6CCS3PRJ Final Year
Fine-tuning pre-trained models for English-Polish machine translation

Final Project Report

Author: Mikolaj Deja

Supervisor: Dr Kevin Lano

Student ID: 20010020

Program of study: BSc Computer Science

April 5, 2023

Abstract

Machine translation is a problem known to humanity for many centuries. In the past decades, many advancements were made. Starting with rule-based systems, moving to statistical models trained on vast amounts of data, to neural network approaches. The ease and quality of translation has been improving throughout the years.

This project explores the effectiveness of fine-tuning pre-trained models for English-Polish machine translation. With the increasing popularity of the general language models, the quality of translation that can be achieved by fine-tuning publicly available models can be near state-of-the-art. Another big push in the Machine Translation research community is towards multilingual translation models. However, these models are typically trained on large datasets for a number of language pair and domains, making them less effective for the specific task of English-Polish translation. In this project, a number of multilingual and bilingual pre-trained models were fine-tuned on a number of different datasets. The effectiveness of this approach is evaluated using standard machine translation metrics. The results show that this approach is a viable way of improving the quality of translation for some bilingual models. The quality of translation provided by multilingual models can be significantly improved by fine-tuning, to the point where it is comparable with state-of-the-art models.

Originality Avowal

Acknowledgements

I would like to thank my supervisor, Dr Kevin Lano for his support in this project. My big thanks go to my partner, Maja Jaworska, without whom this project wouldn't be what it is now. I would also like to thank my flatmates and friends, Simon Virag, Noah Rotteveel and Martha Hornett who by now know way more than they ever wanted about machine translation.

Contents

Chapter 1

Introduction

This project is about Machine Translation with Natural Language Processing for the pair of languages of Polish and English. Machine Translation is a sequence to sequence task in which the input in one language is translated into another language. There are multiple approaches to this problem, described in chapter 2. Neural Machine Translation was selected since it produces the best results and does not require domain knowledge for the languages used. The pair of languages that is the subject of this project is described and compared in the same chapter. English is by far the most common language for Machine Translation but the same cannot be said about Polish [10].

As shown in the literature review, Chapter 3, many model architectures and approaches were proposed over the years, including multilingual models [80, 50, 10] and vast web crawling datasets [73, 5]. Several techniques for improving the quality of translation were introduced, such as fine-tuning pre-trained models [16]. A general parent model is pre-trained on massive datasets. The weights can be reused for many different tasks by fine-tuning on a much smaller dataset. This approach results in near state-of-the-art systems at a fraction of a cost, time, and environmental impact of full training. In this project, this technique was utilised for improving the quality of translation for Polish and English. A number of different datasets and models were selected and fine-tuned. The models were state-of-the-art bilingual and multilingual models and the datasets were the largest high quality parallel corpora available.

A practical application was developed, to showcase the best models created in this project. The requirements for the software component of this project are defined in Chapter 4, with a breakdown for the machine learning and the practical application parts. The design and specification of those two components is further detailed in the Chapter 5. The implementation of the application delineated in the Requirements and Design and Specification chapters is shown and discussed in 6. The focus was placed on the Machine Learning pipeline. The results are shown and discussed in the following chapter, 7. The models were compared using standard metrics for Machine Translation. Professional, ethical, legal, and social issues are discussed in Chapter 8. Those include how the Code of Conduct of the British Computing Society was used throughout the project, discussions about ethical AI, licensing, environmental impact, and recommended use cases. The conclusions and future work plans are discussed in the final chapter, 9.

Chapter 2

Background

2.1 Languages

Let me preface this section by saying that I am not a linguist. I am however fortunate enough to know both of the languages in question. I am Polish and I have been speaking English for most of my life.

2.1.1 Polish

Polish is a West Slavic language of the Lechitic group [8]. It is the official language of Poland [15]. Its alphabet consists of the basic 26 letters in the Latin alphabet, with additional 9 letters {*ą, ć, ę, ł, ń, ó, ś, ź, ż*}. The traditional alphabet does not include (*x, q, v*), these only appear in words that originate in other languages. It is phonetic, and with the inclusion of double character phonemes (*sz, cz, rz, dź, dż, ch*), one can learn the pronunciation of all words in the language by learning all sounds [69, p 1].

In terms of grammar, it is highly inflected [69, p 40]. The word order is rather free, but the most common is the subject-verb-object order [69, p 42]. The subject pronoun information is usually captured within the verb and as result dropped [69, p 267].

Nouns belong to one of the three main classes: masculine, feminine and neuter. There are also subclasses for the masculine: animate and inanimate in the singular and human and nonhuman in the plural [69, Chapter 3].

There are seven cases: nominative, accusative, genitive, dative, locative, instrumental, and vocative. The case choice depends mostly on the verb used before the noun [69, Chapter 2].

Adjectives follow the noun's case, class, and number. They most often precede the noun, apart from some fixed phrases where the noun comes first (*język polski* -> *Polish language* directly "language polish") [69, Chapter 4].

Polish verbs have the following categories:

- tense: (past, present, or future)
- person
- number: (singular or plural)
- class: (as previously discussed)
- aspect: (perfective and imperfective)
- mood: (indicative, imperative, conditional)

and are conjugated for all of them [69, Chapter 6]. That information allows Polish to skip auxiliary verbs or pronouns.

Passive-type constructions are made using *być* or *zostać* (*be* and *become*). There also exists an impersonal construction with no subject but with the reflexive pronoun *się* to indicate a general subject [69, Chapter 6.12].

Yes-no questions are asked with the word *czy* (*if*) at the beginning. Negation uses the word *nie* (*no*) before the verb being negated; there also exists a double negative with *nie* and *nigdy* (*never*) used in the same sentence (*Never gonna give you up* would be *Never* ***not*** *gonna give you up* instead) [69, Chapter 5.7.1].

2.1.2 English

English is a West Germanic language. It is the most spoken language in the world and is the language of choice in many professional contexts, such as science, law or media [63]. Organisations such as United Nations, European Free Trade Association or the Olympic Committee use it as one of the official languages [63].

Its grammar, as described in [11, Part III] can be characterised as follows:

Even though it is a typical example of the Indo-European family, instead of a case system it mostly uses analytic constructions and a substantial set of auxiliary verbs, such as do or have.

Nouns in English are only inflected for possession and number. They do not follow cases and have no class, unlike the ones in Polish.

Adjectives appear before nouns and after determiners. They do not match the form of the nouns, as opposed to the ones in Polish.

The pronouns in English have traces of case and class inflection. There is an optional class distinction in third person singular and a distinction between the subjective and objective cases, as well as a possessive form.

The way in which verbs reflect information about tense, mood and aspect is very different in English than in Polish. The majority of the work is done by auxiliary verbs such as *can*, *will*, *shall*, *may*, *might*, *could*, *should*, *would*. Verbs are inflected for tense and aspect, with the third person singular subject requiring the -s suffix in present tense. The only exception is the copula verb *to be* which agrees with the other subjects as well.

Adverbs accompany the verb to provide additional information and are often created from adjectives.

Questions are often constructed with the auxiliary verb *do* and inversion. To achieve negation *not* is used. Many negative constructions require the auxiliary *do* as well.

Having looked at the two languages in question, we can discuss their similarities and differences. Generally speaking, they are both in the Indo-European language family, but are rather distant relatives. Polish is in the Balto-Slavic branch and English is in the Germanic one [67]. Nowadays, both languages share a lot of borrowed words, from Latin or other European languages, but also between each other [18]. They are grammatically quite distinct so translation between the two is a substantial task [69, Chapter 3.7].

2.2 Translation

Machine Translation (MT) from a computer science point of view is a sequence to sequence natural language task. It takes in a sequence in one language and outputs a result which is the translation of that sequence in another language. It is argued, that to perform a good translation, one must first understand the input sequence.

However, translation was a task centuries before computers were utilised to do it.

2.2.1 Human Translation

The need for translation between humans existed probably as long as humans wanted to communicate but used different languages. The first document translated we have the reference of was the Epic of Gilgamesh which was translated from Sumerian to Asian languages in the Mesopotamian era [52].

Famously, the Bible is the most translated book ever with over 3 thousand languages [85].

It seems like religious scripts and the most famous pieces of literature are the most translated documents in the world. A lot of effort was put into it throughout the ages, for instance one of the earliest translations of the Bible in Polish, the *Brest Bible* took over 6 years of manual labour [24].

This is the main reason why in the 20-th century when the computational power became sufficient, many machine translation systems were created [17]. They are utilised by professional translators as an aid, but for most tasks, the cost of hiring a person is too prohibitive at scale. Hence, fully automated systems are widely used throughout the world. For instance, Google Translate is used by millions of users to translate about 100B words every day [14]. It is worth noting, however, that machine translations systems are not perfect, struggling with context, irony, incomplete information, and creative translation [14]. Even with those problems, they are still an incredibly useful tool for providing solid translations in many context and truly a wonder of technology.

2.2.2 Machine Translation

The beginning of machine translation is thought to be in the 9-th century, when Al-Kindi, an Arabic cryptographer, developed techniques for systemic language translation [17]. The techniques he proposed like cryptoanalysis, frequency analysis, probability and statistics are used in machine translation nowadays [17].

In the Western world, there were attempts at developing a universal, perfect and philosophical language in the sixteenth and seventeenth centuries [17]. Actual translation was not viable; the system was just a word-lookup. Only in the 20-th century, did we see some advancements in the machine translation field. In the 1930s, Petr Troyanskii applied for a patent for a translating machine which included a method for an automated bilingual dictionary and a way of encoding interlingual grammatical rules [42].

The [42] provides a detailed history of machine translation. Summarising here, the first formal proposal for computer based machine translation is attributed to Warren Weaver. The system was fairly limited but it was sufficient to stimulate significant funding in the US and encourage other researchers around the world. There came a decade of optimism and a lot of work with no apparent breakthrough. The funding was running low and the US government set up the Automatic Language Processing Advisory Committee (ALPAC), which created a report in 1966 stating that MT was slower, more expensive and less accurate than human translation. They saw no immediate or predictable prospect of useful machine translation. This report was detrimental to the MT research in the United States and had a great impact in the world during the 1970s. The Cold War stimulated the need for rapid translation of scientific and technical documents for Russian-English and English-Russian, even if the output was crude. Later, the demand shifted more towards commercial demands, regarding trade with Europe and Japan. In the 1980s research continued, especially on more advanced methods, for instance indirect translation via intermediary representations. In the 1990s, major changes in the approach took place, when a group from IBM introduced an experiment based on purely statistical methods. There were also experiments in Japan with example-based translation. Both systems needed no syntactic or semantic rules for analysis of texts.

Since 2000, the Statistical Machine Translation (SMT) methods became the dominant ones. The availability of large corpora, open-source software for basic SMT tasks and the availability of metrics for evaluating the systems were huge factors. Nowadays, the neural approach overtook SMT and is being developed rapidly.

Rule Based

As described in [4], the rule-based machine translation (or the classical approach) is based on linguistic information about the source and target languages. The knowledge is usually represented in two broad ways: dictionaries and grammars. The main premise is based on understanding the syntactic structure of the source, mapping that to the structure of the target sentence and filling in all the information. Words are taken from dictionaries and then rules are applied to inflect them appropriately.

Some advantages of this approach include:

- Full control: the person writing the rules is in full control of them.
- No ceiling: one can correct every error that crops up with an appropriate rule.
- Domain independence: rules can be written in a very general form which makes them applicable in many domains.
- No need for a large corpus: one could create a translation system for languages with no texts in common.

The disadvantages are:

- A lot of the linguistic information needs to be input manually
- The rules are hand written and require expert knowledge

- The correctness is based on dictionaries, which are expensive to maintain and develop
- It is difficult to deal with ambiguity, interactions of rules in big systems, or irony

Statistical Machine Translation

As stipulated in [45], Statistical Machine Translation (SMT) is a machine translation paradigm utilising statistical models. The parameters are obtained by analysing bilingual text corpora. Its underpinnings lie in information theory, similarly to what Weaver proposed in 1949. There are many types of SMT systems, such as word-based, where the fundamental unit of translation is a word, or phrase-based, where the translation is done in blocks.

The main advantages are:

- Efficient use of resources: many parallel corpora, no need for manual rule development
- More fluent translations

The disadvantages include:

- High cost of corpus creation
- Difficult to fix rare errors
- Big problems for languages whose word order is significantly different

Neural Machine Translation

In 2014, the use of a neural network model for machine translation was introduced by [9] and it quickly proved superior to SMT. This mirrors the general tendency in Artificial Intelligence to transition towards machine learning in recent years [49]. All the parts of the neural network are trained jointly to maximise the translation performance. There are no more statistical methods; words and internal states are represented using vectors. The model predicts one word at a time, however, this prediction depends on the entire source sequence and the partially translated sequence before it [9]. There were many architectures proposed, and they will be discussed in the next section.

Chapter 3

Review of relevant literature

The previous section introduced the history and the three main approaches used in Machine Translation. In this section, more detail about the findings relevant to this project will be given.

3.1 Neural networks

For Neural Machine Translation, it is important to start from the very basics. What are neural networks?

3.1.1 General ideas

Neural networks are composed of nodes connected by links. In Figure 3.1 the nodes are the colourful circles and links are the arrows in between. As described in [68, Chapter 18.7], each link has a weight $w_{i,j}$ for a link connecting nodes i and j which encodes the strength of the connection. Each node j can compute a weighted sum of its inputs:

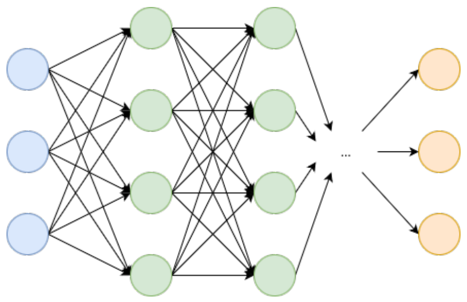


Figure 3.1: A neural network with the input layer in blue, hidden layers in green and output layer in orange

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{3.1}$$

where $a_{i,j}$ is the activation from i to j. It then applies an activation function g to the result of the sum:

$$a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} a_i) \tag{3.2}$$

There are multiple commonly used activation functions:

- Identity: $g(x) = x$
- Binary step: $g(x) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x \geq 0 \end{cases}$
- Sigmoid: $g(x) = \sigma(x) = \frac{1}{1+e^{-1}}$
- Hyperbolic tangent: $g(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Rectified linear unit (ReLU): $g(x) = max(0, x)$
- Softplus: $g(x) = ln(q + e^x)$
- and many more

The activation function ensures that the network as a whole is a non linear function of its inputs.

There are two main types of network architectures: feed-forward networks and recurrent networks. The former has connections only in one direction, whilst the latter can feed its outputs back into the inputs. This allows recurrent networks to have short-term memory.

Networks are arranged into layers. The first layer is called the input layer and in 3.1 is shown in blue. It receives the inputs and propagates it through the network. The green layers are the hidden layers and their number varies depending on the application. The final layer, shown in orange, is the output layer. It allows the network to produce some output to the query asked with the input.

The power of neural networks really comes from their learning abilities. They adjust the weights of the model in order to capture information they have already seen. This improves upon the previous rule based artificial intelligence systems, which are more static. The way this is achieved is through the backpropagation algorithm. As explained in [32], it was introduced by Paul J. Werbos in 1974 and in the coming years rediscovered a couple of times and subsequently used in neural network training. Its basic idea is that the error between the desired output and the actual output is reduced at each step. The network computes the error derivative of the weights. The error is defined in [32] as

$$\mathcal{E} = \frac{1}{2}\sum_j (a_j - \mathbf{d}_j)^2 \tag{3.3}$$

where a_j is the activity function of the jth unit in the top layer and $\mathbf{d}_j$ is the desired output of the jth unit.

The backpropagation algorithm, as defined in [32] consists of four steps:

1. Compute the error derivative with respect to the activity of an output unit

$$EA_j = \frac{\partial \mathcal{E}}{\partial a_j} \tag{3.4}$$

2. Compute the derivative for error with respect to the total input received

$$EI_j = \frac{\partial \mathcal{E}}{\partial in_j} = \frac{\partial \mathcal{E}}{\partial a_j}\frac{\partial a_j}{\partial x_j} \tag{3.5}$$

3. Compute the derivative for error with respect to the weights

$$EW_{i,j} = \frac{\partial \mathcal{E}}{\partial w_{i,j}} = \frac{\partial \mathcal{E}}{\partial in_j}\frac{\partial in_j}{\partial w_{i,j}} = EI_j w_{i_j} \tag{3.6}$$

4. Compute the derivative for error with respect to the output from a unit in the previous layer

$$EA_i = \frac{\partial \mathcal{E}}{\partial a_i} = \sum_j \frac{\partial \mathcal{E}}{\partial in_j}\frac{\partial in_j}{\partial a_i} = \sum_j EI_j w_{i_j} \tag{3.7}$$

By using steps 2 and 4, we can calculate the EAs of a previous layer given EAs of units in one layer. This can be repeated for as many previous layers as we want. Once we know the EA, we can use the steps 2 and 3 to compute the EWs on its inputs. This basic algorithm is used in a slightly more sophisticated form in the training of machine translation models.

