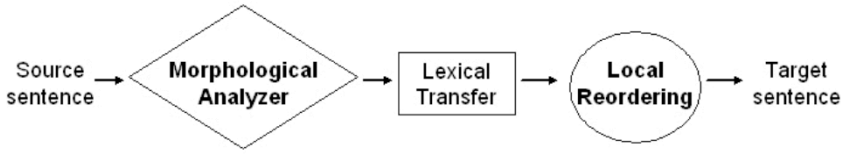

Figure 3.2. Direct approach [39].

3.4.1.2. Transfer Based Approach

The Transfer method in translation operates in three main phases (Figure 3.3). Initially, SL texts are transformed into an intermediate representation, often in the form of parse trees. Next, these intermediary representations are restructured into comparable forms suitable for the TL. Finally, the translation is completed during the generation phase, where the target text is produced [41]. This method involves analyzing the source text to create an abstract representation that retains many features specific to the SL but excludes those of the TL. These representations can range from primarily syntactic to heavily semantic. For syntactic transfer, the source's parse tree undergoes transformations into a TL tree through predefined tree manipulations, often guided by associated feature structures. Regardless of the chosen representation, the transfer process relies on rules to map SL structures to their TL equivalents. During the generation phase, the target structure is adjusted to comply with the linguistic rules and constraints of the TL, culminating in the final translated output.

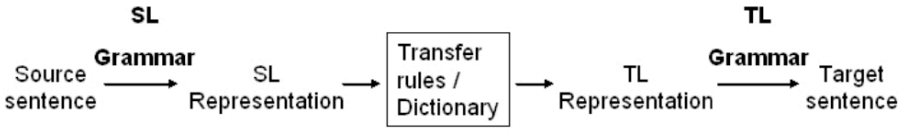

Figure 3.3. Transfer-based approach [39].

3.4.1.3. Interlingua Approach

The Interlingua approach is particularly advantageous for multilingual translation systems and involves two primary phases (Figure 3.4):

1. Analysis which converting the SL into the Interlingua
2. Generation which transforming the Interlingua into the TL.

In the analysis phase, a SL sentence is processed to extract its semantic meaning, which is then encoded in an Interlingua format. The Interlingua serves as a neutral intermediary language, specifically designed to represent the source content without being tied to any particular source or TL.

Once the analysis is complete, the generation phase uses the Interlingua representation to create the TL output. Because the analysis component is tailored to the SL but not dependent on any specific TL, it can be reused across multiple TLs. Similarly, the generation component for a particular TL operates independently of any specific SL, allowing it to be employed for translations from various SLs into that TL [41].

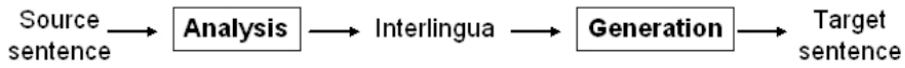

Figure 3.4. Interlingua-based approach [39].

3.4.2. Computational Architecture

The computational architecture employs three fundamental approaches for emphasize computational frameworks and methodologies to achieve efficient and accurate translation [40]. These approaches are rule based, corpus based and hybrid.

3.4.2.1. Rule Based Approach

Rule-based MT (RBMT) systems use linguistic rules to facilitate translation. This is called RBMT because it uses different types of linguistic rules, such as those for morphology, lexical transfer, syntactic generation and syntactic analysis.

The translation process involves the following steps:

1. Conducting syntactic, morphological and semantic analysis of the input text
2. The output text is produced through structural transformations based on internal representations.

The steps outlined above rely on the use of a dictionary and grammar, both of which must be created by linguists. This requirement is the primary challenge of RBMT, as the process of gathering and explicitly defining this knowledge is time-intensive and is often referred to as the knowledge acquisition problem. Developing and maintaining the rules in this type of system is not only highly challenging, but it also does not guarantee that the system will perform as effectively after the addition of new rules. RBMT systems are extensive rule-based systems that involve significant computational costs since they must implement all aspects – such as syntactic, semantic, and structural transfer – through predefined rules [42].

3.4.2.2. Corpus Based Approach

Corpus-Based MT (CBMT) is an alternative approach for MT to overcome the knowledge acquisition problem of RBMT. CBMT automatically acquires translation knowledge from bilingual corpora. [43-46].

3.4.2.3. Hybrid Approach

A hybrid approach to MT combines the strengths of syntactic and morphological analysis with statistical methods to address ambiguities that arise during the translation process (Figure 3.5). Knight et al. discuss a hybrid MT framework that generates an ambiguous semantic representation of the source sentence, which is subsequently refined using a TL model [47]. The "generation-heavy" MT approach described by Habash [48] and Ayan et al. [49] involves creating multiple translation hypotheses using rule-based methods and then applying statistical techniques to determine the most probable translation. Furthermore, statistical methodologies can also be

employed to derive transfer rules that facilitate the transformation of syntactic structures between the source and TLs [50].

Figure 3.5. Hybrid approach [39].

3.4.3. SMT

SMT is a type of MT that uses statistical methods to identify the most likely translation of a given sentence (Figure 3.6) . Specifically, SMT models the translation process as a "noisy channel": a sentence *e* is passed through a "noisy channel" and is transformed into f. The goal is to determine *e* such that the probability of *e* being the translation of the observed output *f* is maximized.

$$e^* = \arg max_e P(e|f) \tag{3.1}$$

Instead of attempting to accurately model this probability with a joint distribution, the problem is broken down using Bayes' rule.

$$e^* = \arg \frac{max_e P(f|e) P(e)}{P(f)} = \arg max_e P(f|e) P(e) \tag{3.2}$$

The denominator *P(f)* can be disregarded as it remains constant for each *e*. Notably, Equation 3.2 provides a clearer representation of the translation process compared to Equation 3.1 by dividing it into two distinct components. In Equation 3.1, the model for P(e|f) must account for both how likely *f* is translated into eee and how well-formed the English string *e* is. Conversely, in Equation 3.2, the model for P*(f|e)* focuses solely on the probability that eee is a translation of *f*, regardless of the quality of the French string *f*. Meanwhile, the model for P(e) addresses the likelihood of *e* being a valid English string, independent of the translation process. The first model is referred to as the translation model, while the second is called the language model [51]. The argmax operator represents the search for the English string eee that maximizes the specified

probability. This search process, known as "decoding," has been proven to be NP-hard by Knight [52].

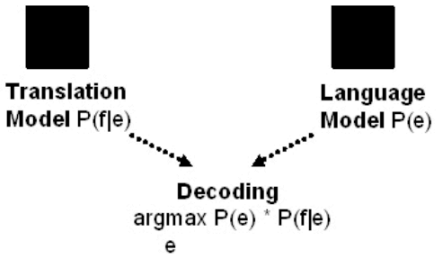

Figure 3.6. SMT approach [39].

3.4.4. Neural Based Approaches To MT

NMT marks a major breakthrough compared to traditional MT methods, utilizing deep learning to handle translation tasks in an end-to-end manner. Unlike conventional systems that rely on manually crafted rules or statistical techniques, NMT employs neural networks to model the translation process, resulting in enhanced fluency and better context management [53].

3.4.4.1. End-To-End Learning

NMT systems are trained in an end-to-end fashion, allowing the model to learn the complete translation process directly from the data without requiring distinct components. This integrated approach enables the model to identify complex patterns and dependencies within the data [54].

3.4.4.2. Encoder-Decoder Architecture

The encoder-decoder framework serves as the cornerstone of NMT. In this framework, the encoder converts the source sentence into a continuous representation, while the decoder generates the target sentence based on this representation. This architecture effectively manages variable-length input and output sequences. Sutskever et al. introduced this approach, showcasing its effectiveness in MT tasks [16].

3.4.4.3. Attention Mechanisms

Attention mechanisms allow the model to focus on specific portions of the source sentence during translation, enhancing its ability to handle long sentences and capture relevant context. The groundbreaking work by Bahdanau et al. introduced the attention mechanism, leading to significant improvements in translation performance [19].

3.4.4.4. Transformer Architecture

Vaswani et al. introduced the transformer model, which replaced recurrent architectures with self-attention mechanisms [1]. This innovation enabled greater parallelization and significantly enhanced performance, establishing the architecture as the cornerstone of many state-of-the-art NMT systems.

PART 4

TRANSFORMER

4.1. TRANSFORMER ARCHITECTURE

The majority of advanced neural sequence transduction models are based on an encoder-decoder framework [16, 18, 19]. In this structure, the encoder transforms an input sequence of symbol representations (x1,...,xn) into a sequence of continuous representations z = (z1,...,zn). The decoder uses z to produce an output sequence (y1, ..., ym) symbol by symbol. The process is auto-regressive [55], where each step incorporates the symbols generated so far as additional input to predict the next symbol.

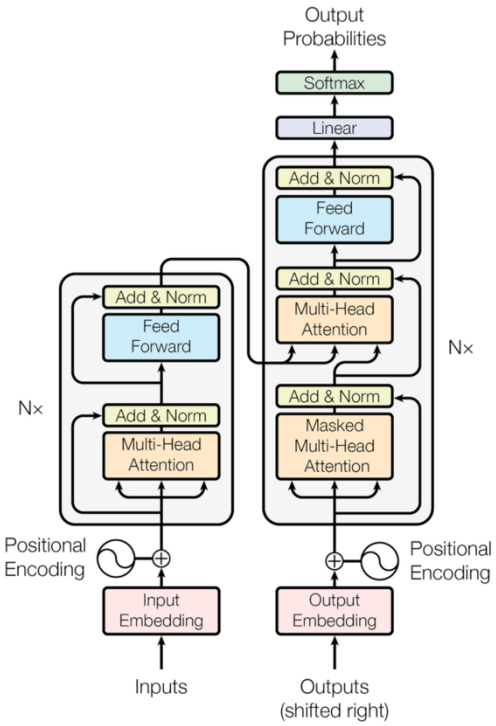

Figure 4.1. The Transformer model architecture [1].

The Transformer adopts this general architecture, utilizing stacked self-attention mechanisms and point-wise fully connected layers in both the encoder and decoder, as illustrated in the left and right sections of Figure 4.1, respectively.

4.1.1. Encoder

The encoder consists of six layers, each containing two distinct sub-layers. The 1st sub-layer employs a multi-head self-attention mechanism, while the 2nd consists of a fully connected feed-forward network. Residual connections are incorporated around each sub-layer, and these are followed by layer normalization [56, 57]. The output of a sub-layer is calculated using the formula: LayerNorm(x + Sublayer(x)), where Sublayer(x) represents the operation carried out by the respective sub-layer. Both the sub-layers and the embedding layers generate outputs with a consistent dimensionality of d_model = 512.

