6CCS3PRJ Final Year

PETRAI: Utilising Machine Learning Techniques for English-Russian Translation

Final Report

Author: Michael A. Higham

Supervisor: Dr. Kevin Lano

Student ID: K21051343

Program of study: BSc Computer Science

April 12, 2024

Abstract

Since the dawn of language, humanity has been seeking methods on how to translate between languages to aid communication and, with the advent of the computer, machine translation has been concepted to ease this process. From rule-based machine translation in the 1950s, to the rise of statistical machine translation in the 1990s, to the various Neural Network architectures that can be seen today, the rise of machine translation is an ever-changing field with frequent innovative discoveries.

This project delves into the question of how accessible machine translation is to individuals and small organisations who are seeking to build machine translation models for low-resource languages. Through the development of PETRAI, the Proficient English To Russian Artificial Intelligence, it is possible to see the various advantages and set backs that exist for those seeking to develop their own bespoke machine translation models with limited resources. PETRAI is trained on a relatively small amount of data and utilises various machine translation techniques in order to develop an accurate and fluent translation model. PETRAI is then evaluated against other models using various standardised machine translation metrics showing that with further training and various tweaks to the model, it is possible for low-resource languages to develop a model using a relatively small training dataset whilst still producing coherent translations.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary.

I also verify that this report does not exceed 25,000 words.

Michael A. Higham

April 12, 2024

Acknowledgements

I would like to thank my supervisor, Dr Kevin Lano for his support and guidance throughout this project. I also thank my mother, Yuliya Higham, for her continued support navigating the Russian Language, with its many intricacies, and my significant other, Cara-Rose Morgan-Heatley, for persevering through my rambles about AI models.

Contents

Introduction

In recent years, Artificial Intelligence and Deep Learning have experienced massive leaps in development, especially given the release of the publicly renowned Large Language Model ChatGPT. In the domain of Natural Language Processing, these techniques have been used to improve Machine Translation, which involves the automatic translation of text from a given source language to a target language. Machine Translation has been crucial in facilitating multilingual communication across the world and is becoming ever more important, given the global and multilingual nature of business and the internet. Therefore, developing more accurate and faster Machine Translation models in more and more languages has become a key focus within the field. The developments of new architectures, and especially the development of Transformer Neural Networks and the Attention Mechanism, has been pivotal in recent innovations in the the domain with recent models scoring highly in translation accuracy metrics.

Whilst Machine Translation has developed, the disparity of language coverage, with lower-resource languages struggling to be represented on the internet. The phenomenon known as "digital language death" [1] has been attributed to the self-fulfilling cycle of rarer languages being less commonly used on the internet, given its lack of support, leading to less resources available in that language. This leads to less native-made resources in order to train translation models which, therefore, leads to lower support. It is clear that there are still strides that need to be made in the democratisation of translation technology in order to preserve language diversity. This project focuses on assessing the current state of machine translation and evaluating its accessibility for the development of models through developing my own translation models for English-Russian translation. Evaluating the creation my own models against established English-Russian models will allow me to research the ease of creation of a well-resourced language pair and, therefore, evaluation of the creation of models for lower-resource languages.

In the following report, the **Background and Literature Review** chapter discusses the history and current state of Machine Translation, discussing various Machine Translation architectures and current difficulties of Machine Translation models. There is also the compar-

ison and discussion of various existing tools for training Russian-English Machine Translation models, including datasets and scoring methods. In the **Requirements and Specification Section**, describe the requirements and specification for the model and the user interface. The **Design Section**, describes the design of the user interface and the technologies used for the translation model. The **Implementation Section** describes how the translation model was trained and how it translates text. The evaluation and results of the trained models and some models from literature are found in the **Evaluation and Analysis Section**. There is the **Legal, Social, Ethical, and Professional Issues** describing and considering the relevant potential issues with the project and an evaluation of the project against the guidelines from the British Computing Society. Finally, the **Conclusion and Further Work** concludes the project and considers the future steps that could be taken for the project.

Background and Literature Review

2.1 Languages

2.1.1 Russian

Russian is an Eastern Slavic branch of the Indo-European language family [2] and is spoken by over 300 million people worldwide [3, p.1]. It the official language of the Russian Federation [4, art.68] and, along with English, is one of two main languages spoken on the International Space Station [5]. Russian's alphabet is based on the Cyrillic writing system and consists of 33 letters with twenty consonants (ш, р, т, п, щ, с, д, ф, г, ч, к, л, ж, з, х, ц, в, б, н, м); ten vowels (я, е, ы, у, и, о, ю, э, а, ё); one semivowel (й); a soft sign (ь); and a hard sign (ъ) [6, p.1]. These latter two letters are modifier signs and are not spoken as individual sounds [3, p.1], but rather they change how the word overall is pronounced.

In Russian grammar, nouns can take on of three genders: masculine, feminine, and neuter [6, p.30]. There are also six cases: nominative, accusative, genitive, dative, instrumental, and prepositional [6, p.50] although remnants of the former vocative case do still show in certain exclamations [6, p.104]. Declension, where certain word endings are conjugated, depends on tense, case, numeracy, and animacy [6, p.50].

2.1.2 English

Belonging to the Western Germanic branch of the Indo-European language family [7], English and Russian show differences in their linguistic structures. One area of notable complexity is in the usage of their respective tense and case systems. In English, there are more tenses than in Russian. In contrast, Russian compensates for its relatively smaller tense system by incorporating more cases. In the Russian language, cases alter the word endings to convey the particular meaning within a sentence [3]. The concept of declension in Russian extends beyond English's usage, where declension is primarily reliant upon case and numeracy. While

English relies more on the tense system, Russian relies more on the usage of cases, reflecting the inherent intricacies of each language.

2.1.3 Difficulties with Translation between English and Russian

The United States Department of Defence's Defense Language Institute describes Russian as a Category III Language in relation to the difficulty of learning the language as a native English speaker. The scale ranges from I to IV where I is consists of relatively easier languages to learn, such as Dutch or French, and IV are relatively harder languages, such as Japanese and Korean [8]. This indicates that Russian is relatively harder to learn and, thus, could be harder to translate between the two languages due to their differences.

2.2 Human Translation

Human translation is the process carried out by humans of converting a text from a natural source language to a natural target language.

Translation does not consist of utilising one skill to convert one text to another, but rather a complex mix of being able to understand two languages both syntactically and semantically. Syntactical understanding involves knowledge of finite and defined rules such as grammatical structure or word conjugation whereas semantic understanding requires a more abstract understanding of culture in order to be able to convey the embedded meaning and context to the target language. Linguist Mario Pei concludes that, learning a language "comes close to being a life-time job" due to its multi-faceted nature from speaking to reading, and writing to understanding [9, p.397].

There is often a conflicting battle in translation between two seemingly opposing qualities of translation: fidelity and fluency, sometimes referred to as transparency. Fidelity refers to how close the translation text has retained the original meaning of the source text without distortion. In contrast, fluency refers to how close the translation is to appearing as if it has been written in the target language. In an example case, word order may be important to the meaning of a sentence as it could reveal information in a intended way. However, a more fluent translated version of this text may lose this as it would not retain the specific word order whereas a higher fidelity translation may be harder to read as it would not follow the regular syntax of the target language. Whilst these qualities are not mutually exclusive, they often conflict with each other depending on the interoperability of the source and target language.

It is clear that, due to the aforementioned factors, that the skill of translation runs much deeper than a surface level of text-conversion from one language to another. As a result, this is what makes expert translators so valuable as the ability of humans to understand context using surrounding sentences and inferring connections with the relevant cultures in order to optimise the balance between fidelity and fluency [10, p.6]. This is especially relevant in texts such as literary works, where meaning is vital to the importance and meaning of the text.

2.3 Machine Translation

As opposed to human translation, Machine Translation is the process carried out by machines of converting a text from a natural source language to a natural target language.

2.3.1 Early Beginnings

The origins of Machine Translation can be traced back to March 1947, where the director of the Natural Sciences Division at the Rockefeller Foundation named Warren Weaver stated that for a given Russian text, one could say that it was "really written in English, but it has been coded in some strange symbols. I will now proceed to decode" [11, p.195]. This suggests that not only could cryptography techniques be used in the art of translation but, as a result, a computer could be designed in order to translate natural languages, as the Enigma machine had decoded German communications.

In 1959, philosopher Bar-Hillel was arguing against the possibility all together of a fully-automatic, high-quality machine translator (FAHQMT), as a candidate FAHQMT would not be able to understand the surrounding context of a sentence in order to provide a high-fidelity translation, especially in the case of homonyms. He gave the paragraph, "Little John was looking for his toy box. Finally he found it. The box was in the pen. John was very happy." as his contradiction. Pen can be defined as 'a small area surrounded by a fence', 'a writing instrument', 'a hill, 'a female swan', etc. [12] and a candidate FAHQMT would not be able to reliably select the correct corresponding word in the target language to translate the word to. As a result, there could be no such true FAHQMT as it would either require human input, and thus not fully-automatic, or it would not produce high-quality translations [13, p.13].

2.3.2 Rule-Based Machine Translation

One of the initial techniques proposed for a usable Machine Translation was Rule-based Machine Translation (RBMT). This improves upon the idea of direct translation proposed by Warren Weaver where the system attempts to "determine a more abstract representation of the content" before attempting to translate it [14, p.18]. RBMT models, or the interlingual approach, work by converting the source text into an interlingua, which is a formal definition and representation of the language which is language-independent [13, p.2], via a process called analysis. It is then possible to write conversion rules which will allow you take the language-independent interlingua and form sentences in the target language using the corresponding language rules in a process called generation.

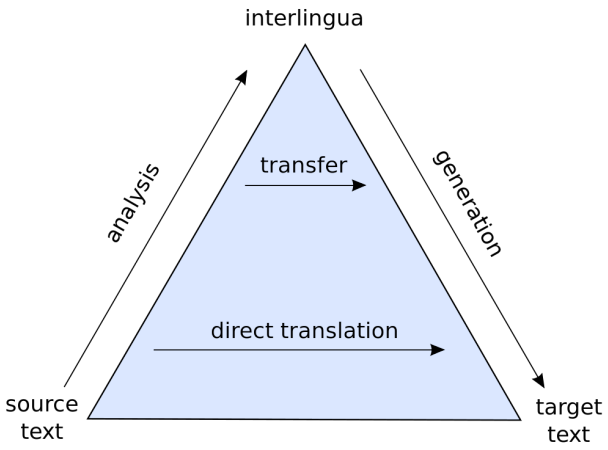

Figure 2.1: Vauquois' triangle [15]

Another related approach is the transfer approach which utilises components of the interlingual and the direct translation approach. Developed in the 1950s, this approach builds upon the notion of direct translation by considering "fragments" of text instead of purely individual words. This enables the translation of idioms and set-phrases by considering the context of the surrounding words [14, p.18]. The approach maintains the analysis process and uses a transfer component which determines how fragments should be translated. This is then run through the generation process in order to convert the fragments back into the target text [13, p.2]. This is often considered a "very ambitious" approach as accurately defining the analysis and generation stages requires a "complete understanding" of both the source and target language for the respective parts of the process [14, p.20].

This approach is limiting as for every pair of languages, individual transfer components need to be produced in order to be developed. However, as interlingual approach provides an abstracted representation of the source language, this provides the benefit of being able to translate of any given target language from said interlingua [14, p.19]. This enables for the independent development of analysis and generation which means that experts in various languages could aid the development of analysis and generation processes in their own areas of expertise. However, the level of control gained by RBMT is faced with the issue that explicitly defining every language rule is labour-intensive and virtually impossible due to the large number intricacies arising from languages developing over thousands of years.

2.3.3 Statistical Machine Translation

In the 1990s, as computing power increased and the internet started to introduce vast sums of data, new approaches were developed in order to utilise this. The basic notion of Statistical Machine Translation (SMT) is to find patterns between sets of data and use probabilistic techniques in order to determine the output translation of a new given source text. This is done by producing two models: the language and the translation model. These models allow us to produce probabilities for strings of words, called fragments, for strings of words from the source text and strings of words from the target text. These are referred to as source and target fragments respectively.

In 1993, IBM researcher Peter Brown defined SMT mathematically in terms of probability [16, p.264-265]. Probability $P(T)$ for target fragment, T, is given by our language model. Poibeau tells us that, $P(T)$ indicates that the "the red car" appears more frequently in the the language model than "car the red" "car red the" or "red car the" [14, p.80]. This helps for us to choose the fragment with the highest chance of being correct whilst still accounting for the possibility of multiple correct options. It also accounts for word order as we assume our language model to be correct.

These probabilities are formed by taking the consecutive conditional probability of each word given the previous words. Given n is the number of words in fragment, F, with f_i being the indexed word in the sentence, we can use the following equation to determine $P(S)$:

$$P(F) = P(f_1) \cdot P(f_2|f_1) \cdot P(f_3|f_1, f_2) \cdot ... \cdot P(f_n|f_1...f_{n-1})$$

In long sentences, the resulting calculations can often become too large and, thus, must be limited. Therefore, it is typical to use 'bigram' and 'trigram' models which only take into account the preceding one or two words respectively. The trigram model would be implemented as follows:

$$P(F) \ = \ P(f_1) \ \cdot \ P(f_2|f_1) \ \cdot \ P(f_3|f_1, f_2) \ \cdot \ P(f_4|f_2, f_3) \ \cdot \ ... \ \cdot \ P(f_n|f_{n-1}, f_{n-2})$$

Given that S represents the source fragment, our translation model provides us with probabilities in the form $P(S|T)$ which represents the probability a given S will be translated into T. Therefore, to find the corresponding target fragment, $\hat{T}$, for a source fragment, S, we need to find a T which produces the product in the following equation:

$$\hat{T} = \arg\max_{T}(P(T) \ \cdot \ P(S|T))$$

This is the "Fundamental Equation of Machine Translation" and is the core equation used in SMT [16, p.265]. To form the translation model, large amounts of data are needed to calculate the conditional probabilities. and this comes in the form of bilingual parallel corpora [13, p.191]. A bilingual parallel corpus is a pair of aligned texts where one text is written in the source language and the other text is its corresponding translation in the target language. It is critical that they are aligned correctly as the model uses the pairs of texts to learn associations and mappings between the pairs. As a result, SMT models are better at grasping meaning and translating idiomatic phrases as they rely on relationships between words within the phrases in order to determine the resulting translation.

Using the bilingual parallel corpora, the conditional probabilities, in the form $P(S|T)$, can then be calculated using the SMT process proposed by IBM. Firstly, the system has to determine the "best possible alignment" between the source and target text using the lengths of both texts to aid this [14, p.91]. The aligning fragments can then be added to a dictionary aligning source fragments with one or more corresponding target fragments and associating a probability with it. These fragments could be words or phrases, depending on the approach taken, with IBM initially proposing a word-based system where as more modern approaches choose to align sentences based on phrases for more fluent translations.

2.3.4 Neural Machine Translation

SMT models excel at learning phrases by identifying patterns within extensive sets of parallel data. However, SMT models face limitations as they can not "[model] long-distance dependencies", particularly over larger phrases [17, p.1]. These models are primarily designed to identify and exploit patterns between smaller fragments of text.

Since 2013, there has been a push to develop Neural Machine Translation (NMT) and, as a result, has has now become the leading paradigm in Machine Translation, swiftly overtaking SMT. NMT employs a single large neural network to manage the translation process, eliminating the need for multiple smaller components as found in traditional SMTs [17, p.1, 4]. NMTs build upon the fundamental idea of probability found in SMTs by trying to find the conditional probability $P(S|T)$ for a source input, S, and an target output, T [17, p.1]. To do this, NMT architectures consist of several key components to perform the translation process. These components include:

1. **Embedding Layer:** This layer converts fragments into one "continuous representation" in the form of a vector. The $\langle bos \rangle$ and $\langle eos \rangle$ symbols are used to denotes the begin and end of sentences respectively as the source and target fragments are contained within the same vector. This is what given the Seq2Seq (Sequence to Sequence) model its name. Continuous representation allows the model to capture semantic relationships between fragments by passing the embeddings through [17, p.2].

2. **Encoder Phase:** The encoder processes the vector passed in from the embedding layer. It is responsible for mapping the intricacies and ordering found in the source language. The result of this phase is hidden and, thus, is known as a hidden state [17, p.2] [18, p.47-48].

3. **Decoder Phase:** The decoder ensures the fluent and accurate translation of the source input by using the encoder information and the information about the target language, modelled in the training process, to "[model] long-distance dependencies between target words" [17, p.2].

4. **Classification Layer:** The classification layer then uses this information passed in from the decoder to generate the target sequence by predicting each word based on the information and on the previous word generated [17, p.2-3].

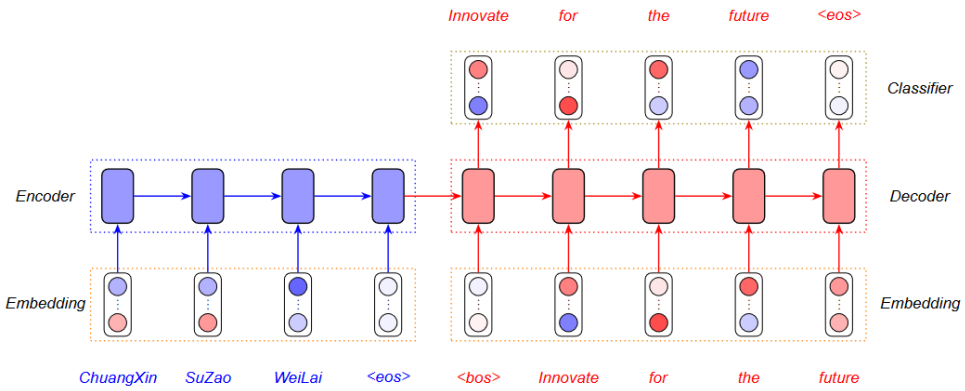

Figure 2.2: An overview of the Neural Machine Translation architecture [17, p.2]

In NMT, there has been three main types of network architecture used for encoders and decoders: Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), and Transformers Neural Networks (TNN).

Recurrent Neural Networks

Typically, neural networks are ”feed forward”, this means that each node can only feed their value into a node on the next layer, dependent on a given parameter. However, RNNs are recurrent and contain ”feedback loops” which allows for a node to feed data back into itself for future calculations. This makes it useful for sequenced data input, such as translation, as the feedback loops are able to act as a type of ”memory”. However, their are held back by this as the simple feedback loops can only retain data for so many iterations before it is lost [19]. This is known as the ”problem of vanishing gradients” [20] and affects the ability of the network architecture to model long-distance dependencies between words and, thus, only really useful for shorter translations.

Long Short-Term Memory (LSTM) Networks are a more complex configuration of RNNs and is widely used in deep learning tasks. They overcome the vanishing gradient problem by developing more complex feedback loops built of gates and memory cells which allow for higher retention of data over longer time spans [21]. This makes them more useful for longer translations as they excel where RNNs struggle.

Convolutional Neural Networks

Despite their development and proprietary use being for image-processing applications [19], CNNs have found their usage in Machine Translation. In 2017, Facebook’s (now known as Meta) AI Research team developed a CNN-based encoder-decoder system that was found to be more

accurate than Google's implementation of a LSTM-based system. CNNs are based convolutions, which is a mathematical operation used to filter wave forms. In terms of translation, multiple layers of convolutions can be used to create a "hierarchical representation of the input [text]" where elements closer together interact with one another on lower-level layers whilst the more distant ones interact on higher-level layers [22, p.1]. This removes the problem of data-retention from RNNs and, subsequently, LSTMs as CNNs can model distant dependencies based on a hierarchical structure. The measure of how well long-distance dependencies can be measured is done by the maximum path length between any combination of elements. For RNNs, this is $O(n)$, where n is the length of the sequence, and for CNNs this is $O(n/k)$, where k is the kernel size of the convolutions. This means, as you increase k, the efficiency of finding long-distance dependencies, and thus the ability to learn such dependencies, increases [23, p.5-6].

CNNs provide the advantage of linearity as each word inputted into the CNN has to be processed by a constant number of layers. However, RNNs apply n operations and non-linearities to the first word of the input, where n is the number of words in the input, with the number of operations decreasing linearly with every word. If the number of non-linearities is fixed, then the learning process is made even easier [22, p.1].