3.1.2 Training

Training of actual machine translation models is done in a number of steps, as described in [12]. First, the input data is pre-processed. A vocabulary of N most frequent words is created and the remaining words are mapped to a single "UNK" token. Usually, subword tokenisation is performed, so that the vocabulary size is smaller. For instance, *token* and *tokenisation* are linguistically related and that information can be kept. Then, a minimisation problem is solved. The goal is to minimise the cross-entropy between the predicted target words and the actual target words in the reference. This is an optimisation problem and gradient descent methods such as stochastic gradient descent or `ADAM` can be used. `ADAM` is widely used due to its speed, however it might fail to converge. Often, a learning schedule is designed where the learning rate changes with the training step. Training is done for a large number of iterations or until the model converges. A convergence is achieved when its evaluation on a development set does not change over several iterations. A number of parameters, such as the number and size of hidden layers or the learning rate can be selected so often a hyperparameter search is performed to find the best combination of parameters. For more detail, [12] is recommended.

3.1.3 Recurrent Neural Networks

A number of machine translation systems, especially in the past, utilised recurrent neural networks. Let us discuss them here.

Recurrent neural networks (RNN) are a class of neural networks that don't follow the forward-only rule. Connections between nodes can create a cycle, allowing the output of certain nodes to affect subsequent input to said nodes [1]. That in turn allows temporal dynamic behaviour. They specialise in sequence recognition, for instance text or speech signal. The cycles in the net are a sort of short-term memory [1]. Some machine translation models (or their parts) follow this general architecture. The intuition for that is that the neural network needs to see more than one word at a time and the aforementioned short-term memory is capable of providing this context.

3.2 Evaluation

It became apparent that machine translation is a problem that is not best qualified using simple metrics used throughout machine learning such as `accuracy`, `recall` or `f-score`. In the early systems, the best way of comparing was to human translators. That process was prohibitively expensive so some automatic metrics were proposed with the goal of correlating highly with human evaluation.

3.2.1 BLEU

In [62], the authors propose such a metric called BiLingual Evaluation Understudy. This metric measures the quality of a translation given some references. It is based on modified n-gram precision, which intuitively is a precision metric which only counts the words up to the number they appear in the reference text. The example used in [62] is:

Candidate: the the the the the the the.

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

Modified Unigram Precision = 2/7 because the maximum count of "the" in all the references is 2 and the count in the candidate is 7.

This idea is then extended to n-grams. The shorter grams (1-grams) corresponds to adequacy of the translation and the longer n-grams correlate with fluency. Then, the modified n-gram precisions are combined using geometric average.

The metric also uses sentence brevity penalty to encourage sentences of similar length.

The metric ranges from 0 to 1. Even human translations rarely achieve the score of 1 though. It is important to minimise the variation by taking the average BLEU score over the whole corpus. The score is usually reported as a percentage, instead of decimal values. Thus, score of 24 actually means 0.24.

It is important to compare the BLEU score on the same corpora and the same languages; even changes in the number of references can affect the scores significantly.

3.2.2 SacreBLEU

The BLEU score has some problems. The last sentence in the previous subsection signifies some problems with comparing the score; it turns out it is quite difficult to compare results in practice. In [64], the author points out that BLEU is in fact a parameterized metric and the changes in the parameters can affect the score massively. Most papers using BLEU do not provide the parameters the authors used to generate that score, so comparison is largely impossible. A big factor is also the preprocessing techniques which can also affect the scores greatly. To combat that, the author proposed `SacreBLEU`, a tool for a more standardised scoring. This is achieved by applying its own preprocessing, handling test sets and documenting all the settings used.

All of the evaluation in this project is performed using `SacreBLEU`, utilising the `Evaluate API` from Huggingface [35].

3.2.3 Test sets

It is important to evaluate models on the same test sets. A lot of big projects introduce their own metrics, which will be discussed in later sections. Still, the WMT sets are widely recognised, starting in 2006, when the first Workshop on Machine Translation was held at the NAACL (North American Chapter of the Association for Computational Linguistics Annual Meeting). With time, it branched off into its own Conference on Machine Translation. Every year, companies and researchers participate and create machine translation systems for their translation tasks. Thus, the authors of many papers say that their models were trained for the WMT 2014 German-English task for instance.

3.3 Data

As always with machine learning, the data is as important, if not more important, than the model used. Getting huge parallel corpora was what enabled the rise of statistical machine translation and later neural machine translation. The biggest and most important project is called Open crPUS (`OPUS`) [78]. In 2012, it already had millions of parallel sentences for some of the highest-resource language pairs, such as English-Spanish, English-French, and English-German. `OPUS` contains parallel corpora from many sources, such as legislative and administrative texts from the European Union and United Nations, translated movie subtitles, localisation data from open source software projects and many smaller collections. The resources were made available to the public and the authors created some search and processing tools. By now, it contains hundreds of gigabytes of data, hundreds of millions of sentences for thousands of language pairs. Most of later introduced web crawling datasets were added to `OPUS` so it is safe to say that it is the main source of data for most of machine translation.

3.4 Architectures

As previously mentioned, most machine translation models were recurrent Neural Networks at the early stages. Many different architectures were proposed, to make the quality of the translation higher, to make the models smaller, easier to train and more general. Below, the main steps to the current state-of-the-art are shown.

3.4.1 Encoder-Decoder

In 2014, the Machine Translation world overcame a massive transformation. A novel architecture called RNN Encoder-Decoder was introduced in [9]. It quickly became the most popular model and was the basis of subsequent breakthroughs and introduction of other new architectures built upon it. The model is based on two recurrent neural networks. The first encodes a sequence of characters into a fixed-length vector representation. That is the Encoder. The second one is the Decoder; it decodes the vector into another sequence of symbols. The authors argue that this architecture improves the scores and allows better syntactic and semantic representation of the phrase.

The encoder, as explained in the paper [9], reads the input sequence x sequentially. At each step, the internal state is updated in accordance with

$$\mathbf{h}_{\langle t\rangle} = f(\mathbf{h}_{\langle t-1\rangle}, x_t) \tag{3.8}$$

after reading the end-of-sequence symbol, the internal state of the network is a summary $\mathbf{c}$ of the input.

The decoder creates the output sequence based on the summary $\mathbf{c}$, the previously generated characters and the function h. The relation is as follows:

$$\mathbf{h}_{\langle t\rangle} = f(\mathbf{h}_{\langle t-1\rangle}, y_{t-1}, \mathbf{c}) \tag{3.9}$$

The two networks are trained together to maximise the conditional log-likelihood

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_{\theta}(y_n | x_n) \tag{3.10}$$

where θ is the model parameters set and each (y_n, x_n) is an (input sequence, output sequence) tuple from the training set.

The authors trained the model for English-French translation task of WMT 2014 workshop and got state-of-the-art results at the time with BLEU of 33.30 on the test set. They trained the language modelling on 418 million words and the Encoder-Decoder on 348 million words. The training time and setup was not specified, other than the fact that it was done on GPUs.

The architecture is visualised in 3.2.

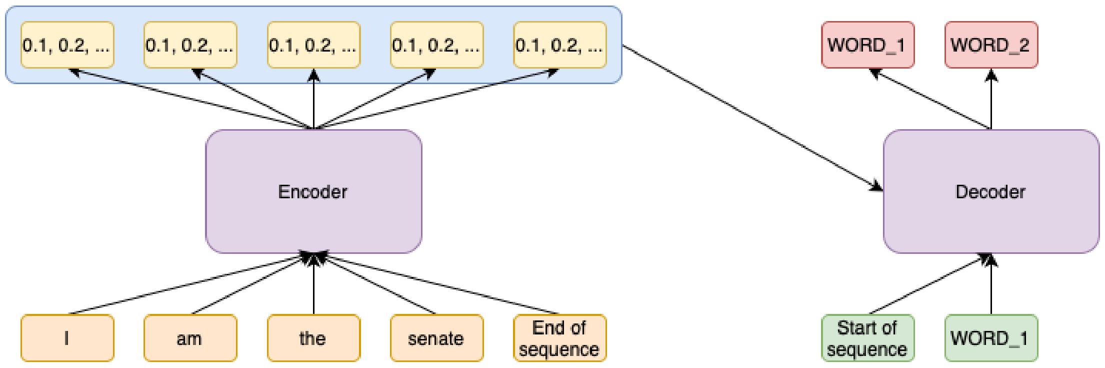


Figure 3.2: The Encoder-Decoder architecture. The input sequence is in orange, the calculated vector representation in blue, the previously generated words in green and the output words in red.

3.4.2 Attention

Let us turn our attention to the next big breakthrough in machine translation - the Attention mechanism. It was popularised in 2015 by a team at Stanford in [48] and was incredibly important to the development of the field. They proposed two architectures for the attention mechanism for machine translation: global approach and local approach.

An attention function is defined in [48] as a mapping of query and a set of key-value pairs to an output, where all the parameters are vectors. The output is calculated as a weighted sum of the values, the weight being computed by a compatibility function of the query with the corresponding key.

In the global attention approach, all source words are attended and in the local one only a subset is considered at a time. Those two types share the fact that at each time step t of the decoding phase, they consider the hidden state $\mathbf{h}_t$ as input to derive a context vector $\mathbf{c}_t$ that contains relevant information for predicting the next target word y_t. In the global attention approach, the model considers all the hidden states of the encoder. The local attention approach does not attend to all the source words, which can be computationally intensive. It focuses on a small window of context. Importantly, it's also differentiable which makes it relatively easy to train.

The authors trained the model on the WMT 2014 English-German task with 4.5 million training sentence pairs. The training of one models took 7-10 days on a single GPU machine. They achieved the BLEU of 23 on the English-German WMT 2014 test set.

3.4.3 Transformers

As the time went on, the architecture became increasingly more complex. Many small improvements were added atop the existing systems, improving the scores, shrinking the memory footprint and improving the speed. In 2017 however, a team from Google proposed a new architecture, in which "attention is all you need" [83]. They based the proposed architecture, called the `Transformer`, on attention only, removing the recurrence and convolution from the neural network.

As most neural machine translation models at this point in time, it follows the encoder-decoder architecture. The encoder in the proposed model consists of a stack of N layers, each composed of two sub-layers. The first is the multi-head self-attention mechanism. The second is a position-wise fully connected feed-forward network. They also employ layer normalisation.

The decoder is also composed of a stack of N layers, each of which has the same two sub-layers as in the encoder and additionally a multi-head attention over the output of the encoder stack. The self-attention is also right-masked to avoid attending to words that haven't been generated yet.

The particular attention mechanism proposed is Scaled Dot-Product Attention. The attention function is computed on a set of queries simultaneously, using the matrices of queries Q, keys K and values V as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{3.11}$$

where d_k are the dimensions of the keys.

The Multi-Head Attention mechanism is the key of this architecture. It is implemented as follows: instead of a single attention function with d_{model} dimensions for keys, values, and queries, they are linearly projected h times with different learned projections to d_k, d_k and d_v dimensions respectively. On each of those, the attention function is performed in parallel. The resulting values are concatenated.

The attention mechanism is used in three places in the model:

1. In the encoder-decoder attention layers, the queries come from the previous decoder layer and the keys and values come from the output of the encoder.

2. In the encoder self-attention layers, all the information comes from the previous output of the encoder.

3. Similarly, in the decoder self-attention layers, all the information comes from the previous output of the decoder.

The attention mechanism can be understood as the context for a word in a sentence. When looking at a particular word in the input sequence, what comes before it and after it affects the role and meaning of the word.

The aforementioned position-wise feed-forward networks consist of two linear transformations with a ReLU (one of activation functions defined above) in between

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{3.12}$$

For the training, the authors used 4.5 million sentence pairs for WMT 2014 English-German task and 36 million parallel sentences for the WMT 2014 English-French task. The training was done on 8 NVIDIA P100 GPUs, with small models being trained for 12 hours and big models for 3.5 days. This resulting architecture allowed the authors to obtain state-of-the-art results on the WMT 2014 English-French translation with the BLEU score of 41.8, and WMT 2014 English-German translation with 28.4 BLEU at a fraction of training cost and time of other systems.

3.4.4 BERT

A crucial development for machine translation recently was the introduction of general language models such as `BERT` in [16]. It stands for Bidirectional Encoder Representations from Transformers and as the name suggests it builds upon the `Transformer` model. It is a language representation model, not specific to translation. It can be pre-trained for deep bidirectional representations from unlabeled text. It takes context from both left and right in all layers. Such a pre-trained model can be subsequently fine-tuned with just one additional output layer to create state-of-the-art models for many language tasks including translation.

As described in [16], `BERT`'s framework comprises of two steps: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data for different language tasks. In fine-tuning, the model is initialised with the parameters obtained in the previous step and further fine-tuned using labeled data specific for the task in question. The model architecture is a multi-layer bidirectional `Transformer` encoder based on [83]. For the pre-training, the model was trained on more than 3.3 billion words. 16 TPU chips were used for 4 days for the model to complete pre-training.

The original paper did not cover translation by itself but `BERT`-like models are the current state-of-the-art systems. The introduction of all-purpose language models means that most of the training time can be skipped and the model can be adjusted to the given task with a relatively small dataset and very little training time. This changed the world of Natural Language Processing as a whole and the smaller tasks, like machine translation.

3.4.5 Pre-trained checkpoints

As previously stated, the introduction of unsupervised pre-training of large neural models dramatically changed Natural Language Processing. Systems such as [16] were subsequently used from publicly available checkpoints to push the state-of-the-art while saving huge amounts of compute time. The authors of [66] pushed the state-of-the-art for sequence generation.

In terms of models used, the authors mixed some plain `Transformer` based models with pre-trained checkpoints from `BERT`, `RoBERTa`, and `GPT-2`. They tried initialising the weights for the encoder and decoder separately and compared the results over a number of tasks, one of which was machine translation.

The authors used the WMT 2014 English-German task, so 4.5 millions of parallel sentences. The training time was not discussed, apart from the fact that the models were trained for 30 epochs. The results were really good for some of the models in question, for instance `BERT2BERT` which achieved the BLEU score of 30.1. They also tried some custom `BERT` checkpoints which improved their score even further.

3.5 Multilingual models

While the classical models for machine translation focus on performing the best possible translation from one source language to one target, in the recent years the idea of multilingual translation has been gaining more traction. The general idea is to leverage the huge and high quality parallel corpora for some high-resource language pairs for improving the quality of translation for low-resource languages. In the paper [12], the authors discuss some of the techniques used for this purpose. The main idea is to use knowledge transferring and allow the model to be able to encode multiple input languages into a shared attention mechanism and then decode to multiple target languages.

Multilingual models are often utilised for low-resource language pairs. It is possible to perform transfer learning to improve low-resource MT significantly [12]. One can train jointly both language pair but fine-tuning is regarded as a better approach.

3.5.1 Polish language models

So far, most of the described models use English-French or English-German. This is because the corpora for those languages are the biggest and of really good quality. The same cannot be said about the target of this project which is English-Polish. This problem was explored in [13]. The authors found that although there are multiple models trained with multilingual corpora, there are only two BERT-base models specifically for Polish. They created two new `BERT`-based models called `Polish RoBERTa`. They trained their models similarly to what the authors of [16] proposed. They managed to push the state-of-the-art for many language tasks in Polish. Nonetheless, machine translation was not one of them.

3.5.2 OPUS-MT

The authors of [80] introduced open source translation services and tools as a part of `OPUS-MT` project. They created and maintain a framework for launching translation services as web applications. They also created over 1000 pre-trained translation models. The models were based on state-of-the-art transformer based architecture. They are trained on data from parallel corpora collected from OPUS [78]. The models are both bilingual and multilingual. The bilingual models for some language pairs are among the state-of-the-art in terms of quality of translation. The multilingual models naturally achieve lower quality translation but they are more general and can be applied in wider domains.

3.5.3 Democratizing Machine Translation with OPUS-MT

Following the previous paper [80], in [81], the authors talk about democratizing machine translation. The proposed `OPUS-MT` project was widened in scope to battle the increasing cost of working on machine translation. As they point out, creating high quality translation models is now incredibly expensive in terms of training, evaluating and maintaining the models. The model size keeps increasing and only a few corporations can afford to pay those costs. The `OPUS-MT` project aims to help with that by providing a range of tools and services for working with data through the `OPUS` project, making thousands of pre-trained models available publicly. It also introduces a data augmentation framework which is particularly important for low resource languages. The authors also introduce a system for easy deploying machine translation at scale. They discuss their collaboration with Huggingface and the European Language Grid.

Finally, the authors discuss the evaluation framework for more fair comparison, using the same development datasets and metrics. They introduce the OPUS leaderboard and competitions such as the Tatoeba Translation Challenge [79].