4.1.2. Decoder

The decoder is consist of N identical layers. Each layer has two sub-layers like the encoder, plus a third that does multi-head attention on the encoder's output. Like the encoder, each sub-layer in the decoder has residual connections and layer normalisation. The self-attention sub-layer is modified to restrict positions from attending to subsequent positions. This ensures that predictions at position i depend only on the known outputs from positions earlier than i [1].

4.1.3. Attention

An attention mechanism takes a query along with a set of key-value pairs and maps them to an output, with all elements represented as vectors. The output is derived by computing a weighted sum of the values, where the weights are assigned based on a compatibility function that evaluates how closely the query aligns with each corresponding key.

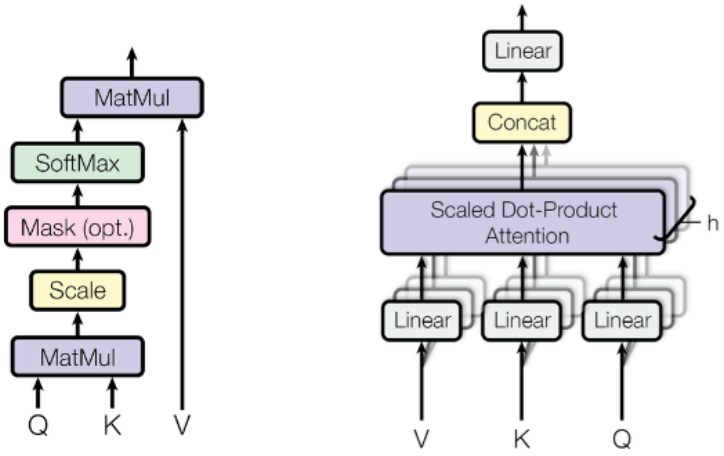

Figure 4.2. Multi-Head Attention consists of several attention layers running in parallel (right) Scaled Dot-Product Attention (left) [1].

4.1.3.1. Scaled Dot-Product Attention

The input of Scaled Dot-Product Attention includes queries and keys with a dimension of d, and values with a dimension of d_k (Figure 4.2). To compute the attention, the dot product of the query with each key is calculated, the result is scaled by dividing by$\sqrt{d_k}$, and the softmax function is applied to derive the weights for the values.

In practice, the attention function is computed for a set of queries simultaneously, organized into a matrix Q. Similarly, the keys and values are combined into matrices K and V . The resulting matrix of outputs is calculated as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (4.1)$$

The two most widely used attention mechanisms are additive attention [19] and dot-product (multiplicative) attention. Dot-product attention aligns with our approach, except for the inclusion of a scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention calculates the compatibility function through a feed-forward network with a single hidden layer. While both methods have similar theoretical complexity, dot-product attention is significantly faster and more memory-efficient in practice due to its implementation with highly optimized matrix multiplication code.

For small values of d_k, both mechanisms perform comparably; however, additive attention outperforms unscaled dot-product attention when d_k is larger [18]. The hypothesis suggests that with large d_k, the magnitude of the dot products becomes excessively high, causing the softmax function to operate in regions with extremely small gradients. To mitigate this issue, the dot products are scaled by $\frac{1}{\sqrt{d_k}}$.

4.1.3.2. Multi-Head Attention

Instead of utilizing a single attention function with d_{model}- dimensional keys, values, and queries, it is more effective to apply h distinct learned linear projections. These projections independently map the queries, keys, and values into dimensions d_k, d_k and d_v, respectively. The attention function is then computed in parallel on these projected versions, producing d_v -dimensional output values. These outputs are concatenated and projected again to generate the final values, as illustrated in Figure 4.2.

Multi-head attention enables the model to simultaneously focus on information from various representation subspaces at different positions. In contrast, using a single attention head can limit this capability, as averaging tends to obscure finer details [1].

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_\text{h})W^O \tag{4.2}$$

where $head_1 = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

4.1.4. Position-Wise Feed-Forward Networks

Alongside the attention sub-layers, each layer in the encoder and decoder includes a fully connected feed-forward network, which is applied independently and identically to each position. This network comprises two linear transformations with a ReLU activation function in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (4.3)$$

Although the linear transformations are consistent across different positions, they utilize distinct parameters for each layer [1].

4.1.5. Embeddings and Softmax

Similar to other sequence transduction models, learned embeddings are employed to convert input and output tokens into vectors with a dimension of d_{model}. Similarly, a conventional learned linear transformation combined with a softmax function is used to transform the decoder's output into probabilities for predicting the next token. In this model, the weight matrix is shared across the two embedding layers and the pre-softmax linear transformation, adopting the methodology described in [58]. Additionally, in the embedding layers, these weights are scaled by $\sqrt{d_{model}}$.

4.1.6. Positional Encoding

As the model does not incorporate recurrence or convolution, it relies on an alternative method to capture the sequence's order. To achieve this, positional encodings are integrated into the input embeddings at the foundation of both the encoder and decoder stacks. These encodings share the same dimensionality, d_{model}, as the embeddings, enabling them to be combined through summation. Positional encodings can be implemented in various ways, either as learned or fixed representations [59].

Used sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right) \quad (4.4)$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right) \quad (4.5)$$

Here *pos* represents the position and *i* is the dimension. denotes the dimension. Each dimension of the positional encoding is defined by a sinusoidal function, with wavelengths following a geometric progression from 2π to 10000 · 2π. This design was chosen based on the hypothesis that it would enable the model to effectively learn

to attend to relative positions. Specifically, for any fixed offset k, PE_{pos+k}pos+k can be expressed as a linear function of PE_{pos} [1].

4.2. TRANSFORMER IN MT

Transformers have transformed MT with their encoder-decoder architecture. In this framework, the encoder processes the input sentence in the SL, while the decoder generates the corresponding output in the TL. A key component of this approach is the attention mechanism, which incorporates self-attention to capture contextual relationships within a sequence and cross-attention to align the source and TLs efficiently. Unlike traditional models like RNNs, Transformers handle all tokens simultaneously, enabling faster and more precise translations [1].

In NMT, Transformers have become the cornerstone of modern systems, surpassing traditional methods such as phrase-based and RNN-based models to achieve state-of-the-art performance. Pre-trained models like BERT, GPT, and T5 are commonly fine-tuned for translation tasks, while multilingual models such as mBART and M2M-100 facilitate translation across numerous language pairs. These advanced models also enable zero-shot translation, allowing them to generalize to unseen language pairs by leveraging powerful representation learning techniques [60, 61].

Transformers offer significant advantages in MT, including their capability to manage long sequences by capturing global dependencies, delivering rich contextual representations through self-attention, and supporting efficient training through parallelization. These features have resulted in translation quality that approaches human-level performance for many language pairs [1, 62].

Transformers drive real-world applications like Google Translate, which utilizes Transformer-based models to deliver high-quality translations, and DeepL, renowned for its nuanced and accurate results. Additionally, platforms such as Amazon Translate and open-source projects like Helsinki-NLP's OPUS-MT showcase the broad adoption and effectiveness of Transformers in MT [63, 64].

PART 5

METHODOLOGY

I used PyTorch to build all the necessary structures and blocks of Transformer. For encoder and decoder blocks I defined a similar value N=6 layers like Vaswani et al. [1]. Hugging Face libraries used for dataset handling and text preprocessing. Word-level tokenization was used, with special tokens for sequence processing. The model was built and trained using Google Colaboratory. The dataset organized for training and validation using dataloaders and attention masks.

5.1. USED LIBRARIES

I used several key libraries to streamline development and enhance performance. PyTorch served as the core deep learning framework, providing tools for tensor operations, model training, while also enabling GPU acceleration via CUDA. The Hugging Face libraries were integral for handling datasets and text preprocessing; dataset.load_dataset function used to load our custom dataset and leveraged the tokenizers module for efficient and customizable text tokenization. Additionally, the standard Python math library was employed for essential mathematical operations, ensuring precision and simplicity in basic computations.

5.1.1. PyTorch

PyTorch is a popular deep learning framework providing tools for building and training machine learning models. Supports computation on GPUs with minimal code changes, leveraging CUDA for high performance.

5.1.1.1. Torch

The core library of PyTorch used for tensor operations, model definition, and training.

5.1.1.2. Torch.nn

A submodule that contains building blocks for neural networks, such as layers, loss functions, and other utilities. It simplifies defining complex neural network architectures.

5.1.1.3. Torch.Utils.Data

Provides utilities for working with data in PyTorch. I used random_split utility function to split datasets into training and validation sets.

5.1.2. Math

A standard Python library for mathematical operations. This library is often used for common mathematical calculations such as exponentials, logarithms, trigonometric functions, etc. In our project used for take the square root.

5.1.3. Hugging Face Libraries

5.1.3.1. Datasets.load_dataset

A function to load datasets from the Hugging Face dataset repository or custom data. I used this function for load our custom dataset.

5.1.3.2. Tokenizers.Tokenizer

A fast and customizable tokenization library used to prepare text data for models. It's designed for high performance, especially for large datasets.

5.1.3.3. Tokenizers.models.WordLevel

Specifies the tokenization model to use. The WordLevel model tokenizes text at the word level.

5.2. GOOGLE COLABORATORY

Google Colaboratory, often referred to as Google Colab, is a free cloud-based platform that allows users to write and execute Python code directly in a web browser. It provides an interactive environment ideal for tasks like data analysis, machine learning, and education, eliminating the need for local setup. Colab hosts a Jupyter Notebook service, enabling users to create and share documents that combine live code, equations, visualizations, and descriptive text [65].

Users can take advantage of free access to computing resources such as GPUs and TPUs, enabling the execution of resource-intensive tasks. This seamless integration ensures that work is saved and can be accessed from any device [66].

I used Google Colab for storing our training model directly inside the cloud drive and for training with high A100 GPU.

5.3. PREPARING TRANSFORMER FOR CODING

5.3.1. Input Embeddings

Embeddings serve as the initial step in the encoder and decoder blocks of the Transformer architecture.

```python
class InputEmbeddings(nn.Module):
    def __init__(self, d_model: int, vocab_size: int) -> None:
        super().__init__()
        self.d_model = d_model
        self.vocab_size = vocab_size
        self.embedding = nn.Embedding(vocab_size, d_model)

    def forward(self, x):
        return self.embedding(x) * math.sqrt(self.d_model)
```

Figure 5.1. InputEmbedding class code block.

