NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

INTERDEPARTMENTAL PROGRAM OF POSTGRADUATE STUDIES
“LANGUAGE TECHNOLOGY”

MSc THESIS

Tradutorium:
An offline cross-platform Machine Translation application

Spyros T. Stravoravdis

Supervisor: George Tambouratzis, Researcher, ILSP / Athena R. C.

ATHENS

OCTOBER 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΙΔΡΥΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΓΛΩΣΣΙΚΗ ΤΕΧΝΟΛΟΓΙΑ»

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Tradutorium:
Μία τοπική εφαρμογή για Μηχανική Μετάφραση

Σπύρος Θ. Στραβοράβδης

Επιβλέπων: Γεώργιος Ταμπουρατζής, Ερευνητής, ΙΕΛ / Ε. Κ. Αθηνά

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2024

MSc THESIS

Tradutorium: An offline cross-platform Machine Translation application

Spyros T. Stravoravdis
S.N.: 7115182100028

SUPERVISOR: **George Tambouratzis**, Researcher, ILSP / Athena R. C.

EXAMINATION COMMITTEE:
George Tambouratzis, Researcher, ILSP / Athena R. C.
Sokratis Sofianopoulos, Researcher, ILSP / Athena R. C.
Vassilis Papavassiliou, Researcher, ILSP / Athena R. C.

Examination Date: October 30, 2024

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Tradutorium: Μία τοπική εφαρμογή για Μηχανική Μετάφραση

Σπύρος Θ. Στραβοράβδης
Α.Μ.: 7115182100028

ΕΠΙΒΛΕΠΩΝ: **Γεώργιος Ταμπουρατζής**, Ερευνητής, ΙΕΛ / Ε. Κ. Αθηνά

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:
Γεώργιος Ταμπουρατζής, Ερευνητής, ΙΕΛ / Ε. Κ. Αθηνά
Σωκράτης Σοφιανόπουλος, Ερευνητής, ΙΕΛ / Ε. Κ. Αθηνά
Βασίλης Παπαβασιλείου, Ερευνητής, ΙΕΛ / Ε. Κ. Αθηνά

Ημερομηνία Εξέτασης: 30 Οκτωβρίου 2024

ABSTRACT

Machine translation has evolved significantly since the field's creation. In the last 30 years the output quality has improved, at first with the use of statistical techniques and later neural networks. A large amount of data was however needed in order to get acceptable results, which practically meant dependencies on major cloud services, with secondary consideration for data privacy. Recently, though, this assumption has been shattered since newer models can be trained and then quantized for size reduction. These models can afterwards be run on affordable, common hardware, opening exciting opportunities for offline machine translation and uses that would previously have been unfeasible.

The creation of an offline machine translation application called Tradutorium is described in detail. It utilizes resources from project Bergamot, which is at the forefront of offline machine translation research and development. It also offers further functionality such as audio and image transcription (using models from the Whisper and Tesseract projects respectively), as well as API integration with the English version of Wiktionary (hosted by the Wikimedia project), for word definitions. Tradutorium would not have been possible without the above projects and the teams that have created or contributed on them.

Tradutorium is available under the open-source Mozilla Public License and its first version targets desktop operating systems. Since it is built using cross-platform technologies, Tradutorium is scalable and can theoretically be extended to run on more platforms, including mobile devices or web browsers. This would require extensions to the existing codebase, including the creation of compatible user interfaces and proper ways to call the various libraries that are being used on each platform. Additional libraries could also be integrated, adding new features or complementing existing ones.

An English - Greek language model is also trained with Marian NMT and then quantized according to the specifications of project Bergamot models. The process that was followed is documented thoroughly.

SUBJECT AREA: Machine translation

KEYWORDS: Machine translation, neural machine translation, neural networks, machine learning, optical character recognition, OCR, language identification, language detection

ΠΕΡΙΛΗΨΗ

Η μηχανική μετάφραση έχει εξελιχθεί σημαντικά από τη δημιουργία του κλάδου. Τα τελευταία 30 χρόνια η ποιότητα του παραγόμενου κειμένου έχει βελτιωθεί, αρχικά με την εκμετάλλευση στατιστικών τεχνικών και αργότερα με χρήση νευρωνικών δικτύων. Χρειαζόταν μεγάλος όγκος δεδομένων για να εξαχθούν αποδεκτά αποτελέσματα, πρακτικά καθιστώντας υποχρεωτική την εξάρτηση από μείζονες διαδικτυακές υπηρεσίες, με το απόρρητο των δεδομένων να είναι δευτερεύουσα προτεραιότητα. Πλέον αυτή η υπόθεση καταρρίπτεται, καθώς τα νεότερα μοντέλα μπορούν να εκπαιδευτούν και στη συνέχεια να μειωθεί το μέγεθος τους μέσω μίας διαδικασίας quantization. Αυτά τα μοντέλα μπορούν στη συνέχεια να εκτελεστούν σε καθημερινό και εύκολα προσβάσιμο hardware, ανοίγοντας νέα μονοπάτια για αυτόματη μετάφραση σε offline περιβάλλοντα και χρήσεις που προηγουμένως δεν ήταν εφικτές.