The Tatoeba Translation competition was introduced in 2020 as a new benchmark for language pair covering over 500 languages [79]. Over 500GB of data for nearly 3000 language pairs was released as a part of the challenge to encourage machine translation development, especially for low-resource language pairs. The data was collected from `OPUS` [78]. As a result of this, many state-of-the-art models were created.

3.5.4 SMaLL-100

In the paper [50], the authors introduce `SMaLL-100` (Shallow Multilingual Machine Translation Model for Low-Resource Languages on 100 languages) which is a distilled model obtained from the `M2M-100` model introduced in [22]. They show how this model outperforms multilingual models of its size and talk about the "curse of multilinguality". It is an important problem, when multilingual models forget their knowledge about high-resource languages when trained for low-resource languages. To battle this, bigger and bigger models were introduced. The authors of this paper chose to distill one of the existing huge multilingual models and showed a big improvement both in the scores and the efficiency of translation for models of similar size. In [51], the authors look into what compressed multilingual machine translation models forget. They investigate what kinds of knowledge is lost during the compression. The model was trained on 456 million parallel sentences sampled from CCMatrix. It was trained for 30 days on 16 GPUs.

3.5.5 No Language Left Behind

The newest advancement came from Meta, with their `No Language Left Behind` (`NLLB`) project [10]. It's been ongoing for many years and in late 2022, they released a number of models. The models are all massively multilingual (200 languages) and the base model is the highest scoring model for many of those language pairs. As mentioned above, training massive models is very expensive and Meta is one of those corporations that certainly can afford it. What's really valuable, they introduced distilled version of their models, that still have very good scores and are much more manageable in size. They also created their own parallel dataset framework which is now commonly used for training and testing. To train NLLB, 18 billion parallel sentences and 51968 GPU hours were used.

Chapter 4

Requirements

The requirements for this project are clearly divided into two parts:

1. The Machine Translation model

2. A practical application showing the model

4.1 Machine Translation model

For the purpose of this project, the aim was to achieve the highest possible quality of translation between the pair of languages English and Polish. The system must be capable of translating both ways and performing it efficiently.

For this project, the expectation is that the language used for translation will be very general. No specific domain is selected so the system is not suitable for use in a specialised context.

The literature review showed that for the selected pair of languages, state-of-the-art models include huge, massively multilingual models. They have incredible capabilities but for any chosen language pair do not necessarily achieve the best scores [50, 51, 10]. Their focus is bringing in low-resource languages. Polish might have been considered one 5 years ago, especially in comparison with French or German, but lately it is rather considered a medium- or high-resource language [10]. The aforementioned multilingual models are capable of achieving high quality translation but they have a big potential for improvement through fine-tuning.

For this project, a number of models were selected, trained and evaluated. For training, a number of different datasets were used. The model and dataset selection will be discussed in detail in Chapter 6. The metric used for evaluation was `SacreBLEU`, introduced in section 3.2.

4.2 A practical application

To show the performance of the best models in practice, it was decided that a web application with translation capabilities should be created.

Its requirements, in the form of user stories, are as follows:

R1. As a visitor to the website, I want to be able to translate English to Polish.

R2. As a visitor to the website, I want to be able to translate Polish to English.

R3. As a visitor to the website, I want to be able to create an account.

R4. As a user of the website, I want to be able to log in to an account I created.

R5. As a logged in user, I want to be able to translate English to Polish.

R6. As a logged in user, I want to be able to translate Polish to English.

R7. As a logged in user, I want to be able to log out.

R8. As a logged in user, I want to be able to change my password.

R9. As a logged in user, I want to be able to see my profile with a list of past translations.

R10. As a logged in user, I want to be able to use my past translations to quickly see the translation again.

Chapter 5

Design and specification

The design and specification of the system can be considered in two distinct parts, same as previously.

5.1 Machine Learning

5.1.1 Technologies

Python was selected as the language used for this task. This is because even though it might not provide the fastest compute time, it is the most common language for Machine Learning and has the biggest support in terms of libraries. Those libraries in turn are highly optimised and might use something like a C++ backend for the computational speed. The `Huggingface APIs` were selected for the relative ease of training, evaluating and subsequent inference. `Pytorch` was selected as the underlying package for machine learning for the ease of use on HPC Create, the selected compute environment.

5.1.2 Model architectures

Even though a number of models were selected for this project, all of them have one of the two following underlying architectures:

- `M2M100`
- `MarianMT`

M2M100

The M2M100 architecture was introduced in the paper [22]. It stands for Many-to-Many multilingual translation model that can translate between any two of 100 languages. As described in the paper, the model is based on the Transformer sequence-to-sequence architecture. It comprises of 12 Encoder and 12 Decoder layers, with 8192 feed-forward network size and 1024 embedding dimension. The total parameter count is 1.2 billion. The training was done with `Adam` optimiser and warm-up for 4000 steps with label smoothing of 0.1.

The model architecture was later reused with different parameter numbers for different model versions. Notably, the main No Language Left Behind (NLLB) model [10] has 3.3 billion parameters.

MarianMT

MarianMT is not an architecture per se, it is a framework for very fast creating, training, and evaluating models in C++ [43]. The implementation details can differ slightly between different models. The models created for the [81] project are Transformers based, with 6 encoder and decoder layers each. There are 8 attention heads for each layer and the feed-forward network size is 2048. They have about 77 million parameters each. They are about 310MB which makes them much easier to handle than some of the big M2M models.

5.2 Practical application

5.2.1 Technologies

Python with Django was selected as the framework for the practical application. The main reason is the ease of creating full stack applications in Python, which is then incredibly easy to connect with the `Huggingface Inference API` for using the trained models in the application.

5.2.2 Architecture

The practical application is designed to be a simple web application in Django, used to showcase the best models created in this project.

It follows the Model-View-Controller (MVC) architectural pattern, although Django refers to it as Model-View-Template (MVT) pattern.

- Model: The model part is responsible for managing data and interacting with the database.

In this case, the application extends the `AbstractUser` model by adding an email field. It also introduces one model: the `Translation` which stores the information related to the translations done by users.

- View: The view is responsible for handling requests and returning responses. In case of this application, the views are defined using `View` classes and provide functionality for `GET` and `POST` methods. They retrieve the necessary information from the database, perform checks, create objects, and use the `Translator API` to make translations.

- Template: The template is responsible for how the data is displayed on the web page. In this case, it is HTML with parameters and some icons. For styling, Bootstrap was used.

Additionally, Django includes several other components to support the MVC architecture, such as:

- URL Dispatcher: It is responsible for mapping the incoming requests with the correct views.

- Middleware: It intercepts requests and responses and performs some additional actions such as authentication, logging, and caching.

- Forms: Forms are used to handle user input and validate data. In the case of this application, `LogIn`, `SignUp`, `Translation`, and `ChangePassword` functionalities are implemented using Forms.

The Unified Modelling Language (UML) representations for the models used is shown in Figure 5.1. As described above, only the `Translation` and `User` models were implemented, the other models are auto-generated by Django. The figure was generated using [82].

5.2.3 Pages

It was decided, that for the requirements described in section 4.2, a small number of pages would be sufficient.

The pages were designed as follows:

- Home page

- Log in page

- Sign up page

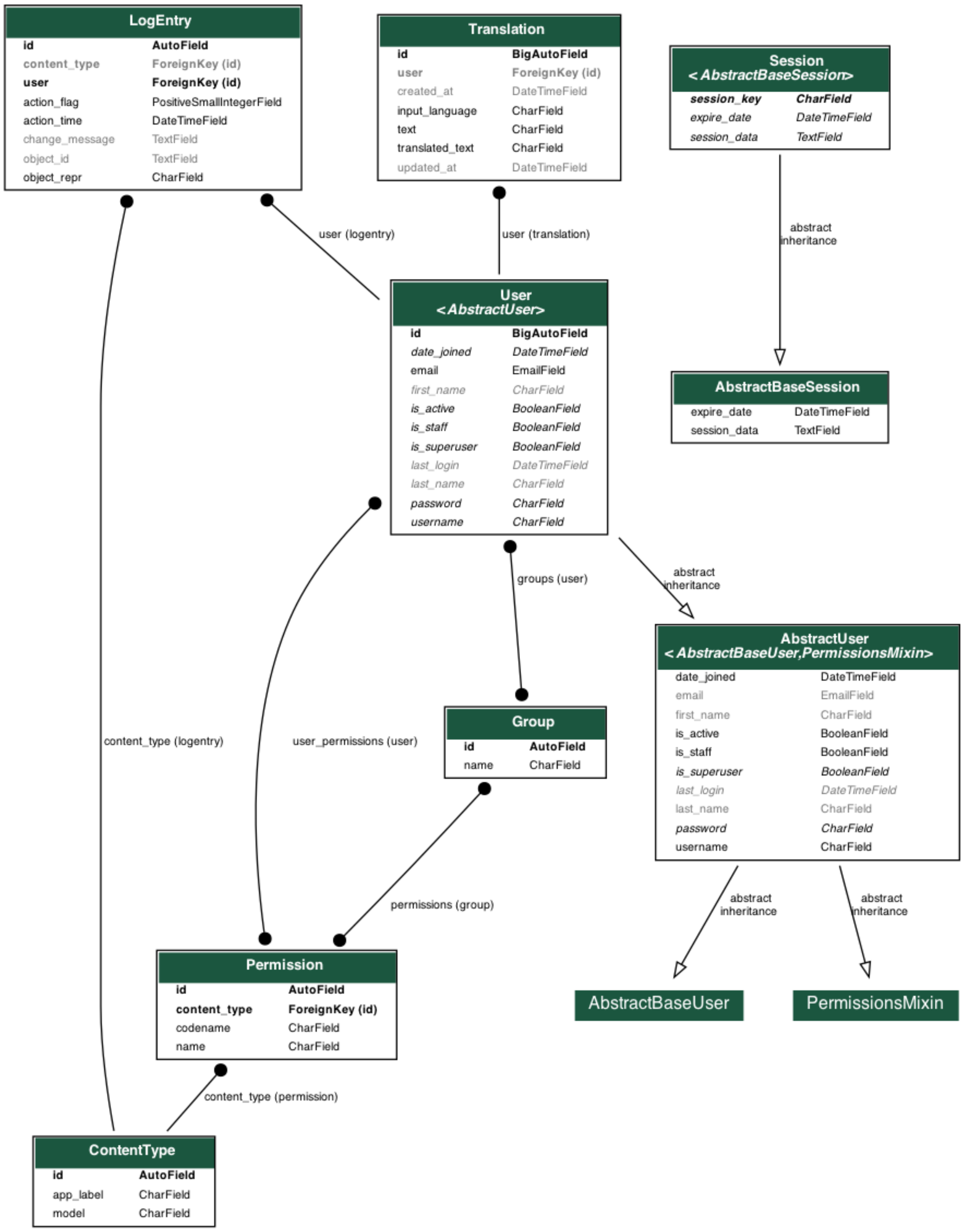


Figure 5.1: A UML representation for the models in the Translator Application.

- Password change page
- Profile page

Home page

The home page is the main page of the application. It houses the main functionality, through textboxes that allow translation. There is a dropdown menu to pick the language the translation is done from and the input box. There is a button which allows the translation request to be submitted. After clicking the button, the user will be able to translate from English to Polish and from Polish to English, depending on what they choose in the dropdown. This covers the requirements R1 and R2.

The home page displays links to the log in and sign up pages. After the user is logged in, they can use the same functionality, translating from Polish to English and vice versa. This covers the requirements R5 and R6.

Log in page

The log in page allows a registered user to log in, using the `LogIn` form. The form contains two fields: the username and password. The form will not accept an empty input and will provide visual indication of that. This ensures the requirement R4 is implemented.

Sign up page

The sign up page allows a visitor to create an account. To do that, the `SignUp` form is used. It asks for a username, an email address, password and password confirmation. Some restrictions are placed on the password: it cannot be too similar to the username or email address, cannot be too common, must contain at least 8 characters and cannot be entirely numeric. The password confirmation must match the password provided, otherwise the form will provide visual feedback about it. It will also indicate that fields cannot be empty. This page ensures the requirement R3 is implemented.

Password change page

The password change page allows a logged in user to change their password. It uses the `ChangePassword` form which takes the old password, the new password and the new password confirmation. The rules for password choice are the same as for the sign up form. The form also gives visual feedback if the passwords do not match or the rules are violated. This implements requirement R8.

Profile page

The profile page is available to logged in users. The page displays some basic information about the user and their history of translations. The history is a list that shows the previous translations made by the user, sorted by date, descending. The items in the list are hyperlinks that lead to the home page with the input box and language dropdown filled in with the correct information. When a logged in user performs a translation, that translation is stored in a database and then displayed on the profile page. Importantly, the same translations are not stored multiple times, only one copy is stored and the associated date is updated. The profile page covers the requirements R9 and R10.

Navbar

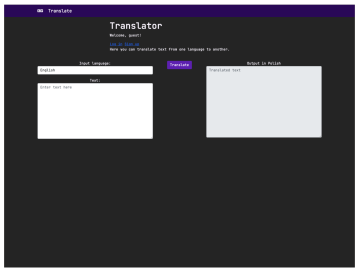
The navbar is a partial included in a number of webpages. It contains a link to the home page, the icon and name of the application, and, if the user is logged in, the user controls. The user controls enable accessing the profile page, the change password page, and the log out button. The log out button, when clicked, logs the user out and redirects to the home page. This implements the requirement R7.

The design of the translator app is shown in Figure 5.2. Bigger images can be seen in the appendix A.2.

5.2.4 Page connections

The pages were linked with each other; the resulting network is shown in Figure 5.3. There are restrictions in place, prohibiting users from accessing certain webpages at certain times. There are two mechanisms:

- `LogInRequired` - this restriction is placed on webpages that can only be accessed by a logged in user. Those websites are: log out, change password, profile page. If a user tries to access one of them, they will be redirected to the log in page.
- `LogInProhibited` - this restriction is placed on webpages that can only be accessed by a user that is not logged in. Those websites are log in and sign up. If a user tries to access of on them, they will be redirected to home page.


Translate

Translator

Welcome, guest!

Log in Sign up

Here you can translate text from one language to another.

Input language: English

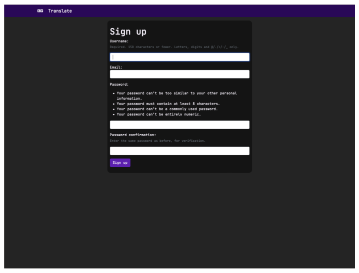
Text: Enter text here

Translate

Output in Polish

Translated text

(a) The home page for a guest


Translate

Sign up

Username:

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email:

Password:

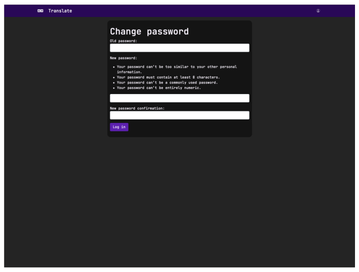
- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation:

Enter the same password as before, for verification.

Sign up

(b) The sign up page


Translate

Change password

Old password:

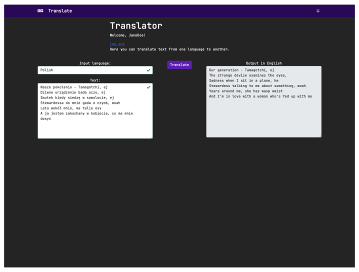
New password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

New password confirmation:

Log in

(c) The change password page


Translate

Translator

Welcome, JaneDoe!

Log out

Here you can translate text from one language to another.

Input language: Polish

Text:

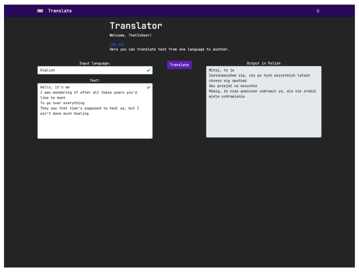
Nasze pokolenie - Tamagotchi, ej
Dziwne urządzenie bada oczy, ej
Smutek kiedy siedzę w samolocie, ej
Stewardessa do mnie gada o czymś, woah
Lata wokół mnie, ma talie osy
A ja jestem zakochany w kobiecie, co ma mnie dosyć

Translate

Output in English

Our generation - Tamagotchi, ej
The strange device examines the eyes,
Sadness when I sit in a plane, he
Stewardess talking to me about something, woah
Years around me, she has wasp waist
And I'm in love with a woman who's fed up with me

(d) The home page for a logged in user who translates from Polish to English


Translate

Translator

Welcome, ThatIsUser!

Log out

Here you can translate text from one language to another.