The *InputEmbedding* class handles the conversion of input text into numerical vectors with dimensions equal to *d_model*. To prevent the embeddings from becoming excessively small, they are scaled by multiplying with $\sqrt{d_{model}}$. Implementation of InputEmbeddings can be seen in Figure 5.1.

5.3.2. Positional Encoding

The positional encodings share the same dimension d_{model} as the embeddings, allowing the two vectors to be summed. This enables the integration of semantic information from the word embeddings with positional information from the positional encodings.

```python
class PositionalEncoding(nn.Module):
    def __init__(self, d_model: int, seq_len: int, dropout: float) -> None:
        super().__init__()
        self.d_model = d_model
        self.seq_len = seq_len
        self.dropout = nn.Dropout(dropout)
        pe = torch.zeros(seq_len, d_model)
        position = torch.arange(0, seq_len, dtype=torch.float).unsqueeze(1) # (seq_len, 1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model)) # (d_model / 2)
        pe[:, 0::2] = torch.sin(position * div_term) # sin(position * (10000 ** (2i / d_model))
        pe[:, 1::2] = torch.cos(position * div_term) # cos(position * (10000 ** (2i / d_model))
        pe = pe.unsqueeze(0) # (1, seq_len, d_model)
        self.register_buffer('pe', pe)
```

Figure 5.2. PositionalEncoding class code block.

In the *PositionalEncoding* class, a matrix of positional encodings, pe is created with dimensions(seq_len, d_model). Initially, the matrix is filled with zeros. The sine function is applied to the even indices of the positional encoding matrix, while the cosine function is applied to the odd indices. These functions are used to enable the

model to infer the position of a word relative to other words in the sequence. Implementation of PositionalEncoding can be seen in Figure 5.2.

5.3.3. Layer Normalization

```python
class LayerNormalization(nn.Module):
    def __init__(self, features: int, eps:float=10**-6) -> None:
        super().__init__()
        self.eps = eps
        self.alpha = nn.Parameter(torch.ones(features)) # alpha is a learnable parameter
        self.bias = nn.Parameter(torch.zeros(features)) # bias is a learnable parameter
    def forward(self, x):
        mean = x.mean(dim = -1, keepdim = True) # (batch, seq_len, 1)
        std = x.std(dim = -1, keepdim = True) # (batch, seq_len, 1)
        return self.alpha * (x - mean) / (std + self.eps) + self.bias
```

Figure 5.3. LayerNormalization class code block.

The *LayerNormalization* class is responsible for performing layer normalization on input data. During the forward pass, it calculates the mean and standard deviation of the input. The input is then normalized by subtracting the mean and dividing by the standard deviation, with a small value, epsilon, added to avoid division by zero. This produces a normalized output with a mean of 0 and a standard deviation of 1.

Next, the normalized output is scaled by a learnable parameter, α\alphaα, and shifted by a learnable *bias* parameter. These parameters are adjusted during training. The resulting output is a layer-normalized tensor, ensuring consistent input scaling for layers in the network. Implementation of LayerNormalization can be seen in Figure 5.3.

5.3.4. Feed-Forward Network

```python
class FeedForwardBlock(nn.Module):
    def __init__(self, d_model: int, d_ff: int, dropout: float) -> None:
        super().__init__()
        self.linear_1 = nn.Linear(d_model, d_ff) # w1 and b1
        self.dropout = nn.Dropout(dropout)
        self.linear_2 = nn.Linear(d_ff, d_model) # w2 and b2
    def forward(self, x):
        return self.linear_2(self.dropout(torch.relu(self.linear_1(x))))
```

Figure 5.4. FeedForwardBlock class code block.

The fully connected feed-forward network applies two linear transformations, with a ReLU activation function placed between them.

In the *FeedForwardBlock* class two linear transformations, self.linear_1 and self.linear_2, are defined along with the inner-layer dimension d_ff . The input data first passes through self.linear_1, which increases its dimensionality d_model to d_ff. The output is then processed through a ReLU activation function, introducing non-linearity to enable the network to learn more complex patterns. A self.dropout layer is applied afterward to reduce overfitting. Finally, the transformed tensor is passed through self.linear_2, which reduces its dimensionality back to the original d_model dimension. Implementation of *FeedForwardBlock* can be seen in Figure 5.4.

5.3.5. Multi-Head Attention

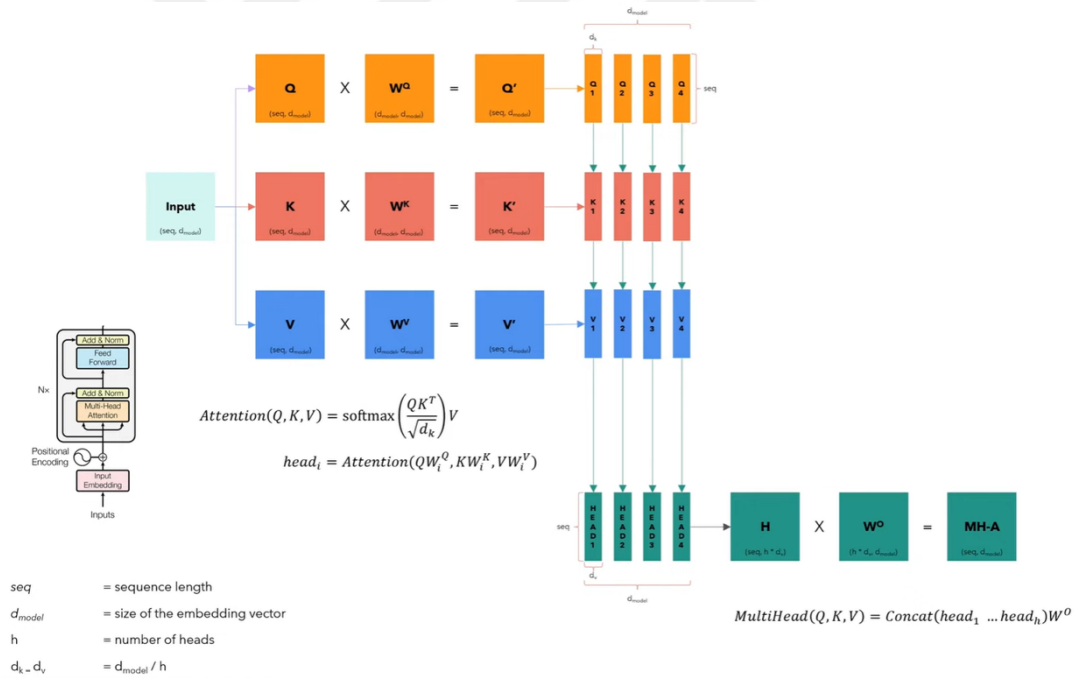

Figure 5.5. Multi-Head Attention block [67].

The Multi-Head Attention mechanism is responsible for enabling the model to capture and understand complex relationships and patterns within the data.

Figure 5.5 illustrates the working of the Multi-Head Attention mechanism. The batch dimension is omitted to focus on the process for a single sentence.

```python
class MultiHeadAttentionBlock(nn.Module):
    def __init__(self, d_model: int, h: int, dropout: float) -> None:
        super().__init__()
        self.d_model = d_model # Embedding vector size
        self.h = h # Number of heads
        assert d_model % h == 0, "d_model is not divisible by h"
        self.d_k = d_model // h # Dimension of vector seen by each head
        self.w_q = nn.Linear(d_model, d_model, bias=False) # Wq
        self.w_k = nn.Linear(d_model, d_model, bias=False) # Wk
        self.w_v = nn.Linear(d_model, d_model, bias=False) # Wv
        self.w_o = nn.Linear(d_model, d_model, bias=False) # Wo
        self.dropout = nn.Dropout(dropout)

    @staticmethod
    def attention(query, key, value, mask, dropout: nn.Dropout):
        d_k = query.shape[-1]
        attention_scores = (query @ key.transpose(-2, -1)) / math.sqrt(d_k)
        if mask is not None:
            attention_scores.masked_fill_(mask == 0, -1e9)
        attention_scores = attention_scores.softmax(dim=-1) # (batch, h, seq_len, seq_len) # Apply softmax
        if dropout is not None:
            attention_scores = dropout(attention_scores)
        return (attention_scores @ value), attention_scores

    def forward(self, q, k, v, mask):
        query = self.w_q(q) # (batch, seq_len, d_model) --> (batch, seq_len, d_model)
        key = self.w_k(k) # (batch, seq_len, d_model) --> (batch, seq_len, d_model)
        value = self.w_v(v) # (batch, seq_len, d_model) --> (batch, seq_len, d_model)
        query = query.view(query.shape[0], query.shape[1], self.h, self.d_k).transpose(1, 2)
        key = key.view(key.shape[0], key.shape[1], self.h, self.d_k).transpose(1, 2)
        value = value.view(value.shape[0], value.shape[1], self.h, self.d_k).transpose(1, 2)
        x, self.attention_scores = MultiHeadAttentionBlock.attention(query, key, value, mask, self.dropout)
        x = x.transpose(1, 2).contiguous().view(x.shape[0], -1, self.h * self.d_k)
        return self.w_o(x)
```

Figure 5.6. FeedForwardBlock class code block.

The *Multi-Head Attention block* processes input data by splitting it into queries, keys, and values, which are organized into matrices Q, K, and V. These matrices capture different aspects of the input while maintaining the same dimensions as the original data.

Each matrix is then linearly transformed using the corresponding weight matrices W^Q, W^K, and W^V, resulting in new matrices Q′, K′, and V′. These transformed matrices are further divided into smaller submatrices, corresponding to different heads *h.* This division enables the model to process information from various representation subspaces simultaneously, creating multiple sets of queries, keys, and values for each head.

Finally, all the heads are concatenated into a single matrix *H,* which is then transformed using another weight matrix W^o to generate the multi-head attention

output MH – A, a matrix that preserves the original input dimensionality. Implementation of *MultiHeadAttentionBlock* can be seen in Figure 5.6.

5.3.6. Residual Connection

```
class ResidualConnection(nn.Module):
        def __init__(self, features: int, dropout: float) -> None:
            super().__init__()
            self.dropout = nn.Dropout(dropout)
            self.norm = LayerNormalization(features)

        def forward(self, x, sublayer):
            return x + self.dropout(sublayer(self.norm(x)))
```

Figure 5.7. ResidualConnection class code block.

In the Transformer's architecture (Figure 4.1), each sub-layer, such as the self-attention and Feed Forward blocks, combines its output with its input before passing it to the Add & Norm layer. This integration of the output and the original input within the Add & Norm layer is known as a skip connection. Skip connections enable the Transformer to train deep networks more efficiently by creating a shortcut for gradients to flow during backpropagation. Implementation of *ResidualConnection* can be seen in Figure 5.7.