Attention Mechanism

The attention mechanism was introduced in a 2014 paper by Dzmitry Bahdanau and aims to resolve a bottleneck found in traditional systems that stems from the use of "fixed-length vector[s]" [24, p.1] and enable the decoder greater access to the input information which is thought to impact longer input sequences. The attention mechanism uses a weighted sum of all the hidden states found within the encoder to feed into the decoder. This sum allows the decoder to assigning varying degrees of importance to different elements of the input sequence [25]. The attention mechanism can be applied to both RNNs and CNNs in order to gain more fluent and higher fidelity translations. A further development of this mechanism, called Multi-head Attention (MHA), is also used in various NMT network architectures. MHA allows for the multiple previous words to be considered in the calculation of attention at a specific point by utilising multiple sets of weighted attention sums [22, p.4].

Transformer Neural Networks

In 2017, a research team at Google proposed a new network architecture called a Transformer Neural Network (TNN) in their paper "Attention Is All You Need" where, as the title suggests,

is ”solely based on attention mechanisms” and removes issues stemming from recurrence and convolutions. Their new architecture was shown to be simultaneously quicker to train and more accurate than some of the ”best models from literature” [23, p.1]. TNNs attempt to process as much data in parallel as opposed to sequentially. This means that Central Processing Units (CPU) and, more relevantly, Graphics Processing Units (GPU) can more efficiently process data as more tasks can be executed simultaneously [19] [23, p.1]. In terms of modelling long-distant dependencies, TNNs massively improve upon CNNs again with a maximum path length of $O(1)$. In comparison to CNNs, whose efficiency is $O(n/k)$, as the sequence length n increases, TNNs get increasingly more efficient in comparison to CNNs. [23, p.6].

2.3.5 Advantages and Disadvantages of Machine Translation

In comparison to human translation, Machine Translation has both its advantages and its drawbacks. In favour of Machine Translation, is that it is far faster than human translation. This becomes particularly evident in the translation of larger documents as, what can be done in minutes or even seconds by a machine, would take humans hours or even days to complete. As a result, Machine Translation proves to be more cost-effective than human translation, making it a more financially viable option for translation needs of everyone allowing for the freedom of small businesses and ordinary people to communicate. The versatility of Machine Translation allows for the translation of a source text into multiple languages, as opposed to humans who tend to specialise in a few languages. The accessibility and user-friendly nature of Machine Translation, paired with its ability to deliver reasonable results, when compared to human translation, makes it the more popular choice for most applications. Warren Weaver argued that Machine Translation was a necessary advancement in technology, viewing it as pivotal for the ”constructive and peaceful future of the planet” [11, p.195].

However, Machine Translation is not without its limitations. Machine Translation often falls short in considering contextual and cultural nuances during the translation process due to the fundamental nature that machines are unable to understand. This causes target texts to have low fluency, as the approach often involves a mechanical, word-for-word translation. As a result, it is widely viewed that Machine Translations are of lower quality when compared to those produced by expert human translators. Machine Translation also was developed on the back of the internet with its seemingly limitless amount of content. However, this is only true for widely used languages with high internet usage, such as English or Russian. For languages with a smaller population, where Machine Translation seems more vital due to the lack of expert

translation, Machine Translation falls short due to the lack of parallel corpora. Furthermore, handling idiomatic expressions poses a challenge as they require specific training on each idiom for the machine to correctly interpret them and translate effectively.

2.3.6 Evaluation of Machine Translation Architectures

Progress in NMT development has experienced exponential growth with early standards of Machine Translation, such as RBMT and SMT, lasted decades whereas the dawn of machine learning has brought along lots of competing network architectures competing to find architectures which work best. Some architectures, such as CNNs, were not developed for usage in translation but have still been found superior to RNNs. So as the field of machine learning becomes more well-understood and researched, the accessibility and the abilities of NMT models and architectures will only grow as researchers invent novel architectures and adapt them for efficient NMT. TNNs are still very novel, having only come around within the last five years within the field of NMT, and widespread knowledge and usage among smaller projects such as mine is still rather sparse. However, their advantages over CNNs and RNNs is clearly apparent, as shown by Google's "Attention Is All You Need" paper published in 2017 [23].

2.4 Machine Translation Tools

In this section, I will describe the range of tools and assets than can and have been used to develop NMT models, specifically for English-Russian translation.

2.4.1 English-Russian Datasets

WMT Datasets

The Workshop on Machine Translation (WMT) started in 2006 as a competition between universities, research facilities, and large technology companies to "exchange and share their experiences and research results" in order to develop Machine Translation. Since 2016, WMT has transformed into an annual Conference on Machine Translation [26]. The organisation still maintains the annual translation tasks, although the tasks have become more varied with fields such as code-mixed Machine Translation, where natural language is combined with code, and low-resource supervised Machine Translation appearing in the 7th Conference in WMT22 [27].

Between 2014 and 2019, the WMT Shared Translation Task included Russian-English as one of the possible pairs of languages to train and test models with. Each workshop and conference

included multiple corpora with incremental additions each year to them, adding more texts each time to the training and validation datasets. For my project, I will focus on the WMT14, which took place in 2014 and was ninth workshop by WMT. Below, are the following corpora included in the WMT14 Shared Translation Task for Russian-English, along with their associated sizes:

News Commentary Parallel Corpus

	French ↔ English		German ↔ English		Czech ↔ English		Russian ↔ English	
Sentences	183,251		201,288		146,549		165,602	
Words	5,688,656	4,659,619	5,105,101	5,046,157	3,288,645	3,590,287	4,153,847	4,339,974
Distinct words	72,863	62,673	150,760	65,520	139,477	55,547	151,101	60,801

Common Crawl Parallel Corpus

	French ↔ English		German ↔ English		Czech ↔ English		Russian ↔ English	
Sentences	3,244,152		2,399,123		161,838		878,386	
Words	91,328,790	81,096,306	54,575,405	58,870,638	3,529,783	3,927,378	21,018,793	21,535,122
Distinct words	889,291	859,017	1,640,835	823,480	210,170	128,212	764,203	432,062

Figure 2.3: An overview of the News Commentary and Common Crawl corpora [28, p.14]

Yandex 1M Parallel Corpus

	Russian ↔ English	
Sentences	1,000,000	
Words	24,121,459	26,107,293
Distinct words	701,809	387,646

Wiki Headlines Parallel Corpus

	Russian ↔ English		Hindi ↔ English	
Sentences	514,859		32,863	
Words	1,191,474	1,230,644	141,042	70,075
Distinct words	282,989	251,328	25,678	26,989

Figure 2.4: An overview of the Yandex 1M and Wiki Headlines corpora [28, p.14]

Within the dataset, there are four main corpora which contain parallel English and Russian text. These can be seen in the figure above. The 'News Commentary Parallel Corpus' contains texts from various news providers, combining a mix of sources where the original texts were written in English and in Russian [28, p.15]. The 'Common Crawl Parallel Corpus' uses data from the Common Crawl organisation, who have built an open repository containing petabytes of data. This allows for websites with multiple translations to be paired together and parallel corpora to be built [29]. Yandex is a Russian-based technology company best known for their search engine and occupies a 61% share of search traffic in Russia, as of 2022, and also operate in Belarus, Kazakhstan, and Turkey [30]. The 'Yandex 1M Parallel Corpus' contains 1 million parallel texts randomly selected from internet resources between 2011 and 2013 [31]. The 'Wiki Headlines Parallel Corpus' uses parallel articles found on Wikipedia and uses the correlating article headlines to produces a parallel set.

OPUS-100

In the 2020 paper, 'Improving Massively Multilingual Neural Machine Translation and Zero-Shot Translation', Zhang et al. developed on Tiedemann's OPUS dataset, introduced in 2012, with a new multilingual dataset called OPUS-100 [32, p.1]. The original OPUS dataset contains parallel corpora for over 90 languages and over 40 billions tokens, including English-Russian corpora [33, p.2214]. OPUS-100 improves on the original dataset by providing parallel corpora covering 100 languages with 55 million sentence pairs. Each language pair uses English as a source or target language, with each language pair contains up to 1 million training pairs sourced from Tiedemann's OPUS dataset. This may cause issues in non-English translation pairs, such as Catalan-Kazakh. This is due to potential lost fidelity caused by the translation process translation from Catalan to English to Kazakh, for instance. However, as my translation model focuses on English-Russian translation, I do not foresee this to be an issue [32, p.1-4].

Issues with Datasets

Issues do arise from this when taking a more scrutinous look at datasets. For instance, there are many occasions of poor or incorrect translations between the languages. Gathering large amounts of training data is tedious work, with millions of parallel training pairs needed in order produce a model with sufficient language coverage, in most use cases. Doing this by hand is simply too time-consuming and expensive, due to the costs of hiring enough language experts to translate sentence pairs. In most cases, web crawling is used to gather such data. This is a process by which bots, most often deployed by search engines, download and analyse web pages in order to extract metadata and index them [34]. This allows for websites with inbuilt parallel web pages to be extracted and automatically converted into parallel texts, taking advantage of already completed translations. Their simplicity and ease of creation makes them an attractive choice for many dataset collators. However, issues arise when said web pages do not translate correctly. For instance, the below sentence pair can be found in the WMT14 training set:

```
{
    "en": "This is a one time charge and you will never be rebilled! You will
           receive direct access to a registration code automatically after
           you place your order."
    "ru": "This LingvoSoft Talking English Russian Dictionary,
           built to operate on mobile devices running the Pocket PC platform,
```

```
            incorporates advanced speech technology that will make your mobile
            device speak English!"
}
```

This is an incorrect translation with it being an English-English sentence pair with no shared semantic meaning. Combined, the 'Yandex 1M' and the 'Common Crawl' parallel corpora form just over 73% of the WMT14 English-Russian training data, and plenty more example of poor fidelity translations can be found. It is hoped, however, that, if provided enough information, then any errors within the dataset will be generalised out over the course of training over such a quantity of training data. As the OPUS and its resulting OPUS-100 datasets are also obtained from crawling, they mirror issues found in the WMT14 dataset with similar errors. Take, for example, the below sentence pair:

```
{
    "en": "Posted: 24 Sep 2007, 18:42"
    "ru": "Posted: 26 Oct 2011, 05:44"
}
```

This sentence pair is not only an English-English sentence pair, but also contains incorrect translations of dates and times. This may cause issues whenever a model, trained on the OPUS-100 dataset, encounters one of the numbers, it may decide to use the pairing below, causing an incorrect translation.

2.4.2 Machine Translation Metrics

BLEU Score

In 2002, IBM researchers proposed a novel metric for Machine Translation that is automatic, language-independent, and provides numerical evaluation which is quick and inexpensive to run to enable easy comparison of translation quality. They named this metric the Bilingual Evaluation Understudy metric, referred to as the BLEU metric. One of the core aims of the metric is to be able to correlate human judgement, and their assessment of the translation qualities such as fidelity and fluency, with a human-comprehensible score [35, p.311]. The following equations are used to describe the BLEU score:

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n \in C} Count_{clip}(n)}{\sum_{C' \in \{Candidates\}} \sum_{n' \in C'} Count(n')}$$

An n-gram is a sub-unit of a sentence made up of n contiguous words from the sentence. For instance, the 3-grams, known as tri-grams, of the sentence "Phil Foden scored twice on Sunday" are "Phil Foden scored", "Foden scored twice", "scored twice on", and "twice on Sunday". The above equation is used to calculate the modified n-gram precision, p_n, for a sentence. Normally, n-gram precision calculates the fraction of n-grams, contained in the candidate sentence, which appear in the references. However, if a candidate sentence were to repeat a word found in a reference sentence, then it would achieve a precision score of 1. For example, the candidate sentence "Haaland Haaland Haaland Haaland Haaland" would achieve a perfect 6/6 unigram, or 1-gram, precision score against the sentence, "Erling Haaland scored a third". To counteract this, the modified n-gram precision equation performs a clipping function to consider the maximum amount of times that an n-gram is allowed to appear in a reference sentence [35, p.312-313].

$$BP = \left\{ \begin{array}{rl} 1 & \text{if} \quad c > r \\ e^{(1-r/c)} & \text{if} \quad c \leq r \end{array} \right\}$$

As lengthy candidate sentences are penalised by the modified n-gram precision score, the sentence brevity penalty, BP, equation is used to penalise any candidates who are shorter than the references, which would allow them to get an undesired high score from using a few keywords from the references. This brevity penalty, is set to 1 if the candidate sentence length, c, is longer than the reference sentence length, r. Otherwise, it is set to a "decaying exponential" proportional lengths of c and r. As to not punish shorter sentences, where a missing word would disproportionately harm the brevity penalty in comparison to a longer sentence, the brevity penalty is calculated over a whole corpus to get a more generalised penalty for the model [35, p.315].

$$GM = \sum_{n=1}^{N} w_n \cdot \log p_n$$

The Geometric Mean, GM, equation is used to combine the modified precision scores of a varying n-gram sizes. It is important to consider different lengths of n-gram as they tend to favour different qualities of translation, with uni-grams favouring references with a higher fidelity whilst longer n-grams tend to favour higher fluency. Normally, the upper bound on n-grams considered, N, is set to 4. To show this n-gram upper bound, the BLEU score unit is often written as BLEU-N to allow for easy comparison and a unmarked BLEU score refers to BLEU-4. In order to produce a unified score, a weight, w_n, is applied to the p_n before

summation. This allows for different n-grams to have differing importance within the final score, depending on the desired translation attributes. However, this is normally set to $\frac{1}{N}$ for all w_n.

$$BLEU = BP \cdot e^{(GM)}$$

Finally, the BLEU score is calculated as shown above to produce a decimal number between 0 and 1 to measure "translation closeness". This score is often multiplied by 100 in literature and, as such, I shall be referring to BLEU scores in the same manner [35, p.312-315].

BERT Score

In response to the growing nature of machine learning usage for natural language generation in 2019, researchers from Cornell University devised a new metric to rival IBM's BLEU score, named the BERT score. They noted that the BLEU score performed a very surface level analysis on the candidate-reference sets due to the Blue's score effective aim, which is to simply calculate n-gram overlap. The aim of the BERT score is to more accurately take into account preservation of semantic meaning in order to prevent paraphrased sentences with similar semantic meaning from being punished. Consider the following reference sentence, "the citizens are revolting against the crown". The BLEU score would give a higher score to the candidate sentence 1 (c_1) "the citizens are visiting the crown jewels" over candidate sentence 2 (c_2) "the people are rioting against the royalty". As a human, we would attribute c_2 as being a higher fidelity translation due to the higher preservation of semantic meaning, even if there is a higher count of n-gram overlaps in c_1 [36, p.1].

To prepare the sentences for scoring, the candidate sentence, c, and the reference sentence, r, are tokenised to the corresponding sets of $\langle c_1, ..., c_p \rangle$ and $\langle r_1, ..., r_q \rangle$, with each c_i and r_i representing a token of the sentence. These are then mapped via the pre-trained BERT embedding model to produce a set of vectors $\langle \hat{c}_1, ..., \hat{c}_m \rangle$ and $\langle \hat{r}_1, ..., \hat{r}_n \rangle$.

$$R_{BERT} = \frac{1}{|r|} \cdot \sum_{r_i \in r} \max_{c_j \in c} \hat{r}_i^\top \hat{c}_j \qquad P_{BERT} = \frac{1}{|c|} \cdot \sum_{c_j \in c} \max_{r_i \in r} \hat{r}_i^\top \hat{c}_j$$

Above are the equations for the recall score, R_{BERT}, and the precision score, P_{BERT}. In each equation, the $\hat{r}_i^\top \hat{c}_j$ term produces the cosine similarity between the reference token, r_i, and the candidate token, c_j [36, p.4]. Whilst the BLEU score does use a modified precision score, the team at IBM did consider and reject the idea of including a recall factor. The recall factor is used to score how well the candidate translation "recalls" words from the reference

sentences. However, a potential issue is that a candidate could over-recall words, meaning that the candidate has "recalled" many words from various references. A "naïve recall" score would give it a higher score for doing so over another higher-fidelity candidate. For example, consider the following candidate sentences, c_i, and the reference sentences, r_i:

```
c1:  "Rodri accurately passed the ball"
c2:  "Rodri accurately, exactly, flawlessly passed the ball"
r1:  "Rodri accurately passed the ball"
r2:  "Rodri exactly passed the ball"
r3:  "Rodri flawlessly passed the ball"
```

Its clear that c_1 is a better translation than c_2 as it exactly matches r_2. However, c_2 has recalled more words by recalling the adverbs from all three reference sentences [35, p.315]. This is why the IBM team opted against using a recall score as the modified n-gram precision score would capture the similarity information whilst crediting sentences of similar lengths. However, as a pair, the precision and recall scores work to credit sentences of similar lengths whilst independently capturing information about recall and precision.

$$F_{BERT} = 2 \cdot \frac{P_{BERT} \cdot R_{BERT}}{P_{BERT} + R_{BERT}}$$

To make comparison between BERT scores easier, the F1-score is calculated. Whilst the precision and recall are good measures of their independent translation attributes, the F1-score provides a reliable, single figure for easy comparison against various models. Similarly to the BLEU score, the BERT score is a decimal number between 0 and 1 and, for lexical equivalence to the BLEU score, I shall also be multiplying the score by 100 in future references with the BERT score referring to the F_{BERT} score specifically.

ChrF Score

Another metric used is the ChrF score, introduced in a 2015 paper by Maja Popovic. It draws similarities from the BLEU and BERT score calculations by considering both n-grams and F scores. However, speciality of the ChrF score is found in its n-gram calculations. Instead of using tokenised words to perform n-gram overlaps, the ChrF score looks at character n-grams in order to gain a more human-like understanding of a text [37, p.329]. For example, consider the following candidate sentence "Ederson is protecting the goal" and the reference sentence "Ederson protects the goal". Whilst the candidate is written in the present continuous case and

the reference in the present simple case, it is clear that both sentences are in the present tense and strongly share a semantic meaning [38]. A word-based n-gram score, such as the BLEU score, would not be able to match "protects" with "protecting" as they are not the same word. However, due to them both sharing the root word "protect", a character-based n-gram score would score such as sentence higher and more accurately align with human evaluation. Below, is the formula for the ChrF score:

$$F\beta_{CHR} = (1 + \beta^2) \cdot \frac{P_{CHR} \cdot R_{CHR}}{\beta^2 \cdot P_{CHR} + R_{CHR}}$$

The equation follows the simple formula for calculating the F-score, with P_{CHR} and R_{CHR} representing the precision and recall scores respectively. These precision and recall scores are calculated as the percentage of overlaps of character n-grams from the candidate sentence in the reference sentence, in the case of P_{CHR}, or the reference sentence in the candidate sentence, in the case of R_{CHR}. A parameter, β, is used to calculate a parametered $F\beta$-score. For instance, if β is set to 1, then the F1-score is calculated. This parametered F-score allows the user to adjust the balance between precision and recall, where the recall score is considered β times more important than the precision score. Popovic found that $F3_{CHR}$ performed the best out of all the β variants, outperforming other metrics, such as the BLEU score, around 75% of the time [37, p.329-330]. As the ChrF score also produces an F-score between 0 and 1, similar to the BERT score, I shall also be multiplying the score by 100 in future references for simplified comparison between the three metrics.

2.5 Existing Machine Translation Models

2.5.1 Multilingual Translation Models

In this section, I will be discussing a few existing multilingual translation models found in literature and their comparative accuracies.

OPUS-MT

In 2020, in a fight against "digital language death", where smaller languages become extinct online due to the difficulty of conversing with non-speakers and a lack of support for their languages by web-hosters, Tiedemann started the OPUS-MT project in collaboration with the Wikimedia Foundation. The focus of the project was to make automatically translating between

languages easier by providing open source models for anyone to implement, using "transparent and secure" methods for easy accessibility. The OPUS-MT project has support for over 500 languages covered by over 1000 pre-trained one-way models. The focus was to develop models for minority languages, however models were also built for more well-established languages, such as German or Russian. The project also has support for both bilingual and multilingual models, with multilingual models having a more widespread use case for easier implementation whilst the bilingual models place a greater emphasis on accuracy [39, p.479].