Input language: English

Text:

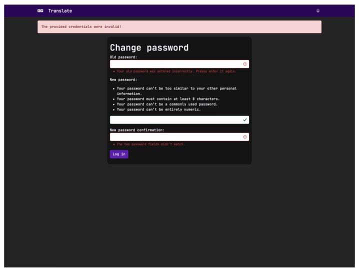
Hello, it's me
I was wondering if after all these years you'd like to meet
To go over everything
They say that time's supposed to heal ya, but I ain't done much healing

Translate

Output in Polish

Witaj, to ja
Zastanawiałem się, czy po tych wszystkich latach chcesz się spotkać
Aby przejść na wszystko
Mówią, że czas powinien uzdrowić ya, ale nie zrobić wiele uzdrawiania

(e) The home page for a logged in user who translates from English to Polish


Translate

The provided credentials were invalid!

Change password

Old password:

- Your old password was entered incorrectly. Please enter it again.

New password:

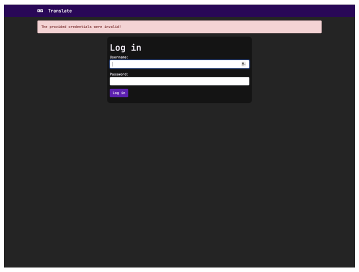
- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

New password confirmation:

- The two password fields didn't match.

Log in

(f) The change password page with incorrect password confirmation


Translate

The provided credentials were invalid!

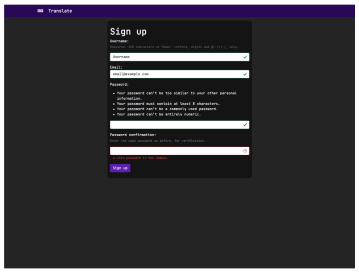
Log in

Username:

Password:

Log in

(g) Log in page with incorrect credentials


Translate

Sign up

Username:

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Username

Email:

email@example.com

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

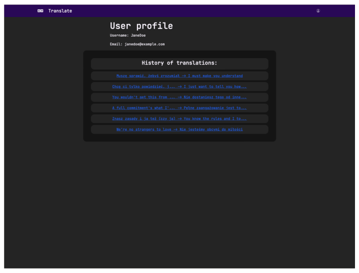
Password confirmation:

Enter the same password as before, for verification.

- This password is too common.

Sign up

(h) The sign up page with too common password


Translate

User profile

Username: JaneDoe

Email: janedoe@example.com

History of translations:

Muszę sprawić, żebyś zrozumiał → I must make you understand
Chcę ci tylko powiedzieć, j... → I just want to tell you how...
You wouldn't get this from ... → Nie dostaniesz tego od inne...
A full commitment's what I'... → Pełne zaangażowanie jest to...
Znasz zasady i ja też (czy ja) → You know the rules and I to...
We're no strangers to love → Nie jesteśmy obcymi do miłości

(i) The profile page with the history of translations

Figure 5.2: The design of web pages.

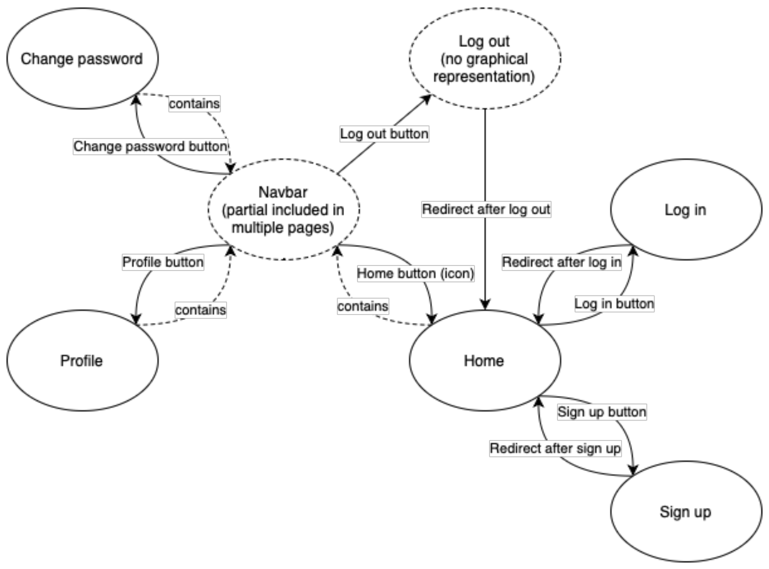


Figure 5.3: A graph representation of the pages included in the application with links between them.

Chapter 6

Implementation

6.1 Machine translation

The main task of the project was to create a system capable of machine translation. With the convenience of some existing libraries such as `Huggingface Transformers` [40], the task of translation requires three lines of code. One can create a `pipeline` for the task of translation and provide the pair of languages in question as below:

```
from transformers import pipeline

translator = pipeline(task="translation_en_to_fr")
translator("I love machine translation")
```

This outputs a translation of the input in French. Now, three lines of code is possibly not enough for a final year project, so I decided to do a bit more. The focus of this project was placed on fine-tuning existing bilingual and multilingual models on English-Polish parallel datasets in order to achieve high quality translations.

6.1.1 Dataset selection

The most important thing for machine learning is data. High quality parallel corpora are necessary for successful applications of machine translation with the neural approach. Thus, multiple datasets were tested and evaluated. The `Huggingface Dataset` library [34] was used for easy dataset manipulation. For practical reasons, the largest datasets were constrained. The bigger the dataset the longer the training is and the infrastructure in use had limited resources. The particular datasets were selected based on their size and quality. At the end, the following train sets were used:

- KDE4 - a parallel corpus of KDE4 localisation files [78, 56]
- Opus100 - an English-centric multilingual dataset [87, 78, 59]
- ParaCrawl - a big web crawling multilingual dataset [5, 78, 61]
- CCMatrix - a massive multilingual dataset from web crawling [73, 22, 78, 54]

KDE4

This dataset is a part of OPUS and comes from KDE4 localisation files [78, 55]. It was made available on the Huggingface Hub [56].

It is a rather toy dataset for this project, it has about 200 thousand parallel sentences for English and Polish and it was the first dataset that was used due to the speed of training. The domain is very technical; a lot of examples talk about plugins and use a lot of proper names.

Opus100

Opus100 is a dataset created in [87]. It was created by random sampling from the OPUS collection [78]. It is English centric and it contains 100 languages. For each, up to 1 million sentences were selected for the training set and 2000 for each test and development sets. The data was chosen randomly, no attempt was made to balance the different domains. A filter was applied during sampling to make sure the test and validation data does not appear anywhere in the train data. The dataset is now available in the OPUS collection [58]. It was made available on the Huggingface Hub [59].

It contains 1 million parallel sentences for English and Polish. The quality is rather high and the data covers a lot of different domains.

ParaCrawl

ParaCrawl is a dataset created as a part of the [5] project. It was created by crawling hundreds of thousands of websites using open source tools and was the largest publicly available parallel corpus for many language pairs at the time. A pipeline was created for harvesting the parallel corpora from multilingual websites and past web crawls. It was used for official European Union languages and a small selection of others. The pipeline consists of several steps: document alignment, sentence alignment, sentence pair filtering, and comparable corpus mining. For the amount of data generated, its quality is surprisingly high but manual inspection shows some irregularities. It is shared publicly in the `OPUS` collection [60]. It was made available on the `Huggingface Hub` [61].

It provides about 3 million parallel sentences for English and Polish. `ParaCrawl` is used to train models in two ways in this project. First, a training on 1 million sentences sampled from it was performed and then the whole train set was used (apart from a separate validation set used in both workflows). This decision was made for the speed of training. Some training configurations with the full `ParaCrawl` dataset took a couple of hundreds of hours.

CCMatrix

The paper [73] introduced by far the biggest dataset - `CCMatrix`. It was obtained through mining bitexts by comparing huge collections of monolingual data. The authors achieves scales previously unheard of - 10.8 billion parallel sentences in 90 different languages. The resulting data is freely available and the authors demonstrate its quality on a number of machine translation benchmarks. The focus of the paper was on the problem of scaling of known pipelines to billions of sentences and making the process highly parallelisable. Their mining procedure has three steps: text extraction, index creation, and mining. Another difference with previous work such as `ParaCrawl`, is that the alignment is not limited to English, which creates unprecedented amounts of data for non-English language pairs. The data is available as a part of the `OPUS` collection [53]. It was made available on the `Huggingface Hub` [54].

It has about 70 million parallel sentences for English and Polish. For this project, 1 million sentences was sampled to avoid very long training times. The quality of the data is similar to that of `ParaCrawl` - very impressive for datasets of this size but with some visible problems.

6.1.2 Model selection

A selection of models available through the `Huggingface Hub` [38] was chosen. The focus was placed on relatively small multilingual models. The size of the model is crucial for making this task viable. Fine-tuning a larger model requires much more time and computation power.

The models that were used are as follows:

- gsarti/opus-mt-tc-en-pl - a small bilingual model ported by Gabriele Sarti from the Helsinki NLP models made for the `OPUS-MT` project [81, 70]

- Helsinki-NLP/opus-mt-pl-en - a small bilingual model developed by the Helsinki NLP group for the `OPUS-MT` project [81, 29]

- Helsinki-NLP/opus-mt-en-mul - a small multilingual model developed by the Helsinki NLP group for the `OPUS-MT` project [81, 30]
- Helsinki-NLP/opus-mt-mul-en - a small multilingual model developed by the Helsinki NLP group for the `OPUS-MT` project [81, 31]
- alirezamsh/small100 - a massively multilingual compact model [51, 50, 3]
- facebook/nllb-200-distilled-600M - a massively multilingual model developed by Meta researchers distilled to 600M parameters [10, 19]

The underlying model architectures were described in 5.1.

gsarti/opus-mt-tc-en-pl

This model is based on the `MarianMT` architecture. It was released as a part of the `OPUS-MT` project [81] and included in the `Tatoeba challenge` [79]. It was ported from C++ to Python by Gabriele Sarti [70]. It is a bilingual model, with English as the source, and Polish as the target language. As most `OPUS-MT` models, it is a `Transformers` based model with 6 layers of encoders and decoders. There are 8 attention heads for each layers and the feed-forward network size is 2048. The total number of trainable parameters is 77419008 (77.4 million).

Helsinki-NLP/opus-mt-pl-en

This model is based on the `MarianMT` architecture. It was released as a part of the `OPUS-MT` project [81] and included in the `Tatoeba challenge` [79] and made available on the `Huggingface Hub` [29]. It is a bilingual model, with Polish as the source, and English as the target language. As most `OPUS-MT` models, it is a `Transformers` based model with 6 layers of encoders and decoders. There are 8 attention heads for each layers and the feed-forward network size is 2048. The total number of trainable parameters is 76614656 (76.6 million).

Helsinki-NLP/opus-mt-en-mul

This model is based on the `MarianMT` architecture. It was released as a part of the `OPUS-MT` project [81] and included in the `Tatoeba challenge` [79] and made available on the `Huggingface Hub` [30]. It is a one-to-many multilingual model, with English as the source, and more than 300 possible target languages. As most `OPUS-MT` models, it is a `Transformers` based model with 6 layers of encoders and decoders. There are 8 attention heads for each layers and the feed-forward network size is 2048. The total number of trainable parameters is 76962816 (76.9 million).

Helsinki-NLP/opus-mt-mul-en

This model is based on the `MarianMT` architecture. It was released as a part of the `OPUS-MT` project [81] and included in the `Tatoeba challenge` [79] and made available on the `Huggingface Hub` [31]. It is a many-to-one multilingual model, with more than 300 possible source languages, and English as the target language. As most `OPUS-MT` models, it is a `Transformers` based model with 6 layers of encoders and decoders. There are 8 attention heads for each layers and the feed-forward network size is 2048. The total number of trainable parameters is 76962816 (76.9 million).

alirezamsh/small100

`SMaLL-100` is a multilingual model, covering more than 10 thousand language pairs introduced in [50, 51] and available on the `Huggingface Hub` [3]. Its architecture is based on `M2M-100`. It is a distilled `M2M-100` model, showing competitive results in comparison with the parent model, while being significantly smaller and faster. It is `Transformers` based and contains a 12-layer encoder and a 3-layer decoder. The embedding dimensions for both the encoder and decoder are 1024 and the feed-forward network dimensions are 4096 for both layers. There are 16 attention heads. The total number of trainable parameters is 332735488 (332.7 million).

facebook/nllb-200-distilled-600M

The `NLLB-200 distilled 600M` variant was introduced in the `NLLB` project [10] and was made available on the `Huggingface Hub` [19]. It is a massively multilingual model, offering translation between 200 languages. It is based on the `M2M-100` architecture. It is a large `Transformers` based model, built from 12 encoder and 12 decoder layers. Each layer contains 16 attention heads, 1024 embedding dimensions and the size of the feed-forward network is 4096. The total number of trainable parameters is 615073792 (615 million).

Bilingual and multilingual models

The behaviour of the multilingual models differs vastly from the more traditional bilingual models in terms of training and later performance. It only makes sense that models that are able to translate between hundreds of languages are not as accurate for one of those pairs as models that are purpose built. Thus, as will be demonstrated in the coming sections, fine-tuning bilingual models shows limited progress. However, fine-tuning multilingual models results in a significant improvement of scores and high quality translations.

6.1.3 Training process

The training process for the neural network models consists of forward passes to calculate the output and the following backpropagation of error. It is described in more detail in Section 3.1. For bigger models and datasets, training can be a very costly and time-consuming process. In practice, it is infeasible to train large models without specialised hardware, such as powerful GPUs or TPUs [84]. At the beginning, some small training loops were tried on Google Colaboratory [26] and discovered some limitations of this platform, namely the fact that the tab must be kept open for the time of the training which becomes infeasible for longer processes. The King's Create High Performance Computing infrastructure [47] was used to solve that problem.

For the training itself, the Huggingface's fine-tuning guide was adapted [41, 36]. Their `Trainer API` [39] was used for automated training loop and the `Evaluate API` [35] for evaluating the resulting models. A 90-10 train-test data split for each of the datasets was used.

Each model was trained for a set number of epochs on each dataset. The appropriate number of epochs was determined experimentally by inspecting the training loss and gain at the end of each epoch. For the largest datasets, a part of the set was taken for training to avoid very long training times. Depending on dataset and model choice, the training time ranged from about 20 to 300 hours. The longest training times occurred for the biggest models (`alirezamsh/small100`) and the largest dataset (full `ParaCrawl`).

The training was abstracted away from the dataset and model choice. An example entry level script is as follows:

```python
from datasets import load_dataset
from transformers import AutoTokenizer

from training import train_model

DATASET = 'opus100'
EPOCH_NUM = 100
max_length = 128
MODEL_NAME = 'Helsinki-NLP/opus-mt-en-mul'

model_name_cleaned = MODEL_NAME.replace('/', '-')
```

```
raw_dataset = load_dataset(DATASET, 'en-pl')

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, return_tensors='pt')


# The preprocessing function was adapted from the huggingface example
# https://huggingface.co/docs/transformers/tasks/translation
def tokenize_help(examples):
    inputs = [ex['en'] for ex in examples['translation']]
    targets = [ex['pl'] for ex in examples['translation']]
    model_inputs = tokenizer(
        inputs, text_target=targets, max_length=max_length, truncation=True
    )
    return model_inputs


train_model(raw_dataset, tokenizer, MODEL_NAME, model_name_cleaned, DATASET,
                                        EPOCH_NUM, tokenize_help, max_length)
```

This script loads the `Opus100` dataset, downloads the tokeniser for the parent model and creates a pre-processing function for tokenising. Then all the data is passed to the `train_model` function.

The function is defined as follows:

```
def train_model(raw_dataset, tokenizer, model_name, model_name_cleaned,
                                        dataset_name, epoch_num, tokenize_help,
                                        max_length, resume_from_checkpoint=False
                                        ):
    global metric
    login('*******API KEY*******')
    split_datasets = raw_dataset['train'].train_test_split(train_size=0.9, seed=
                                            20)
    split_datasets['validation'] = split_datasets.pop('test')
    tokenized_datasets = split_datasets.map(
        tokenize_help,
        batched=True,
        remove_columns=split_datasets['train'].column_names,
    )
    model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
    data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
    metric = evaluate.load('sacrebleu')
```

```
dataset_name = dataset_name.replace('/', '-')

def compute_metrics(eval_preds):
    preds, labels = eval_preds
    # In case the model returns more than the prediction logits
    if isinstance(preds, tuple):
        preds = preds[0]

    decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)

    # Replace -100s in the labels as we can't decode them
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True
                                            )

    # Some simple post-processing
    decoded_preds = [pred.strip() for pred in decoded_preds]
    decoded_labels = [[label.strip()] for label in decoded_labels]

    result = metric.compute(predictions=decoded_preds, references=
                                            decoded_labels)
    return {'bleu': result['score']}

args = Seq2SeqTrainingArguments(
    f'{model_name_cleaned}-{dataset_name}-finetune',
    evaluation_strategy='no',
    save_strategy='epoch',
    learning_rate=2e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=64,
    weight_decay=0.01,
    save_total_limit=2,
    num_train_epochs=epoch_num,
    predict_with_generate=True,
    fp16=True,
    push_to_hub=True,
)
trainer = Seq2SeqTrainer(
    model,
    args,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['validation'],
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
before_training = trainer.evaluate(max_length=max_length)
file = open(dataset_name + '-' + str(epoch_num) + '-' + model_name_cleaned +
                                          '.txt', 'w')
file.write(str(before_training))
file.write('\n')
file.close()

trainer.train(resume_from_checkpoint=resume_from_checkpoint)
trainer.push_to_hub()

after_training = trainer.evaluate(max_length=max_length)
file = open(dataset_name + '-' + str(epoch_num) + '-' + model_name_cleaned +
                                          '.txt', 'a')
file.write(str(after_training))
file.close()
```

The function first authenticates for the `Huggingface Hub`. Then, the dataset is split into `train` and `test` sets. The `split_datasets` are tokenised using the helper function created in the entry script, shown in Listing 6.1.3. The model is downloaded from the `Hub`, together with the data collator and the metric of choice. A `compute_metric` method is defined for evaluation of the model. Training arguments are then defined and training can begin. Before that, an evaluation on the parent model is performed and the results are saved to an appropriately named file. The training is then performed using the `Trainer API` and when it is done the model is pushed to the `Hub`. Another evaluation is performed and the results are then saved to the same file.