5.3.7. Encoder And Decoder

```
class EncoderBlock(nn.Module):
    def __init__(self, features: int, self_attention_block: MultiHeadAttentionBlock,
                 feed_forward_block: FeedForwardBlock, dropout: float) -> None:
        super().__init__()
        self.self_attention_block = self_attention_block
        self.feed_forward_block = feed_forward_block
        self.residual_connections = nn.ModuleList([ResidualConnection(features, dropout) for _ in range(2)])
    def forward(self, x, src_mask):
        x = self.residual_connections[0](x, lambda x: self.self_attention_block(x, x, x, src_mask))
        x = self.residual_connections[1](x, self.feed_forward_block)
        return x

class Encoder(nn.Module):
    def __init__(self, features: int, layers: nn.ModuleList) -> None:
        super().__init__()
        self.layers = layers
        self.norm = LayerNormalization(features)
    def forward(self, x, mask):
        for layer in self.layers:
            x = layer(x, mask)
        return self.norm(x)
```

Figure 5.8. EncoderBlock and Encoder class code blocks.

I implemented the Encoder class as a composition of multiple EncoderBlocks and included layer normalization as a final step after the input has been processed through all the blocks. Implementation of EncoderBlock and Encoder can be seen in Figure 5.8.

```
class DecoderBlock(nn.Module):
    def __init__(self, features: int, self_attention_block: MultiHeadAttentionBlock,
                 cross_attention_block: MultiHeadAttentionBlock, feed_forward_block: FeedForwardBlock, dropout: float) -> None:
        super().__init__()
        self.self_attention_block = self_attention_block
        self.cross_attention_block = cross_attention_block
        self.feed_forward_block = feed_forward_block
        self.residual_connections = nn.ModuleList([ResidualConnection(features, dropout) for _ in range(3)])
    def forward(self, x, encoder_output, src_mask, tgt_mask):
        x = self.residual_connections[0](x, lambda x: self.self_attention_block(x, x, x, tgt_mask))
        x = self.residual_connections[1](x, lambda x: self.cross_attention_block(x, encoder_output, encoder_output, src_mask))
        x = self.residual_connections[2](x, self.feed_forward_block)
        return x

class Decoder(nn.Module):
    def __init__(self, features: int, layers: nn.ModuleList) -> None:
        super().__init__()
        self.layers = layers
        self.norm = LayerNormalization(features)
    def forward(self, x, encoder_output, src_mask, tgt_mask):
        for layer in self.layers:
            x = layer(x, encoder_output, src_mask, tgt_mask)
        return self.norm(x)
```

Figure 5.9. DecoderBlock and Decoder class code blocks.

The primary distinction of the Decoder, compared to Vaswani et al. [1], is the inclusion of an additional sub-layer. This sub-layer performs multi-head attention with a *cross-attention* mechanism, using the Encoder's output as keys and values while utilizing the Decoder's input as queries.

For the Output Embedding, I reused the same InputEmbeddings class as in the Encoder. The self-attention sub-layer is *masked* to prevent the model from accessing future elements in the sequence. Implementation of DecoderBlock and Decoder can be seen in Figure 5.9.

```
class ProjectionLayer(nn.Module):
    def __init__(self, d_model, vocab_size) -> None:
        super().__init__()
        self.proj = nn.Linear(d_model, vocab_size)
    def forward(self, x) -> None:
        return self.proj(x)
```

Figure 5.10. ProjectionLayer class code block.

The ProjectionLayer class converts the model's output into a probability distribution over the vocabulary, enabling the selection of each output token from the set of possible tokens. Implementation of ProjectionLayer can be seen in Figure 5.10.

5.3.8. Building The Transformer

```
def build_transformer(src_vocab_size: int, tgt_vocab_size: int, src_seq_len: int, tgt_seq_len: int,
                      d_model: int=512, N: int=6, h: int=8, dropout: float=0.1, d_ff: int=2048) -> Transformer:
    src_embed = InputEmbeddings(d_model, src_vocab_size)
    tgt_embed = InputEmbeddings(d_model, tgt_vocab_size)

    src_pos = PositionalEncoding(d_model, src_seq_len, dropout)
    tgt_pos = PositionalEncoding(d_model, tgt_seq_len, dropout)

    encoder_blocks = []
    for _ in range(N):
        encoder_self_attention_block = MultiHeadAttentionBlock(d_model, h, dropout)
        feed_forward_block = FeedForwardBlock(d_model, d_ff, dropout)
        encoder_block = EncoderBlock(d_model, encoder_self_attention_block, feed_forward_block, dropout)
        encoder_blocks.append(encoder_block)

    decoder_blocks = []
    for _ in range(N):
        decoder_self_attention_block = MultiHeadAttentionBlock(d_model, h, dropout)
        decoder_cross_attention_block = MultiHeadAttentionBlock(d_model, h, dropout)
        feed_forward_block = FeedForwardBlock(d_model, d_ff, dropout)
        decoder_block = DecoderBlock(d_model, decoder_self_attention_block, decoder_cross_attention_block, feed_forward_block, dropout)
        decoder_blocks.append(decoder_block)
    encoder = Encoder(d_model, nn.ModuleList(encoder_blocks))
    decoder = Decoder(d_model, nn.ModuleList(decoder_blocks))
    projection_layer = ProjectionLayer(d_model, tgt_vocab_size)
    transformer = Transformer(encoder, decoder, src_embed, tgt_embed, src_pos, tgt_pos, projection_layer)
    for p in transformer.parameters():
        if p.dim() > 1:
            nn.init.xavier_uniform_(p)

    return transformer
```

Figure 5.11. Build_transformer class code block.

In the Transformer class, I integrate all components of the model's architecture, using the same parameters specified in Vaswani et al. [1]: d_{model} = 512, N = 6, h = 8, dropout rate P_{drop}= 0.1, and d_{ff} = 2048. Implementation of build_transformer can be seen in Figure 5.10.

5.4. LOADING DATASET

```
def get_all_sentences(ds, lang):
    for item in ds:
        yield item['translation'][lang]
```

Figure 5.12. Get_all_sentences class code block.

Implementation of get_all_sentences function which iterates through the dataset and extracts sentences based on the specified language pair (Figure 5.12).

```python
def get_ds(config):
    dataset_path = 'drive/MyDrive/Colab Notebooks/dataset/tr-ru.json'
    ds_raw = load_dataset('json', data_files=dataset_path, split='train')

    tokenizer_src = get_or_build_tokenizer(config, ds_raw, config['lang_src'])
    tokenizer_tgt = get_or_build_tokenizer(config, ds_raw, config['lang_tgt'])

    train_ds_size = int(0.9 * len(ds_raw))  # 90% for training
    val_ds_size = len(ds_raw) - train_ds_size  # 10% for validation
    train_ds_raw, val_ds_raw = torch.utils.data.random_split(ds_raw, [train_ds_size, val_ds_size])

    # Create tokenized datasets for training and validation
    train_ds = BilingualDataset(train_ds_raw, tokenizer_src, tokenizer_tgt, config['lang_src'], config['lang_tgt'], config['seq_len'])
    val_ds = BilingualDataset(val_ds_raw, tokenizer_src, tokenizer_tgt, config['lang_src'], config['lang_tgt'], config['seq_len'])

    max_len_src = 0
    max_len_tgt = 0
    for pair in ds_raw:
        src_ids = tokenizer_src.encode(pair['translation'][config['lang_src']]).ids
        tgt_ids = tokenizer_tgt.encode(pair['translation'][config['lang_tgt']]).ids
        max_len_src = max(max_len_src, len(src_ids))
        max_len_tgt = max(max_len_tgt, len(tgt_ids))

    train_dataloader = DataLoader(train_ds, batch_size=config['batch_size'], shuffle=True)
    val_dataloader = DataLoader(val_ds, batch_size=1, shuffle=True)

    return train_dataloader, val_dataloader, tokenizer_src, tokenizer_tgt
```

Figure 5.13. Get_ds class code block.

The *get_ds* function is designed to load and prepare the dataset for training and validation. It handles tokenizer creation or loading, dataset splitting, and DataLoader creation, enabling the model to efficiently process the dataset in batches. Implementation of *get_ds* can be seen in Figure 5.13.

```python
def causal_mask(size):
    mask = torch.triu(torch.ones((1, size, size)), diagonal=1).type(torch.int)
    return mask == 0
```

Figure 5.14. Causal_mask class code block.

The *causal_mask* function is defined to generate a mask for the decoder's attention mechanism, ensuring the model cannot access information about future elements in the sequence. Implementation of causal_mask can be seen in Figure 5.14.

5.5. TOKENIZER

Tokenization is an essential preprocessing step for the Transformer model, converting raw text into a numerical format that the model can process. Various tokenization

strategies exist; I employed word-level tokenization (Figure 5.15), which converts each word in a sentence into a corresponding token. Implementation of *get_or_build_tokenizer* can be seen in Figure 5.16.

Figure 5.15. Word level tokenization [74].

```
def get_or_build_tokenizer(config, ds, lang):
    tokenizer_path = Path(config['tokenizer_file'].format(lang))
    if not Path.exists(tokenizer_path):
        tokenizer = Tokenizer(WordLevel(unk_token="[UNK]"))
        tokenizer.pre_tokenizer = Whitespace()
        trainer = WordLevelTrainer(special_tokens=["[UNK]", "[PAD]", "[SOS]", "[EOS]"], min_frequency=2)
        tokenizer.train_from_iterator(get_all_sentences(ds, lang), trainer=trainer)
        tokenizer.save(str(tokenizer_path))
    else:
        tokenizer = Tokenizer.from_file(str(tokenizer_path))
    return tokenizer
```

Figure 5.16. Get_or_build_tokenizer class code block.

After tokenizing a sentence, each token is mapped to a unique integer ID based on the vocabulary generated from the training corpus during tokenizer training. Each integer corresponds to a specific word in the vocabulary.

In addition to the words in the training corpus, Transformers utilize special tokens for specific purposes. Here are some that I will define immediately:

[UNK]: This token is utilized to denote a word in the sequence that is unrecognized.

[PAD]: A padding token is employed to ensure uniform length among all sequences in a batch, achieved by adding this token to shorter sentences. Attention masks are

applied to "tell" the model to disregard the padded tokens during training, as they lack any true relevance to the task.

[SOS]: This token serves as an indicator for the Start of a Sentence.

[EOS]: This token is used as a marker for the End of a Sentence.