Model	**Test**	**BLEU**	**ChrF**
OPUS-MT-ru-en	WMT14 News Test	31.9	59.1
OPUS-MT-en-ru	WMT12 News Test	31.1	58.1
OPUS-MT-en-ru	WMT13 News Test	23.5	51.3
OPUS-MT-en-ru	WMT15 News Test	27.5	56.4

Figure 2.5: Reported evaluation data for the OPUS-MT models [40] [41]

As with many recent machine translation models, the OPUS-MT models uses a Seq2Seq encoder-decoder system based on standard TNN architecture. The encoder and decoder layers contain 6 self-attentive layers and uses the **multi-head attention mechanism** by incorporating 8 attention heads per layer [39].

SeamlessM4T-v2

Given the widespread nature fictional depictions of universal translation computers, the realisation for the need for such a device seems clear, given the diverse nature of multilingual conversations both online and offline. This lead to Meta's development of their SeamlessM4T model in 2023 with hopes to building towards this idealistic, multilingual model which can translate in real time. The SeamlessM4T model is a multilingual, multi-modal model and is able, as such, to perform Text to Speech Translation (T2ST), Speech to Text Translation (S2TT), Text to Text Translation (T2TT), and Speech to Speech Translation (S2ST) with T2TT covering translations between English to 95 languages and 95 languages to English [42, p.4].

The SeamlessM4T model is based on FAIRSEQ2, which is Meta's implementation of a Seq2Seq model using a Transformer Neural Network architecture. This base architecture is very similar to most recent implementations for machine translation [42, p.106].

The following figure shows the system architecture of the SeamlessM4T model and how its able to handle multi-modal inputs and convert them into the correct mode of output. The SeamlessM4T model has two separate encoders, one for text and one for speech. Similarly

to the text encoder, the speech encoder parses, tokenises, and converts its input into a set of vectors. A single decoder is used for both inputs to produce a text output which than can be returned for X2TT (unspecified input to text translation) or passed through to the Text-to-Unit decoder and the Vocoder in order to convert this text into speech [42, p.28].

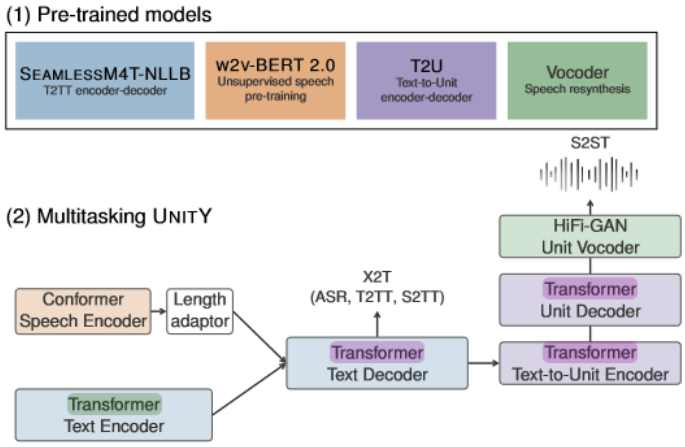

Figure 2.6: An overview of the SeamlessM4T model [42, p.28]

Building off the original SeamlessM4T model, SeamlessM4T-v2 was introduced as a basis for developing real-time translation models. Version 2 includes a new Text-to-Unit for conversion of a translation into speech. The new version has been trained on more data, with special focus being placed on low-resource languages for better coverage. Additionally, SeamlessM4T-v2 has support for all languages pairs within its supported set of 96 languages [43, p.12].

Model	**Test**	**BLEU**
SeamlessM4T-Large v2 xx-en	Fleurs	30.17
SeamlessM4T-Large v2 en-xx	Fleurs	26.05

Figure 2.7: Reported evaluation data for the SeamlessM4T-Large v2 models on Fleurs [43, p.117]

Model	**Test**	**ChrF**
SeamlessM4T-Large v2 xx-en	Flores	60.8
SeamlessM4T-Large v2 en-xx	Flores	50.9

Figure 2.8: Reported evaluation data for the SeamlessM4T-Large v2 models on a T2TT dataset [43, p.39]

An important note here is the datasets and tests performed in the above evaluations. The first table shows tests performed on the Fleurs dataset, which is one of the largest S2TT benchmark datasets in the field. In opposition to a T2TT dataset, Fleurs contains parallel pairs of speech in the source language with their corresponding texts in the target language.

Flores, on the other hand, is a multilingual T2TT dataset and is considered to be one of the main benchmarks for low-resource languages.

MADLAD-400

Instead of focusing on producing a translational model, Google's MADLAD-400 project intends to solve the problem of many widely used web-crawled datasets containing a lot of "undesirable content" and noise. To solve this, they curated their own Multilingual And Document-level Large Audited Dataset (MADLAD) and trained three models of varying size to verify their results, with the final models being able to translate between 419 languages [44, p.1].

To filter the datasets, they took the web-crawled data and filtered out any "questionable content" using a set of heuristics to score and filter out any sentence scoring over 20%. The heuristics took into account mismatched language identifiers, abnormal character counts, abnormally high counts of technical characters, and whether they contained any cursed regexes or substrings. The cursed regexes and substrings help to remove any pairs containing unwanted known words or phrases, with the Chinese dataset requiring its own separate filter given its high levels of pornographic content contained. All of these factors can hamper translational quality and their removal can simultaneously prevent this and also decrease training time. There were also other filters for more specific cases such as the inclusion of the word "JavaScript" or braces ("{", "}"), indicating that unwanted code was included in the dataset, and the inclusion of the "Lorem ipsum..." text, which is often used as filler in many websites. Finally, a self-audit was conducted to assure quality and approve the languages for training. 19 languages were deemed unfit datasets via this process and were omitted from the final models [44, p.3].

Model	**Test**	**BLEU**	**ChrF**
MT-3B-ru-en	WMT Test Data	39.9	64.4
MT-7.2B-ru-en	WMT Test Data	40.7	64.9
MT-10.7B-ru-en	WMT Test Data	40.5	64.9
MT-3B-en-ru	WMT Test Data	32.9	57.8
MT-7.2B-en-ru	WMT Test Data	33.6	58.4
MT-10.7B-en-ru	WMT Test Data	33.6	58.3

Figure 2.9: Reported evaluation data for the MADLAD-400 models of varying sizes [44, p.35]

The final three models include a 3 billion parameter, 32-layer model; a 7.2 billion parameter, 48-layer model; and a 10.7 billion parameter, 32-layer model with their reported evaluated results between the two relevant language pairs shown in the table above [44].

2.5.2 English-Russian Translation Models

In this section, I will be discussing a few existing translation models found in literature built specifically for English-Russian translation. Both of these models focus on submissions to the WMT News Translation Tasks.

Facebook FAIR's WMT19 News Translation Task Submission

In 2019, Facebook's AI Research team, known as FAIR, sought to improve on their submission to the previous year's WMT18 entry by building off of it. They built two models each for English-Russian and English-German translation. Their base system was built using FAIRSEQ, which is discussed in the SeamlessM4T model, for Seq2Seq encoding-decoding and a BPE-based TNN model [45, p.1]. Byte Pair Encoding, or BPE, is an algorithm which is able to perform subword tokenisation, where words are split up into units called subword tokens. These subword tokens can then be used to represent frequently used subwords in order to compress texts. For instance, consider the following words:

```
Press, Cress, Rest, Fresh, Resin
```

These words all share the subword "res" and, thus, the subword "res" could be tokenised as token number 27. Given that the other letters are tokenised as the their position in the alphabet, you would gain the following tokenisation:

```
{9,27,19}, {3,27,19}, {27,20}, {6,27,8}, {27,9,14}
```

The above tokenisation gives us 14 tokens where as a tokenisation system based on the position of the character in the alphabet, such as ASCII encodings, would result in 24 tokens. BPE was originally developed for text compression but has found itself usage in Machine Translation tasks to make model training more efficient by compressing vocabularies and its flexible handling of new, unknown words [46]. In addition to the base WMT18 entry, the WMT19 submission also uses further techniques to improve the model, including back-translation, data filtering, and fine-tuning on domain-specific data. The reported results for the English to Russian and Russian to English models can be seen below.

Model	**Test**	**BLEU**
Facebook-FAIR ru-en	WMT19 News test	40.0
Facebook-FAIR en-ru	WMT19 News test	36.3

Figure 2.10: Reported evaluation data for Facebook-FAIR's models [45, p.318]

Ariel Xv's WMT20 News Translation Task Submission

Ariel Xv, a student of Beihang University in Beijing, China, also produced a submission for the WMT20 news translation task. They produced a bidirectional English-Russian translation system comprised of two models, one for each direction of translation [47]. The models use the Sockeye toolkit, which is an open-source Seq2Seq framework for machine translation written in Python and was developed by Amazon's research team. Sockeye provides many advantages for training models. It has support for a variety of languages and is optimised for training on multiple GPUs, allowing for quicker training times via parallelisation. Sockeye is also modular, allowing for many components to be swapped and selected. For instance, Sockeye supports a variety of encoders and decoders as well as various methods of optimisation. This customised is useful in facilitating further research into different neural machine translation models via its quick training and deployment methods [48].

Ariel Xv's model also researches various machine translation techniques, such as data filtering, back-translation, reordering, and ensemble models. To train the model, data was taken from various WMT corpora consisting of 10 million monolingual texts, for back-translation, and 102 million parallel texts [47]. The reported results for the English to Russian and Russian to English models can be seen below.

Model	**Test**	**BLEU**
Ariel-Xv ru-en	WMT20 News test	39.1
Ariel-Xv en-ru	WMT20 News test	24.8

Figure 2.11: Reported evaluation data for Ariel Xv's models [47, p.324]

Requirements and Specification

The purpose of this project is to create a translation model called PETRAI (Proficient English To Russian AI) which can allow users to bidirectionally translate between English and Russian. This will be able to be accessed via a simple user interface.

3.1 Requirements

The below requirements state how the system should operate from the point of view of a user or stakeholder and how they expect such an application to function.

3.1.1 User Interface Requirements

- U1 - Users should be able to bidirectionally translate text between English and Russian.
- U2 - Users should be able to type using the English Latin alphabet when using English input.
- U3 - Users should be able to type using the Russian Cyrillic alphabet when using Russian input.
- U4 - Users should be able to translate the inputted text.
- U5 - The translation of the inputted text should occur quickly after submission.
- U6 - The UI should then display the translation.
- U7 - The UI should be accessible to a range of users.
- U8 - The UI should be usable on a range of devices.

3.1.2 Translator Requirements

- T1 - The English to Russian translation should be relatively accurate.

- T2 - The Russian to English translation should be relatively accurate.
- T3 - The translation model should translate the input text quickly.

3.2 Specification

In this section, the above requirements are considered and gives a specification on how to they should be implemented in technical terms. They are also associated with a priority level which indicates how important they are to the project. Higher priority specifications should be implemented first in order to offer the basic functionality to the project whereas lower priority specifications offer less critical, but still useful functionality to the project

3.2.1 User Interface Specifications

Code	Details	Specification	Priority
U1	Users should be able to bidirectionally translate text between English and Russian.	The UI displays two options for input text: English and Russian. The user is able to select the source language before typing in their input.	High
U2	Users should be able to type using the English Latin alphabet when using English input.	The input type of the English input uses the Unicode UTF-8 encoding and accepts it as input.	High
U3	Users should be able to type using the Russian Cyrillic alphabet when using Russian input.	The input type of the Russian input uses the Unicode UTF-8 encoding and accepts it as input.	High
U4	Users should be able to translate the inputted text.	The user will then be able to submit their input text for translation using the 'Enter' key.	High
U5	The translation of the inputted text should occur quickly after submission.	The two models for English to Russian and Russian to English will need to first be initialised before translation to allow for quick translation of the given input.	Medium

Code	Details	Specification	Priority
U6	The UI should then display the translation.	The UI will take the output from the translation model and display it for the user to see.	High
U7	The UI should be accessible to a range of users.	The UI should be simple to use and require little technical expertise to operate.	Low
U8	The UI should be usable on a range of devices.	The UI will be accessible via a python terminal interface and will be available to those on both UNIX and Windows based machines.	Low

3.2.2 Translator Specifications

Code	Details	Specification	Priority
T1	The English to Russian translation should be relatively accurate.	The English to Russian model will be scored using a combination of the BLEU, BERT, and ChrF score in order to evaluate its ability.	High
T2	The Russian to English translation should be relatively accurate.	The Russian to English model will be scored using a combination of the BLEU, BERT, and ChrF score in order to evaluate its ability.	High
T3	The translation model should translate the input text quickly.	As the model is pre-trained, the resulting translation algorithm will be relatively quick as it will be querying from pre-calculated data.	Medium

Design

In the the Design section, I will lay out the technologies that will be used for the project along with the design of the final program, considering the advantages and disadvantages of each approach used.

4.1 Technologies

In this project, I will be using a range of technologies in order to develop the translation models and to develop a user interface where the users can interact with the model.

4.1.1 Machine Learning

In order to translate between English and Russian, I will need to develop two distinct models, one for each direction of translation. To complete this task, I have three different options of possible paths to consider: building and training a model from scratch, fine-tuning existing Seq2Seq models for translation, or fine-tuning an established translation model to increase accuracy.

Self-developed Model

As of researching this topic, the current state-of-the-art architecture are Transformer Neural Networks (TNN). With its increased efficiency in training and its ability to process data in parallel, it makes a suitable option for my model. As the goal of my model is to take a sequence of characters and produce a corresponding sequence of characters, I will naturally be using a Seq2Seq encoder-decoder. Due to their relative advantages, this TNN-Seq2Seq pairing is a common one among most recent models in literature. For instance, Meta's SeamlessM4T model uses a TNN architecture for its model and FAIRSEQ for text generation, which is their implementation of a Seq2Seq encoder-decoder [42].

To do this, there are two libraries, concerned with the development of new machine learning models: PyTorch and TensorFlow. Similarly to Python, PyTorch is a more user-friendly and well-documented when compared to its alternative TensorFlow, which tends to be more technical but is available for usage in a variety of programming languages. For instance, PyTorch provides a simple, easy-to-follow guide on creating a simple German to English, Seq2Seq transformer-based model in [49].

Fine-tuning Generic Seq2Seq Models

Another option is to use Hugging Face's API to load in a generic Seq2Seq model and train this to perform a translations from English to Russian, and vice-versa. One potential candidate model is Meta's BART model. Whilst the model is not able to translate between languages out-of-the-box, it provides a pre-trained encoder-decoder for improved translation accuracy in a shorter training period [50]. For its pre-training, the model is passed a corrupted sequence of tokens, with the decoder being passed the original sequence. Using this, the encoder-decoder pair can learn to reconstruct sentences by calculating and forming the relevant contextual embeddings [51]. This pre-training makes it useful for a wide range of different tasks, from text classification to text generation. Of interest is the text generation ability, as we aim to generate a new sentence in the target language from our source sentence.

Using a model, such of the likes of BART, offers several advantages over a self-developed model. Using a model with pre-trained encoder-decoders offers a well-established base for a model to be trained off of, providing both flexibility in the languages that I am able to train the model on as well as performance and efficiency benefits. As it is pre-trained and built for generalisation, the model is able to perform better, following appropriate training, in a variety of different tasks. It is then able to handle new data better, than a self-developed model initially could. This ability to perform many tasks could be of hindrance due to a lack of control over the original training data causing potential issues in being able to adapt to a new domain which, in our case, is English-Russian translation. As this is a third-party model, a level of trust must be placed that the training methods and the training data are of ethical origins. Despite these potential concerns, I am confident that this method will provide promising results.

Fine-tuning Established Translation Models

Another method of consideration for our translation models, is to take a pre-trained translation model for English-Russian and fine-tune it. Hugging Face's APIs make it simple to import any

model and train it further. This could be especially useful and promising for improving said models to cover a wider range of language, vocabulary, and domains. One example could be training a model such as SeamlessM4T-v2 or MADLAD-400 to translate slang terms in both English and Russian, especially as slang terms are by definition both niche and novel.

However, there are potential issues with this method. One being that training a model with an established model with satisfactory metrics may cause said scores to go down if the new training data containing errors overwrites or overcomes existing embeddings. This means that any new training data must be well-reviewed for correctness before training. A more pertinent issue revolves about my goal for this project. If I wanted to produce the highest scoring set of models, fine-tuning an established English-Russian model would result in the best yielding results. Given the nature of translational models for many languages, such an initial model is hard to find or poorly trained. Therefore, it would be more relevant to the aims of the project to train the models myself. On the other hand, building a model from scratch would require a lot of computation resources, especially temporal, as such a model would not have any pre-training to work from. Therefore, I aim to strike a balance between effort and high-scoring metrics by using BART as a base model to build two translator models, one from English to Russian and the other vice-versa.

Other Training Technologies

I will also be using a variety of tools and technologies to train the model. One of the most common programming languages used for machine learning is Python. Despite issues surrounding its inefficient usage of memory and its slower speeds in comparison to a language like C++, Python's user-friendly nature and easy-to-read syntax has made it a staple of analytics and data science applications. Due to Python's resulting high support for data handling and analysis libraries, Python has also become a key programming language for machine learning too [52]. These libraries are often implemented in Rust or C++, despite being provided for usage in Python.

One such website, which supports machine learning development, is Hugging Face. Hugging Face provides APIs which are used to standardise and collate various models from literature and from the general machine learning community in order to create a more democratic way of accessing said models for usage. Their APIs allow for simple training, evaluating, and usage of models and, for these reason chosen for this project.

As the project revolves around the accessibility of translation and the ability for anyone to train an AI model, I will be using my own personal laptop to complete the training process. The laptop contains a 13th Gen Intel® Core™ i9-13900H CPU, 32GB of DDR5 RAM in a 2x16GB configuration, and an NVIDIA® GeForce™ RTX 4080 Mobile GPU with 12GB of dedicated GPU memory. Whilst this does represent a powerful laptop, it does strike a good balance between computing power and accessibility in order to develop a pair of satisfactory models

4.1.2 User Interface

For the User Interface (UI), I had two main options: a simple, terminal-based UI and a website- or application-based Graphical User Interface (GUI).

Simple UI

For simplicity, I have chosen to create a simple UI using the terminal and the in-built `print()` functions found within Python. I chose the terminal-based UI due to its simplicity of implementation and due to the scope of the project. Whilst a clean, user-friendly UI or GUI is a good quality for a translation system, I decided that the project aims of developing an accessible model which provides interfaces which are easy to implement. As a result, the focus should be on developing the model without worrying about the outer shell of a GUI. The following result will also provide an easy and accessible way to extend the project to include any desired UI, if a possible extension to the project was to be done.

4.2 System Architecture

The below diagram explains the control flow of the PETRAI translator system.