6.1.4 Hyperparameter search

For all of the initial training, most hyperparemeters were set to their default values, included in the translation guide [41]. Then, the hyperparameter search in the `Trainer API` with `Optuna` backend [2] was utilised. This proved infeasible for the model and dataset size, especially in terms of memory. The search was ran for a smaller version of the dataset and a smaller number of epochs. Unfortunately, this optimal parameter set for a smaller dataset and number of epochs did not generalise well and did not improve the results of the full trained models.

Hyperparameter search is done in a similar way to the training scripts shown above 6.1.3. The main difference is that instead of `trainer.evaluate()` followed by `trainer.train()`, the following code is used

```
best_trial = trainer.hyperparameter_search(
    direction="maximize",
    backend="optuna",
    n_trials=10,
)
print(best_trial)
# save to file
with open(f"{model_name_cleaned}-{dataset_name}-hyper.txt", "w") as f:
    f.write(str(best_trial))
```

This lack of major improvements is not surprising, the pre-trained models were most likely already optimal and the hyperparameters in the fine-tuning process do not affect the model as much.

6.1.5 Tweaking

One of the datasets, `Opus100`, which looked the most promising in terms of the quality of data, turned out to be quite difficult to use effectively. This is most likely because all the `Helsinki-NLP`, `gsarti`, and `alirezamsh` models were trained on `Tatoeba` data [79] which contains some of the same `OPUS` data. Thus, fine-tuning on `Opus100` for 50 or 100 epochs proved detrimental to the ultimate score. The models were starting to overfit. A different training strategy was used which evaluated the model after each epoch to determine the best cut-off point. To do that, the Huggingface's training guide with the `Accelerate API` was used [33, 37].

6.1.6 Evaluation and comparison

The exact results will be described in the subsequent section 7. Here, more detail about the process will be provided.

Evaluation is a crucial part of machine learning development. There are two separate parts of evaluation of the models in this project.

First, there is evaluation during training to see if the model is improving on the dataset it is trained on. For that, a 90-10 train-test split and the Huggingface `Evaluate API` [35] were used, for calculating the `SacreBLEU` score as described in section 3.2. The evaluation was ran at the beginning of the training and at the end to see the effect of training.

Second, there is a more objective evaluation, on completely unrelated data. This is more fair across all the models but is not necessarily more "correct". It measures the score on the same dataset but, as will be shown in the subsequent section 7, depending on the exact test set used, the scores vary greatly. This evaluation was also done using `SacreBLEU` through the Huggingface `Evaluate API`. The scores for all the models were evaluated on a number of test sets.

All the testing scripts follow the same structure. An example of an entry level script is shown below:

```python
from datasets import load_dataset
from testing import evaluate_models

en_pl_dataset = load_dataset("gsarti/wmt_vat", "wmt20_en_pl")
pl_en_dataset = load_dataset("gsarti/wmt_vat", "wmt20_pl_en")

evaluate_models(en_pl_dataset, pl_en_dataset, "results-wmt.txt")
```

The datasets for en-pl and pl-en directions are loaded.

The `evaluate_models` is defined as follows:

```python
def evaluate_models(dataset_en_pl, dataset_pl_en, filename):
    model_params = get_model_params()
    for param in model_params:
        evaluate_model(
            param['model_name'],
            param['tokenizer_to_use'],
            filename, dataset_pl_en,
            dataset_en_pl, param['source_lang'],
            param['additional_model_data'])



def evaluate_model(model_name, tokenizer_to_use, filename, pl_en_dataset,
                                         en_pl_dataset, source_lang=None,
                                         additional_model_data=None):
    second_lang = 'pl' if source_lang == 'en' else 'en'
    if source_lang == 'pl':
        dataset = pl_en_dataset
    else:
        dataset = en_pl_dataset
    if additional_model_data is not None:
        pipeline_to_use = pipeline(
            'translation',
            tokenizer=tokenizer_to_use,
            model=model_name,
            src_lang=source_lang,
            tgt_lang=second_lang)
    else:
        pipeline_to_use = pipeline('translation', tokenizer=tokenizer_to_use,
                                                   model=model_name)

    task_evaluator = evaluate.evaluator('translation')

    with open(filename, 'a') as file:
        file.write('Evaluating model: ' + model_name + '\n')
    print('Evaluating model: ' + model_name)

    results = get_results(task_evaluator, pipeline_to_use, dataset)

    with open(filename, 'a') as file:
        file.write('Results ' + str(results['score']) + '\n')
    print('Results ' + str(results['score']))


def get_results(task_evaluator_to_use, pipeline_to_use, dataset_to_use):
    results = task_evaluator_to_use.compute(
        model_or_pipeline=pipeline_to_use,
        data=dataset_to_use,
        input_column='source',
        label_column='reference',
        metric='sacrebleu'
    )

    return results
```

The `evaluate_models` function first gets all the model parameters. They are stored as a list of `(model_name, tokenizer, source_language, additional_info)`. The model name is needed to download it from the `Hub`, the tokeniser to be able to tokenise the input. Source language helps determine the tokenisation and additional information is needed for multilingual models who need to know the target language. It then iterates through that list and calls `evaluate_model` on each of them. That sets up the dataset to use and the translation pipeline.

Finally, `get_results` is called to compute the actual results. All of the results are stored in an output file together with the model name.

6.2 Practical application

What good is a language model that can produce high quality translations if it cannot be easily used? To showcase the best models for English to Polish and vice versa, a web application in Django was created. It is a rather simple albeit fairly usable implementation of a translator application.

6.2.1 Using a machine learning model

The backend of the application is written in Python, using the Django framework [65]. For ease of use, the models trained earlier were uploaded to the `Huggingface Hub` [38]. One can then easily use a `pipeline`, as demonstrated above in listing 6.1, to download the models from the `Hub` and use them for inference tasks.

6.2.2 User management

The main functionality of the application is translation which is done using the `pipeline`. To make the experience nicer for the user, all the basic user management functionality was implemented - sign up, log in, log out, user profile page and changing passwords. As described in section 5.2, Django follows the Model-View-Controller architecture.

As mentioned, a `User` model was created by adapting the `AbstractUser` available in Django and a `Translation` model was introduced. The `User` model is very similar to that of Django, the main difference is that an `email` field was added.

For the View part, Django `View` classes were used to deal with `GET` and `POST` requests.

A `LoginProhibitedMixin` was introduced to prevent the user from accessing the `LogIn` and `SignUp` pages if they are already logged in. It redirects the user to the `Home` page. A `LoginRequiredMixin` is utilised to make sure the user is logged in before they can access the `Profile`, `ChangePassword` and `LogOut` pages. The user is redirected to the `LogIn` page. This behaviour was implemented using Django's `LoginRequiredMixin` was added to the `View` classes that are responsible for the pages that require the user to be logged in. The `login_url` must be set to properly redirect. `LoginProhibitedMixin` was implemented and added to the appropriate classes. The `Mixin` was adapted from a project from 5CCS2SEG called *Clucker*.

```python
class LoginProhibitedMixin:
    """The LoginProhibitedMixin is used to prevent logged in users from
                                                            accessing the login and signup pages
                                                            .
    It is adapted from the Clucker project from 5CCS2SEG."""
    redirect_when_logged_in_url = None

    def dispatch(self, *args, **kwargs):
        if self.request.user.is_authenticated:
            return self.handle_already_logged_in(*args, **kwargs)
        return super().dispatch(*args, **kwargs)

    def handle_already_logged_in(self, *args, **kwargs):
        url = self.get_redirect_when_logged_in_url()
        return redirect(url)

    def get_redirect_when_logged_in_url(self):
        if self.redirect_when_logged_in_url is None:
            raise ImproperlyConfigured(
                'LoginProhibitedMixin requires either a value for
                redirect_when_logged_in_url or an implementation for
                get_redirect_when_logged_in_url().')
        else:
            return self.redirect_when_logged_in_url
```

While most views are very standard in terms of implementation, the `HomeView` merits a brief description. In terms of functionality, it is the landing page of the application and contains a greeting and `sign up` and `log in` links if no user is authenticated. If there is an authenticated user, they are greeted by their username. Below, two text boxes are present. On the left, there is the input and on the right there is the output. Above the input box, there is a dropdown menu for choosing the source language. The available options are Polish and English. After having selected the source language and typing in some text to translate, the user can press the `Translate` button. This triggers a `POST` request which is processed by the Django server side. The request contains a `TranslationForm` which is validated. If the form is invalid, the page is rendered again with form error messages. Otherwise, the translate method of the form is called. It checks the input language and performs the appropriate call to the backend. In the case of this application, the backend is incredibly simple due to the `Inference API`. There is then a database lookup for an existing translation object with the same input and language, performed by the same user. If there is one, it is updated instead of creating a new object. This is to avoid clutter on the Profile page which contains a list of past translations. The page is then rendered with the translated text in the output box.

The code for this view is given in 6.2.2.

```
class HomeView(View):
    """View that shows the home page."""
    template_name = 'home.html'
    translated_text = ''
    text = ''
    form = TranslatorForm()

    def get(self, request):
        translation_pk = request.GET.get('translation', None)
        if translation_pk:
            translation = Translation.objects.get(pk=translation_pk)
            self.form = TranslatorForm(initial={'text': translation.text, '
                                                language': translation.
                                                input_language})

        return self.render()

    def post(self, request):
        self.form = TranslatorForm(request.POST)
        if self.form.is_valid():
            self.translated_text = self.form.translate()

            # try to find an existing translation
            try:
                translation = Translation.objects.get(
                    text=self.form.cleaned_data['text'],
                    input_language=self.form.cleaned_data['language'],
                    user_id=request.user.pk
                )
                translation.translated_text = self.translated_text
                translation.save()
            except Translation.DoesNotExist:
                # create a new translation
                translation = Translation.objects.create(
                    text=self.form.cleaned_data['text'],
                    translated_text=self.translated_text,
                    user_id=request.user.pk,
                    input_language=self.form.cleaned_data['language']
                )

            translation.save()
            self.text = self.form.cleaned_data['text']

        return self.render()

    def render(self):
        return render(
            self.request,
            self.template_name,
            {'form': self.form,
            'text': self.text,
            'translations': self.translated_text,
            'second_language': self.get_second_language()})

    def get_second_language(self):
        try:
            if self.form.cleaned_data['language'] == 'English':
                return 'Polish'
            else:
                return 'English'
        except AttributeError:
            return 'Polish'
```

6.2.3 Availability

The application can be run locally. The instructions are included in Appendix B.2.

Chapter 7

Evaluation and findings

7.1 Model evaluation

7.1.1 Understanding BLEU

Before we compare the model scores and results, it is important that we understand what the scores mean. If we interpret the scores as percentages rather than the decimal, the following table, proposed by [25] shows how to interpret certain 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
30-39	Understandable to good translations
40-49	High quality translations
50-60	Very high quality, adequate, and fluent translations
>60	Quality often better than human

A gradient bar proposed by [46] is as follows:

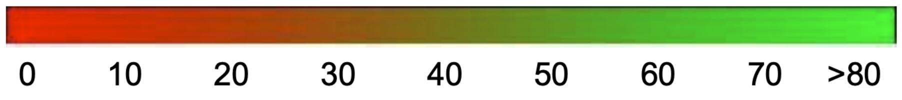


Figure 7.1: A gradient bar showing the quality of translation

7.1.2 Training BLEU scores

The table 7.1 shows the scores achieved by the models before the training and after for each model/dataset pair. The evaluation was done using `SacreBLEU` and the `Huggingface Evaluate`

Model	kde4	opus100	para_crawl/3	para_crawl	CCMatrix
gsarti/opus-mt-en-pl	32.1740	31.7925	42.9107	42.8877	28.2460
	41.5168	29.8405	45.1861	48.4535	27.8847
Helsinki-NLP/opus-mt-pl-en	35.0419	35.7256	46.9451	46.8731	33.3759
	47.1161	33.4208	49.3041	52.6786	33.2961
Helsinki-NLP/opus-mt-en-mul	7.6379	2.1136	5.2623	5.2107	2.0236
	42.0394	27.1127	43.8723	47.2868	26.4820
Helsinki-NLP/opus-mt-mul-en	32.8551	26.2340	31.8774	31.8897	25.1841
	45.3379	30.9884	47.8372	51.3262	32.1390
alirezamsh/small100 en-pl	8.8490	3.4451	6.0168	5.9689	2.7372
	43.0903	28.3231	45.3020	50.2128	26.1180
alirezamsh/small100 pl-en	26.6362	23.5657	32.9474	32.9429	31.9940
	48.0588	32.0733	49.2573	54.4578	31.5621
facebook/nllb-200-distilled-600M en-pl	NA	3.0558	4.9688	NA	2.6648
		25.2004	43.5070		32.5112
facebook/nllb-200-distilled-600M pl-en	NA	2.3795	3.6778	NA	2.2435
		27.7541	48.1756		26.9911

Table 7.1: A table showing the comparison of the BLEU scores before and after training.

and `Trainer APIs`. Each test set was created by taking 10% of the training data by creating a train-test split. As can be seen, the scores improve significantly for most model/dataset combinations, perhaps with the exception of bilingual models trained on `opus100`. Some of the fine-tuned models offer a score about 50 on the training set, which corresponds to very high quality translation.

7.1.3 Testing on additional datasets

As described in section 6.1.6, all of the fine-tuned models were evaluated and compared with the parent models on a suite of test sets. 3 test sets were selected and an average was taken for the results.

- gsarti/wmt_vat - variance-aware test sets introduced in [86], based on the Fifth Conference on Machine Translation task [6]
- facebook/flores - benchmark dataset for machine translation introduced for work on low-resource languages [10, 27, 28]
- cartesinus/iva_mt_wslot - training and test set created for training and evaluating machine translation models in the context of Virtual Assistants [75]

WMT

The WMT test sets, also known as `newstest`, are a well known metric in machine translation. Their history was outlined in section 3.2.3. For this project, a test set based on WMT was used. It is called variance aware machine translation WMT test set and was introduced in [86]. The test sets are created automatically from the WMT competitions. The general idea is that if the sentences are very similar to one another, different machine translation systems will either all answer right or wrong and the differences in score will be small. To combat that, the authors introduce variance-aware filtering, to remove the data that does not provide diversity in the measuring. Experiments and analysis shows that this approach is beneficial for the correlation performance of automatic evaluation results across test sets and languages. It also improves the efficiency of evaluation, since about 60% of the test sets are filtered out. The data set was made available on the `Huggingface Hub` [71].

The sets selected for the evaluation here are `wmt20_en_pl` and `wmt20_pl_en`. They both have 400 sentences. WMT20 was the only year when Polish-English translation was a challenge so there are no more test sets available from WMT.

Flores

The `Flores` benchmark dataset was developed by Meta over a number of years and is now the main multilingual and low-resource language benchmark in the field [10, 27, 28]. The first big iteration, `Flores101` was a test set containing many-to-many parallel data for 101 languages. Since Polish is in the top 10 languages in terms of resources, it was included in that benchmark. Then, `Flores200` added additional languages to a total of 204.

The process of creation was as follows: 3001 sentences were sampled from English-based Wikimedia projects: Wikinews, Wikijunior, and Wikivoyage. The content was professionally translated into 200+ languages. Since all the content is the same, `Flores200` is a many-to-many multilingual benchmark. The inclusion of professional human translators for this task is what makes this test set incredibly high quality. As can be seen later though, the sentences included are quite sophisticated and long which results in relatively low BLEU scores even for good machine translation systems. The data set was made available on the `Huggingface Hub` [20].

For this project, evaluation was done on the combined `dev` and `devtest` sets. The `dev` split contains 997 and the `devtest` split contains 1012 sentences, resulting in the total of 2009.