In the get_or_build_tokenizer function, I make certain that a tokenizer is prepared to train the model. It verifies whether an existing tokenizer is available; if not, it creates and trains a new one.

5.6. GRADIO

Gradio is an open-source Python library that enables developers to create interactive web interfaces for machine learning models, APIs, or any Python functions with minimal code. This facilitates the sharing and testing of models in real-world scenarios. By setting share=True in the launch() method, Gradio generates a public URL, allowing others to interact with your model directly through their browsers [73]. I used GradIO as interface for testing our translation model (Figure 5.17).

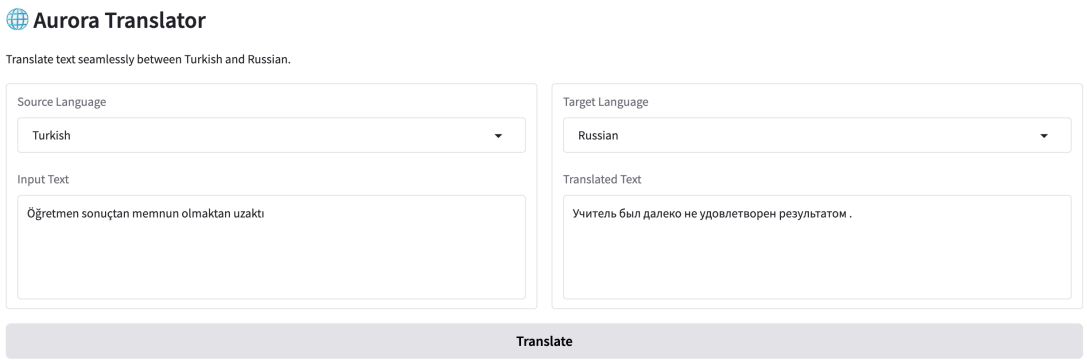

Figure 5.17. Design of our project in GradIO.

PART 6

EXPERIMENTAL STUDIES AND RESULTS

I created a Turkish-to-Russian translation dataset using the Adoxalim/Tr-En-Translator dataset from Hugging Face. The translations were manually verified, corrected for errors, and will be uploaded to Hugging Face for future research. The transformer-based model trained on this dataset used six encoder and decoder layers, eight self-attention heads, and various hyperparameters. Training on an A100 GPU for 30 epochs over 3.5 days achieved a BLEU score of 0.4903, indicating moderate-to-good translation quality, a WER of 0.2915 and a CER of 0.3935, highlighting challenges in capturing linguistic nuances between the morphologically complex languages.

6.1. MT EVALUATION

MT systems generate a vast number of translated sentences, each of which must be assessed for quality. MT evaluation involves determining whether these outputs are good or bad translations, ensuring that the assigned scores align with human judgments. Evaluating MT systems is crucial as it provides researchers with valuable feedback, serves as evidence of a system's success or failure, and allows for the comparison of different systems. Automated evaluation is particularly important because human evaluation is time-consuming and expensive. While a computer can assess thousands of sentences in seconds at no cost, human evaluation of the same text would take weeks and require significant resources. In essence, the demand for automation in translation extends to evaluation as well.

A good translation can be evaluated along two dimensions: quality and fidelity. Quality refers to a syntactically well-formed output, while fidelity ensures that the output preserves the meaning of the input [20]. This section introduces three evaluation

metrics: BLEU, WER, and TER. Each of these metrics measures the distance between the system's translation output and a reference translation, considering both quality and fidelity. Based on this distance, they assess how closely the system's translation aligns with the reference translations and assign a score accordingly.

6.1.1. BLEU Score

BLEU scores are expressed on a scale of 0 to 1. It is rare for a translation to receive a score of 1 unless it is an exact replication of the reference text. It is unlikely that even a human translator would achieve a score of 1. The number of reference translations per sentence has been found to have a significant impact on the score, with higher scores being associated with more references [68]. BLEU was developed based on the premise that the closer a MT is to a professional human translation, the better it is. The BLEU metric is designed to measure the degree of alignment between SMT output and that of human reference translations. It should be noted that translations may differ in terms of word usage, word order, and phrase length [68]. BLEU is a metric that is used to evaluate the similarity between variable-length phrases in SMT output and reference translations. The weighted match averages determine the translation score. There are various versions of BLEU, but the fundamental metric necessitates a brevity penalty (PB), which is calculated as follows:

$$P_b \begin{cases} 1, c > r \\ e(1^{-rc}), c \leq r \end{cases} \tag{6.1}$$

where r represents the length of the reference corpus, while c denotes the candidate translation length. The fundamental BLEU metric is subsequently calculated according to the following formula:

$$\text{BLEU} = P_B \, exp \sum_{n=0}^{N} w_n \log p_n \tag{6.2}$$

The n-gram precision pn is calculated using n-grams with a maximum length of N. wn are positive weights that sum to one. It should be noted that BLEU does not evaluate word and phrase position. This prevents SMT systems from increasing their scores by using words that are known with an elevated level of confidence. The number of permissible words for each candidate word is limited by the word count of the

reference translation. The geometric mean of the sentence scores is calculated for the entire corpus. [69].

6.1.2. WER

The traditional method for evaluating MT is WER [70]. It is based on the Levenshtein distance, which calculates the minimum number of deletionsi, insertions, and substitutions required to transform the system's translation into the reference translation. In this context, N denotes the total number of words in the reference translation, S represents the count of substituted words, D signifies the number of deleted words, and I refers to the count of inserted words. The WER is calculated using the following formula:

$$WER = S+I+D/N \quad (6.3)$$

6.1.3. CER

CER is a metric commonly used to assess the performance of systems such as Optical Character Recognition and Automatic Speech Recognition. It quantifies the proportion of characters in the reference text that the system predicts incorrectly. The CER is calculated using the following formula:

$$CER =D+S+I/N \quad (6.4)$$

Here, D represents the number of deletions, S indicates the count of substitutions, I refers to the number of insertions, and N is the total number of characters in the reference text. This formula determines the minimum number of single-character edits (including insertions, deletions, or substitutions) required to match the system's output with the reference text [71].

6.2. DATASET

To create a Turkish-to-Russian dataset, I used the Adoxalim/Tr-En-Translator dataset available on Hugging Face [72]. First, I separated the Turkish and English sentences. I then selected only the English sentences for translation into Russian, as translation

from English to Russian is structurally more similar. By analyzing translations from English to Russian, I translated longer sentences using Google Translator, DeepL, and Yandex Translator to identify the best results. Ultimately, I chose Google Translator based on its accuracy in meaning and punctuation. To utilize document translation in Google Translator, I converted the .txt files into .xlsx format using the Python script provided in Figure 6.1.

```python
import os
import math
import pandas as pd  # Ensure pandas is installed: pip install pandas

# File paths
source_file = 'en.txt'  # Source file with text
output_folder = './chunks-5/'  # Folder to store Excel files

# Ensure the output folder exists
os.makedirs(output_folder, exist_ok=True)

# Read the source file
with open(source_file, 'r', encoding='utf-8') as f:
    lines = [line.strip() for line in f.readlines()]

# Calculate chunk size
total_files = 5
chunk_size = math.ceil(len(lines) / total_files)  # Lines per file

# Split the lines into chunks
chunks = [lines[i:i + chunk_size] for i in range(0, len(lines), chunk_size)]

# Ensure exactly 20 files (truncate or pad with empty chunks if needed)
chunks = chunks[:total_files] + [[] for _ in range(total_files - len(chunks))]

# Save each chunk to a separate Excel file
for i, chunk in enumerate(chunks, start=1):
    # Create a DataFrame for the chunk
    df = pd.DataFrame(chunk, columns=["Text"])

    # Save to Excel
    excel_file = os.path.join(output_folder, f"chunk_{i}.xlsx")
    df.to_excel(excel_file, index=False)
    print(f"Created {excel_file} with {len(chunk)} lines.")

print("Chunking complete. Excel files saved in the 'chunks-5' folder.")
```

Figure 6.1. Script for chunking .txt file and saving in .xlsx format.

After translating each chunk_*.xlsx file by manually importing them into the Google Translator service, I obtained the translation results from English to Russian in .xlsx format. All the translation files were merged and formatted into .txt files, then mapped with the corresponding Turkish sentences. For some translations, I identified mistakes and corrected them manually. For example, the English word "Who?" was incorrectly

translated into Russian as "ВОЗ?", which refers to the World Health Organization. As a contribution to this research study, I will upload our created AuroraDataset to the Hugging Face platform to support future MT research tasks.

6.3. HYPERPARAMETERS

I trained our transformer model with six layers each in the encoder and decoder, utilizing eight self-attention heads. The model was configured with a sequence length of 350, an embedding and hidden dimension of 512, and a feedforward network size of 2048. Training was conducted using a batch size of 8, a learning rate of 0.1, and over 30 epochs. The hyperparameters is presented in the Table 6.1.

Table 6.1. The Hyperparameters Used In Training Our MT Model.

Name	Value
Encoder & Decoder	6
Number of Epochs	30
Multi Head Attention	8
Batch Size	8
Sequence Length	350
Learning Rate	0.1
Feedforward Dimension	2048
Embedding and Hidden Size	512

6.4. RESULTS

During the evaluation of our Turkish-to-Russian MT model, the results demonstrated the effectiveness of the transformer-based architecture. The model was trained for 30 epochs on the Google Colab platform with an A100 GPU. Training process took 3.5 days. We can see the validation results of the BLEU, WER and CER metric in Figure 6.2. The model achieved an average BLEU score of 0.4903, reflecting a moderate to good quality of translation outputs, with substantial alignment to the reference translations. Additionally, the WER averaged at 0.2915, indicating that approximately 29% of words in the model's translations differ from the ground truth, showcasing a reasonably accurate performance. On a more granular level, the CER was 0.3935, suggesting that the model encounters challenges in handling morphological and

orthographic nuances. These results highlight the model's ability to effectively translate between two morphologically complex languages, while also pointing to areas for improvement in capturing fine-grained linguistic details.