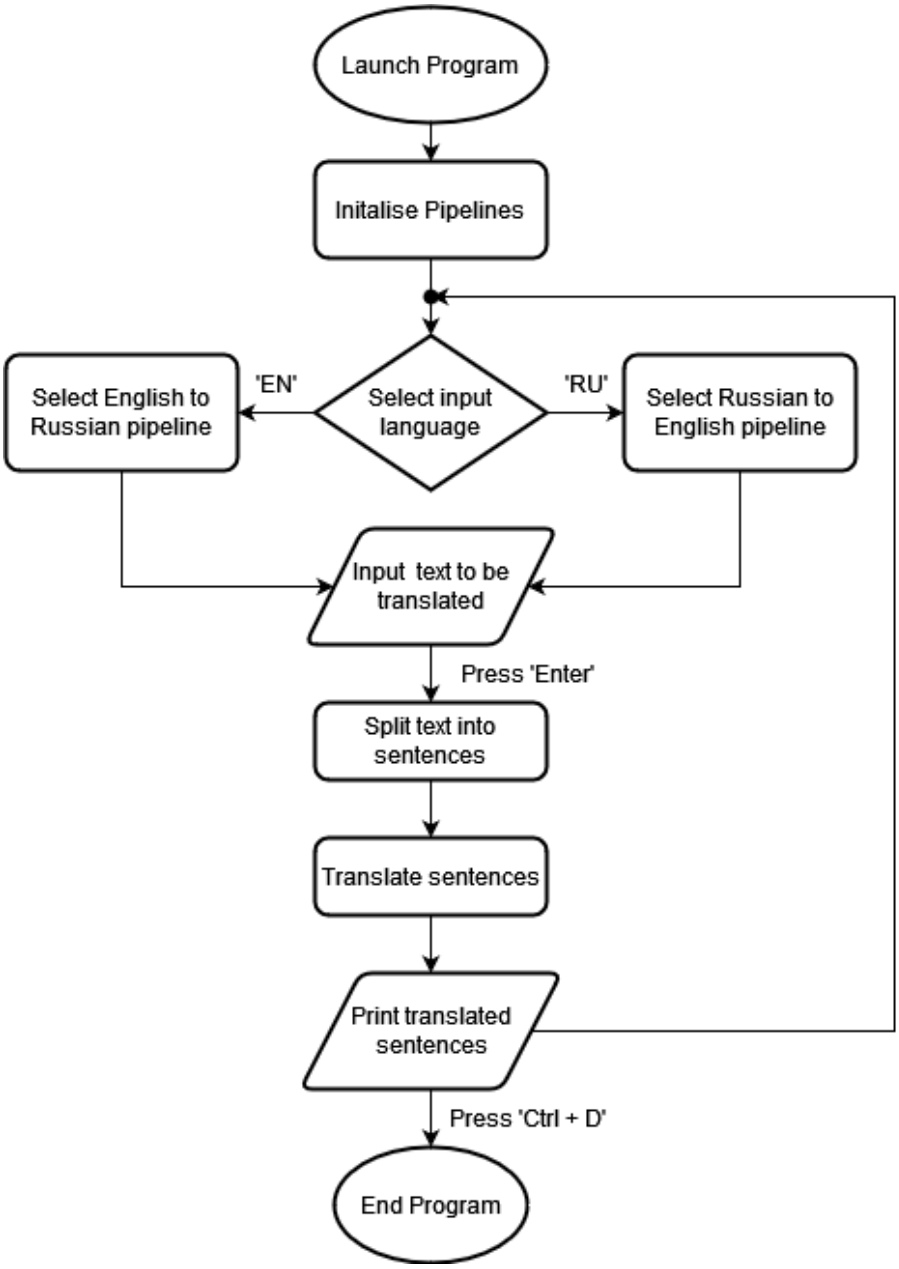

Figure 4.1: Control Flow for the PETRAI translator program

The small program will first initialise the translator before giving the user the option to select the source language of the input, with the target language being assumed to be the alternate option. This is because PETRAI is limited to English to Russian or Russian to English translation with two models being used for the translation, one model for each direction

of translation. The user will then press the 'Enter' key, in line with the specifications, in order to translate the input text. This process of selecting a language and translating the input text is then repeated until the user exits the program.

In order to translate any given input, it should be expected that a user may want to translate a text. Due to limitations on most Seq2Seq models with the amount of tokens they can read and generate, I will split the text up by sentences in order to ease the workload on the translator. The two models are also initialised in the form of a pipeline. Pipelines provides an easy way to complete a multitude of tasks, including translation. The initialisation of the pipelines does take time to complete and, therefore, will be completed at the start of the program, rather than before each translation, in order to save time.

4.3 Limitations

Unfortunately, there are some limitations that can be for seen in this project.

Spelling Mistakes

Due to the nature of the model, the model assumes that the input is accurate and therefore will try to translate the input, no matter what has been inputted. Simple recognition could be done to ensure the correct alphabet has been used for input, such as Russian Cyrillic over English Latin, but the user will not be indicated of any spelling errors.

Knowledge of all Words

As a result of limited input, the model will not be able to recognise all words, especially in the case of names where names are numerous and unique or rare spellings are used. The fix for this issue will be to directly transcribe the word letter by letter or by letter pairs. This is often done by translators for unusual names or for many city names.

Computing Capability

Even though my laptop is relatively powerful, its computing capabilities is still rather limited in comparison to more mainstream translators, like Google Translate. The training time will also be longer as I must produce two satisfiable models, one for English to Russian and one for Russian to English. Additionally, my method is still relatively limited as finding and verifying larger amounts of training data becomes increasingly more difficult and the training time and

computing power needed to train this data also increases. However, I am still confident on my ability to produce two satisfiable models for this project.

Cultural Understanding

One fundamental issue with machine translation is the inability for computers to understand. Machine learning in its simplest form is about finding connections within and between large sets of data. However, the computer at no point is able to actual understand the data and is an ongoing problem in AI research and development. As explained in the section about 'Human Translation', cultural understanding and being able to use context of a given text, such as time period and emotion, is important to being able to properly achieve high fidelity in a given translation.

Implementation

In the implementation section, I will describe the workings of the PETRAI system, including how I trained the PETRAI models along with the datasets used in this training process. I will also discuss the final implementation of the translator program and how the PETRAI models perform the translation process.

Please note that the following scripts listed in the implementation have been adapted from the source code to improve readability. However, the core functionality of the code has been maintained. For instance, logging print statements have been removed and long lines of code spread over multiple lines.

5.1 Machine Translation

For the actual translation element of the project, this can be split into two parts: the training of the models and the usage of these models for translation.

5.1.1 Training

For the training scripts, I followed guides from Hugging Face for building a one-way translation model found in [53] and [54]. I then adapted these to fit the goal of translating between English and Russian, including the usage of the OPUS-100 dataset.

Datasets Used

After evaluating the different English-Russian datasets earlier on in the report, I decided that I would use the OPUS-100 dataset for the training of my models. Both datasets do present issues in terms of quality with both datasets containing many issues often found with web crawling. In this project, I have decided upon the usage of the OPUS-100 dataset due to its proven nature of translation with Tiedemann's OPUS-MT models and its ideals of the OPUS project being of an open-source nature, designed for a more open internet. In terms of choosing OPUS-100

over the original OPUS dataset, I chose the OPUS-100 dataset due to resource limitations, especially temporal.

```
from datasets import load_dataset, DatasetDict, Dataset

iteration = 1
iter_perc = iteration/3
from_perc, to_perc = (iter_perc-1/3), (iter_perc)
src_lang = "en"
tgt_lang = "ru"
model_name = "petrai_" + src_lang +"-"+ tgt_lang + "_bart_opus100-" + iteration

train_dataset = load_dataset("opus100", "en-ru")
training_len = len(train_dataset['train'])
from_index, to_index = int(training_len*from_perc), int(training_len*to_perc)
train_dataset =
    DatasetDict({
        'train': Dataset.from_dict(train_dataset['train'][from_index:to_index]),
        'test':  train_dataset['test']
    })

checkpoint = "facebook/bart-large"
             #petrai_src-tgt_bart_opus100-n used for iterations, n > 1
```

petrai_train.py

Here, the `load_dataset` API function from Hugging Face makes it easy to download the English-Russian dataset from the OPUS-100 repository on Hugging Face, for usage in training. Using the `from_perc` and `to_perc` variables, I was able to train three iterations of each model, each training on a third of the OPUS-100 dataset using the last iteration or the original BART model as its base model, seen in the `checkpoint` variable. This was done as each iteration took between 10-15 hours to complete and the system was required for other tasks.

In comparison with each other, the OPUS-100 dataset contains exactly 1 million sentence pairs whilst the WMT14 dataset contains just over 1.68 million sentence pairs. Whilst the are training times are not exactly proportionate to the number of sentence pairs, they do give a good heuristic for training times. Therefore, training two models with three iterations, each taking 10-15 hours, results in a training time would have, in the worst case, taken over 150 hours total of training for the WMT14 dataset. Even more significantly, the current OPUS

dataset for English-Russian contains over 390 million sentence pairs [55], spread over various corpora. Using my current configuration, training of such a dataset would be simple infeasible.

Training Preparation

The following is the dataset preparation section of the model training script. As previously mentioned, accreditation goes to Hugging Face's translation guides, especially for the pre- and post-processing text functions [53] [54].

```
from transformers import BartTokenizer, DataCollatorForSeq2Seq

tokenizer = BartTokenizer.from_pretrained(checkpoint)
prefix = "translate " + src_lang + " to " + tgt_lang + ": "

def preprocess_function(examples):
    inputs =
        [prefix + example[src_lang] for example in examples["translation"]]
    refs = [example[tgt_lang] for example in examples["translation"]]
    model_inputs = tokenizer(inputs, text_target=refs, max_length=128,
    truncation=True)
    return model_inputs

def postprocess_text(preds, refs):
    preds = [pred.strip() for pred in preds]
    refs = [[ref.strip()] for ref in refs]
    return preds, refs

tokenized_dataset = training_dataset.map(preprocess_function, batched=True)
data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=checkpoint)
```

petrai_train.py

In this project, parallel datasets are made up of a tuples containing inputs in the source language and references in the target language, with each reference being a translation of the input language. When calculating metrics, a third label is added to the tuple, labelled predictions. This contains the model's predicted translation of the input in the target language. As such, this is how the three labels will be referred to throughout the code and project.

The text pre-processor takes every sentence in the source language and appends it with a prefix. This prefix is useful for indicating to the model about the positioning of a given token, and whether it is at the start of a sentence or not. The pre-processing function then passes

the prefixed inputs through the tokeniser. The base model chosen for this project was Meta's BART model and is used as the initial checkpoint for the training of both the English-Russian and Russian-English translation models. One of the advantages of the BART model was its pre-trained tokeniser and encoder-decoder, and therefore the `BartTokeniser` has been used here to tokenise the inputs.

Finally, the `DataCollatorForSeq2Seq` is used in Seq2Seq encoder-decoder systems to inform the model of the details of the tokenisation process, passing along the tokeniser and the relevant model. In the training process, the `data_collator` it pads inputs so that the encoder-decoder can work on a specified length of input and output each and every time. Attention masks are used to apply various levels of priority on different tokens, see the **Attention Mechanism** section for more details on this process. Here, attention masks are used to prevent the model from creating any embeddings linked to the padding tokens [56].

Use of Metrics in Training

The following is the metrics calculation section of the model training script. As previously mentioned, the `compute_metrics()` function has been adapted from Hugging Face's translation guides [53] [54].

```
import numpy
import evaluate

bleu = evaluate.load("sacrebleu")

def compute_metrics(eval_preds):
    preds, refs = eval_preds
    if isinstance(preds, tuple):
        preds = preds[0]
    decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)

    refs = numpy.where(refs != -100, refs, tokenizer.pad_token_id)
    decoded_refs = tokenizer.batch_decode(refs, skip_special_tokens=True)

    decoded_preds, decoded_refs = postprocess_text(decoded_preds, decoded_refs)

    bleu_score = bleu.compute(predictions=decoded_preds,references=decoded_refs)
    metrics = {"bleu": bleu_score["score"]}

    prediction_lens =
```

```
        [numpy.count_nonzero(pred != tokenizer.pad_token_id) for pred in preds]
    metrics["gen_len"] = numpy.mean(prediction_lens)

    metrics = {metric: round(val, 4) for metric, val in metrics.items()}
    return metrics
```

petrai_train.py

During the training of the model, the `compute_metrics()` function is called and is passed in a tuple of references and predictions. The decoder is used the output of the model's predictions from a list of tokenised symbols into (hopefully) comprehensible text. The `skip_special_tokens` parameter ensures that the tokeniser does not attempt to decode any special tokens, such as padding tokens, as this would cause issues with the inclusion of unexpected extra text. The this function also calculates the generation length of each tokenised prediction, discounting any padding tokens, and averages it over the dataset to produce a `gen_len` metric. The BLEU score is also calculated via the sacreBLEU API, provided by Hugging Face. These metrics are useful for debugging and assessing the performance of the model and track any variation as it trains.

SacreBLEU

Hugging Face also provide an API which enables easy calculation of metrics via its `evaluate` module, allowing users to quickly and easily select and swap different methods of evaluating metrics. In this script, sacreBLEU is used to calculate a BLEU score. SacreBLEU was developed by Matt Post of Amazon Research as a way to easily produce "shareable, comparable, and reproducible BLEU scores" [57] as a response to variation of up to 1.8 BLEU often found with other methods, caused by many issues such as different tokenisation methods [58]. SacreBLEU's ease of use and reproducibility is a key factor in my decision to use this metric, as it prevents any randomness within evaluation and, thus, allows for confidence in comparison between models. Another reason for its usage, is that it provides a corpus-wide score. As the BLEU score was designed to score on a sentence level, the BLEU score over an entire corpus of size n, would result in n individual BLEU scores. SacreBLEU, on the other hand, produces a single score, allowing for a streamlined and more meaningful analysis between models.

Training Setup

The following code describes the training arguments and the configuration used for the training process. The `per_device_train_batch_size` and `per_device_eval_batch_size` were chosen

to maximise GPU memory usage and, thus, speed of training without causing the system to crash due to unavailable memory.

```
from transformers import AutoModelForSeq2SeqLM, Seq2SeqTrainingArguments,
    Seq2SeqTrainer

model = AutoModelForSeq2SeqLM.from_pretrained(checkpoint)
modelLocation = "./translator_models/"

training_args = Seq2SeqTrainingArguments(
    output_dir=model_name + "_checkpoints",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=20,
    per_device_eval_batch_size=20,
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=4,
    predict_with_generate=True,
    fp16=True,
    push_to_hub=False,
)

trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

trainer.train()
trainer.save_model(modelLocation+model_name)
```

petrai_train.py

5.1.2 Translation

In this section, I will cover the translation segment of the model and how it converts text from the source to the target language. As with the previous sections, the code listed below is an

adapted version of the final source code or an adapted examples from their referenced sources. This is done for readability, whilst maintaining the core functionality of the original code.

Pipelines

Another benefit of using the Hugging Face ecosystem is the ease of translation. Hugging Face provides a `pipeline` API which allows the user a unified way to choose one of a range of tasks and the option to specify a model to complete the task. The stated task is "translation" with the model selected being either one of the models created in the **Training** section of the implementation. Below is an example of how the pipeline can be used to translate English text into Russian:

```
from transformers import pipeline

text = "And Sergio Aguero sings champions."
translator = pipeline("translation", model="petrai_en-ru")
translation = translator(text)
```

Example of a translation pipeline, adapted from [53]

In order to get a sense of how the function operates, Hugging Face provides a method to manually recreate the outputs of the pipeline [53]. This is seen below:

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

text =  "And Sergio Aguero sings champions."

tokenizer = AutoTokenizer.from_pretrained("petrai_en-ru")
inputs = tokenizer(text, return_tensors="pt").input_ids

model = AutoModelForSeq2SeqLM.from_pretrained("petrai_en-ru")
outputs = model.generate(inputs, max_new_tokens=40, do_sample=True, top_k=30,
    top_p=0.95)
translation = tokenizer.decode(outputs[0], skip_special_tokens=True)
```

Example of a manual implementation of a translation pipeline, adapted from [53]

The imported `AutoModelForSeq2Seq` and `AutoTokenizer` classes by Hugging Face are used to automatically select correct model and tokeniser depending on the model provided allowing for easy replacement of the model chosen. In this instance, the corresponding BART classes will be chosen as PETRAI was fine-tuned from Meta's original BART model. The tokeniser

then converts the input text into inputs and the model carries out the Seq2Seq task, encoding the input tokens ready for decoding. Finally, the tokeniser takes these encodings and decodes them into human-readable text.

Translator Models

Given the substitutability of pipelines provided by Hugging Face, I have built upon these ideas of Clean Architecture within my own code. The `translator_models.py` is used to provide a unified way of accessing various Hugging Face English-Russian models via the `EnRuTranslatorModel` base class. This class allows for both pipelines for English to Russian and Russian to English translation to be initialised once for each instance of a specific `EnRuTranslatorModel`, as translation models tend to be unidirectional.

```
from transformers import pipeline

class EnRuTranslatorModel:
    def __init__(self):
        self.symbols = ['.', '!', '?', ';', ':']
        self.initialise_translators()

    def get_pipeline(self, src, tgt):
        model_translators_dir = "./translator_models/"
        if (src[:2] == "en" and tgt[:2] == "ru"):
            model_name = self.en_ru_model_name
        elif (src[:2] == "ru" and tgt[:2] == "en"):
            model_name = self.ru_en_model_name
        else:
            raise Exception(" ** TRANSLATOR NOT FOUND ** ")

        pipe = pipeline("translation", src_lang=src, tgt_lang=tgt, model=
    model_translators_dir+model_name, max_length=1000)
        return pipe

    def initialise_translators(self):
        self.en_ru_translator = self.get_pipeline("en", "ru")
        self.ru_en_translator = self.get_pipeline("ru", "en")
```

translator_models.py

As the project is only concerned with English-Russian translation, the `EnRuTranslatorModel` is so too only concerned with English-Russian translation. Future development could see this

model being abstracted into a general translation class for various languages. The initialisation of both pipelines prevents the need to continually reinitialise the translation pipelines upon every translation. This is needed as there is a delay every time a pipeline is initialised.

```
from nltk.tokenize import sent_tokenize
nltk.download('punkt')
#class EnRuTranslatorModel:
    def choose_translator(self, src, tgt):
        if (src[:2] == "en" and tgt[:2] == "ru"):
            return self.en_ru_translator
        elif (src[:2] == "ru" and tgt[:2] == "en"):
            return self.ru_en_translator
        else:
            raise Exception(" ** TRANSLATOR NOT FOUND ** ")

    def split_into_sentences(self, text):
        if isinstance(text, str):
            sentences = sent_tokenize(text)
            return sentences
        else:
            raise Exception(" ** TRANSLATOR CAN ONLY TRANSLATE TEXT ** ")

    def translate_single_sentence(self, sentence, translator):
        return translator(sentence)[0].get('translation_text')

    def translate_text(self, text, translator):
        sentences = self.split_into_sentences(text)
        translated_sentences = [self.translate_single_sentence(sentence,
    translator) for sentence in sentences]
        translated_text = " ".join(translated_sentences)
        return translated_text

    def translate(self, text, src, tgt, log=True):
        translator = self.choose_translator(src.lower(), tgt.lower())
        translated_text = self.translate_text(text, translator)
        return translated_text
```

translator_models.py

Seen here is the base implementation of translation within the `EnRuTranslatorModel` class. The premise of the translation is that the `translate()` method takes in a given text in a stated source language in order to be translated into the stated target language. The text is then split

up into sentences, as the pipeline can only handle a max number of tokens to generate from a given input. Each sentence is then translated via the appropriate pipeline and then stitched back together to form the full translated text. This is then return as the final translation.

To perform the partitioning of the text into sentences, the `sent_tokenise()` function is imported from the NLTK library, as it provides a simple and easy way to perform this. One potential issue with the function is the ambiguous nature of various ending punctuation as the full stop found in the sentence "Dr. Smith" does not indicate the end of a sentence, but rather indicates an abbreviation of the word "doctor". Despite this, I will use this function as, without defining an extensive list of exceptions, this will always remain an issue. Furthermore, sentences can still be adequately translated in the resulting text partitions. The `split_into_sentences()` function also is able to raise an exception if it receives a non-string input, preventing unwanted results from the `sent_tokenise()` function.