IVA

The `IVA` set was created by Marcin Sowanski as a training and evaluation dataset for machine translation models used in Virtual Assistant Natural Language Understanding (NLU) context [75]. It contains some additional NLU information which is not used in this project. The data mostly comes from Amazon's `Massive 1.1` [23, 7] and the `Leyzer` project [76]. A small amount of data is also sampled from the `OPUS` collection (`OpenSubtitles`, `KDE`, `CCMatrix`, Ubuntu, Gnome). The data set was made available on the Huggingface Hub [75].

For this project, the test set was used and it contains 5394 parallel sentences.

Own dataset creation efforts

An effort was made to create a new evaluation test set. The general idea was "If all of the datasets used for training have some domains, than a mix test set will cut across domains and thus will be a good generalisation". While this assumption is good on paper, it should have been done before the training. Afterwards, it was impossible to get test data that was not seen by any model.

With the first split,

- 1000 examples from KDE4
- 1000 examples from Opus100
- 1000 examples from ParaCrawl
- 1000 examples from CCMatrix

the best models seemed to be the ones trained on KDE4.

This made no sense, since they are consistently the worst scoring ones for other tests.

Another mix was tried,

- 100 examples from KDE4
- 1300 examples from Opus100
- 1300 examples from ParaCrawl
- 1300 examples from CCMatrix

which produced a much more sensible distribution which still heavily favoured the full ParaCrawl trained models. The results are however fundamentally flawed.

The problem is where exactly the data is coming from.

- 100 examples from KDE4 *train* set
- 1300 examples from Opus100 test set
- 1300 examples from ParaCrawl *train* set
- 1300 examples from CCMatrix *train* set

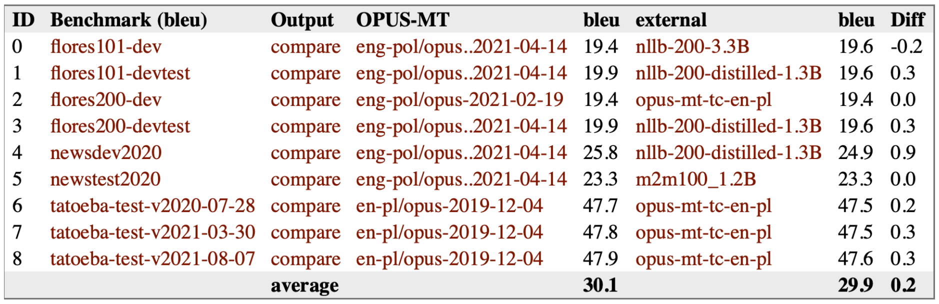
Since the models know (at least a part of) the train set, testing on the data they have already seen makes no sense. That is why separate validation sets are kept for early stopping conditions and training evaluation. Creating a test set from a mixture of training sets favours the models that know more of the sets than other, for instance the models trained of 200 thousand sentences from `KDE4` are then asked about 1000 of those, they will have an advantage over the `Opus100` models which do not know any of the test sets. The mix test set approach is thus infeasible. It would be possible if before the training, some data was put aside for subsequent testing from the train sets. That way, the examples would not be known to any of the models and this test might be a good metric for which model generalises best. At this point in the development of the project this cannot be done but it might be a technique used for other projects.

7.1.4 Score comparison and discussion

The scores for all of the models evaluated on `WMT`, `Flores` and `IVA` test sets are shown in table 7.2

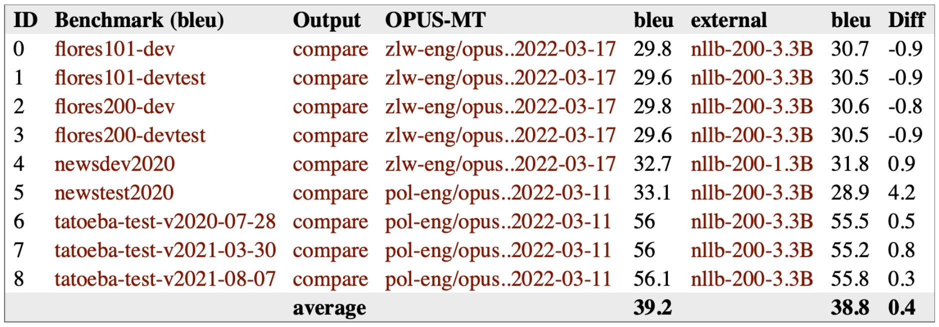
Model	Test	Parent	kde4	opus100	para_crawl/3	para_crawl	CCMatrix
gsarti/opus-mt-en-pl	WMT	23.9568	9.4402	18.6401	18.3960	19.1269	21.4978
	Flores	19.3243	10.0729	16.2144	16.3702	16.9693	18.0919
	IVA	21.6577	15.0135	16.4496	19.9175	20.7564	17.4397
	Average	**21.6463**	11.5089	17.1014	18.2279	18.9508	19.0098
Helsinki-NLP/opus-mt-pl-en	WMT	27.0836	10.5122	21.3460	24.2143	27.8591	22.6275
	Flores	25.6131	13.3708	22.1692	23.1013	24.9931	24.0142
	IVA	26.8445	21.7442	23.3764	27.1739	28.7714	20.4027
	Average	26.5137	15.2091	22.2972	24.8298	**27.2079**	22.3481
Helsinki-NLP/opus-mt-en-mul	WMT	1.5862	6.7916	16.9086	16.2419	18.7067	19.6990
	Flores	1.2076	7.3174	13.4360	15.2980	16.6876	17.2681
	IVA	2.2437	13.9861	16.3908	18.7874	20.2517	17.3465
	Average	1.6792	9.3650	15.5785	16.7758	**18.5487**	18.1045
Helsinki-NLP/opus-mt-mul-en	WMT	17.7349	8.1706	18.1763	23.6789	27.9135	22.2378
	Flores	18.4275	10.5287	19.7691	22.4319	24.3612	23.1831
	IVA	19.4891	17.2139	20.1560	25.9096	27.2979	19.0260
	Average	18.5505	11.9711	19.3671	24.0068	**26.5242**	21.4823
alirezamsh/small100 en-pl	WMT	2.1158	8.5600	19.1775	17.5006	20.8312	20.3080
	Flores	1.5892	9.5190	15.6288	15.5393	17.4684	17.4243
	IVA	6.1844	14.6917	16.9494	19.4603	21.0765	15.9665
	Average	3.2965	10.9236	17.2519	17.5000	**19.7920**	17.8996
alirezamsh/small100 pl-en	WMT	2.6102	2.7366	19.5285	23.3407	26.6330	22.5210
	Flores	1.5514	5.1803	20.9452	21.3172	24.5728	22.6863
	IVA	5.5992	12.2989	16.6471	23.1621	24.7405	16.0901
	Average	3.2536	6.7386	19.0402	22.6066	**25.3154**	20.4324
facebook/nllb-200-distilled-600M en-pl	WMT	2.0036	NA	14.9459	18.7648	NA	22.2834
	Flores	1.3253		12.6295	15.7129		17.6746
	IVA	3.3697		8.7065	12.3226		12.1967
	Average	2.2329		12.0940	15.6001		**17.3849**
facebook/nllb-200-distilled-600M pl-en	WMT	8.2844	NA	14.8167	22.1415	NA	20.9444
	Flores	11.4550		18.4771	22.1326		23.1920
	IVA	5.1466		15.2926	18.8591		19.1361
	Average	8.2953		16.1955	21.0444		**21.0908**

Table 7.2: The comparison of all BLEU results on the test sets for all models with their parents. For each model family, the highest score is shown in bold.



ID	Benchmark (bleu)	Output	OPUS-MT	bleu	external	bleu	Diff
0	flores101-dev	compare	eng-pol/opus..2021-04-14	19.4	nllb-200-3.3B	19.6	-0.2
1	flores101-devtest	compare	eng-pol/opus..2021-04-14	19.9	nllb-200-distilled-1.3B	19.6	0.3
2	flores200-dev	compare	eng-pol/opus-2021-02-19	19.4	opus-mt-tc-en-pl	19.4	0.0
3	flores200-devtest	compare	eng-pol/opus..2021-04-14	19.9	nllb-200-distilled-1.3B	19.6	0.3
4	newsdev2020	compare	eng-pol/opus..2021-04-14	25.8	nllb-200-distilled-1.3B	24.9	0.9
5	newstest2020	compare	eng-pol/opus..2021-04-14	23.3	m2m100_1.2B	23.3	0.0
6	tatoeba-test-v2020-07-28	compare	en-pl/opus-2019-12-04	47.7	opus-mt-tc-en-pl	47.5	0.2
7	tatoeba-test-v2021-03-30	compare	en-pl/opus-2019-12-04	47.8	opus-mt-tc-en-pl	47.5	0.3
8	tatoeba-test-v2021-08-07	compare	en-pl/opus-2019-12-04	47.9	opus-mt-tc-en-pl	47.6	0.3
		average		**30.1**		**29.9**	**0.2**

(a) The OPUS leaderboard for English Polish



ID	Benchmark (bleu)	Output	OPUS-MT	bleu	external	bleu	Diff
0	flores101-dev	compare	zlw-eng/opus..2022-03-17	29.8	nllb-200-3.3B	30.7	-0.9
1	flores101-devtest	compare	zlw-eng/opus..2022-03-17	29.6	nllb-200-3.3B	30.5	-0.9
2	flores200-dev	compare	zlw-eng/opus..2022-03-17	29.8	nllb-200-3.3B	30.6	-0.8
3	flores200-devtest	compare	zlw-eng/opus..2022-03-17	29.6	nllb-200-3.3B	30.5	-0.9
4	newsdev2020	compare	zlw-eng/opus..2022-03-17	32.7	nllb-200-1.3B	31.8	0.9
5	newstest2020	compare	pol-eng/opus..2022-03-11	33.1	nllb-200-3.3B	28.9	4.2
6	tatoeba-test-v2020-07-28	compare	pol-eng/opus..2022-03-11	56	nllb-200-3.3B	55.5	0.5
7	tatoeba-test-v2021-03-30	compare	pol-eng/opus..2022-03-11	56	nllb-200-3.3B	55.2	0.8
8	tatoeba-test-v2021-08-07	compare	pol-eng/opus..2022-03-11	56.1	nllb-200-3.3B	55.8	0.3
		average		**39.2**		**38.8**	**0.4**

(b) The OPUS leaderboard for Polish English

Figure 7.2: The OPUS leaderboard for the language pairs considered in the project.

7.1.5 Low BLEU score?

An attentive reader having seen the results might question the quality of the translation. This is a valid point, however the appartent low score is generally the case for the chosen test sets. The BLEU scores are much lower for some of the selected test sets, than for instance for the `Tatoeba` test set, which is another commonly used benchmark. In table 7.3, a small number of models is shown evaluated on 5 different test sets. The scores for `WMT`, `Flores`, and `IVA` are generally in the 20s which is above described as "The gist is clear, but has significant grammatical errors". The scores for `Tatoeba` and `opus_euconst` are much higher however, some climbing up to 50s, which corresponds to "Very high quality, adequate, and fluent translations". This same tendency can be seen in the OPUS leaderboard, as shown in figure 7.2 [57], which was introduced in [81]. Directly comparing scores makes little sense since BLEU depends so much on so many factors, but the scores achieved in this paper are comparable with the state-of-the-art systems. In the leaderboard for English-Polish, the scores for `flores200-devtest` and `flores200-dev` are nearly 20 and the score for `tatoeba-test-v2021-08-07` is 47.9. The reason for this is that the test sets differ vastly in terms of the length of sentences and the "difficulty" of the translation. `Flores` and `WMT` have longer, more sophisticated sentences than `Tatoeba`. The first three parallel sentences in `Tatoeba` are:

- A baby deer can stand as soon as it's born.
- Abandon ship!
- A bear can climb a tree.

and the first three in `Flores` are:

- On Monday, scientists from the Stanford University School of Medicine announced the invention of a new diagnostic tool that can sort cells by type: a tiny printable chip that can be manufactured using standard inkjet printers for possibly about one U.S. cent each.
- Lead researchers say this may bring early detection of cancer, tuberculosis, HIV and malaria to patients in low-income countries, where the survival rates for illnesses such as breast cancer can be half those of richer countries.
- The JAS 39C Gripen crashed onto a runway at around 9:30 am local time (0230 UTC) and exploded, closing the airport to commercial flights.

This tendency can be seen throughout the test sets which makes the vast differences in scores make sense.

Also, in table 7.3 it can be seen that the scores for the models fine-tuned used `ParaCrawl` are much better for the `WMT`, `Flores`, and `IVA` test sets. The models fine-tuned on the `Opus100` dataset are much better for the `Tatoeba` and `opus_euconst` sets though. This is most likely due to the fact that the `Tatoeba` and `opus_euconst` sets are a part of the `OPUS` collection, and `Opus100` was created by randomly sampling all of `OPUS`. Thus, the models would be trained on data from the same domain or maybe even know some of the sentences from training.

Model	WMT	Flores	IVA	Tatoeba	opus_euconst
MikolajDeja/gsarti-opus-mt-tc-en-pl-para_crawl-finetune	19.1269	16.9693	20.7564	36.3076	31.6512
MikolajDeja/gsarti-opus-mt-tc-en-pl-opus100-finetune	18.6401	16.2144	16.4496	41.1963	42.8206
MikolajDeja/Helsinki-NLP-opus-mt-pl-en-para_crawl-finetune	27.8591	24.9931	28.7714	46.4735	42.1135
MikolajDeja/Helsinki-NLP-opus-mt-pl-en-opus100-finetune	21.3460	22.1692	23.3764	49.3316	54.0756

Table 7.3: A table showing the BLEU scores for a small selection of models for different datasets.

Translation direction

There is one more interesting question that can be asked about the results, both presented in this paper and in the OPUS leaderboard in figure 7.2. The scores for the direction Polish - English are much higher than the opposite. Why could that be the case? The author of [69] provides a number of tricky grammar rules for the pair of languages. In the context of machine translation, it is likely due to the way information is encoded in the language itself. For instance, verbs in Polish are conjugated in many more ways than in English and all of (*jem*, *zjem*, *pojem*, *dojem*, *nadjem*, *wyjem*) are verbs in first person present tense from the verb *to eat* and all of the prefixes add some nuance to the meaning (the translations could be: *I am eating*, *I will eat it all*, *I will eat a lot*, *I will finish eating*, *I will start eating*, *I will eat until the food is gone*). During the preprocessing of the data, a lot of that nuance will be lost so the model will not be able to choose one of the correct forms. This is to do with the limit in vocabulary size. English utilises auxiliary verbs instead of conjugating the verbs so the number of different forms is smaller, which enables learning more useful words.

Findings

Now, having taken all of this into consideration, we can summarise the findings of this project.

Fine-tuning is an incredibly useful technique for improving the quality of translation. With the right choice of datasets, the quality of translation for multilingual models improves significantly and reaches similar values as state-of-the-art bilingual models. The multilingual models such as those based on `Helsinki-NLP/opus-mt-mul-en` and `Helsinki-NLP/opus-mt-mul-en` and fine-tuned on `ParaCrawl` show a lot of promise. Perhaps with longer training time, more data, or some additional techniques, the models can supersede the bilingual models. Fine-tuning models based on `facebook/nllb-200-distilled-600M` and `alirezamsh/small100` show a significant increase in the quality of translation. This is to be expected for massively multilingual models; during this fine-tuning, the information about other languages was most likely lost. The smaller multilingual models seem to have adapted better. The models based on `Helsinki-NLP/opus-mt-en-mul` and `Helsinki-NLP/opus-mt-mul-en` achieve higher scores, comparable to bilingual models. This might be because even though they are multilingual, they are one-to-many and many-to-one respectively, whereas the other multilingual models are many-to-many models.

Some improvements can also be found for bilingual models. For Polish-English, an improvement was found with the `MikolajDeja/Helsinki-NLP-opus-mt-pl-en-para_crawl-finetune` which achieves a higher average score than the parent model `Helsinki-NLP/opus-mt-pl-en`, which is one of the state-of-the-art models. No direct improvement was found for English-Polish; `gsarti/opus-mt-en-pl` still achieves the highest average.

All in all, this projects clearly shows that fine-tuning is a viable technique for improving the scores of near state-of-the-art bilingual models and allows bringing multilingual models increasingly close to the state-of-the-art even for high-resource language pairs.