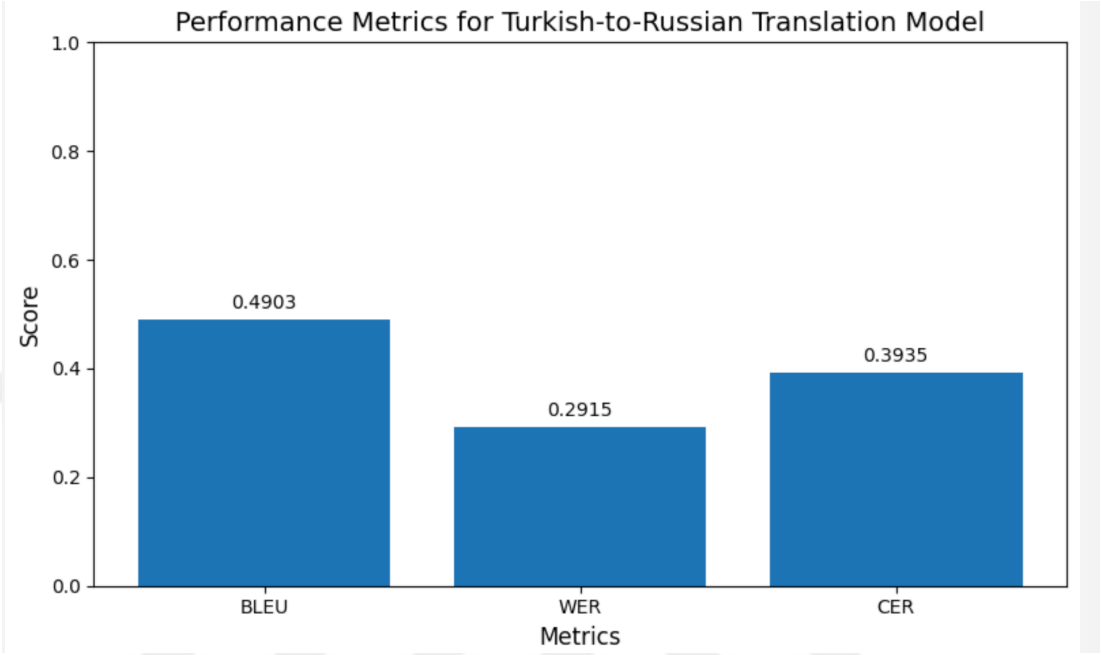

Figure 6.2. The validation results of the BLEU, WER and CER metric after 30 epochs training.

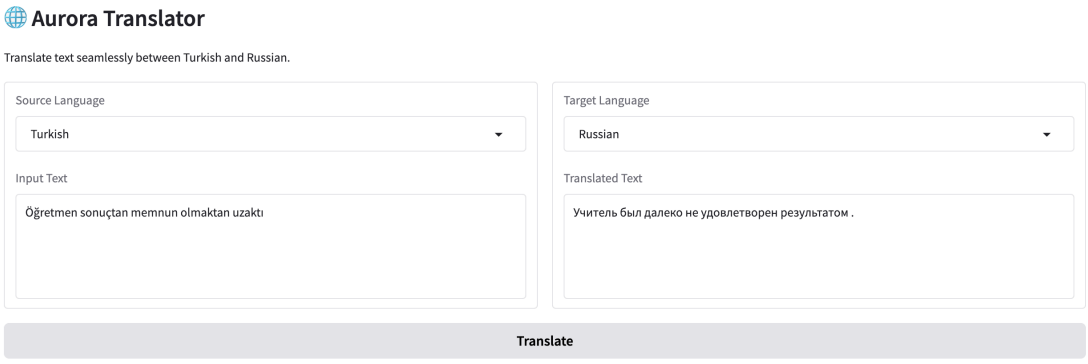

Figure 6.3. Design of our project in GradIO interface with translation example.

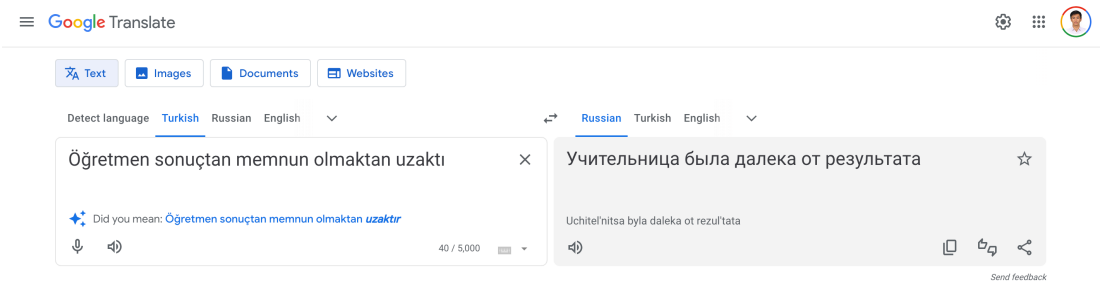

Figure 6.4. Testing example text in Google Translator.

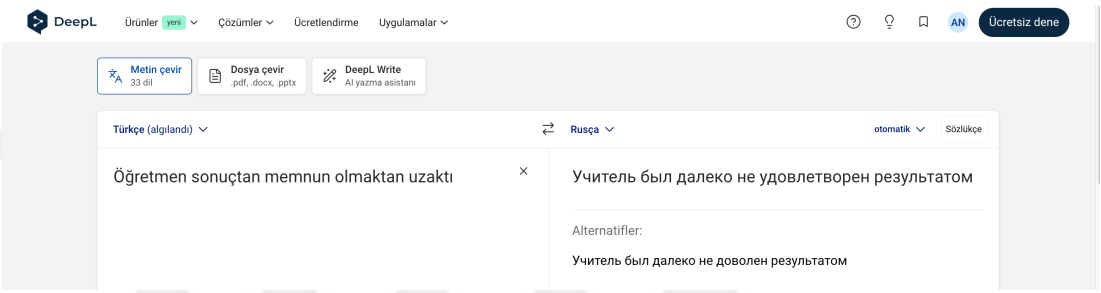

Figure 6.5. Testing example text in DeepL.

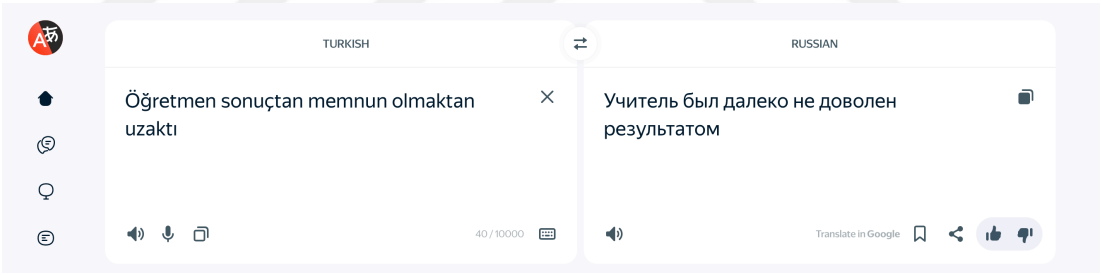

Figure 6.6. Testing example text in Yandex Translator.

Table 6.2. Comparison Of The Translated Sentences Among Google Translator, DeepL, Yandex Translator, And Our Aurora Translator.

Source Sentences	Öğretmen sonuçtan memnun olmaktan uzaktı
Google Translate (Figure 6.4)	Учительница была далека от результата
DeepL (Figure 6.5)	Учитель был далеко не удовлетворен результатом
Yandex Translator (Figure 6.6)	Учитель был далеко не доволен результатом
Our Aurora Translator (Figure 6.3)	Учитель был далеко не удовлетворен результатом .

The source sentence "Öğretmen sonuçtan memnun olmaktan uzaktı" translates to "The teacher was far from being satisfied with the result." We can see from Table 6.2 comparison of results. Among the provided translations, DeepL, Yandex, and Aurora Translator provide fairly accurate interpretations, though nuanced differences exist. DeepL and Aurora Translator produce nearly identical outputs ("Учитель был далеко не удовлетворен результатом"), which convey the intended meaning accurately. Yandex Translator is also close in meaning but uses "доволен" (satisfied) instead of "удовлетворен" (fulfilled/satisfied), introducing a subtle variation. Google Translate's version, however, deviates by stating "Учительница была далека от результата" (The teacher was far from the result), which misses the "satisfaction" nuance and changes the focus of the sentence. This analysis shows that DeepL, Yandex, and Aurora capture the sentiment better, with Aurora standing out for including proper punctuation, indicating refinement.

6.5. COMPARISON OF RESULTS FOR TURKISH MT TASKS

Table 6.3 summarizes the approaches, language pairs, datasets, and BLEU scores for Turkish in previous MT studies compared to our model.

Table 6.3. Comparison Of Translation Results With Our Model.

Study	Approach	Language Pair	Dataset	BLEU
El-Kahlout & Oflazer (2006) [12]	SMT with morphological information	English-Turkish	192K parallel sentences	0.2517
Yıldırım & Tantug (2012) [11]	SMT with re-ranking N-best lists	English-Turkish	Google Translate Research API	BLEU improvement
Gülçehre et al. (2015) [21]	Language model fusion with NMT, RNN-based NMT, shallow/deep fusion	Turkish-English	IWSLT Turkish-English corpus	BLEU improvement: +2 points
Firat et al. (2016) [22]	Multilingual NMT with shared encoder – decoder Attention	Turkish-English, Uzbek-English	Parallel corpora for multiple languages	Comparable to SMT

Ataman et al. (2017) [10]	Vocabulary reduction strategies for NMT, attention mechanism	Turkish-English	BOUN-ITU Corpus	BLEU improvement of ~3 points
Tukeyev et al. (2019) [23]	Morphological segmentation for NMT, encoder-decoder architecture	Turkic languages (incl. Turkish)	Turkic language corpora	BLEU improvement
Yirmibeşoğlu & Güngör (2019) [24]	Hybrid BiDeep + Transformer model (with 8 layer Transformer, data Augmentation)	Turkish-English	SETimes corpus (English-Turkish parallel)	0.2638
Sel et al. (2021) [75]	Bi-LSTM-based NMT with sentence alignment	Turkish-English	1M parallel sentence pairs from academic theses	0.158 (TED)
Aurora Translator	Transformer-based neural machine translation with self-attention	Turkish-Russian	AuroraDataset (476K parallel pairs)	0.4903