Η δημιουργία μιας offline εφαρμογής αυτόματης μετάφρασης που ονομάζεται Tradutorium περιγράφεται αναλυτικά. Χρησιμοποιεί πόρους από το project Bergamot, το οποίο βρίσκεται στην πρώτη γραμμή της έρευνας και ανάπτυξης offline μηχανικής μετάφρασης. Προσφέρονται επίσης περαιτέρω λειτουργίες, όπως μεταγραφή ήχου και εικόνας (χρησιμοποιώντας μοντέλα από τα projects Whisper και Tesseract αντίστοιχα), καθώς και ενσωμάτωση με το API της αγγλικής έκδοσης του Wiktionary (το οποίο φιλοξενείται από το Ίδρυμα Wikimedia), για ορισμούς λέξεων. Το Tradutorium δε θα υπήρχε χωρίς τα παραπάνω έργα στα οποία βασίστηκε και τις ομάδες που δημιούργησαν ή συνεισέφεραν σε αυτά.

Το Tradutorium είναι διαθέσιμο υπό την Mozilla Public License (άδεια ανοιχτού κώδικα) και η πρώτη του έκδοση στοχεύει λειτουργικά συστήματα για προσωπικούς υπολογιστές. Επειδή έχει κατασκευαστεί με χρήση τεχνολογιών που υποστηρίζουν πολλαπλές πλατφόρμες, το Tradutorium μπορεί θεωρητικά να επεκταθεί για να εκτελείται σε περισσότερες πλατφόρμες, συμπεριλαμβανομένων κινητών συσκευών ή προγραμμάτων περιήγησης ιστού. Αυτό θα απαιτούσε κάποιες επεκτάσεις στον υπάρχοντα κώδικα, όπως τη δημιουργία κατάλληλων διεπαφών χρήστη (UI) για κάθε πλατφόρμα και χρήση σε αυτές των διαφόρων βιβλιοθηκών από τις οποίες εξαρτάται. Θα μπορούσαν επίσης να ενσωματωθούν πρόσθετες βιβλιοθήκες, προσθέτοντας νέες δυνατότητες ή συμπληρώνοντας τις υπάρχουσες.

Ένα μεταφραστικό μοντέλο αγγλικής - ελληνικής γλώσσας εκπαιδεύτηκε επίσης με Marian NMT και στη συνέχεια υπέστη quantization, σύμφωνα με τις προδιαγραφές του project Bergamot για δημιουργία συμβατών μοντέλων. Η διαδικασία που ακολουθήθηκε τεκμηριώνεται διεξοδικά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική μετάφραση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μηχανική μετάφραση, νευρωνική μηχανική μετάφραση, νευρωνικά δίκτυα, μηχανική μάθηση, οπτική αναγνώριση χαρακτήρων, OCR, αναγνώριση γλώσσας, ανίχνευση γλώσσας

ACKNOWLEDGMENTS

I wish to sincerely thank my supervisor, George Tambouratzis, for his invaluable assistance, thoughtful feedback and guidance over the past year. His insights and direction through obstacles were instrumental to this project's completion and shaped the final result in significant ways.

I would also like to extend my gratitude to my friends and family for their continuous encouragement and support; their presence made this thesis possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. HISTORY OF MACHINE TRANSLATION

1.1 Origins

> *One naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.*
>
> *—Warren Weaver, Letter to Norbert Wiener, March 4, 1947* [1]

Machine translation started taking shape in the late 1940s. One of the first proposals was presented by Warren Weaver in his "Translation memorandum" (which includes the above illustrative quotation). In the text he discussed the possibility of automatic translation without human involvement. Tangible results began to surface in the 1950s, with the first public demonstration happening in 1954, jointly by Georgetown University and IBM. The experiment presented the translation of 250 words and 50 carefully selected Russian sentences into English and created a huge public interest in the field, with the governments of the United States and Soviet Union allocating large-scale funding for machine translation research. Researchers claimed that machine translation could be a solved problem in a few years. The expectations failed to materialize because of the simplistic nature of the experiment; only six grammar rules were used and the vocabulary was limited and carefully selected, with no significant problems compared to those that would be encountered in a less-contained linguistic sample.