7.1.6 Testing inference efficiency

Scoring the quality of the resulting translations is one thing. It is also important, if the models are used in a practical application, to make sure that the efficiency of inference is high. To test that, a test script was created.

```python
import time
from transformers import pipeline
from testing import get_model_params


def time_model(model_name, tokenizer_to_use, source_lang=None,
                                        additional_model_data=None):
    second_lang = 'pl' if source_lang == 'en' else 'en'
    if additional_model_data is not None:
        pipeline_to_use = pipeline('translation', tokenizer=tokenizer_to_use,
                                                model=model_name, src_lang=
                                                source_lang, tgt_lang=
                                                second_lang)
    else:
        pipeline_to_use = pipeline('translation', tokenizer=tokenizer_to_use,
                                                model=model_name)

    # time the model for 10 characters, 100 characters, and 300 characters
    times = ''
    for c in [10, 100, 300]:
        # take an average of 10 runs
        runs = []
        for j in range(10):
            text = 'a' * c
            start = time.time()
            pipeline_to_use(text)
            end = time.time()
            runs.append(end - start)
        times += f'{sum(runs) / len(runs):.2f}, '

    print(f'{model_name}: {times}')
    with open('results-efficiency.txt', 'a') as file:
        file.write(f'{model_name}: {times}\n')
```

```
params = get_model_params()


for param in params:
    time_model(param['model_name'], param['tokenizer_to_use'], param['
                                                          source_lang'], param['
                                                          additional_model_data'])
```

Similarly to the previous test scripts, a list of parameters is acquired and then iterated through. The `time_model` function creates a pipeline for the given model and times the inference for input sizes of 10, 100, and 300. Each test is ran 10 times and the average is taken. The timing results are shown in table 7.1.6.

The efficiency of translation is quite important, especially for the end users expecting a responsive application.

Model	Input length	Parent	kde4	opus100	para_crawl/3	para_crawl	CCMatrix
	10	1.34	0.88	0.71	0.54	0.58	27.99
gsarti/opus-mt-en-pl	100	28.11	8.66	28.04	3.43	5.40	28.25
	300	28.53	28.46	28.59	6.70	28.48	28.54
	10	27.85	0.48	0.59	0.58	0.62	0.56
Helsinki-NLP/opus-mt-pl-en	100	14.57	28.89	5.21	28.35	28.37	28.86
	300	29.81	30.27	30.23	29.72	29.75	30.23
	10	3.15	0.59	1.01	3.81	0.70	0.77
Helsinki-NLP/opus-mt-en-mul	100	24.54	5.16	5.28	5.26	6.07	5.25
	300	25.26	7.71	25.16	25.23	25.23	7.40
	10	0.86	0.50	0.50	0.56	0.56	0.53
Helsinki-NLP/opus-mt-mul-en	100	28.04	28.07	8.44	10.16	5.71	27.53
	300	23.76	27.67	10.31	28.76	28.84	19.16
	10	31.97	1.86	1.17	1.16	1.73	0.95
alirezamsh/small100 en-pl	100	31.74	15.55	15.56	7.17	15.47	14.74
	300	32.09	16.07	15.95	15.75	15.84	16.06
	10	1.06	0.96	1.51	1.05	1.73	1.44
alirezamsh/small100 pl-en	100	17.36	11.69	15.68	15.61	15.68	2.22
	300	24.92	15.99	16.02	16.10	16.16	18.56
	10	1.58		1.28	2.28		1.28
facebook/nllb-200-distilled-600M en-pl	100	2.04	NA	2.47	4.21	NA	5.75
	300	29.21		4.46	18.81		28.39
	10	1.22		1.28	1.71		1.87
facebook/nllb-200-distilled-600M pl-en	100	29.05	NA	1.47	4.61	NA	28.83
	300	3.62		1.88	13.05		29.09

Table 7.4: A table showing a comparison of the time taken for translating inputs of length 10, 100, and 300 characters. The time is reported in seconds.

The inference times show no clearly visible pattern but they are reproducible. Similar results were obtained multiple times. It can be seen that many of the fine-tuned models are faster than the parent models. For instance, `alirezamsh/small100 en-pl` is incredibly slow in comparison to the fine-tuned models based on it. `Helsinki-NLP/opus-mt-pl-en` is also slower than `MikolajDeja/Helsinki-NLP-opus-mt-pl-en-para_crawl-finetune`, and has a lower average score. Generally speaking, the `KDE4` based models are the fastest, followed by `Opus100` and `ParaCrawl/3`.

7.1.7 Model choice

Having combined the results of the score and efficiency tests, the following two models were selected to be used in the practical application.

1. `MikolajDeja/Helsinki-NLP-opus-mt-pl-en-para_crawl-finetune` - for Polish to English. This model has the highest average score for translation from Polish to English and offers a significant improvement in terms of efficiency in comparison with its parent model.

2. `MikolajDeja/alirezamsh-small100-en-pl-para_crawl-finetune` - for English to Polish. This model offers the second highest average for translation from English to Polish, bested only by `gsarti/opus-mt-en-pl`. It offers a significant improvements in terms of efficiency in comparison with its parent model and `gsarti/opus-mt-en-pl`.

7.2 Testing the web app

The web application comprises of several parts. Unit tests were created for all of the components.

1. Models

2. Forms

3. Views

There are 87 tests in total, all of them passing. The unit tests provide a 100% coverage on all the code written. The report is shown below.

```
Name                                              Stmts   Miss  Cover
---------------------------------------------------------------------
manage.py                                            10      0   100%
translate/__init__.py                                 0      0   100%
translate/admin.py                                    1      0   100%
translate/apps.py                                     4      0   100%
```

```
translate/forms.py                                                     48      0    100%
translate/migrations/0001_initial.py                                    8      0    100%
translate/migrations/0002_translation.py                                6      0    100%
translate/migrations/0003_alter_translation_user.py                     6      0    100%
translate/migrations/0004_alter_translation_user.py                     6      0    100%
translate/migrations/0005_translation_input_language.py                 4      0    100%
translate/migrations/0006_remove_translation_created_at_and_more.py     4      0    100%
translate/migrations/0007_translation_created_at_translation_updated_at.py  5  0    100%
translate/migrations/__init__.py                                        0      0    100%
translate/models.py                                                    18      0    100%
translate/tests/__init__.py                                             0      0    100%
translate/tests/forms/__init__.py                                       0      0    100%
translate/tests/forms/test_log_in_form.py                              53      0    100%
translate/tests/forms/test_sign_up_form.py                             80      0    100%
translate/tests/forms/test_translator_form.py                          22      0    100%
translate/tests/helpers.py                                              3      0    100%
translate/tests/models/__init__.py                                      0      0    100%
translate/tests/models/test_translation_model.py                       44      0    100%
translate/tests/models/test_user_model.py                              44      0    100%
translate/tests/views/__init__.py                                       0      0    100%
translate/tests/views/test_change_password_view.py                     51      0    100%
translate/tests/views/test_home_view.py                                85      0    100%
translate/tests/views/test_log_in_view.py                              52      0    100%
translate/tests/views/test_logout_view.py                              26      0    100%
translate/tests/views/test_profile_view.py                             40      0    100%
translate/tests/views/test_sign_up_view.py                             61      0    100%
translate/views.py                                                    128      0    100%
translator/__init__.py                                                  0      0    100%
translator/settings.py                                                 25      0    100%
translator/urls.py                                                      4      0    100%
translator_backend.py                                                   9      0    100%
-----------------------------------------------------------------------------------------
TOTAL                                                                 847      0    100%
```

The unit tests are set up using fixtures for some default database objects. There is a `TestCase` class for each Model, View and Form. For instance, the tests for the `Translator` form are as follows:

```python
from django.test import TestCase

from translate.forms import TranslatorForm

"""Tests of the translator form."""


class TestTranslatorForm(TestCase):
    def setUp(self):
        self.form_input = {
            'text': 'Hello World',
            'language': 'English'
        }

    def test_form_contains_required_fields(self):
        form = TranslatorForm()
        self.assertIn('text', form.fields)
        self.assertIn('language', form.fields)

    def test_form_accepts_valid_input(self):
        form = TranslatorForm(data=self.form_input)
        self.assertTrue(form.is_valid())

    def test_form_rejects_blank_text(self):
        self.form_input['text'] = ''
        form = TranslatorForm(data=self.form_input)
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors['text'], ['This field is required.', '
                                                Please enter some text.'])

    def test_form_rejects_blank_language(self):
        self.form_input['language'] = ''
        form = TranslatorForm(data=self.form_input)
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors['language'], ['This field is required.'])
```

There are 4 unit tests, each starting with the word `test`, which allows Django to find them. `test_form_contains_required_fields` checks if the form has the fields that are expected. `test_form_accepts_valid_input` checks if a form with some valid input results in a valid form. `test_form_rejects_blank_text` and `test_form_rejects_blank_language` check that the form correctly does not accept blank fields and provides error messages. The other unit tests are built in a similar way. Each test checks one expected behaviour, both positive and negative. The unit tests for views are more complex, since they require simulating requests. An example can be seen below.

```
def test_post_home_with_user(self):
    self.client.login(username=self.user.username, password='Password123')
    count_before = Translation.objects.count()
    response = self.client.post(self.url, {'text': 'Hello there!', 'language': '
                                            English'})
    count_after = Translation.objects.count()
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'home.html')
    self.assertInHTML('Hello there!', response.content.decode('utf-8'))
    self.assertInHTML('Witaj!', response.content.decode('utf-8'))
    self.assertEqual(count_after, count_before + 1)
    translation = Translation.objects.last()
    self.assertEqual(translation.text, 'Hello there!')
    self.assertEqual(translation.translated_text, 'Witaj!')
    self.assertEqual(translation.user, self.user)
```

The above test checks if a `POST` request when a user is logged in results in a correct page. It checks the status code, the template used, if the text to be translated and the translation result are present, if the `Translation` object was created correctly and it recorded the correct information.

Chapter 8

Legal, Social, Ethical and Professional Issues

The legal, social, ethical and professional issues for this projects were considered from a number of different points of view. First of all, the project as a whole adhered to the British Computing Society Code of Conduct. Secondly, the main focus of the project was developing an Artificial Intelligence system for translation. Ethical AI development was a core consideration of the work, as shown below. The future usage of created application and models is described later.

8.1 British Computing Society

As outlined in the British Computing Society code of conduct, there are four main points [74]. All of them were adhered to throughout the project:

1. Public Interest - no personal identifiable data was used in this project.
2. Professional Competence and Integrity - good practices were followed, no competence was claimed without proof.
3. Duty to Relevant Authority - there was no conflict with the Authority, here being my Project Supervisor.
4. Duty to the Profession - this project is focused on sharing knowledge and understanding of the relevant technologies. It can serve as a guide to fellow members should they want to enter the field of fine-tuning machine translation models.

8.2 Ethical Artificial Intelligence development

It is crucial to develop Artificial Intelligence in an ethical manner. Machine learning is notorious for struggling with bias, explainability, and rectification.

Since this project does not deal with the whole process of development of machine learning models, but only fine-tuning existing pre-trained models, the focus of this consideration is placed on model/dataset selection.

Licensing agreements were followed for all the models selected, as shown before:

- Helsinki-NLP/opus-mt-mul-en - this model was released as part of [81] and featured in [79] and is free for academic use (CC-BY 4.0 License)
- Helsinki-NLP/opus-mt-en-mul - this model was released as part of [81] and featured in [79] and is free for academic use (CC-BY 4.0 License)
- Helsinki-NLP/opus-mt-pl-en - this model was released as part of [81] and featured in [79] and is free for academic use (CC-BY 4.0 License)
- gsarti/opus-mt-tc-en-pl - this model was released as part of [81] and featured in [79] and is free for academic use (CC-BY 4.0 License)
- alirezamsh/small100 - this model was released as part of [50]
- facebook/nllb-200-distilled-600M - this model was released as part of [10] under a CC-BY-NC 4.0 license

Similarly, licensing agreements were followed for all of the datasets used:

- KDE4 - released as a part of the open source `OPUS` project [78]
- Opus100 - released as a part of the open source `OPUS` project [87, 78]
- ParaCrawl - released under CC0 license [5, 78]
- CCMatrix - released as a part of the open source `OPUS` project [73, 22, 78]
- gsarti/wmt_vat - created in [86] based on data released as a part of the Fifth Conference on Machine Translation [6] for academic purposes
- facebook/flores - released as a part of the `NLLB` project [10] under Creative Commons Attribution Share Alike 4.0
- cartesinus/iva_mt_wslot - released under (CC-BY 4.0 License) for research purposes [75]

8.2.1 Environmental impact

Training machine learning models has a huge adverse environmental impact. This comes from the fact that a lot of computational time is required to train big neural networks. Carbon emission estimates are rather imprecise but the general metric is power consumption estimates of GPU devices [10]. The fine-tuning approach offers much shorter training times and thus produces much smaller carbon footprint [81]. This approach is much more environment friendly, than full training cycles [72].

8.2.2 The use of machine translation

It can be argued that machine translation is a way to allow people from different cultures and backgrounds to more easily communicate and exchange ideas [81, 10]. In that sense, further developments in this field allow the voice of people who use less known languages to be heard better. The recent focus on bringing in low-resource languages is certainly a step in the good direction [10].

8.2.3 Possible biases

As always with Artificial Intelligence, there can be biases in the system introduced through the choice of data.

There are for instance issues with gendered languages where machine translation systems are likely to connect professions with genders [77]. When translating from a language with no gender inflection on the nouns like English (a *chef* could be any gender) to a language with gender inflection like Polish (*kucharka* is female and *kucharz* is male), the gender bias is picked up.

Another interesting problem is with what the audience of the translation might understand. In [44], the author discusses how a human translator changed Heath Ledger to Tom Cruise in a translation from English to Spanish, which was later picked up by Google Translate. This was not an error, but rather a good reformulation; this was an example of a famous male movie star and the human translator judged that the audience in Argentina would be more familiar with Tom Cruise than Heath Ledger. In this context, it can be judged as a good translation but in other contexts, it might not. Machine Translation systems will not have any of that reasoning and might swap people out in a completely wrong context. This can lead to the translation not being culturally relevant or appropriate

8.3 The use of the application

The project was made as a research tool and the practical application is to showcase the models trained as a part of this project. It is not suitable for critical systems and the translation quality cannot be relied on. It is also trained on general data so it is not suitable for any domain specific contexts.

8.4 Models trained

The models trained as a part of this application were made publicly available on the `Huggingface Hub`. They are meant for research in machine translation. No responsibility is taken for wrong results of the translation. The whole list of models is available in Appendix A.1.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

Fine-tuning pre-trained machine translation models proved a viable way of improving the quality of translation for bilingual and multilingual models. The improvement is very significant for multilingual models, bringing them very close to bilingual models in terms of results. An improvement was found for the Polish-English translation with a fine-tuned model, `MikolajDeja/Helsinki-NLP-opus-mt-pl-en-para_crawl-finetune`, which achieves a higher average score than the parent model `Helsinki-NLP/opus-mt-pl-en`, which is one of the state-of-the-art models. The quality of translation for the fine-tuned multilingual models is very close to state-of-the-art models and perhaps with some additional training, more data, or some other techniques, they could achieve the best scores. The best models are showcased in a practical application - a simple translator web app.

9.2 Further work

There are many ways in which this project can be improved. As in many chapters before, this can be broken down to the machine translation part and the practical application.

9.2.1 Machine translation

The project can be taken further by considering different available pre-trained models, different datasets and some additional techniques. Hyperparameter tuning could be performed at scale and possibly improve the models further. Training can possibly be ran for longer and on larger datasets. The computational resources for this project were limited; with more time and compute power, possibly with distributed training, even better results could be achieved.

Data augmentation

As described in [21], data augmentation is a technique widely used in many fields of machine learning. It has been successfully applied to machine translation too and many projects (especially focused on low-resource languages) use it to improve the scores [10]. The general idea is to create bigger and higher quality training sets.

Backtranslation

Backtranslation is one data augmentation technique which is very common in the field. As described in [10], which uses it extensively, it is a technique which involves creating parallel corpora from monolingual data through translating the original data and then using the parallel sentences for training. The initial translation is done by a teacher model and can be used to improve the scores of models. For instance, in all of [10, 80, 87, 22] the authors improve significant improvements after performing backtranslation, sometimes by a couple of BLEU points.

Domain manipulation

The domain of the training and test data can be manipulated to achieve the highest scores possible. The simplest way of doing that is using a separate part of the training set as a test set later on. Before the training, a random sample from the training set is taken out to be the test set. This looks fair but the results can be much higher because most datasets have some sort of domain. For instance, if the training set was the Bible and the test set was from KDE4, the results would be much lower than if the test was also from the Bible. This however, does not improve the model, just the scores obtained at the end.

Training techniques

There are many training techniques that are used in bigger projects for making sure the model is as good as it can be which were not explored here. Those include different optimisers, varying learning schedules, warm up steps, regularisation, pre-normalisation, sharding for distributed learning, and many more.

New models

New models are released all the time. The `NLLB` project [10] was released in late 2022 so it was incorporated into the project. A new big batch of `OPUS-MT` models were released after the training phase of the project (in late January) [80, 79, 81]. Should a similar project be done in the future, they might be included. They are multilingual and operate on subgroups and language families. This allows better knowledge sharing so they achieve better scores.