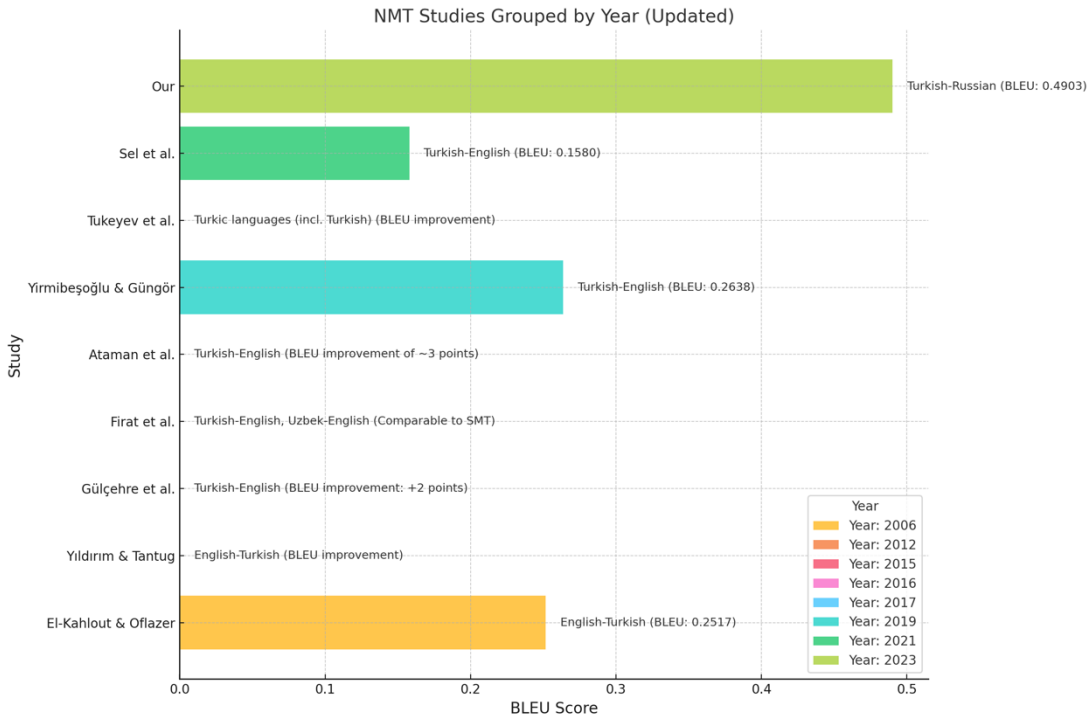

Figure 6.7. Comparison of BLEU scores and language pairs for Turkish MT task studies.

Figure 6.7 highlights a significant gap in the existing literature on Turkish-to-Russian MT. As we can see from the Figure 6.7 AuroraTranslator achieves the highest BLEU score 0.4903 among all studies, demonstrating exceptional performance in Turkish-Russian translation, a less-explored low-resource language pair. Most previous works in the literature have concentrated on Turkish-English or other Turkic languages, leaving Turkish-Russian MT largely unexplored. My contribution paves the way for future advancements in Turkish-Russian MT tasks. Early SMT approaches, like El-Kahlout & Oflazer (2006) [12], struggle with morphological complexity, whereas Transformer-based models, such as AuroraTranslator, consistently outperform other architectures. Larger datasets, like Sel et al. [75] 1M parallel pairs, improve BLEU scores for general-purpose MT but lack domain-specific accuracy, highlighting the adaptability of domain-focused models for specialized contexts. Overall, the results emphasize the robustness of Transformer-based models for morphologically complex, low-resource languages like Turkish and Russian, with AuroraTranslator setting a new benchmark for Turkish-Russian MT while underscoring the need for domain-specific features and bias mitigation in future research.

PART 7

CONCLUSION

In this study, I developed and evaluated a transformer-based NMT model for Turkish-to-Russian translation. Leveraging the transformer architecture, which has demonstrated superior performance over traditional RNNs and BiLSTMs, our model successfully addressed the complexities inherent in translating between Turkish and Russian. Our experiments utilized the AuroraDataset, consisting of 476K parallel Turkish-Russian sentence pairs, and achieved notable results. After training for 30 epochs, our model attained a BLEU score of 0.4903. These metrics underscore the effectiveness of the transformer architecture in capturing the contextual information necessary for high-quality translations. As a result, I developed an effective, efficient, and high-quality transformer-based translation system for Turkish to Russian. The findings from this study demonstrate the potential of transformer-based models in addressing the challenges associated with Turkish-to-Russian translation. This research not only advances the understanding of transformer applications in MT but also sets the stage for further exploration into optimizing such models for languages with limited text corpora.

Despite the promising results achieved in this study, there are several areas where further research and improvements can be made to enhance the performance and robustness of our transformer-based NMT model for Turkish-to-Russian translation, as well as extending its application to Turkish-to-Russian translation. Future work should explore various regularization techniques such as dropout, weight decay, and data augmentation to address overfitting. Future work should focus on correcting the dataset fully by native speakers and expanding the dataset with additional parallel corpora. High resource dataset can provide the model with more diverse linguistic examples, potentially improving its ability to generalize. Further optimization of hyperparameters, incorporating pre-trained language models, and focusing on domain

adaptation techniques are also crucial. Additionally, utilizing comprehensive evaluation metrics, enhancing model interpretability, and investigating cross-lingual transfer learning approaches could significantly advance the capabilities of transformer-based NMT models for Turkish-to-Russian translations.

REFERENCES

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I., "Attention Is All You Need," ***Adv Neural Inf Process Syst***, 2017-December, 5999–6009 (2017).

2. Yapıcı, F., "The Problem of Realia's Translation from Turkish to Russian," Araştırma/Research, **https://www.academia.edu/44814928/The_Problem_of_Realias_Translation_from_Turkish_to_Russian_Araştırma_Research_Fatih_YAPICI.**

3. "A Comparative Study on Google Translate: An Error Analysis of Turkish-to-English Translations in Terms of the Text Typology of Katherina Reiss," **https://dergipark.org.tr/en/pub/rumelide/article/606217.**

4. "Turkic languages," Wikipedia, **https://www.en.wikipedia.org/wiki/Turkic_languages**

5. Bayatlı, S., Kurnaz, S., Salimzianov, I., Washington, J. N., and Tyers, F. M., "Rule-Based Machine Translation from Kazakh to Turkish," ***Proceedings of the 21st Annual Conference of the European Association for Machine Translation***, 49–58 (2018). **https://doi.org/10.5281/zenodo.2708275.**

6. Gökırmak, M., Tyers, F. M., and Washington, J. N., "A Free/Open-Source Rule-Based Machine Translation System for Crimean Tatar to Turkish," ***Proceedings of the 3rd Workshop on the Use of Computational Methods in the Study of Endangered Languages*** (2019).

7. Brown, F., Pietra, S. D., Pietra, V. J. D., and Mercer, R. L., "A Statistical Approach to Machine Translation," ***Computational Linguistics***, 16 (2): 79–85 (1990). **https://aclanthology.org/J90-2002.**

8. Callison-Burch, P., Koehn, P., Fordyce, C. S., and Monz, C., "Proceedings of the Second Workshop on Statistical Machine Translation," (2007). **https://aclanthology.org/W07-0700.**

9. Türe, F., "A Hybrid Machine Translation System from Turkish to English," ***Yüksek Lisans Tezi, Sabancı Üniversitesi***, (2008).

10. Ataman, D., Negri, M., Turchi, M., and Federico, M., "Linguistically Motivated Vocabulary Reduction for Neural Machine Translation from Turkish to English," ***The Prague Bulletin of Mathematical Linguistics***, 108: 331–342 (2017). **https://doi.org/10.1515/pralin-2017-0031.**

11. Yıldırım, A., and Tantug, A. C., "The Feasibility Analysis of Re-Ranking for N-Best Lists on English-Turkish Machine Translation," ***IEEE International Symposium on Innovations in Intelligent Systems and Applications, IEEE INISTA***, doi: 10.1109/INISTA.2013.6577652 (2013).

12. El-Kahlout, I. D., and Oflazer, K., "Exploiting Morphology and Local Word Reordering in English-to-Turkish Phrase-Based Statistical Machine Translation," ***IEEE Trans Audio Speech Lang Process***, 18 (6): 1313–1322 (2010). doi: 10.1109/TASL.2009.2033321.

13. Yeniterzi, R., and Oflazer, K., "Syntax-to-Morphology Mapping in Factored Phrase-Based Statistical Machine Translation from English to Turkish," ***Association for Computational Linguistics***, 454–464 (2010). Erişim: 13 Temmuz 2024, **https://aclanthology.org/P10-1047.**

14. Yılmaz, M., and Uğuz, İ., "TURKISH-ENGLISH MACHINE TRANSLATION SYSTEM," (2020).

15. Oflazer, İ., and El-Kahlout, D., "Exploring Different Representational Units in English-to-Turkish Statistical Machine Translation," ***Association for Computational Linguistics***, 25–32 (2007). **https://aclanthology.org/W07-0704.**

16. Sutskever, I., Vinyals, O., and Le, Q. V., "Sequence to Sequence Learning with Neural Networks," ***Adv Neural Inf Process Syst***, 4 (January): 3104–3112 (2014). **https://arxiv.org/abs/1409.3215v3.**

17. Kalchbrenner, N., and Blunsom, P., "Recurrent Continuous Translation Models," ***Association for Computational Linguistics***, 1700–1709 (2013). https://aclanthology.org/D13-1176.

18. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," ***EMNLP***, 1724–1734 (2014). doi: 10.3115/V1/D14-1179.

19. Bahdanau, D., Cho, K. H., and Bengio, Y., "Neural Machine Translation by Jointly Learning to Align and Translate," ***ICLR Conference Track Proceedings***, (2015). Erişim: 13 Temmuz 2024, **https://arxiv.org/abs/1409.0473v7.**

20. Junczys-Dowmunt, M., Dwojak, T., and Hoang, H., "Is Neural Machine Translation Ready for Deployment? A Case Study on 30 Translation Directions," (2016). **https://aclanthology.org/2016.iwslt-1.5.**

21. Gülcehre, C., Firat, O., Xu, K., Cho, K., and Bengio, Y., "On Integrating a Language Model into Neural Machine Translation," ***Computational Speech and Language***, 45: 137–148 (2017). doi: 10.1016/J.CSL.2017.01.014.

22. Firat, O., Cho, K., Sankaran, B., Yarman Vural, F. T., and Bengio, Y., "Multi-Way, Multilingual Neural Machine Translation," ***Computational Speech and Language*****,** 45: 236–252 (2017). doi: 10.1016/J.CSL.2016.10.006.

23. Tukeyev, U., Karibayeva, A., and Zhumanov, Z., "Morphological Segmentation Method for Turkic Language Neural Machine Translation," ***Cogent Engineering***, 7 (1): 1856500 (2020). https://doi.org/10.1080/23311916.2020.1856500.

24. Yirmibeşoğlu, Z., and Güngör, T., "Morphologically Motivated Input Variations and Data Augmentation in Turkish-English Neural Machine Translation," ***ACM Transactions on Asian and Low-Resource Language Information Processing***, 22 (3): (2023). doi: 10.1145/3571073/ASSET/AC619299-699C-4C6F-982F-EE94A8828F87.