The most common techniques used in these early systems were rudimentary, utilizing bilingual dictionaries and hand-coded linguistic rules [2]. They required extensive human input to create rules for syntax and morphology, and they were often limited to specific language pairs and domains. Generative linguistics and transformational grammar were also used to improve the quality of translations. As part of the ongoing Cold War, most efforts by the United States concentrated mostly on the English-Russian / Russian-English language pair, as can be seen by the Georgetown-IBM experiment. The goal was mostly to achieve a rough translation outline for a basic understanding, with articles or reports that seemed interesting being selected for human translation. Soviet Union also focused on strategic language pairs, developing dictionaries and algorithms for translation of English, Chinese, Japanese and German into Russian [3].

In 1966, the Automatic Language Processing Advisory Committee (ALPAC) was set up by the United States government, due to concerns over the effectiveness and efficiency of machine translation efforts. The committee was tasked with assessing the current state of MT research, evaluating its practical applications and analyzing the field's future prospects and direction. Its assessment, known as the ALPAC report, was highly negative, doubting the practical need for MT or a near-future breakthrough and arguing that despite years of research and substantial investment, machine translation had not advanced to a point where

it could be practically and reliably used for large-scale translation tasks. The quality of machine-generated translations was still far below the level needed for practical use, and most systems required extensive human post-editing [1]. It also reasoned that use of human translators was more affordable and recommended focusing on developing tools and technologies that could aid them.

> There is no emergency in the field of translation. The problem is not to meet some nonexistent need through nonexistent machine translation. There are, however, several crucial problems of translation. These are quality, speed, and cost.
>
> Alpac Report, 1966 [4]

As a consequence, the funding for the field in the United States rapidly declined, and since machine translation stopped being a priority, the field stagnated. One of the few machine translation systems to survive the termination of funding was SYSTRAN, which would later serve as the original engine behind Yahoo! Babelfish (nowadays replaced by Microsoft Translator) and certain linguistic pairs in Google Translate. Both services later developed in-house technology that replaced the previous engines.

Nonetheless, research continued in Europe and Canada, in countries where multilingual communities and multinational trade flourished and even imperfect results from working systems could be utilized effectively. The European Commission used SYSTRAN, while Canada developed its own METEO system for weather forecast translations from English to French. During that time, rule-based approaches were the norm.

1.2 Statistical and example-based machine translation

In 1981, Makoto Nagao and his group proposed using methods based on large numbers of translation examples, a technique that is now termed example-based machine translation (EBMT) [5]. Instead of handling increasingly more complex rules as in previous approaches, EBMT translates by finding analogous examples in an existing database of parallel corpora between the source and target languages. If a new sentence is similar to one or more sentences that have been translated before, the system can use the translations of those sentences as a basis for generating the new translation. When a new sentence is to be translated, EBMT searches the database for examples that are similar in structure or content. It looks for matches that can be reused directly or adapted to produce the correct translation. If a full sentence match is not found, the system may break down the sentence into smaller segments or phrases and find corresponding segments in the database. These segments are then recombined to form the final translation. EBMT often needs to adapt the retrieved examples to fit the new sentence's structure. This may involve adjusting word order, conjugating verbs correctly, or inferring translations for parts of the sentence that do not match

exactly [1].

Around the end of the decade, IBM published the results of experiments on systems based purely on statistical methods, beginning the era of statistical machine translation (SMT) [6]. SMT involves collecting and preprocessing large corpora, training translation, language, and distortion models (the latter category accounting for the reordering of words or phrases between the source and target languages), and using these models to decode and translate new sentences. The process includes multiple stages of alignment, modeling, decoding, and evaluation, with the goal of producing the most accurate and fluent translation possible based on statistical probabilities derived from the training data.

EBMT has occasionally been incorporated as a component within more complex SMT systems. Combining the example-based approach with statistical analysis of large corpora has produced compelling results, as statistical methods are known for their strong recall and can be enhanced by the precision provided by the example-based approach [1].These two techniques, known as corpus-based approaches, did not focus on syntactic and semantic rules but instead relied on the manipulation of large text corpora. Because of their flexibility to support multiple languages, their superior results compared to their predecessors, the availability of multilingual text corpora and the increasing computing power to process them, these two methods dominated the field for more than twenty years.