```
class Generic(EnRuTranslatorModel):
    def __init__(self, name, model_name):
        self.name = name
        if isinstance(model_name, tuple):
            self.en_ru_model_name = model_name[0]
            self.ru_en_model_name = model_name[1]
        else:
            self.en_ru_model_name = model_name
            self.ru_en_model_name = model_name
        super().__init__()

class PETRAI(EnRuTranslatorModel):
    def __init__(self):
        self.name = "Opus-MT"
        self.en_ru_model_name = "petrai_en-ru_bart_opus100-3"
        self.ru_en_model_name = "petrai_ru-en_bart_opus100-2"
        super().__init__()
    def choose_translator(self, src, tgt):
        if (src[:2] == "en"):
            self.prefix = "translate " + src[:2] + " to " + tgt[:2] + ": "
        else:
            self.prefix = ""
        return super().choose_translator(src, tgt)
    def translate_single_sentence(self, sentence, translator):
        return super().translate_single_sentence(self.prefix+sentence,
    translator)

```

```
class Petrai_2(EnRuTranslatorModel):
    def __init__(self):
        self.name = "PETRAI_2"
        self.en_ru_model_name = "petrai_en-ru_bart_opus100-2"
        self.ru_en_model_name = "petrai_ru-en_bart_opus100-2"
        super().__init__()

class PrefixedPetrai_2(Petrai_2):
    def __init__(self):
        super().__init__()
    def choose_translator(self, src, tgt):
        self.prefix = "translate " + src[:2] + " to " + tgt[:2] + ": "
        return super().choose_translator(src, tgt)
    def translate_single_sentence(self, sentence, translator):
        return super().translate_single_sentence(self.prefix+sentence,
    translator)

class Petrai_3(EnRuTranslatorModel):
    def __init__(self):
        self.name = "PETRAI_3"
        self.en_ru_model_name = "petrai_en-ru_bart_opus100-3"
        self.ru_en_model_name = "petrai_ru-en_bart_opus100-3"
        super().__init__()

class PrefixedPetrai_3(Petrai_3):
    def __init__(self):
        super().__init__()
    def choose_translator(self, src, tgt):
        self.prefix = "translate " + src[:2] + " to " + tgt[:2] + ": "
        return super().choose_translator(src, tgt)
    def translate_single_sentence(self, sentence, translator):
        return super().translate_single_sentence(self.prefix+sentence,
    translator)
```

translator_models.py

In order to function, these classes need to be extended via a child class to define the model location for each pipeline. Above, is an definition of the `Generic` class which allows the user to easily define a new `EnRuTranslatorModel` by providing the model locations for each pipeline, or just the singular location if the model is bidirectional. Also contained in the code above, is the implementation of PETRAI-2 and PETRAI-3, where their names and the model locations

are defined. As they uses the base implementation of translation via pipelines contained within the `EnRuTranslatorModel` class, there is no need for any method overriding.

One point of interest is the existence of two PETRAI models: PETRAI-2 and PETRAI-3. During implementation, I decided to split the training cycles into three segments due to the long training times for a model, given my current configuration. As a result, there are three models, where each iteration provides a base model to the next iteration, with PETRAI-2 being the base model for PETRAI-3. The training set up also provides checkpoints containing evaluation data calculated by the `compute_metrics()` function in `petrai_train.py`. Noticeably, it seemed to be that PETRAI-3 was not as large of an improvement as PETRAI-2, therefore, I decided to test both models in order to evaluate their qualities in order to decide which model to use within my final implementation.

Due to the usage of a prefix in training, another potential point of research was how the inclusion of a prefix would affect the quality of a translation. As a result, the PETRAI-2 and PETRAI-3 models will be scored four times each, twice for the English to Russian variant and twice for the Russian to English variant. Each variant will have two trials each, one with a prefix prepended to each input, and one without. To support this prefix option, each PETRAI model has a child class for the purposes of prefixing. These prefixed classes override the `choose_translator()` function, where the prefix for that translator is defined, and the `translate_single_sentence()` function, which prepends a prefix to the input in preparation for translation.

Finally, we have the implementation for the main PETRAI class, which uses the prefixed version of PETRAI-3 for English to Russian translation and the prefixed version of PETRAI-2 for Russian to English translation. For more information on the reasoning behind this decision, see the **Evaluation Section** for analysis of the PETRAI variants.

5.1.3 Testing

To ensure that the model performs as expected, I also developed a small set of tests to ensure that the PETRAI model was working as intended.

```
def test(name, result):
    if result:
        return True
    else:
        raise Exception(name + " test case has failed")

```

```
def can_split_into_sentences(model, text, expected_result):
    split_text = model.split_into_sentences(text)
    return test("can_split_into_sentences", len(split_text)==expected_result)

def test_can_split_into_sentences_acceptable(model):
    text = "Hello there General Kenobi. I am Ben. What is your name?"
    return can_split_into_sentences(model, text, 3)

def test_can_split_into_sentences_zero_text(model):
    text = ""
    return can_split_into_sentences(model, text, 0)

def test_can_split_into_sentences_lots_of_text(model):
    text = "hi. hi. hi. hi. hi. hi. hi. hi. hi. hi. hi. hi. hi. hi. hi. hi. hi.
    hi. hi. hi."
    return can_split_into_sentences(model, text, 20)

def test_can_split_into_sentences_None(model):
    text = None
    try:
        can_split_into_sentences(model, text, None)
    except (Exception):
        return True
    else:
        return False
```

test_models.py

The above code shows the `test()` function which is used to raise an exception if the result passed in is false, therefore ending the tests as a test case has failed. Additionally, the test cases regarding the ability to split a text into sentences are shown. The following texts test a range of various inputs from the user, including long texts with 20 sentences to tests where an invalid input is passed through. If an input of an invalid type is passed in, then an exception is raised. The function also is shown to handle situations where a string of length zero is passed into the function.

```
def can_translate(model, text, src, tgt, expected_results):
    translation = model.translate(text, src, tgt, log=False)
    return test("can_translate", translation in expected_results)

def test_can_translate_en_to_ru(model):
```

```
    text = "Hello."
    return can_translate(model, text, "en", "ru", ["**RUSSIAN REFERENCES**"])

def test_can_translate_ru_to_en(model):
    text = "**RUSSIAN TEXT**"
    return can_translate(model, text, "ru", "en", ["Hello.", "Hey.", "Hi."])

def test_can_translate_en_to_ru_no_text(model):
    text = ""
    return can_translate(model, text, "en", "ru", "")

def test_can_translate_ru_to_en_no_text(model):
    text = ""
    return can_translate(model, text, "ru", "en", "")

def can_translate_text(model, text, src, tgt):
    translation = model.translate(text, src, tgt, log=False)
    return test("can_translate", isinstance(translation, str))

def test_can_translate_text_en_to_ru(model):
    text = "Hello. I am the policeman. Hands in the air!"
    return can_translate_text(model, text, "en", "ru")

def test_can_translate_text_ru_to_en(model):
    text = "**RUSSIAN TEXT**"
    return can_translate_text(model, text, "ru", "en")
```

test_models.py

The tests above measure whether the `translate()` function works as intended. The first two test functions, `test_can_translate_en_to_ru()` and `test_can_translate_ru_to_en()`, test whether a given input in the source language matches one of the given references stated in the function. This is to allow for acceptable variations in translation. The model also correctly handles the translation of a string of length zero and also is able to translate texts containing multiple sentences, as shown by the last two functions.

```
from translator_models import PETRAI

def test_model(model):
    tests = [
        test_can_split_into_sentences_acceptable(model),
        test_can_split_into_sentences_zero_text(model),
```

```
        test_can_split_into_sentences_lots_of_text(model),
        test_can_split_into_sentences_None(model),
        test_can_translate_en_to_ru(model),
        test_can_translate_en_to_ru_no_text(model),
        test_test_can_translate_ru_to_en_no_text(model),
        test_can_translate_ru_to_en(model),
        test_can_translate_text_en_to_ru(model),
        test_can_translate_text_ru_to_en(model),
    ]
    return False not in tests

def test_models(models):
    for model in models:
        if test_model(model):
            print(model.name + " has passed")

test_models([PETRAI()])
```

test_models.py

The code here shows the tests being carried out for the inputted models. As the models were designed around the `EnRuTranslatorModel` base class, the testing methods are able to function on all of the models within the `translator_models.py` file. Therefore, this testing file has been built with the option to test further models in mind.

5.2 User Interface

The code below shows the implementation of my simple user interface that is displayed in the terminal.

```
from translator_models import PETRAI

seperator = "-----------------------------------------"
print(seperator+"\n             WELCOME TO PETRAI\n\n      Proficient English To
     Russian AI\n"+seperator+"\n")
print("Press Ctrl + D to quit the program\n")

petrai = PETRAI()
langs = ["en", "ru"]

while True:
```

```
    src_lang = ""
    while (src_lang not in langs):
        src_lang = input("choose input language ("+ langs[0] + "/" + langs[1] +
"): ")
        if (src_lang not in langs):
            print('## INCORRECT INPUT ##')
    src_text = input("text to translate: ")
    langs.remove(src_lang)
    tgt_lang = langs[0]
    tgt_text = petrai.translate(src_text, src_lang, tgt_lang)
    print("\noutput: " + tgt_text)
    langs.append(src_lang)
    print("\n"+seperator+"\n")
```

petrai_translate.py

Upon starting the program, the PETRAI model is instantiated, with both the English to Russian and the Russian to English pipelines being instantiated too. The user can then select their input language before typing in their input text. The input is translated and then displayed to the user. This process is repeated until the user exits the program using the 'Ctrl + D' key combination.

Evaluation and Analysis

Please note that, as with previous sections, the following scripts listed in the evaluation have been adapted from the source code to improve readability. However, the core functionality of the code has been maintained.

6.1 Model Evaluation

In order to be able to properly evaluate my models, I have found three models on Hugging Face from established authors. These are the Meta's SeamlessM4T-v2 model, Facebook's (now Meta) FAIR team submission for the WMT19 Translation Task, and Tiedemann's OPUS-MT model. As Tiedemann is part of Helsinki University's Natural Language Processing department, the OPUS-MT mode is often referred to by the University's name, Helsinki, for differentiation between Tiedemann's other creation, the OPUS data set. The model descriptions and reported metrics can be seen in the **Existing Machine Translation Models Section**.

```
from translator_models import PETRAI, Petrai_2, PrefixedPetrai_2, Petrai_3,
    PrefixedPetrai_3, M4T, FAIR, Helsinki
def get_translator_model(model):
    match(model):
        case("petrai"):
            return PETRAI()
        case("petrai_2"):
            return Petrai_2()
        case("prefixed_petrai_2"):
            return PrefixedPetrai_2()
        case("petrai_3"):
            return Petrai_3()
        case("prefixed_petrai_3"):
            return PrefixedPetrai_3()
        case("m4t"):
            return M4T()
```

```
        case("fair"):
            return FAIR()
        case("helsinki"):
            return Helsinki()
```

evaluate_models.py

I have created child classes of the `EnRuTranslatorModel` to implement these models into my code. This helps to make training easy as it follows SOLID principles, which removes the need for the evaluation code to be dependent on the implementation of the translation models and allows for easy substitution of translation models. These implementations can be seen in the code below:

```
from transformers import AutoProcessor, SeamlessM4Tv2Model

class M4T(EnRuTranslatorModel):
    def __init__(self):
        self.name = "M4T"
        self.model_name = "facebook/seamless-m4t-v2-large"
        self.processor = AutoProcessor.from_pretrained(self.model_name)
        self.model = SeamlessM4Tv2Model.from_pretrained(self.model_name)

    def translate_single_sentence(self, sentence, src, tgt):
        text_inputs = self.processor(text=sentence, src_lang=src, return_tensors
    ="pt")
        output_tokens = self.model.generate(**text_inputs, tgt_lang=tgt,
    generate_speech=False)
        translated_text_from_text = self.processor.decode(output_tokens[0].
    tolist()[0], skip_special_tokens=True)
        return translated_text_from_text

    def translate_text(self, text, src, tgt):
        sentences = self.split_into_sentences(text)
        translated_sentences = [self.translate_single_sentence(sentence, src,
    tgt) for sentence in sentences]
        translated_text = " ".join(translated_sentences)
        return translated_text

    def translate(self, text, src, tgt, log=True):
        translated_text = self.translate_text(text, src, tgt)
        return translated_text

```

```
class FAIR(EnRuTranslatorModel):
    def __init__(self):
        self.name = "FAIR"
        self.en_ru_model_name = "facebook/wmt19-en-ru"
        self.ru_en_model_name = "facebook/wmt19-ru-en"
        super().__init__()

class Helsinki(EnRuTranslatorModel):
    def __init__(self):
        self.name = "Opus-MT"
        self.en_ru_model_name = "Helsinki-NLP/opus-mt-en-ru"
        self.ru_en_model_name = "Helsinki-NLP/opus-mt-ru-en"
        super().__init__()
```

translator_models.py

One point of interest in the above code is that the implementation of the internal translation mechanism for the SeamlessM4T-v2 model does override some of the functionality of the parent class, `EnRuTranslatorModel`. This is because the **SeamlessM4T-v2** model is designed to be a multi-modal, multilingual model and therefore uses a single model to complete a variety of tasks. Whilst a pipeline would somewhat function for this purpose, I have noticed some unusual mistakes caused by the implementation provided by the pipeline. Therefore, this implementation for the SeamlessM4T-v2 model uses an adaptation of the code found in its model description on Hugging Face [59] which generally follows the manual implementation of **Pipelines**, as seen earlier. In terms of key differences between the above implementations, the SeamlessM4T-v2 model takes uses an `AutoProcessor` instead of an `AutoTokenizer`, as the . In terms of the FAIR and OPUS-MT models, they follow the **standard implementation** found in the `EnRuTranslatorModel` class as there are separate, dedicated models for English to Russian and Russian to English.

6.1.1 Evaluation Datasets

```
def prepare_test_data(src, tgt, num_of_test_data=None):
    en_test_data_source = "data/newstest2014-ruen-src.en.txt"
    ru_test_data_source = "data/newstest2014-ruen-src.ru.txt"

    with open(en_test_data_source) as en_file:
        with open(ru_test_data_source) as ru_file:
            en_lines = en_file.read().splitlines()
```

```
            ru_lines = ru_file.read().splitlines()

    if src[:2] == "en":
        inputs = en_lines
        unformatted_references = ru_lines
    else:
        inputs = ru_lines
        unformatted_references = en_lines

    predictions = list.copy(inputs)

    formatted_references = []
    for ref in unformatted_references:
        formatted_references.append([ref])

    if num_of_test_data == None:
        num_of_test_data = len(inputs)

    predictions = predictions[:num_of_test_data]
    inputs = inputs[:num_of_test_data]
    collated_references = {
        "formatted_references": formatted_references[:num_of_test_data],
        "unformatted_references": unformatted_references[:num_of_test_data]
    }

    return inputs, predictions, collated_references
```

evaluate_models.py

The `prepare_test_data()` function is a helper function which parses and formats the English-Russian WMT14 News Test, ready for the given model to form its translation predictions. In the **Training Section of the Implementation**, I discussed the usage of the OPUS-100 dataset for training the model and its advantages over the WMT14 and the standard OPUS datasets. However, I decided to use the WMT14 dataset for the evaluation of the models. The main reason for this is its widespread usage in literature for being one of the standards for English-Russian translation evaluation. This allows for more insightful analysis and the subtle differences in quality and difficulty of translation over various datasets makes it more complicated to perform comparative evaluations. Furthermore, it allows me to compare scores, that I have calculated, with scores found in literature for the same models in order to confirm that the models have been accurately ported over to Hugging Face and that the metrics

used are also accurate.

At time of implementation, the WMT14 training dataset has a few errors in its parsing of the original file, notably with its representation of quotation marks being shown as (\"). To solve this, I downloaded the original Standard Generalized Markup (SGM) files from the WMT14 Translation Task description [60] and formatted it for easy parsing. The references come in two parts as well: formatted and unformatted. This is to aid with the differences in evaluation strategies. Scoring metrics, like the BLEU and ChrF score, check each candidate sentence against a set of reference sentences. In our case, the WMT14 test dataset only provides one reference per input and we use the formatted references for this. In comparison, the ChrF metric scores a set of candidates against a set of references so the unformatted references are apt for said usage.

6.1.2 Calculation of Metrics

```
1 def get_corpus_predictions(model, src, tgt, num_of_test_data=None):
2     inputs, predictions, references = prepare_test_data(src, tgt,
      num_of_test_data)
3     predictions, timings = translate_texts(predictions, src, tgt, model, True)
4     return inputs, predictions, references, timings
```

evaluate_models.py

For the calculation of the metrics, both speed and accuracy metrics use the `get_corpus_predictions()` function to calculate a stated amount of texts from the parallel testing corpus from the given source language into the given target language. The function returns the inputs and references for the test corpus with the model's predictions for the inputs as well as the timings for the model, including how long it takes to initialise the model and how long it takes to translate the given number of texts.

Accuracy

As stated previously, to effectively evaluate my PETRAI models, I have imported and implemented three models, the OPUS-MT models from Helsinki University, the WMT19 Translation Task submission by Facebook's (now Meta) FAIR team, and Meta's SeamlessM4T-v2 model. Their implementations can be seen in the **Model Evaluation Section**.

```
1 from datetime import datetime
2 import csv
3 predictionsLocation = "./predictions/"
```

```
scoresLocation = "./scores/"
models = {
    "petrai_2":          {"notes": "** INSERT NOTES **"},
    "prefixed_petrai_2": {"notes": "** INSERT NOTES **"},
    "petrai_3":          {"notes": "** INSERT NOTES **"},
    "prefixed_petrai_3": {"notes": "** INSERT NOTES **"},
    "m4t":               {"notes": "** INSERT NOTES **"},
    "fair":              {"notes": "** INSERT NOTES **"},
    "helsinki":          {"notes": "** INSERT NOTES **"}
}
def evaluate_models():
    model_scores = {}
    for model_key in models.keys():
        lang_pairs = {"eng": "rus", "rus": "eng"}
        for src in lang_pairs:
            tgt = lang_pairs.get(src)
            model = model_key + "_" + src + "-" + tgt
            tested_models = ["** INSERT TESTED MODELS HERE **"]
            if model not in tested_models:
                inputs, predictions, references, timings =
    get_corpus_predictions(model_key, src, tgt)
                with open(predictionsLocation+model+"_predictions.csv", 'w',
    newline='') as predictionsFile:
                    writer = csv.writer(predictionsFile)
                    writer.writerow(["inputs","predictions","references"])
                    refs = references.get("unformatted_references")
                    for i in range(0,len(predictions)):
                        writer.writerow([inputs[i],predictions[i],refs[i]])

                model_scores[model] = score_model(predictions, references, tgt)
                with open(scoresLocation+"model_scores.csv", "a") as
    modelScoresFile:
                    writer = csv.writer(modelScoresFile)
                    scores = model_scores[model]
                    writer.writerow([model, scores["bleu"], scores["bert"],
    scores["chrf"], datetime.now(),models[model_key]["notes"]])
    return model_scores

print(test_model_speed())
```

evaluate_models.py

The `evaluate_models()` function is used to go through all of the stated models and evaluate them based on their BLEU, BERT, and ChrF scores on both their English to Russian and Russian to English variants. For each variant of each model, the function produces a Comma-separated values (CSV) file, containing the inputs, predictions, and the references produced by the `get_corpus_predictions()` function. This was done in order to debug any errors that may arise and retest the dataset, if required. The function also appends the BLEU score (formed of the score itself and its associated brevity penalty), the BERT score (formed of the F1, precision, and recall score), and its ChrF score to the `model_scores` CSV file, for ease of data collection and comparison of models.

```
from torchmetrics.text import CHRFScore
from bert_score import BERTScorer
from statistics import mean
import evaluate

def extract_tensor_value(t):
    if len(t.shape) > 0:
        return [val.item() for val in t]
    else:
        return t.item()

def get_bleu_score(translations, references):
    bleu = evaluate.load("sacrebleu")
    bleu_score = bleu.compute(predictions=translations, references=references)
    return {"score": bleu_score.get("score"), "bp": bleu_score.get("bp")}

def get_bert_score(translations, references, tgt):
    bert = BERTScorer(model_type='bert-base-multilingual-cased')
    p,r,f = bert.score(translations, references)
    bert_score = {
        "f1": extract_tensor_value(f),
        "precision": extract_tensor_value(p),
        "recall": extract_tensor_value(r)
    }
    average_bertscore = {
        "f1": mean(bert_score.get("f1")),
        "precision": mean(bert_score.get("precision")),
        "recall": mean(bert_score.get("recall"))
    }
    return average_bertscore
```

```
31
32 def get_chrf_score(translations, references):
33     chrf = CHRFScore()
34     return extract_tensor_value(chrf(translations, references))
35
36 def score_model(translations, references, tgt):
37     model_scores = {}
38     model_scores["bleu"] = get_bleu_score(translations, references.get("
       formatted_references"))
39     model_scores["bert"] = get_bert_score(translations, references.get("
       unformatted_references"), tgt)
40     model_scores["chrf"] = get_chrf_score(translations, references.get("
       formatted_references"))
41     return model_scores
```

evaluate_models.py

To evaluate the models, I have chosen three key metrics: the BLEU score, the BERT score, and the ChrF score. For more information on how they function, see the **Machine Translation Metrics Section**. In the case of the BLEU and ChrF score, their widespread usage across literature, acting as the de facto method of scoring and evaluating models. Both BLEU and ChrF scores provide their own benefits given their differences, mainly the difference between token-based versus character-based n-grams. Therefore, they together provide a satisfiable evaluation of the model. In addition, The BERT score is a more interesting metric to consider. Despite its lack of usage in literature, the BERT score uses pre-trained contextual embeddings to provide greater insight into the quality of a translation, allowing for translations using linguistically correct reorderings or close synonyms to be given a higher quality score than of its BLEU or ChrF score. As a result, I believe its a worthy inclusion amongst the evaluation metrics.

In terms of their implementation, I have used various sources for the different metrics. As discussed earlier, I have also used **sacreBLEU** API from Hugging Face to measure the BLEU score of each model. In addition, I have also imported modules to handle the calculations of the BERT score [61] and the ChrF score [62]. As these functions return `Tensor` values, the `extract_tensor_value()` function is used to provide a simple method to extract a value from a tensor.

Speed

Despite accuracy being a vital factor of any given translator, human or machine, another key metric to take into account is how fast the translator is as its one of the main reasons that most translation is done via Machine Translation, and not by human translators. As a result, I have chosen to include this as one of the metrics for each model in the evaluation.