25. Abduvalieva, G. A., "Features of Russian-Turkish and Turkish-Russian Simultaneous Interpretation," ***Linguistics and Culture Review***, 5 (S4): 1451–1463 (2021). https://doi.org/10.21744/lingcure.v5nS4.1766.

26. Mirzakhalov, J., "Turkic Interlingua: A Case Study of Machine Translation in Low-Resource Languages," ***Master's Thesis*****,** ***University of South Florida*** (2021).

27. Hutchins, W. J., and Somers, H. L., An Introduction to Machine Translation, Academic Press, London (1992).

28. Baumgartner-Bovier, "La Traduction Automatique, Quel Avenir? Un Exemple Basé sur Les Mots Composés," ***Cahiers de Linguistique Française***, N°25 (2003).

29. Chéragui, M. A., "Theoretical Overview of Machine Translation," ***Proceedings ICWIT 2012, African University, Adrar, Algeria*** (2012).

30. Hutchins, J. W., ***Machine Translation: Past, Present, Future***, Halsted Press, New York (1986).

31. Hutchins, J. W., and Somers, H. L., An Introduction to Machine Translation, Academic Press, London (1992).

32. Buchmann, B., Warwick, S., and Shann, P., "Design of a Machine Translation System for a Sublanguage," ***Proceedings of the 22nd Annual Meeting on Association for Computational Linguistics***, 334–337 (1984).

33. Varile, G. B., and Lau, P., "Eurotra: Practical Experience with a Multilingual Machine Translation System Under Development," ***Proceedings of the Second Conference on Applied Natural Language Processing***, 160–167 (1988).

34. Nagao, M., and Tsujii, J., "The Transfer Phase of the Mu Machine Translation System," ***Proceedings of the Eleventh Conference on Computational Linguistics***, 97–103 (1986).

35. Koehn, P., Och, F. J., and Marcu, D., "Statistical Phrase-Based Translation," ***NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology***, 48–54 (2003).

36. Brown, P. F., Della Pietra, V. J., Della Pietra, S. A., and Mercer, R. L., "The Mathematics of Statistical Machine Translation: Parameter Estimation," ***Computational Linguistics***, 19 (2): 263–311 (1993).

37. Chiang, D., "A Hierarchical Phrase-Based Model for Statistical Machine Translation," ***Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics***, 263–270 (2005).

38. Jurafsky, D., and Martin, J. H., Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Prentice Hall (2006).

39. Türel, F., "A Hybrid Machine Translation System from Turkish to English," ***Master's Thesis*****, Sabancı University** (2008).

40. Chéragui, M. A., "Theoretical Overview of Machine Translation," **Proceedings ICWIT 2012*, African University, Adrar, Algeria*** (2012).

41. Hutchins, J., "Machine Translation: A Brief History," ***Concise History of the Language Sciences: From the Sumerians to the Cognitivists***, Koerner, E. F. K., and Asher, R. E. (eds.), Pergamon Press, Oxford, 431–445 (1995).

42. Sumita, E., Iida, H., and Kohyama, H., "Translating with Examples: A New Approach to Machine Translation," ***The Third International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*****,** 203–212 (1990).

43. Lavie, L., Vogel, S., Peterson, E., Probst, K., Font-Llitjós, A., Reynolds, R., Carbonell, J., and Cohen, R., "Experiments with a Hindi-to-English Transfer-Based MT System Under a Miserly Data Scenario," ***ACM Transactions on Asian Language Information Processing*** (2004).

44. Imamura, K., Okuma, H., Watanabe, T., and Sumita, E., "Example-Based Machine Translation Based on Syntactic Transfer with Statistical Models," ***Proceedings of the 20th International Conference on Computational Linguistics***, Vol. 1, University of Geneva, Switzerland, 99–105 (2004).

45. Imamura, K., Automatic Construction of Translation Knowledge for Corpus-Based Machine Translation, ***Doktora Tezi, 10 Mayıs*** (2004).

46. Lavie, L., Vogel, S., Peterson, E., Probst, K., Wintner, S., and Eytani, Y., "Rapid Prototyping of a Transfer-Based Hebrew-to-English Machine Translation System," ***Proceedings of the 10th International Conference on Theoretical and***

Methodological Issues in Machine Translation **(TMI-04), Baltimore, MD, USA, 1–10** (2004).

47. Knight, K., Chander, I., Haines, M., Hatzivassiloglou, V., Hovy, E., Iida, M., Luk, S. K., Whitney, R., and Yamada, K., "Filling Knowledge Gaps in a Broad-Coverage MT System," ***Proceedings of the 14th IJCAI Conference*** (1995).

48. Habash, N., "Generation-Heavy Hybrid Machine Translation," ***Proceedings of the 2nd International Natural Language Generation Conference*** (INLG-02) (2002).

49. Ayan, N. F., Dorr, B. J., and Habash, N., "Application of Alignment to Real-World Data: Combining Linguistic and Statistical Techniques for Adaptable MT," ***Proceedings of the 6th Conference of the Association for Machine Translation in the Americas*** (AMTA-04) (2004).

50. Probst, K., Learning Transfer Rules for Machine Translation with Limited Data, ***Doktora Tezi, Carnegie Mellon University, Language Technologies Institute*** (2005).

51. Brown, P. F., Della Pietra, V. J., Della Pietra, S. A., and Mercer, R. L., "The Mathematics of Statistical Machine Translation: Parameter Estimation," ***Computational Linguistics***, 19 (2): 263–311 (1993).

52. Knight, K., "Decoding Complexity in Word-Replacement Translation Models," ***Computational Linguistics***, 25 (4): 607–615 (1999).

53. Yang, S., Wang, Y., and Chu, X., "A Survey of Deep Learning Techniques for Neural Machine Translation," *arXiv* (2020). https://arxiv.org/abs/2002.07526

54. Stahlberg, F., "Neural Machine Translation: A Review and Survey," *arXiv* (2019). https://arxiv.org/abs/1912.02047

55. Graves, A., "Generating Sequences with Recurrent Neural Networks," *arXiv Preprint*, arXiv:1308.0850 (2013).

56. He, K., Zhang, X., Ren, S., and Sun, J., "Deep Residual Learning for Image Recognition," ***Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition***, 770–778 (2016).

57. Lei Ba, J., Kiros, J. R., and Hinton, G. E., "Layer Normalization," ***arXiv Preprint***, arXiv:1607.06450 (2016).

58. Press, O., and Wolf, L., "Using the Output Embedding to Improve Language Models," ***arXiv Preprint***, arXiv:1608.05859 (2016).

59. Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N., "Convolutional Sequence to Sequence Learning," ***arXiv Preprint***, arXiv:1705.03122v2 (2017).

60. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., and Amodei, D., "Language Models are Few-Shot Learners," ***Advances in Neural Information Processing Systems*** (NeurIPS), 33: 1877–1901 (2020).

61. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L., "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," ***Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)***, 7871–7880 (2020).

62. Devlin, J., Chang, M. W., Lee, K., and Toutanova, K., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," ***Proceedings of NAACL-HLT 2019***, 4171–4186 (2018). **https://arxiv.org/abs/1810.04805.**

63. Schuster, M., Johnson, M., and Thorat, N., "Zero-Shot Translation with Google's Multilingual Neural Machine Translation System," ***Google AI Blog (2016).*** **https://ai.googleblog.com/2016/11/zero-shot-translation-with-googles.html.**

64. Tiedemann, J., and Thottingal, S., "OPUS-MT: Building Open Translation Services for the World," ***arXiv Preprint***, arXiv:2001.09907 (2020 **https://arxiv.org/abs/2001.09907.**

65. Google Colab, "Colaboratory Hakkında," **www.colab.google.**

66. "Colaboratory SSS", ***Google AI Research FAQ***, **www.research.google.com/colaboratory/intl/tr/faq.html.**

67. "Coding a Transformer from Scratch on PyTorch, with Full Explanation, Training, and Inference", ***YouTube*** (2020). **www.youtube.com/watch?v=ISNdQcPhsts.**

68. Papineni, S., Roukos, S., Ward, T., and Zhu, W. J., "BLEU: A Method for Automatic Evaluation of Machine Translation," *Proceedings of the 40th Annual* ***Meeting on Association for Computational Linguistics*** (ACL '02), 311–318 (2002). doi: 10.3115/1073083.1073135.

69. Axelrod, A. E., "Factored Language Models for Statistical Machine Translation," (2006).

70. McCowan, I., Moore, D., Dines, J., Gatica-Perez, D., Flynn, M., Wellner, P., and Bourlard, H., "On the Use of Information Retrieval Measures for Speech Recognition Evaluation," ***IDIAP-RR 73***, IDIAP, Martigny, Switzerland (2004).

71. Thennal, D. K., James, J., et al., "Advocating Character Error Rate for Multilingual ASR Evaluation," (2020).

72. HuggingFace, “Tr-En Translator Dataset,” ***Datasets Repository***, **www.huggingface.co/datasets/adoxalim/Tr-En-Translator.**

73. Gradio, “Gradio Interactive Application,” **https://www.gradio.app/.**

74. Shaankhosla, “Technical Blog Posts,” **www.shaankhosla.substack.com.**

75. Sel, İ., Üzen, H., and Hanbay, D., “Creating a Parallel Corpora for Turkish-English Academic Translations,” ***Journal of Computer Science, IDAP-2021, Special Issue***, 335–340 (2021).

RESUME

Nurzhan AMANTAY commenced his educational journey at Astana Polytechnic College in Kazakhstan, where he studied from 2011 to 2015 and earned his diploma as a Technician Programmer. He then advanced his studies in Computer Engineering at Istanbul Commerce University, Turkey, from 2015 to 2021, further solidifying his expertise in the field. Currently, he is pursuing a master's degree in Computer Engineering at Karabuk University, Turkey, having started in 2021.

Nurzhan's professional career includes diverse experiences across various tech companies. In 2013, he worked at QB Solutions in Kazakhstan-Astana, contributing to the development of the "EXPO2017" mobile application. From 2018 to 2019, he engaged with Seven Gates in Istanbul, Turkey, where he was involved in a blockchain-based project called Further Network. His next role was with Edugineer-Mimcrea in Istanbul from 2019 to 2020, focusing on an online education platform using the Python with Django framework. Nurzhan then worked as a Front-End Developer at Pometop in Istanbul from 2020 to 2021. Since 2021, he has been serving as a Full-Stack Java Developer at Aurora Bilişim in Istanbul, where he continues to apply and expand his skills in the industry.