1.2.1 IBM models

The aforementioned IBM alignment models, aligning words between source and target languages, were highly influential and formed the basis of statistical machine translation. They are related to the following equation, coined at the same time, which is often called the **fundamental equation of machine translation** [6]:

$$\hat{e} = \arg\max_{\mathbf{e}} \left(P(\mathbf{e}) \cdot P(\mathbf{f}|\mathbf{e}) \right)$$

where:

- $\hat{e}$ represents the best translation in the target language of the source sentence $\mathbf{f}$.
- $P(\mathbf{e})$ is the language model of the target language
- $P(\mathbf{f}|\mathbf{e})$ is the translation model, which represents the probability of the source sentence $\mathbf{f}$ given the target sentence $\mathbf{e}$.

In other words, implementations of this equation traverse through all possible target translations $\mathbf{e}$ and select the one that maximizes the product of the two probabilities. The IBM models offer a probabilistic framework for computing $P(\mathbf{f}|\mathbf{e})$, which represents the translation model in this equation. They address several key challenges, including determining the likelihood that a word in the target sentence translates to a word in the source sentence (word-to-word translation probabilities) and estimating the alignments between words in the

source and target sentences (alignment probabilities).

The first model operates on the assumption that each word in the source sentence can independently translate into any word in the target sentence. While it is relatively straightforward, it ignores word order and context, making it less effective for complex language pairs or longer sentences. Nonetheless, it laid the foundation for more advanced models. Model 2 builds on the first by introducing positional alignment, which accounts for the position of a target word relative to the source word. This improvement results in better accuracy and faster convergence during training.

Model 3 is more sophisticated, addressing cases where one word in the source language corresponds to multiple words in the target language (such as translating “potato” in English to “pomme de terre” in French). It also tackles the omission of words (e.g., the English article “the” often missing in French) and non-contiguous word pairs. By introducing “fertility probabilities” to predict the number of target words generated from each source word, the model resolves many of these issues. Additionally, “distortion probabilities” were added to account for extra words, such as articles, that appear in the target language but not in the source.

Model 4 refined the distortion probabilities to handle word reordering between source and target sentences more effectively. Finally, Model 5 further reduced irrelevant alignments, where some words in the target sentence were left unaligned or improperly aligned, although it required more training data and was more complex than the earlier models.

Overall, the IBM alignment models were a significant advancement because they provided a structured and quantifiable approach to translation, which helped improve the quality and efficiency of machine translation systems. They helped shift the field from rule-based translation approaches to data-driven methods, where translation probabilities are learned from actual usage rather than predefined rules and inspired subsequent innovations in the field [1].

1.2.2 New fields and advancing technologies

During the 1990s, research into speech translation began to advance, focusing on integrating speech recognition, speech synthesis, and translation modules. *Figure 1.1* presents a common speech-to-text pipeline with translation capabilities. As machine translation technology quickly gained traction among corporations, software for personal use also became widely available and popular. The growing demand for translation on the World Wide Web led to the emergence of the first online machine translation services, such as Altavista/Yahoo! Babelfish [2].

In the 2000s, the previous trends continued, with the main difference being ample Internet access. The aforementioned online machine translation services began to surpass offline translation software in use. Collaboration between researchers worldwide increased, resulting in the development of open-source software for performing basic SMT processes, such