- Helsinki-NLP/opus-mt-ine-en - Indo-European to English
- Helsinki-NLP/opus-mt-en-ine - English to Indo-European
- Helsinki-NLP/opus-mt-ine-ine - Indo-European to Indo-European
- Helsinki-NLP/opus-mt-sla-en - Slavic to English
- Helsinki-NLP/opus-mt-en-sla - English to Slavic
- Helsinki-NLP/opus-mt-zlw-en - West Slavic to English
- Helsinki-NLP/opus-mt-en-zlw - English to West Slavic
- Helsinki-NLP/opus-mt-tc-big-zlw-en - West Slavic to English, about twice as many parameters as the other `OPUS-MT` models

Some of those models, together with `NLLB` are the current state-of-the-art according to the OPUS leaderboard [57].

However, the same techniques and principles can be applied. The results might be better, since the parent models are better by themselves but even the same training and evaluation scripts can be applied.

9.2.2 Practical application

To be able to use the practical application in the real world, one would probably introduce more features. The user interface can be made better, possibly with some additional functionalities such as detecting the language used or more dynamic translation without fully refreshing the webpage. Also, user feedback option could be useful for continuous improvement of the models. The user corrections could be picked up and the model retrained periodically.

There already exist many more sophisticated machine translation products, Google Translate being the biggest and most recognisable. It is rather dubious that a simple Polish-English web application would attract many users, it was only created to show the trained models in a real application. If the fine-tuning was done on some domain specific data, perhaps a specialised tool could be useful, for instance in medical or legal context.

Bibliography

[1] Oludare Isaac Abiodun et al. “State-of-the-art in Artificial Neural Network Applications: A survey”. In: *Heliyon* 4.11 (Nov. 2018). DOI: 10.1016/j.heliyon.2018.e00938.

[2] Takuya Akiba et al. “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[3] alirezamsh. *alirezamsh/small100*. URL: https://huggingface.co/alirezamsh/small100. (accessed: 18.03.2023).

[4] D.J. Arnold et al. *Machine Translation: an Introductory Guide*. London: Blackwells-NCC, 1993. URL: http://www.essex.ac.uk/linguistics/clmt/MTbook/.

[5] Marta Bañón et al. “ParaCrawl: Web-Scale Acquisition of Parallel Corpora”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 4555–4567. DOI: 10.18653/v1/2020.acl-main.417. URL: https://aclanthology.org/2020.acl-main.417.

[6] Loïc Barrault et al., eds. *Proceedings of the Fifth Conference on Machine Translation*. Online: Association for Computational Linguistics, Nov. 2020. URL: https://aclanthology.org/2020.wmt-1.0.

[7] Emanuele Bastianelli et al. “SLURP: A Spoken Language Understanding Resource Package”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 7252–7262. DOI: 10.18653/v1/2020.emnlp-main.588. URL: https://aclanthology.org/2020.emnlp-main.588.

[8] The Editors of Encyclopaedia Britannica. *Lekhitic languages. Encyclopedia Britannica*. 2015. URL: https://www.britannica.com/topic/Lekhitic-languages. (accessed: 02.12.2022).

[9] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *CoRR* abs/1406.1078 (2014). arXiv: `1406.1078`. URL: `http://arxiv.org/abs/1406.1078`.

[10] Marta R Costa-jussà et al. “No language left behind: Scaling human-centered machine translation”. In: *arXiv preprint arXiv:2207.04672* (2022).

[11] David Crystal. *The Cambridge encyclopedia of the English language*. eng. Third edition. Cambridge: Cambridge University Press, 2019. ISBN: 9781108528931.

[12] Raj Dabre, Chenhui Chu, and Anoop Kunchukuttan. “A Survey of Multilingual Neural Machine Translation”. In: *ACM Comput. Surv.* 53.5 (Sept. 2020). ISSN: 0360-0300. DOI: `10.1145/3406095`. URL: `https://doi.org/10.1145/3406095`.

[13] Sławomir Dadas, Michał Perełkiewicz, and Rafał Poświata. *Pre-training Polish Transformer-based Language Models at Scale*. 2020. DOI: `10.48550/ARXIV.2006.04229`. URL: `https://arxiv.org/abs/2006.04229`.

[14] Dalibor. *How accurate is Google Translate?* Nov. 2022. URL: `https://phrase.com/blog/posts/is-google-translate-accurate/`.

[15] Norman Davies et al. *Poland. Encyclopedia Britannica.* 2022. URL: `https://www.britannica.com/place/Poland`. (accessed: 02.12.2022).

[16] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: `10.48550/ARXIV.1810.04805`. URL: `https://arxiv.org/abs/1810.04805`.

[17] Quinn DuPont. *The cryptological origins of Machine Translation*. Mar. 2018. URL: `http://amodern.net/article/cryptological-origins-machine-translation/`.

[18] Radoslaw Dylewski and Zuzanna Witt. “Polish loanwords in English revisited”. In: *Linguistica Silesiana* 42 (2021). DOI: `10.24425/linsi.2021.137235`.

[19] Facebook. *facebook/nllb-200-distilled-600M*. URL: `https://huggingface.co/facebook/nllb-200-distilled-600M`. (accessed: 18.03.2023).

[20] Facebook. *Flores dataset*. URL: `https://huggingface.co/datasets/facebook/flores`. (accessed: 18.03.2023).

[21] Marzieh Fadaee, Arianna Bisazza, and Christof Monz. "Data Augmentation for Low-Resource Neural Machine Translation". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 2017. DOI: 10.18653/v1/p17-2090. URL: https://doi.org/10.18653%5C%2Fv1%5C%2Fp17-2090.

[22] Angela Fan et al. *Beyond English-Centric Multilingual Machine Translation*. 2020. DOI: 10.48550/ARXIV.2010.11125. URL: https://arxiv.org/abs/2010.11125.

[23] Jack FitzGerald et al. *MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages*. 2022. arXiv: 2204.08582 [cs.CL].

[24] D.A. Frick. *Polish Sacred Philology in the Reformation and the Counter-Reformation: Chapters in the History of the Controversies (1551-1632)*. Moderm Philology. University of California Press, 1989, pp. 67–68. ISBN: 9780520097407.

[25] Google. *Evaluating models | automl translation documentation | google cloud*. URL: https://cloud.google.com/translate/automl/docs/evaluate#bleu. (accessed: 28.02.2023).

[26] Google. *Google colaboratory*. URL: https://colab.research.google.com/notebooks/intro.ipynb. (accessed: 28.02.2023).

[27] Naman Goyal et al. "The FLORES-101 Evaluation Benchmark for Low-Resource and Multilingual Machine Translation". In: *CoRR* abs/2106.03193 (2021). arXiv: 2106.03193. URL: https://arxiv.org/abs/2106.03193.

[28] Francisco Guzmán et al. "Two New Evaluation Datasets for Low-Resource Machine Translation: Nepali-English and Sinhala-English". In: *CoRR* abs/1902.01382 (2019). arXiv: 1902.01382. URL: http://arxiv.org/abs/1902.01382.

[29] Helsinki-NLP. *Helsinki-NLP/opus-mt-en-mul*. URL: https://huggingface.co/Helsinki-NLP/opus-mt-pl-e. (accessed: 18.03.2023).

[30] Helsinki-NLP. *Helsinki-NLP/opus-mt-en-mul*. URL: https://huggingface.co/Helsinki-NLP/opus-mt-en-mul. (accessed: 18.03.2023).

[31] Helsinki-NLP. *Helsinki-NLP/opus-mt-mul-en*. URL: https://huggingface.co/Helsinki-NLP/opus-mt-mul-en. (accessed: 18.03.2023).

[32] Geoffrey E. Hinton. "How Neural Networks Learn from Experience". In: *Scientific American* 267.3 (1992), pp. 144–151. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24939221 (visited on 12/12/2022).

[33] Huggingface. *Huggingface Accelerate documentation.* URL: https://huggingface.co/docs/accelerate/index. (accessed: 28.02.2023).

[34] Huggingface. *Huggingface Datasets documentation.* 2023. URL: https://huggingface.co/docs/datasets/index. (accessed: 28.02.2023).

[35] Huggingface. *Huggingface Evaluate documentation.* URL: https://huggingface.co/docs/evaluate/index. (accessed: 28.02.2023).

[36] Huggingface. *Huggingface fine-tune example.* URL: https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/translation.ipynb. (accessed: 03.03.2023).

[37] Huggingface. *Huggingface fine-tune example with Accelerate.* URL: https://colab.research.google.com/github/huggingface/notebooks/blob/master/course/en/chapter7/section4_pt.ipynb. (accessed: 03.03.2023).

[38] Huggingface. *Huggingface Hub documentation.* 2023. URL: https://huggingface.co/docs/hub/index. (accessed: 28.02.2023).

[39] Huggingface. *Huggingface Training documentation.* URL: https://huggingface.co/docs/transformers/main_classes/trainer. (accessed: 28.02.2023).

[40] Huggingface. *Huggingface Transformers documentation.* 2023. URL: https://huggingface.co/docs/transformers/index. (accessed: 28.02.2023).

[41] Huggingface. *Huggingface Translation Task Guide.* URL: https://huggingface.co/docs/transformers/tasks/translation. (accessed: 28.02.2023).

[42] John Hutchins. "The history of machine translation in a nutshell". In: *Retrieved December 20.2009* (2005).

[43] Marcin Junczys-Dowmunt et al. "Marian: Fast Neural Machine Translation in C++". In: *Proceedings of ACL 2018, System Demonstrations.* Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 116–121. URL: http://www.aclweb.org/anthology/P18-4020.

[44] Dorothy Kenny. "The ethics of machine translation". In: (2011).

[45] Philipp Koehn. *Statistical Machine Translation.* Cambridge University Press, 2009. DOI: 10.1017/CBO9780511815829.

[46] Alon Lavie. “Evaluating the output of machine translation systems”. In: *Proceedings of the 9th Conference of the Association for Machine Translation in the Americas: Tutorials.* 2010.

[47] King’s College London. *King’s Computational Research, Engineering and Technology Environment (CREATE).* URL: `https://doi.org/10.18742/rnvf-m076`. (accessed: 28.02.2023).

[48] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation.* 2015. DOI: `10.48550/ARXIV.1508.04025`. URL: `https://arxiv.org/abs/1508.04025`.

[49] T.M. Mitchell. *Machine Learning.* McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673. URL: `https://books.google.pl/books?id=EoYBngEACAAJ`.

[50] Alireza Mohammadshahi et al. “SMaLL-100: Introducing Shallow Multilingual Machine Translation Model for Low-Resource Languages”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing.* Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 8348–8359. URL: `https://aclanthology.org/2022.emnlp-main.571`.

[51] Alireza Mohammadshahi et al. “What Do Compressed Multilingual Machine Translation Models Forget?” In: *Findings of the Association for Computational Linguistics: EMNLP 2022.* Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 4308–4329. URL: `https://aclanthology.org/2022.findings-emnlp.317`.

[52] Language Network. *Brief history of translation: Everything you need to know.* Sept. 2021. URL: `https://www.languagenetworkusa.com/blog/brief-history-of-translation-everything-you-need-to-know`.

[53] OPUS. *CCMatrix.* URL: `https://opus.nlpl.eu/CCMatrix.php`. (accessed: 15.03.2023).

[54] OPUS. *CCMatrix dataset.* URL: `https://huggingface.co/datasets/yhavinga/CCMatrix`. (accessed: 18.03.2023).

[55] OPUS. *KDE4.* URL: `https://opus.nlpl.eu/KDE4.php`. (accessed: 15.03.2023).

[56] OPUS. *KDE4 dataset.* URL: `https://huggingface.co/datasets/kde4`. (accessed: 18.03.2023).

[57] OPUS. *OPUS leaderboard.* URL: `https://opus.nlpl.eu/leaderboard/`. (accessed: 15.03.2023).

[58] OPUS. *Opus100*. URL: https://opus.nlpl.eu/opus-100.php. (accessed: 15.03.2023).

[59] OPUS. *Opus100 dataset*. URL: https://huggingface.co/datasets/opus100. (accessed: 18.03.2023).

[60] OPUS. *ParaCrawl*. URL: https://opus.nlpl.eu/ParaCrawl.php. (accessed: 15.03.2023).

[61] OPUS. *ParaCrawl dataset*. URL: https://huggingface.co/datasets/para_crawl. (accessed: 18.03.2023).

[62] Kishore Papineni et al. "BLEU: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia*. 2002, pp. 311–318.

[63] World Population Review. *English Speaking Countries 2022*. 2022. URL: https://worldpopulationreview.com/country-rankings/english-speaking-countries.

[64] Matt Post. "A call for clarity in reporting BLEU scores". In: *Proceedings of the Third Conference on Machine Translation: Research Papers* (2018). DOI: 10.18653/v1/w18-6319.

[65] Django Project. *Django*. URL: https://www.djangoproject.com/. (accessed: 03.03.2023).

[66] Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. "Leveraging Pre-trained Checkpoints for Sequence Generation Tasks". In: *Transactions of the Association for Computational Linguistics* 8 (Dec. 2020), pp. 264–280. DOI: 10.1162/tacl_a_00313. URL: https://doi.org/10.1162%5C%2Ftacl_a_00313.

[67] B.M. Rowe and D.P. Levine. *A Concise Introduction to Linguistics*. Taylor & Francis, 2015, p. 341. ISBN: 9781317349280. URL: https://books.google.co.uk/books?id=ePQ5CgAAQBAJ.

[68] Stuart J. (Stuart Jonathan) Russell. *Artificial intelligence : a modern approach*. eng. 3rd ed. Prentice Hall series in artificial intelligence. Upper Saddle River, N.J: Prentice Hall, 2010. ISBN: 9780132071482.

[69] Iwona Sadowska. *Polish: A comprehensive grammar*. Routledge, 2012.

[70] Gabriele Sarti. *gsarti/opus-mt-en-pl*. URL: https://huggingface.co/gsarti/opus-mt-en-pl. (accessed: 18.03.2023).

[71] Gabriele Sarti et al. *WMT dataset*. URL: https://huggingface.co/datasets/gsarti/wmt_vat. (accessed: 18.03.2023).

[72] Roy Schwartz et al. “Green AI”. In: *Commun. ACM* 63.12 (Nov. 2020), pp. 54–63. ISSN: 0001-0782. DOI: 10.1145/3381831. URL: https://doi.org/10.1145/3381831.

[73] Holger Schwenk et al. *CCMatrix: Mining Billions of High-Quality Parallel Sentences on the WEB*. 2019. DOI: 10.48550/ARXIV.1911.04944. URL: https://arxiv.org/abs/1911.04944.

[74] British Computing Society. *Code of Conduct - British Computing Society*. URL: https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf. (accessed: 01.02.2023).

[75] Marcin Sowanski. *WMT dataset*. URL: https://huggingface.co/datasets/cartesinus/iva_mt_wslot. (accessed: 18.03.2023).

[76] Marcin Sowański and Artur Janicki. “Leyzer: A Dataset for Multilingual Virtual Assistants”. In: *International Conference on Text, Speech, and Dialogue*. Springer. 2020, pp. 477–486.

[77] Gabriel Stanovsky, Noah A. Smith, and Luke Zettlemoyer. “Evaluating Gender Bias in Machine Translation”. In: *CoRR* abs/1906.00591 (2019). arXiv: 1906.00591. URL: http://arxiv.org/abs/1906.00591.

[78] Jörg Tiedemann. “Parallel Data, Tools and Interfaces in OPUS”. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC’12)*. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, pp. 2214–2218. URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

[79] Jörg Tiedemann. “The Tatoeba Translation Challenge – Realistic Data Sets for Low Resource and Multilingual MT”. In: *Proceedings of the Fifth Conference on Machine Translation*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1174–1182. URL: https://www.aclweb.org/anthology/2020.wmt-1.139.

[80] Jörg Tiedemann and Santhosh Thottingal. “OPUS-MT – Building open translation services for the World”. In: *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*. Lisboa, Portugal: European Association for Machine Translation, Nov. 2020, pp. 479–480. URL: https://aclanthology.org/2020.eamt-1.61.

[81] Jörg Tiedemann et al. *Democratizing Machine Translation with OPUS-MT*. 2022. DOI: 10.48550/ARXIV.2212.01936. URL: https://arxiv.org/abs/2212.01936.

[82] Michael Trier and Bas van Oostveen. *Graph models*. URL: https://django-extensions.readthedocs.io/en/latest/graph_models.html. (accessed: 07.03.2023).

[83] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[84] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. *Benchmarking TPU, GPU, and CPU Platforms for Deep Learning*. 2019. DOI: 10.48550/ARXIV.1907.10701. URL: https://arxiv.org/abs/1907.10701.

[85] Wycliffe. *Scripture access statistics*. Oct. 2019. URL: https://web.archive.org/web/20200826123441/https://www.wycliffe.net/resources/scripture-access-statistics/.

[86] Runzhe Zhan et al. "Variance-Aware Machine Translation Test Sets". In: *Thirty-fifth Conference on Neural Information Processing Systems, Datasets and Benchmarks Track*. 2021. URL: https://openreview.net/forum?id=hhKA5k0oVy5.

[87] Biao Zhang et al. *Improving Massively Multilingual Neural Machine Translation and Zero-Shot Translation*. 2020. arXiv: 2004.11867 [cs.CL].