```
import time

def translate_texts(texts, src, tgt, model, is_logging=False):
    timings = {}

    start_time = time.time()
    translator = get_translator_model(model)
    end_time = time.time()-start_time
    timings["load_translator"] = end_time

    translated_texts = []
    start_time = time.time()
    for i in range(0,len(texts)):
        translated_texts.append(translator.translate(texts[i], src, tgt, False))
    end_time = time.time()-start_time
    timings["translate_dataset"] = end_time

    return translated_texts, timings
```

evaluate_models.py

The `translate_texts()` function is the main method in the evaluation scripts which interacts with each model in order to translate a set of given texts. The translator instance is first initialised before translating the set of texts one-by-one, which are then returned for scoring. Here we can see two key areas, where speed could be an issue: the time it takes to initialise the translator instance and the time taken to translate a given text. These are both factors that will affect the user's experience when using the translator in the User Interface.

```
from datetime import datetime
import csv
scoresLocation = "./scores/"
models = {
    "petrai_2":          {"notes": "** INSERT NOTES **"},
    "prefixed_petrai_2": {"notes": "** INSERT NOTES **"},
    "petrai_3":          {"notes": "** INSERT NOTES **"},
```

```
    "prefixed_petrai_3": {"notes": "** INSERT NOTES **"},
    "m4t":               {"notes": "** INSERT NOTES **"},
    "fair":              {"notes": "** INSERT NOTES **"},
    "helsinki":          {"notes": "** INSERT NOTES **"}
}
def test_model_speed():
    model_timings = {}
    for model_key in models.keys():
        lang_pairs = {"eng": "rus", "rus": "eng"}
        for src in lang_pairs:
            tgt = lang_pairs.get(src)
            model = model_key + "_" + src + "-" + tgt
            tested_models = []
            if model not in tested_models:
                sample_num = 10
                translations_num = 50
                mean_timings = {}
                translator_timings = []
                translation_timings = []
                for i in range(0, sample_num):
                    inputs, predictions, references, timings =
    get_corpus_predictions(model_key, src, tgt, models.get(model_key)["prefix"],
     translations_num)
                    translator_timings.append(timings["load_translator"])
                    translation_timings.append(timings["translate_dataset"])
                mean_timings["load_translator"] = mean(translator_timings)
                mean_timings["translate_dataset"] = mean(translation_timings)
                model_timings[model] = mean_timings
                with open(scoresLocation+"model_times.csv", "a") as
    modelTimesFile:
                    writer = csv.writer(modelTimesFile)
                    timings = model_timings[model]
                    seconds_per_text = timings["translate_dataset"] /
    translations_num
                    writer.writerow([model, timings["load_translator"],timings["
    translate_dataset"], translations_num, seconds_per_text, sample_num,
    datetime.now()])
    return model_timings

print(test_model_speed())
```

evaluate_models.py

The `test_model_speed()` function uses the timings provided by `translate_texts()` to provide a score for each model variant, with the variants being English to Russian and Russian to English. Whilst the time taken to translate a text was highlighted as a key metric in terms of speed of a model, randomly selecting a sentence from the WMT14 News Test corpus and timing how long it takes to translate it is unlikely to provide an accurate measure for an average use case. To mitigate this, each model variant is tasked with translating 50 texts from the test corpus, with each model variant being handed the same 50 texts for apt comparison.

A further potential issue is computing constraints. Due to variations in memory usage, power availability, and temperature, among other things, it is possible that a given result may be slower or different to the average. To mitigate this, 10 samples are taken for each model variant of both the loading of the translator instance and the translating of the 50 texts. These samples are then averaged via the usage of the mean function, before being saved to the CSV file. In addition, the final mean seconds per text metric is calculated using the mean total translation time and the number of translation performed. This gives us the average time taken to translate a text, as desired.

6.1.3 Accuracy

How to Interpret the Scores

In order to better compare the accuracy of the models, it is best to understand the scoring systems that we are using to score said models. In terms of the BLEU score, Google propose a recommended guideline for interpreting the BLEU scores.

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality, often better than human

Figure 6.1: Recommended guideline for interpretation of BLEU scores [63]

As we will see later, it is very difficult to achieve high BLEU scores on most testing datasets. This is because scorers, like the BLEU and ChrF scorers, offer multiple references to be passed in, whereas most testing corpora only include a single reference per input. BLEU and ChrF scorers encourage multiple references as a way of compensating for the lack of creative freedom

provided by machines, in terms of translations, as both of these scoring systems take the references to be them to be the only suitable translations for the input.

As a result, both BLEU and ChrF could be considered metrics measuring sentence similarity rather than translation quality, as this quality is expected as part of a reference. The danger of this is that aiming to improve a BLEU or ChrF score is, in fact, aiming to provide a more similar sentence to the references, not necessarily a better quality translation. BERT provides more freedom, as it uses pretrained contextual embeddings to try acknowledge the issue of multiple translations. Despite this, the BLEU scores act as a sufficient, human-comprehensible metric for easy evaluation and comparison against other models. The BERT scores and ChrF are more vague, in terms of a direct, linear correlation between a score and an interpretation. However, they are still satisfiable metrics to form as part of our analysis.

Accuracy Metrics Data

The following data is a combination of data found in literature and data collected myself, with the implementation of this referenced and explained in the Calculation of Metrics Section. For evaluation purposes, I have evaluated three different model pairs from literature against two versions of my self-developed model PETRAI. Each version of PETRAI has also been split into two further variants: one which prefixes each translation with the prefix that it was trained on, and one which translates the text as inputted. As a result, there are fourteen individual variants which have been tested: seven for English to Russian, and seven for Russian to English. The metrics included are the BLEU scores (including its score and brevity penalty), the BERT score (including the F1 score, the precision score, and its recall score), and the ChrF score.

Models (Russian to English)	**BLEU (Score/BP)**	**BERT (F1/P/R)**	**ChrF**
petrai_ru-en_bart_opus100-2 (w/o)	21.949 / 1.000	84.587 / 84.452 / 84.694	48.188
petrai_ru-en_bart_opus100-2 (pref)	21.948 / 1.000	84.596 / 84.590 / 84.642	47.826
petrai_ru-en_bart_opus100-3 (w/o)	21.246 / 0.990	84.270 / 84.470 / 84.117	46.061
petrai_ru-en_bart_opus100-3 (pref)	20.970 / 0.966	84.274 / 84.606 / 83.992	45.410
Helsinki-NLP/opus-mt-ru-en	31.794 / 1.000	88.638 / 88.774 / 88.534	57.125
facebook/wmt19-ru-en	42.194 / 0.984	91.149 / 91.333 / 90.930	64.483
facebook/seamless-m4t-v2-large	40.283 / 1.000	90.538 / 90.715 / 90.388	62.974

Figure 6.2: Metrics collected on Russian to English models from the WMT14 News Test

The following tables contain data found in literature for Russian to English and multilingual translation models. I have also included data on the models which I have evaluated, as a way to check my own evaluation techniques against established literature. There is also recent data,

gathered by Jungha Son and Boyoung Kim, containing information about the accuracies of many large, well-established translators, such as Microsoft Translator and Google Translate, as well as the two newest iterations of OpenAI's ChatGPT-3.5 and ChatGPT-4 [64].

Models (Russian to English)	**Test Dataset**	**BLEU**	**ChrF**
OPUS-MT-ru-en	WMT14 News Test	31.9	59.1
Facebook-FAIR ru-en	WMT19 News Test	40.0	—
SeamlessM4T-Large v2 ru-en	Fleurs	30.17	—
SeamlessM4T-Large v2 xx-en	Flores	—	60.8
MADLAD MT-3B-ru-en	WMT Test Data	39.9	64.4
MADLAD MT-7.2B-ru-en	WMT Test Data	40.7	64.9
MADLAD MT-10.7B-ru-en	WMT Test Data	40.5	64.9
Ariel-Xv ru-en	WMT20 News Test	39.1	—
ChatGPT-3.5 ru-en [64]	WMT18/20 Test Data	32.9	63.7
ChatGPT-4 ru-en [64]	WMT18/20 Test Data	34.7	63.6
Google Translate ru-en [64]	WMT18/20 Test Data	36.4	65.8
Microsoft Translator ru-en [64]	WMT18/20 Test Data	36.5	64.8
AVERAGE		**36.6**	**63.6**

Figure 6.3: Metrics found on Russian to English models from literature, see **Existing Machine Translation Models** and [64].

Similarly to the Russian to English models, the following tables contains metric data on their English to Russian counterparts.

Models (English to Russian)	**BLEU (Score/BP)**	**BERT (F1/P/R)**	**ChrF**
petrai_en-ru_bart_opus100-2 (w/o)	16.049 / 0.728	82.725 / 84.207 / 81.403	39.152
petrai_en-ru_bart_opus100-2 (pref)	16.512 / 0.718	83.000 / 84.536 / 81.631	39.830
petrai_en-ru_bart_opus100-3 (w/o)	16.511 / 0.726	82.820 / 84.327 / 81.474	39.502
petrai_en-ru_bart_opus100-3 (pref)	16.661 / 0.713	83.027 / 84.551 / 81.667	39.950
Helsinki-NLP/opus-mt-en-ru	32.550 / 1.000	88.252 / 88.319 / 88.225	58.048
facebook/wmt19-en-ru	43.136 / 1.000	90.392 / 90.595 / 90.219	64.467
facebook/seamless-m4t-v2-large	40.630 / 1.000	90.397 / 90.596 / 90.225	63.888

Figure 6.4: Metrics collected on English to Russian models from the WMT14 News Test

Reported versus Tested Scores

Whilst their is some standardisation in scoring, with the BLEU and ChrF scores being the methods of choice in most recent literature on the topic, standardisation in the field of testing seems to be still rather limited. WMT is the most popular choice in evaluating models, given its widespread usage due to their annual competitions to try and solve different translation tasks in various manners. This ease of accessibility makes it a good choice to test a model on, with its resources being free and open for anyone to download on their website. One

Models (English to Russian)	Test Dataset	BLEU	ChrF
OPUS-MT-en-ru	WMT12 News Test	31.1	58.1
OPUS-MT-en-ru	WMT13 News Test	23.5	51.3
OPUS-MT-en-ru	WMT15 News Test	27.5	56.4
Facebook-FAIR en-ru	WMT19 News Test	36.3	—
SeamlessM4T-Large v2 en-ru	Fleurs	26.05	—
SeamlessM4T-Large v2 en-xx	Flores	—	50.9
MADLAD MT-3B-en-ru	WMT Test Data	32.9	57.8
MADLAD MT-7.2B-en-ru	WMT Test Data	33.6	58.4
MADLAD MT-10.7B-en-ru	WMT Test Data	33.6	58.3
Ariel-Xv en-ru	WMT20 News Test	24.8	—
ChatGPT-3.5 en-ru [64]	WMT18/20 Test Data	25.8	58.2
ChatGPT-4 en-ru [64]	WMT18/20 Test Data	26.8	58.5
Google Translate en-ru [64]	WMT18/20 Test Data	33.4	63.2
Microsoft Translator en-ru [64]	WMT18/20 Test Data	31.6	62.7
AVERAGE		**29.8**	**57.6**

Figure 6.5: Metrics found on English to Russian models from literature, see **Existing Machine Translation Models** and [64].

issue comes with the various editions of the conference. As the years have gone by, the WMT organisation have issued new datasets for various, ever-changing language sets. With this, each new edition takes the previous year's dataset and appends it with new data. One of the reasons for choosing WMT14, was that it is one of the early and yet well-established versions of the WMT Translation Tasks involving English-Russian datasets. However, as can be seen with the above tables, the reported scores found in literature come from a range of different WMT editions ranging from 2013 to 2020, where some models have been testing using a mix of editions with some evaluations not stating which editions they are using.

To counteract this, I decided to evaluate the models using my own standardised methods of scoring and datasets used in order to properly evaluate my models against those from literature. One point of interest lies within the OPUS-MT models. Whilst the OPUS-MT-en-ru model, developed by Tiedemann of Helsinki University, had not been evaluated against the WMT14 dataset, the Russian to English variant has been with the BLEU score being just over 0.1 BLEU out from my own evaluation. In terms of its ChrF score, their evaluation was only around 2 ChrF out from my ChrF score. These scores prove my evaluation methods to be roughly in line with Tiedemann's and, thus, literature's scoring methods. For the English to Russian model, a similarity in score is harder to gather due to the mismatching datasets, although there is a score difference of about 1 between their WMT12 and my WMT14 evaluation.

For Meta's SeamlessM4T-v2 model, both the variants saw an increase in BLEU score when tested using my configuration, with the English to Multilingual variant seeing an increase of

around 10 BLEU and the Multilingual to English variant seeing an increase of over 14 BLEU. One key disclaimer is that the SeamlessM4T-v2 model is intended to be a multilingual modal, with generic multilingual models being used to translate from and to English. Despite this, their BLEU scores of 40.6 and 40.3 show the model's strength at translation, exhibiting a "high quality translations" from Google's BLEU interpretation guide [63]. Another issues surrounds their mismatching datasets. Whilst the WMT dataset is one of the key text-to-text translation benchmarks, the Fleurs dataset, developed by Google, is considered one of the largest speech-to-text translation benchmarks in its field. Whilst the input modes vary between speech and text, both methods of translation use the same decoder in order to produce an output text. However, there is an extra level of difficulty of parsing and tokenising speech, which can often include a lot of noise resulting from interference or even from accents and, thus, contains less clarity than its equivalent text. As a result, this extra level of difficulty provides reasoning behind the differences in evaluation scores. The report on the SeamlessM4T-v2 model does provide an alternate text-to-text dataset in the form of Flores, which is another renowned text-to-text test dataset developed by Meta. In this regard, I tested the English to Multilingual and Multilingual to English models to be approximately 13 ChrF and 2 ChrF better off in my evaluation than reported. The large gap in differences is, again, to do with its Multilingual targets. The reported tests are done on a variety of languages of varying resource-levels and, thus, translation qualities. Therefore, the averages of the score are likely to return a lower score than one of a higher resource language, such as Russian and English.

Another model that I evaluated was Facebook's (now Meta) FAIR team's Translation Task submission to WMT19, referred to as FAIR within this report. The FAIR model tested 7 BLEU higher in English to Russian on my configuration than the reported WMT19 News Test results, and only 2 BLEU higher on the Russian to English model. Out of all the models that I tested, it was one of the best performing models with the highest BLEU scores and ChrF scores in both English to Russian and Russian to English variants. Only the SeamlessM4T-v2 model was able to beat the FAIR model, scoring 0.005 better on the BERT F1 score. An arising question from this statement could be that of why Meta would even bother with this large SeamlessM4T model, when a much smaller team successfully built a 'better' model? However, this would be a naive statement as the FAIR model was both trained and tested on WMT19 data, which is an extended version of the WMT14 datasets. This supposed specialisation in WMT translation makes the FAIR model a great translator of WMT data, reaching the desirable status of "high quality translations" from Google's BLEU interpretation guide [63]. However, it is highly likely

that these results would not be repeatable, given a different testing dataset. Therefore, whilst its performance should be commended in this instance, it should also be taken with reservations for its potential performance in other fields.

Why does EN-RU perform worse than RU-EN?

One noticeable trend within the reported metrics tables is the difference in scores between the English to Russian and Russian to English models, with the later often performing better than the former. On average, there is a 7 BLEU and 6 ChrF difference between the model variants. In terms of PETRAI, there was an average of roughly a 5 BLEU and 7 ChrF increase between the Russian to English and the English to Russian variants. This provokes the question of why this may be the case?

One reasoning may be shown within the brevity penalties of the PETRAI models. In the Russian to English PETRAI variants, the brevity penalties are 1.0 and 0.978 on average for PETRAI-2 and PETRAI-3 respectively. In contrast, these averages decrease drastically to 0.723 and and 0.720 for their corresponding English to Russian variants. In the BLEU score, the brevity penalty is intended to punish translations that are considered too short by being shorter than the references. This prevents short sentences from receiving unexpectedly high scores from a high n-gram overlap, due to it only using a few keywords from the references. This brevity penalty is multiplied by the geometric mean of the n-gram overlaps. Therefore, if this brevity penalty was to be removed, the English to Russian PETRAI models would achieve an average BLEU score of 22.784, with the Russian to English models receiving an average score of 21.768 BLEU. This shows that the main disadvantage of the English to Russian PETRAI models is their brevity in translation and that further training to fix this would aid in closing the gap between the language variants. Looking at the CSV files containing the predictions of each model, it seems that the PETRAI-2 model also prefers to use contractions. For instance the input Russian sentence "Украине не нужен блок НАТО", which is given the reference ”Ukraine does not need the NATO alliance”, gets translated to ”Ukraine doesn’t need a NATO block” by PETRAI-2. The contraction of ”does not” to ”doesn’t” produces a shorter sentence with two tokens being converted to a singular one. This leads to a lower brevity score as the resulting sentence is shorter than the original, even if the contraction is semantically the same to the original. Another point of note is that "блок" is pronounced the same as ”block” in English, and even translates to ”block”. However, the context would indicate the translation to ”bloc”, meaning an alliance of several countries, so the given reference translation would be

a more fluent translation.

On a more abstract level, English to Russian seems to a harder translational challenge than Russian to English. English tends to use more words of a shorter nature to describe the same meaning as Russian, with English words being between 30-50% shorter on average than their Russian counterparts. As a result, Russian tends to be more artistic with sentences and phrases tending to be "long and colorful" whilst English is more analytical in nature with more "brief" and succinct. Furthermore, the various genders and their resulting gender agreements found in Russian simply does not appear in English, whose grammar does not contain gender. This makes it harder for English to Russian translators where declension and gender are more difficult to create embeddings for than for Russian to English translators, where genders play a significantly less important role [65].

To Prefix or not to Prefix?

As with the Shakespearean quote, the results prove to not be quite so clear. For the Russian to English PETRAI models, both PETRAI-2 and PETRAI-3 both performed better without the prefixation method, outlined earlier, with both models resulting in either equivalent or better BLEU and ChrF scores without a prefix. However, the BERT F1 score do both show minor improvements, when prefixed than without. Whilst the prefixed versions both exhibit worse recall scores, they both improve on the precision scores, in comparison to their standard variant counterparts. This shows that they are producing more precise translations at the cost of recalling less information from the references than the non-prefixed models. Despite their lower BLEU and ChrF scores, they are not dramatically inferior to their unprefixed variants in these metrics and, perhaps, the BERT score tells us more about their performances than the de facto metrics. One thing to note is that the differences in scores are marginal, making it hard to see differences in performances through example translations.

The story, however, shifts when examining the scores of the English to Russian models. Both the prefixed PETRAI-3 and PETRAI-3 enjoy a relatively sizeable bump their scores over their non-prefixed counterparts, seeing a near 0.5 increase in both BLEU and ChrF scores with an average increase of roughly 0.25 for the BERT F1 scores. Unlike with the English to Russian PETRAI models, the Russian to English variants both do see a decrease in their brevity penalty score. Ideally, this brevity penalty score is kept as high as possible, as this helps maximise the BLEU score. The decrease might be caused by the models having to deal with more tokens during the Seq2Seq encoding and decoding, caused by the additional prefix.

PETRAI-2 versus PETRAI-3

Similarly to the prefixation question, there also seems to be a fairly uncertain choice of PETRAI-2 or PETRAI-3. In the Russian to English category, the PETRAI-2 model outperforms the PETRAI-3 model in both its standard and prefixed variants, with the standard variant being the higher scoring variant in both the BLEU and the ChrF score. The standard, unprefixed variant of PETRAI-2 did receive a slightly lower BERT score, however. Whilst the recall score was higher than that of the prefixed variant, it received a lower precision score and, thus, a lower BERT score. As discussed earlier, the prefixed versions of both PETRAI-2 and PETRAI-3 seemed to perform slightly better in both language variants, despite the brevity issues found in the English to Russian variants. For the Russian to English models, its clear that PETRAI-2 performs slightly better than PETRAI-3, beating it across the board in all three metrics. This is the reason behind choosing the prefixed PETRAI-2 model for English to Russian translation. For English to Russian, the results change with the prefixed version of PETRAI-3 producing the best metrics and, therefore, was chosen for the final PETRAI model.

PETRAI versus the Rest

Now that the PETRAI variants have been chosen, how do they measure against the models from literature? The English to Russian half of the PETRAI translator received a score of 16.66 BLEU, meaning that it produces translations that are '[h]ard to get the gist' of, with the Russian to English half receiving a score of 21.95 BLEU, meaning that '[t]he gist [of the translations are] clear, but [have] significant grammatical errors' [63]. A reader may point out that these results are fairly poor in comparison to the averages of 29.8 BLEU and 36.6 BLEU respectively. However, I am quite happy with the results produced by my translator. Having researched comparitive examples from literature, most models have been trained on more substantial amounts of data than my translator. For instance, Ariel Xv's submission to WMT20 contained over 90.8 million sentence pairs for their most filtered model [47], producing an average increase of 12.6 BLEU score over my PETRAI model, which was trained on 1 million sentence pairs. Meta's FAIR team's submission to the WMT19 Translation Task scored an average of 23.4 BLEU, 7.0 BERT F1, and 20.6 ChrF higher than PETRAI on my WMT14 News Test configuration. Their model was trained on a far smaller dataset than the more recent model by Ariel Xv, having been trained on 26.0 million sentence pairs. Despite its smaller size, it still dwarfs the dataset that PETRAI was trained on. Their team with access to hardware and resources makes it hard to compete to produce a model with similar or better scores.

Given the highly-resourced nature of the companies, I have also collated accuracy metrics on Google Translate, Microsoft Translator, and OpenAI's ChatGPT from Jungha Son's and Boyoung Kim's research in July 2023 [64]. Using test data from WMT18 and WMT20, they analysed the results of 18 different language pairs covering 12 languages against the four models, measuring both OpenAI's ChatGPT-3.5 and its newer ChatGPT-4. Interestingly, the report found that the models did not perform as well as one may expect. Despite Google having researched and developed the BLEU metric, including guides on its usage and interpretation, finding their reported metrics for their model seems to be hard to find and, considering the results found in Son's and Kim's report, the reason seems to be clear. Despite the given computing, research, and monetary power at the given companies, PETRAI is relatively not far behind with the largest gap being between itself and Google Translate. In comparison, Google Translate is only 16.7 BLEU and 14.5 BLEU ahead in the English to Russian and the Russian to English tests respectively. For ChatGPT-3.5, which is the closest competitor to PETRAI, it scores 9.2 BLEU and 11.0 BLEU higher than PETRAI respectively. Considering that ChatGPT-3.5 is able to consider 175 billion parameters and that ChatGPT-4 able to consider 1 trillion parameters [66], it is impressive that PETRAI is able to keep up. The main caveat though is that, whilst ChatGPT is a text generative AI like PETRAI, it is not specifically trained for translation and, therefore, does not aim to produce direct translations as PETRAI does.

6.1.4 Speed

Within the implementation of the `pipeline()` function from Hugging Face, the function takes the model name and searches for a local version of the model. This allows for models to be stored and ran locally, such as my PETRAI models, and allows for models to be downloaded and stored within cache, without the need for downloading them from the web each time. For a fair comparison, the following tests were done after each model's language variants had been downloaded to cache, ensuring consistency across trials and tests.

Speed Metrics Data

As described in the Calculation of Metrics Section, each sample consists of timing the initialisation of the translation models and the timing of the time it takes to translate 50 texts from the source to the target language. These texts were kept the same across both samples and model variants. For each model variant, 10 samples were taken with the scores being averaged

via their mean. The results can be seen in the table below.

Models	**Avg. Model Load Speed (s)**	**Avg. TTT 50 texts (s)**	**Avg. Secs/Text (s)**
petrai_ru-en_bart_opus100-2 (w/o)	28.666	65.723	1.314
petrai_ru-en_bart_opus100-2 (pref)	28.655	63.636	1.273
petrai_en-ru_bart_opus100-2 (w/o)	28.829	180.217	3.604
petrai_en-ru_bart_opus100-2 (pref)	28.738	183.661	3.673
petrai_ru-en_bart_opus100-3 (w/o)	28.391	59.111	1.182
petrai_ru-en_bart_opus100-3 (pref)	28.433	55.137	1.103
petrai_en-ru_bart_opus100-3 (w/o)	29.456	186.264	3.725
petrai_en-ru_bart_opus100-3 (pref)	28.617	169.992	3.400
Helsinki-NLP/opus-mt-ru-en	1.745	16.223	0.324
Helsinki-NLP/opus-mt-en-ru	1.831	15.420	0.308
facebook/wmt19-ru-en	0.918	25.588	0.512
facebook/wmt19-en-ru	1.082	28.891	0.578
facebook/seamless-m4t-v2-large (ru-en)	10.223	106.038	2.121
facebook/seamless-m4t-v2-large (en-ru)	10.678	121.588	2.432

Figure 6.6: Speed metrics collected on various models (TTT stands for Time to Translate)

In terms of model loading speed, PETRAI still has a lot of room for further work and optimisation. Comparatively, all of the PETRAI models and variants initialise in roughly 28 to 29 seconds, whilst both the FAIR and OPUS-MT models take roughly 1.0 seconds and 1.8 seconds, respectively. Their lightweight nature also translated to their running times, with the OPUS-MT translating 50 texts just under 3.4 times faster in the Russian to English tests and just over 11 times faster in the English to Russian tests than the best PETRAI model. The OPUS-MT models were designed for open-source usage to provide support and implementation of translation for over 500 languages, with a focus on smaller languages. The lightweight and speedy nature of the OPUS-MT models shows with an average time of 0.316 seconds per text, the best found in my evaluations.

A notable finding in my dataset is the difference between the speed in translating between the model variants, with the Russian to English models completing, on average, the 50 text translation task in a third of the time than their equivalent English to Russian counterparts. As explained in the earlier section on accuracy, English to Russian seems to be the trickier task as Russian texts often tend to be shorter than English texts of the same semantic meaning. There are also further issue including struggles of gender agreements, contractions, and declension. The shorter texts lead to less tokens being produced via the tokeniser, making the encoding process far easier for the system than an English sentence. This is also seen in the SeamlessM4T-v2 models, where the Russian to English tests ran nearly 15% quicker than its English to Russian counterpart.

In terms of the SeamlessM4T-v2 models, their results seem disappointing as this model seams to lag behind other, smaller translator models. One possibility is that this model was not designed for usage with Hugging Face. Whilst it has successfully been ported over to the website for public use, it has not been properly implemented into the `pipeline` API and requires a manual implementation for useful results. Being the only imported model to use this manual implementation, this may put the model at a disadvantage as it must use Python code to perform translation, rather than a much more memory efficient and faster language like C++ or Rust. Another issue is that such a model may be such a massive model, given its multilingual, multi-modal nature. This may lead to slower translation in comparison to a smaller, leaner model which concentrates only on a single language pair.

The prefixed PETRAI-3 models outperformed every both the PETRAI-2 variants and the unprefixed PETRAI-3 variant. This seemed to be surprising initially as one would suspect that adding more tokens would increase translation time, as prefixation of the texts is included as part of the timings. This, however, did not turn out to be the case with the prefixed models outperforming their unprefixed counterparts more often than not. One explanation is that the prefixes help the encoder to signify the start of a sentence, making the encoding process of converting the trained embeddings into vectors for the decoder easier. This, in turn, helps to speed up the translation process with this difference being exacerbated over the set of 50 texts.

6.1.5 Satisfaction of Requirements

- **T1 - The English to Russian translation should be relatively accurate.**
- **T2 - The Russian to English translation should be relatively accurate.**

  In terms of translational accuracy, I believe that this requirement has been met to the best of my ability. One of the aims of the project was to explore how well I could build a Machine Translation model using my own hardware. Despite the resource limitations, I still managed to achieve a decent score of 21.95 BLEU on the Russian to English variant, with the English to Russian variant scoring slightly lower at 16.66 BLEU.

  Whilst this fall behind in comparison to other models, direct evaluations of model performance are not as simple as measuring BLEU scores. PETRAI was trained on a mere 1 million sentence pairs with the closest models, in terms of training set size, coming in at roughly 26.0 and 90.8 million sentence pairs for the FAIR's and Ariel Xv's models respectively. Given more training time and better hardware, such as the usage of high performance machine with GPUs designed for Machine Translation, its reasonable to argue that PETRAI could easily catch up with these better performing models. Therefore, I am satisfied with the results of my model.

- **T3 - The translation model should translate the input text quickly.**

  Similarly to translational accuracy, the speed of my model does fall slightly short of well-established models. Despite this, my model only takes 3.4 seconds on average to translate a English text into Russian. When translating from Russian to English, this speed is nearly a third of its counterpart, translating a text in 1.3 seconds on average. I would consider this speed to be satisfiable, as it allows for users to translate texts at relative speed. There may be issues with long texts, but its expected that longer texts take longer for translators to translate. Furthermore, my implementation breaks down texts into sentences to prevent issues with max token generation limits.

6.2 User Interface Evaluation

6.2.1 Satisfaction of Requirements

- **U1 - Users should be able to bidirectionally translate text between English and Russian.**

The user interface is able to take a users input in order to select the source and target language for the upcoming input translation. The UI assumes that English and Russian are the only two possible languages for translation, given the nature of the project.

- **U2 - Users should be able to type using the English Latin alphabet when using English input.**
- **U3 - Users should be able to type using the Russian Cyrillic alphabet when using Russian input.**

  The terminal accepts all UTF-8 input, which allows for input in both the English Latin alphabet and the Russian Cyrillic alphabet. The model is also able to accept these input alphabets and tokenise them in the translation process.

- **U4 - Users should be able to translate the inputted text.**

  The user can press the 'Enter' key after they have typed their input text. This will then pass on the input text to the translation model to be translated.

- **U5 - The translation of the inputted text should occur quickly after submission.**

  As discussed in the **Satisfaction of Translation Requirements Section**, the model is able to translate reasonably speedily in both directions. This is aided by the initialisation of the translation pipelines upon starting the program and the ability to select and reuse these pipelines multiple times for multiple translations.

- **U6 - The UI should then display the translation.**

  The terminal also is able to display all UTF-8 input, which allows for both the English Latin alphabet and the Russian Cyrillic alphabet to be displayed. This means that the resulting translations can be displayed for the user to see, in either English or Russian.

- **U7 - The UI should be accessible to a range of users.**

  The UI is simple and has instructions for usage. These instructions, in combination with the **User Guide** in the appendix, should make it easy for a range of users to translate with PETRAI. One issue is that PETRAI is still limited to those users who have a basic grasp of the terminal. The UI is also only accessible to those who can read English and to those with the ability to type, with speech-to-speech or text-to-speech not being implemented in this model. These are, however, possible extensions and reach beyond the scope of this project.

- **U8 - The UI should be usable on a range of devices.**

  As the project was written in Python, it is accessible to both UNIX-based and Windows-based machines along with any other system that supports running Python scripts. The translation process only consumes CPU and RAM resources. This means machines without GPUs are still more than capable of running the model, despite the usage of a GPU in the training of PETRAI.

Legal, Social, Ethical, and Professional Issues

7.1 British Computing Society

The following are the four main qualities for members of the British Computing Society (BCS) to uphold [67].

1. **Public Interest** - show due regard for public well-being and the environment.
2. **Professional Competence and Integrity** - maintain a professional level of competence and integrity.
3. **Duty to Relevant Authority** - carry out any work with due care and diligence, avoiding potential conflicts of interest.
4. **Duty to the Profession** - uphold the integrity and respect of the BCS and the profession.

Throughout this project, I ensured that these qualities were maintained. I showed due regard for public well-being by using established datasets, rather than scraping potentially sensitive or revealing information off the internet. I followed the point about professional competence and integrity by doing my research into the dangers of AI and machine learning, making sure to develop my professional skills and competence in order to carry out the project as best I could. I also maintained my duty to my relevant authority by avoiding any conflicts of interest with my supervisor and the University. Finally, I upheld the integrity and respect of the University and BCS by following these guidelines and supporting professional development through the writing and development of this exploratory work.

7.2 Issues Pertaining to Artificial Intelligence

As with all Artificial Intelligence related developments, there are a multitude of ethical issues to address and consider throughout the whole developmental process of PETRAI.

7.2.1 Bias

One issue relates to potential bias contained within my models. For instance, one of the qualities of languages is their ability to adapt and evolve as society changes. This means, however, that its very likely that many words have been missed from the learning process, rendering the sentences unable to be translated. Nouns tend to be quite specific in usage and come into existence constantly as new concepts are conceived and new items are made. For instance, place names have not been thoroughly covered by the learning process due to the innumerable number of places in the world. Similarly with names as they rise and fall from popularity, with variations on spelling changing all the time in an effort for uniqueness. This, however, may lead to the feeling of exclusion as anyone using PETRAI may not be able to translate their own name or the name of their pen-pal, whom they are writing to. A particular bias may be to non-western names, as they tend to be poorly represented on the internet. So, whilst unintentional, there is still an existence of bias against minority groups. Slang is encapsulated by the idea of novelty. As English is ever-evolving, it is almost impossible to translate new slang words like ”rizz” or ”delulu” as these words have been developed by the English-speaking youth and a consensus on a translation into another language, such as Russian, simply does not exist yet.

7.2.2 Usage

There are also issues with its potential issues, caused directly and indirectly by the usage of PETRAI. One potential use for PETRAI is for harm. For instance, PETRAI could give a malicious spammer the tools needed to automatically translate spam emails into Russian, giving them the ability to target a wider range of audiences. Malicious use cases, such as the aforementioned spammer, are real issues that current translation models have to face with most APIs, such as Google Translate, requiring a fee for usage. This helps to cost out most malicious users, but also prices out small developers or well-intentioned members of the public from usage. I do believe that this will not be an issue as PETRAI will not be released to the general public via the Hugging Face API.

A more abstract and indirect consequence of PETRAI, and AI in general, is the loss of work for human translators. Concerns about AI taking over jobs has been prevalent in the news recently, as generative AI becomes ever more powerful, and this is true in the translation sector. Due to their speed, cost, and availability, Machine Translation has become increasingly more prevalent in the work force. Whilst this originally allowed for human translators to focus on work where the accuracy of a given translation was paramount, such as translating company contracts, even this sector has become an area of risk as companies try to cut costs. Duolingo, an app which allows users to learn languages through a variety of daily tasks, layed off 10% of their human translators with the job of generating meaningful translation tasks for the users of Duolingo to complete has been passed of to generative AI machine translators instead [68]. Whilst PETRAI is not able to generate translation tasks, such as the AI that Duolingo uses, it could be used to translate phrases instead of a human. Another issue with this is that the usage of Machine Translation removes any human creativity and flair from the translation. Translation is an art-form and the ability to translate any given text by understanding the surrounding context, it still a quality unique to humans as machines have no real-world knowledge about the context of languages.

7.2.3 Questionable Translations

Another ethical concern surrounds the potentially sensitive content produced by the PETRAI model. One of the significant concerns of the OPUS dataset, and with datasets which utilise web crawling in general, is their potentially explicit or controversial topics. In the English-Russian OPUS corpora [55], there are many pornographic references within the datasets. This would lead to the assumption that it is possible to translate explicit data through models trained on these corpora. Whilst the impact of this is unlikely to cause wide-scale distress, other more controversial topics may lead to discussions on possible censorship for a machine translator. For instance, if web-crawled data happened to include explicit discrimination phrases or terms, a decision would likely need to be made whether to censor the specific words, in the case of a curse word for example, or a more comprehensive censorship of the sentences or texts, in the case of blatant discrimination.

This, however, raises the question of where the boundary between maintaining decency and preserving free speech lies. These two qualities can be seen as conflicting and even contradicting depending on the views of the individual. PETRAI, like many Machine Translation models, faces the dilemma of how to handle such content appropriately. Context recognition is

such a difficulty quality to find in Machine Translation with machines not being able to truly understand, unlike a human. In Meta's SeamlessM4T model, they performed research into their model to discover how much "added toxicity" was found in their model. The term 'added toxicity' refers to how much toxic content, including discrimination and expletive language, was present in the output that was not present in the input. They found that their SeamlessM4T-Large model produced a 63% reduction in added toxicity, showing that their model does not add more bias to a translation than there already is in the input. Such testing is necessary to find any bias introduced by malicious or bad training data [42, p.67-69]. In my implementation, I decided to not filter any explicit content due to the difficulty of preventing content with many implementations found in use often being rather basic in nature and easy to evade. As PETRAI is not a publicly hosted model, I do not see this to be a prevalent issue.

7.2.4 Environmental Impact

The environmental impact of AI and machine learning is a current issue and is of widespread concern, therefore it is important to evaluate the impact of PETRAI's development. A study was conducted in 2021 to measure the carbon dioxide (CO_2) emissions associated with various architectures. In total, eight models were trained, four using an LSTM network architecture and four using a TNN architecture. In each type of architecture, there were four models covering the English-Spanish and English-French translation. As PETRAI was used the TNN architecture, I will be using those results to measure PETRAI's environmental impact. Below are the figures gathered from the experiment:

Model	**GPU**	**Elapsed time (h)**	**Avg. power (W)**	**kWh**	CO_2 **(kg)**
en-fr	1080Ti	5.22	176.70	3.64	1.33 ± 0.45
en-es	1080Ti	6.60	176.54	4.60	1.68 ± 0.56
fr-en	1080Ti	6.15	176.64	4.29	1.56 ± 0.53
es-en	1080Ti	6.36	179.48	4.50	1.64 ± 0.55
en-fr	P100	5.06	153.47	2.27	1.44 ± 0.12
en-es	P100	6.06	152.08	2.69	1.71 ± 0.14
fr-en	P100	4.85	151.43	2.15	1.37 ± 0.11
es-en	P100	6.20	151.59	2.74	1.74 ± 0.14

Figure 7.1: Power usage and estimated CO_2 emissions of training TNN models, as adapted from [69, p.14]

In comparison to the NVIDIA® GeForce™ RTX 4080 M GPU [70], referred to here on as the 4080M, that PETRAI was trained on, the NVIDIA® GeForce™ GTX 1080 Ti Laptop GPU [71], referred to here as the 1080Ti, have a few differences. The 4080M is a much newer GPU having

been released in February 2023, just under six years after the 1080Ti. However, as a mobile chip, which finds itself being used in mostly laptop machines, is a far less powerful chip than its desktop relative. The 1080Ti has a TDP (Thermal Power Draw) of 250W in comparison to the 4080M, which has a TDP of 110W. The TDP of a GPU can be used as a good heuristic on the maximum power draw of a GPU under the same conditions, such as ambient temperature and relative load. Total power draw of the whole system, however, also depends on components such as the CPU and memory as well as any possible external components powered by the machine. On 1st April 2024, it is calculated that the average CO_2 emissions per kWh is 0.202kg/kWh [72]. Therefore, we were to assume a similar power draw from both systems of 175W and a total training time of 30 hours per model, then both models combined would have produced about 2.12kg of CO_2 emissions simply for the training alone.

Whilst relatively small, this figure is still existent and cannot be simply ignored. For much larger models running training over weeks, rather than days, this figure will be much higher. Carbon emission have been going up globally and consideration in how to reduce training times and develop more efficient architectures needs to be done in order to lower the carbon cost of translation.

Conclusion and Further Work

8.1 Conclusion

Throughout the project, the main research point was about how democratic is the process of developing translation systems and how viable is it for individuals and small organisations to train translation models for new languages, especially for low-level languages.

Through the development of PETRAI, it is clear to see that great strides have been made throughout the past few years in developing new architectures and utilising various machine translation techniques in order to improve the accuracy of models. Whilst, PETRAI did perform around 10 BLEU and 17 ChrF worse than the average reported evaluations of various models, these models are considered to be high-performing and are some of the best examples of open-source English-Russian translational models to date. PETRAI was also trained on a considerably small set of data, when compared to other models. Whilst PETRAI was trained on 1 million pairs of parallel sentences, Ariel Xv's model was trained on 10 million monolingual and 102 million parallel texts [47] whilst the reported score on the WMT20 News Test was, on average, only 12.7 BLEU higher than PETRAI. Russian and English are considered high-resource languages, considering the number of people online who use both languages to communicate and to create resources with. However, the usage of 1 million parallel texts more accurately represents a more low-resource language where the use of targeted web-scraping and manual verification could be undertaken in order to gather resources for model training, as with Google's MADLAD-400 project.

This shows that it could be feasible for an individual or small team to develop a low-language translation model, even with limited hardware. For ease, this could be a pair of English-X and X-English models, where X represents the low-resource language, as the prevalence and strength of translation models to and from English allow for English to provide an interlingua between X and a large variety of languages.

8.2 Further Work

8.2.1 User Interface

There is also further work that could be undertaken with the user interface. Currently, it is a simple terminal interface which requires the user to install various python packages in order for it to be usable. In order to render this more accessible to the public, a more abstracted application would need to be developed so that users could access it via a simple install or via a website. This could allow for the implementation of various features such as automatic detection of spelling and grammar errors and real-time translation.

8.2.2 Techniques to Improve Machine Translation Model

Despite the achievements of PETRAI, there are many ways that the methodology could be improved to develop and train better translation models. This could include improving accuracy or speed metrics.

Back-translation

One method of improving a neural machine translation is with back-translation. It uses a pre-trained translation model and uses monolingual data by translating it into the target language. This translated data is then re-translated back into the source language in order for the system to better understand various linguistic qualities and improve its embeddings. Due to the nature of this technique, it can be comparatively more computational expensive to perform as it has to translate any given source text twice. In FAIR's WMT19 submission, they found that the inclusion of back-translation improved both their of their English-Russian models by 3 BLEU on average [45, p.315-318] [47, p.323]. The inclusion of monolingual data also increases ease of data collection as the validity of one set of data is required to be checked and knowledge of only one language is needed for verification. This makes it ideal for low-resource languages.

Fine-tuning

As many translation models are trained for general translation, many lack knowledge of technical words or phrases. In order to rectify this, pretrained translation models can be fine-tuned by training them with parallel corpora containing language and phrases from the desired technical domain. This is especially useful in many scientific domains where phrases may be complex

and rare within general public use.

Another usage of fine-tuning is to improve existing models for greater accuracy or fine-tuning a generic model in order to translate into a new language. In 2022, Tiedemann and the OPUS-MT project released a large set of new OPUS-MT models for English to X, X to English, or X to X translation, where X represents a language or language class. Of interest, was the language class models which where able to translate between a range of related languages. Some of these language classes were: Slavic, West Slavic, East Slavic, South Slavic, Uralic, Indo-European, North European, Romance, and Scandinavian. These models can be found on the Hugging Face website [73]. This enables the possibility of choosing a language contained within one of these language classes and fine-tuning them to fit your language. This is beneficial for low-resource languages, as the more specific the language class, the more closely aligned the pretrained embeddings will be with the desired outcomes for the language. This will help to improve translational accuracy for those low-resource languages.

Ensemble Models

Another technique is the usage of ensemble models. This approach seeks to combine the results various models and take advantage of their various qualities to increase translational accuracy and quality. The hope is that if various "diverse" and independently devised models produce a similar or the same result, then the likelihood of a translation error is lower [74]. In Ariel Xv's WMT20 submission, they found that the ensemble model improved their Russian to English model by 2.4 BLEU whilst their English to Russian model was only improved by 0.3 BLEU [47, p.324]. This is likely due to the difficulties of translating between English to Russian and shows that the various models were already predicting similar texts, even if they were not as accurate as the predictions of the Russian to English models.

Improve Training Data

The simplest way to improve an machine translation model is to provide a lot of accurate parallel translation data. This can be further split up into improving the quality of translation data and improving the quantity of translation data. Web-crawling has been a massive proponent in the rise of machine translation, even in the era of SMT. There are various massive online parallel corpora with the OPUS Corpus providing over 390 million sentences for training machine translation models [55]. PETRAI was only trained on 1 million sentences. So with further time and improved computing power, it would be reasonable to assume that PETRAI would

perform much better and would be more comparable to the state-of-the-art models out there.

As proved by the MADLAD-400 project, cleaning the data is also shown to improve data and reduce training times by focusing on only training data that will improve the machine and preventing any inaccurate or inconsistent data from harming any of the trained embeddings. Future work could see a more comprehensive review of the test data to prevent any inaccurate translations from making the cut using techniques learned in the **overview of MADLAD-400**. This could range from ensuring matching language identifiers, prevention of explicit material, and checking of character lengths of texts.

Bibliography

[1] András Kornai. Digital language death . *PLoS One*, October 2013.

[2] Wayles Browne and Vyacheslav Vsevolodovich Ivanov. Slavic languages. `https://www.britannica.com/topic/Slavic-languages`, November 2023.

[3] Ousseynou Tall. The Linguistic Difference Between English and Russian. *Advances in Social Sciences Research Journal*, 9(5):159–164, May 2022.

[4] Russian Federation. Constitution of the Russian Federation. `http://www.constitution.ru/en/10003000-04.htm`.

[5] Wolfestone. Why Russian is spoken on the International Space Station. `https://blog.wolfestone.co.uk/why-russian-is-spoken-on-the-international-space-station`, July 2019.

[6] Terrence Wade. *A comprehensive Russian grammar*. Wiley-Blackwell, 2011.

[7] Simeon Potter and David Crystal. English language. `https://www.britannica.com/topic/English-language`, November 2023.

[8] Robins Air Force Base. Defense Language Aptitude Battery. `https://www.robins.af.mil/Portals/59/documents/Base%20Training/DLABInformation.pdf`.

[9] Mario Pei. *The Story of Language*. Lippincott, 1949.

[10] Sijeta Braha. Translation: Human vs Machine. May 2016.

[11] John Hutchins. From First Conception to First Demonstration: the Nascent Years of Machine Translation, 1947-1954. A Chronology. *Machine Translation*, 12(3):195–252, 1997.

[12] Oxford English Dictionary. Definitions of 'pen'. `https://www.oed.com/search/dictionary/?scope=Entries&q=pen`, December 2023.

[13] Doug Arnold, Lorna Balkan, Siety Meijer, R.Lee Humphreys, and Louisa Sadler. *Machine Translation: an Introductory Guide*. January 1994.

[14] Thierry Poibeau. Machine Translation. 2011.

[15] Francis Tyers and Chris Martin. Direct translation and transfer translation pyramid. `https://en.m.wikipedia.org/wiki/File:Direct_translation_and_transfer_translation_pyramid.svg`, July 2006.

[16] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311, June 1993.

[17] Zhixing Tana, Shuo Wang, Zonghan Yanga, Gang Chena, Xuancheng Huanga, Maosong Suna, and Yang Liu. Neural Machine Translation: A Review of Methods, Resources, and Tools. *AI Open*, December 2020.

[18] Philipp Koehn. Neural Machine Translation. September 2017.

[19] Questions and Answers in MRI. Types of Deep Neural Networks.

[20] Nova. From RNNs to Transformers: The Evolution of NLP Models. March 2023.

[21] Aravindpai Pai. Analyzing Types of Neural Networks in Deep Learning. July 2023.

[22] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional Sequence to Sequence Learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252. PMLR, August 2017.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. 2017.

[24] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. 2016.

[25] Stefania Cristina. The Attention Mechanism from Scratch. January 2023.

[26] Machine Translate. The Conference on Machine Translation. `https://machinetranslate.org/wmt`, Feburary 2024.

[27] Machine Translate. WMT22 - Seventh Conference on Machine Translation. https://machinetranslate.org/wmt22, Feburary 2022.

[28] Matous Machacek and Ondrej Bojar. Findings of the 2014 Workshop on Statistical Machine Translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.

[29] Common Crawl. Common Crawl - Overview. https://commoncrawl.org/overview, February 2024.

[30] Yandex. Yandex - About. https://yandex.com/company/general_info/yandex_today, March 2024.

[31] Yandex. Anglo-Russian Parallel Corpus. https://translate.yandex.ru/corpus?lang=en, February 2024.

[32] Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. Improving Massively Multilingual Neural Machine Translation and Zero-Shot Translation. April 2020.

[33] Jorg Tiedemann. Parallel Data, Tools and Interfaces in OPUS. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

[34] Cloudflare. What is a web crawler? — How web spiders work. https://www.cloudflare.com/en-gb/learning/bots/what-is-a-web-crawler/, March 2024.

[35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[36] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating Text Generation with BERT. *CoRR*, abs/1904.09675, 2019.

[37] Maja Popovic. chrF: character n-gram F-score for automatic MT evaluation. In Ondrej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, and Pavel Pecina, editors, *Proceedings of the Tenth Workshop*

on Statistical Machine Translation, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[38] LearnEnglish. Present Tense - Intermediate. `https://learnenglish.britishcouncil.org/grammar/english-grammar-reference/present-tense`, March 2024.

[39] Jorg Tiedemann and Santhosh Thottingal. OPUS-MT - Building open translation services for the World. In Andre Martins, Helena Moniz, Sara Fumega, Bruno Martins, Fernando Batista, Luisa Coheur, Carla Parra, Isabel Trancoso, Marco Turchi, Arianna Bisazza, Joss Moorkens, Ana Guerberof, Mary Nurminen, Lena Marg, and Mikel L. Forcada, editors, *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 479–480, Lisboa, Portugal, nov 2020. European Association for Machine Translation.

[40] Helsinki NLP. Helsinki-NLP/opus-mt-en-ru. `https://huggingface.co/Helsinki-NLP/opus-mt-en-ru`, March 2024.

[41] Helsinki NLP. Helsinki-NLP/opus-mt-ru-en. `https://huggingface.co/Helsinki-NLP/opus-mt-ru-en`, March 2024.

[42] Seamless Communication, Loïc Barrault, Yu-An Chung, Mariano Cora Meglioli, David Dale, Ning Dong, Paul-Ambroise Duquenne, Hady Elsahar, Hongyu Gong, Kevin Heffernan, John Hoffman, Christopher Klaiber, Pengwei Li, Daniel Licht, Jean Maillard, Alice Rakotoarison, Kaushik Ram Sadagopan, Guillaume Wenzek, Ethan Ye, Bapi Akula, Peng-Jen Chen, Naji El Hachem, Brian Ellis, Gabriel Mejia Gonzalez, Justin Haaheim, Prangthip Hansanti, Russ Howes, Bernie Huang, Min-Jae Hwang, Hirofumi Inaguma, Somya Jain, Elahe Kalbassi, Amanda Kallet, Ilia Kulikov, Janice Lam, Daniel Li, Xutai Ma, Ruslan Mavlyutov, Benjamin Peloquin, Mohamed Ramadan, Abinesh Ramakrishnan, Anna Sun, Kevin Tran, Tuan Tran, Igor Tufanov, Vish Vogeti, Carleigh Wood, Yilin Yang, Bokai Yu, Pierre Andrews, Can Balioglu, Marta R. Costa-jussà, Onur Celebi, Maha Elbayad, Cynthia Gao, Francisco Guzmán, Justine Kao, Ann Lee, Alexandre Mourachko, Juan Pino, Sravya Popuri, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, Paden Tomasello, Changhan Wang, Jeff Wang, and Skyler Wang. SeamlessM4T: Massively Multilingual Multimodal Machine Translation, August 2023.

[43] Seamless Communication, Loïc Barrault, Yu-An Chung, Mariano Cora Meglioli, David Dale, Ning Dong, Paul-Ambroise Duquenne, Hady Elsahar, Hongyu Gong, Kevin Heffer-

nan, John Hoffman, Christopher Klaiber, Pengwei Li, Daniel Licht, Jean Maillard, Alice Rakotoarison, Kaushik Ram Sadagopan, Guillaume Wenzek, Ethan Ye, Bapi Akula, Peng-Jen Chen, Naji El Hachem, Brian Ellis, Gabriel Mejia Gonzalez, Justin Haaheim, Prangthip Hansanti, Russ Howes, Bernie Huang, Min-Jae Hwang, Hirofumi Inaguma, Somya Jain, Elahe Kalbassi, Amanda Kallet, Ilia Kulikov, Janice Lam, Daniel Li, Xutai Ma, Ruslan Mavlyutov, Benjamin Peloquin, Mohamed Ramadan, Abinesh Ramakrishnan, Anna Sun, Kevin Tran, Tuan Tran, Igor Tufanov, Vish Vogeti, Carleigh Wood, Yilin Yang, Bokai Yu, Pierre Andrews, Can Balioglu, Marta R. Costa-jussà, Onur Celebi, Maha Elbayad, Cynthia Gao, Francisco Guzmán, Justine Kao, Ann Lee, Alexandre Mourachko, Juan Pino, Sravya Popuri, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, Paden Tomasello, Changhan Wang, Jeff Wang, and Skyler Wang. SeamlessM4T: Massively Multilingual Multimodal Machine Translation. October 2023.

[44] Sneha Kudugunta, Isaac Caswell, Biao Zhang, Xavier Garcia, Christopher A. Choquette-Choo, Katherine Lee, Derrick Xin, Aditya Kusupati, Romi Stella, Ankur Bapna, and Orhan Firat. MADLAD-400: A Multilingual And Document-Level Large Audited Dataset. 2023.

[45] Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. Facebook FAIR's WMT19 News Translation Task Submission. In Ondrej Bojar, Rajen Chatterjee, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Andre Martins, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Matt Post, Marco Turchi, and Karin Verspoor, editors, *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 314–319, Florence, Italy, August 2019. Association for Computational Linguistics.

[46] Hugging Face. Byte-Pair Encoding tokenization. `https://huggingface.co/learn/nlp-course/en/chapter6/5`, April 2024.

[47] Ariel Xv. Russian-English Bidirectional Machine Translation System. In Loic Barrault, Ondrej Bojar, Fethi Bougares, Rajen Chatterjee, Marta R. Costa-jussa, Christian Federmann, Mark Fishel, Alexander Fraser, Yvette Graham, Paco Guzman, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Andre Martins, Makoto Morishita, Christof Monz, Masaaki Nagata, Toshiaki Nakazawa, and Matteo Negri, editors, *Proceedings of the Fifth Conference on Machine Translation*, pages 320–325, Online, November 2020. Association for Computational Linguistics.

[48] Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. Sockeye: A Toolkit for Neural Machine Translation. 2018.

[49] PyTorch. Language Translation with nn.Transformer and torchtext. https://pytorch.org/tutorials/beginner/translation_transformer.html, March 2024.

[50] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. 2019.

[51] Hugging Face. BART - Overview and Usage. https://huggingface.co/docs/transformers/en/model_doc/bart, March 2024.

[52] Taylor Karl. 6 Reasons Why Is Python Used for Machine Learning. https://www.newhorizons.com/resources/blog/why-is-python-used-for-machine-learning, August 2023.

[53] Hugging Face. Translation. https://huggingface.co/docs/transformers/tasks/translation, March 2024.

[54] Hugging Face. Translation. https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/translation.ipynb, March 2024.

[55] OPUS. Resources for English (en) - Russian (ru). https://opus.nlpl.eu/results/en&ru/corpus-result-table, March 2024.

[56] Hugging Face. Data Collator. https://huggingface.co/docs/transformers/main_classes/data_collator#datacollatorforseq2seq, March 2024.

[57] Matt Post. sacrebleu. https://github.com/mjpost/sacreBLEU, January 2024.

[58] Matt Post. A Call for Clarity in Reporting BLEU Scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, October 2018. Association for Computational Linguistics.

[59] Hugging Face. SeamlessM4T-v2 - Overview and Usage. https://huggingface.co/docs/transformers/model_doc/seamless_m4t_v2, March 2024.

[60] Machine Translate. ACL 2014 NINTH WORKSHOP ON STATISTICAL MACHINE TRANSLATION - Shared Task: Machine Translation. https://www.statmt.org/wmt14/translation-task.html, June 2014.

[61] Tiiiger. bert_score. https://github.com/Tiiiger/bert_score, February 2023.

[62] Lightning AI. ChrF Score. https://lightning.ai/docs/torchmetrics/stable/text/chrf_score.html, March 2024.

[63] Google Cloud. Evaluating models. https://cloud.google.com/translate/automl/docs/evaluate, March 2024.

[64] Jungha Son and Boyoung Kim. Translation Performance from the User's Perspective of Large Language Models and Neural Machine Translation Systems. *Information*, 14(10), 2023.

[65] CCJK. Difference Between English and Russian Language. https://www.ccjk.com/difference-english-russian-language/, September 2014.

[66] Jon Martindale. GPT-4 vs. GPT-3.5: how much difference is there? https://www.digitaltrends.com/computing/gpt-4-vs-gpt-35/, April 2023.

[67] The Chartered Institute for IT BCS. Code of conduct for bcs members. https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf, June 2022.

[68] Gerrit De Vynck. Duolingo cuts workers as it relies more on AI. https://www.washingtonpost.com/technology/2024/01/10/duolingo-ai-layoffs/, January 2024.

[69] Dimitar Sht. Shterionov and Eva Vanmassenhove. The Ecological Footprint of Neural Machine Translation Systems. *CoRR*, abs/2202.02170, March 2021.

[70] TechPowerUp. NVIDIA GeForce RTX 4080 Mobile Specs. https://www.techpowerup.com/gpu-specs/geforce-rtx-4080-mobile.c3947, April 2024.

[71] TechPowerUp. NVIDIA GeForce GTX 1080 Ti Specs. https://www.techpowerup.com/gpu-specs/geforce-gtx-1080-ti.c2877, April 2024.

[72] Nowtricity. Current emissions in United Kingdom. https://www.nowtricity.com/country/united-kingdom/, April 2024.

[73] Hugging Face. Language Technology Research Group at the University of Helsinki. https://huggingface.co/Helsinki-NLP, April 2024.

[74] Vijay Kotu and Bala Deshpande. Chapter 2 - Data Mining Process. In Vijay Kotu and Bala Deshpande, editors, *Predictive Analytics and Data Mining*, pages 17–36. Morgan Kaufmann, Boston, 2015.

User Guide

In order to use PETRAI, follow this guide in order to set up and begin your translational foray. The following instructions are for Linux/Mac-based machines. These can either be adapted for Windows or followed using the WSL application. The following scripts were developed in Python 3.10 and must be installed for usage of PETRAI.

Instruction Guide:

1. Download the PETRAI source files.
2. Open the Terminal.
3. Install the requirements via the command: `pip3 install -r requirements.txt`
4. Launch the PETRAI translate program via the command: `python3 petrai_translate.py`
5. Wait for the program to launch and the models to be initialised.
6. When the prompt for input language appears, type ”en” if you intend to use English as your source language and ”ru” if you intend to use Russian as your source language. Press Enter to submit input.
7. Type your input into the `text to translate` field. Make sure to include proper grammar, including punctuation and capital letters. Press Enter to submit input.
8. If you wish to continue, go back to step 6.
9. If you wish to exit the program, use the `'Ctrl + D'` key combination to exit the program.