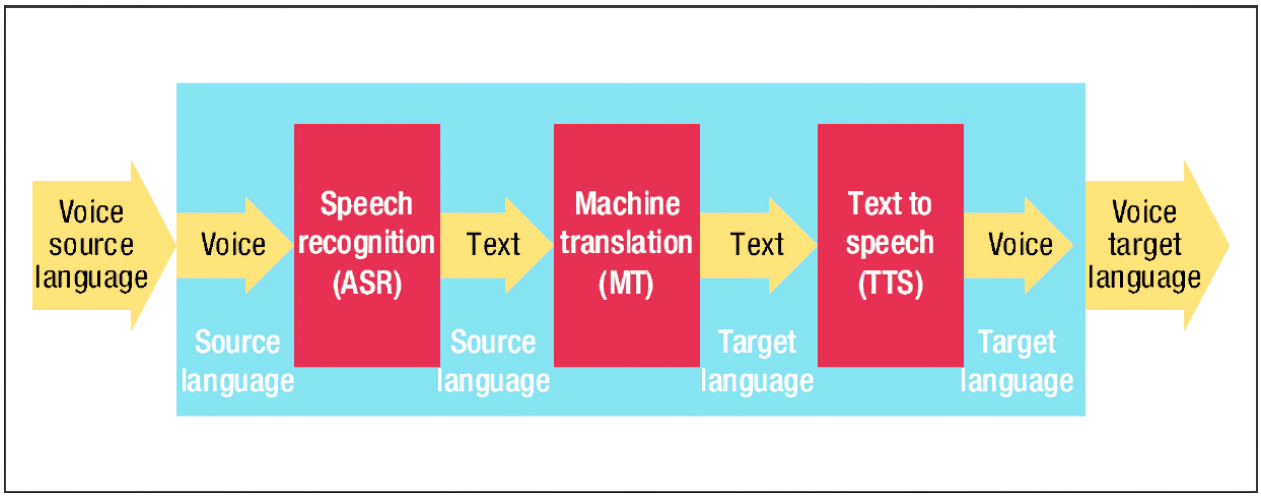


Figure 1.1: The components of a speech-to-speech translation system that utilizes machine translation [8].

as Moses [1] or GIZA(++) [7]. Finally, for certain languages, hybrid approaches began to be used that combined statistical and rule-based translations for better results in cases where syntax and semantics matter [2].

1.2.3 Methods of evaluation

BLEU, an algorithm for the evaluation of machine translation results, was also created by IBM in 2002 [9]. Their idea, called an "evaluation understudy", compares MT output with expert reference translations in terms of the statistics of short sequences of words (word N-grams). The more of these N-grams that a translation shares with the reference translations, the better the translation is judged to be. BLEU also includes a brevity penalty to "punish" candidate strings that are shorter than the reference translations. The final score is calculated with the following formula:

$$BLEU\ Score = exp\left\{\sum_{n=1}^{N} w_n\ log(p_n) - max\left(\frac{L*_{ref}}{L_{sys}} - 1,\ 0\right)\right\}$$

where

$$p_n = \frac{\sum_i \begin{pmatrix} the\ number\ of\ n-grams\ in\ segment\ i, \\ in\ the\ translation\ being\ evaluated,\ with \\ a\ matching\ reference\ cooccurence\ in\ segment\ i \end{pmatrix}}{\sum_i \begin{pmatrix} the\ number\ of\ n-grams\ in\ segment\ i, \\ in\ the\ translation\ being\ evaluated \end{pmatrix}}$$

$$w_n = N^{-1}$$

$$N = 4$$

[1] https://www2.statmt.org/moses/

and

- $L*_{ref}$ — the number of words in the reference translation that is closest in length to the translation being scored
- L_{sys} = the number of words in the translation being scored

The idea is elegant in its simplicity. But far more important, IBM showed a strong correlation between these automatically generated scores and human judgments of translation quality [10]. BLEU remains one of the most popular evaluation metrics today [11], although it suffers from several identified limitations. These include brevity penalty issues (with BLEU unfairly scoring lower translations that are slightly shorter but still convey the full meaning), inattention to semantic errors or grammatical correctness, sensitivity to word order (small variations in the sequence of words can lead to a significantly lower score) and in general lower correlation with human judgment than metrics which take into account linguistic resources [12].

Other metrics following similar underlying principles have also been proposed. NIST is an adaptation of BLEU. It takes into account how informative a particular n-gram is and gives more importance to the less frequent n-grams [10]. METEOR *(Metric for Evaluation of Translation with Explicit ORdering)* is based on an explicit word-to-word matching between the MT output being evaluated and one or more reference translations, and demonstrates improved correlation with human judgments [13]. A newer metric is chrF *(CHaRacter-level F-score)*, which proposes the use of character n-gram F-score (instead of word n-grams) for automatic evaluation of machine translation output [14]. The main equation calculating the chrF score is:

$$CHRF\beta = (1+\beta^2)\frac{CHRP \cdot CHRR}{\beta^2 \cdot CHRP + CHRR}$$

where CHRP and CHRR stand for character n-gram precision and recall arithmetically averaged over all n-grams:

- CHRP: percentage of n-grams in the hypothesis which have a counterpart in the reference;
- CHRR: percentage of character n-grams in the reference which are also present in the hypothesis.
- β is a parameter which assigns β times more importance to recall than to precision – if β = 1, they have the same importance.

1.3 Neural machine translation (NMT)

Artificial neural networks (ANNs) are collections of connected units or nodes called artificial neurons, that are inspired by the structure and function of the neurons in human brains. The

perceptron, a foundational concept in neural networks, was created in 1943 by Warren McCulloch and Walter Pitts [15]. Frank Rosenblatt later implemented the first hardware version, the Mark I Perceptron, at Cornell Aeronautical Laboratory in 1957 [16]. Despite the initial excitement surrounding these developments in the 1950s and early 1960s, the field did not deliver the expected breakthroughs. By the late 1960s, limitations in the perceptron's capabilities were highlighted, such as its inability to solve non-linearly separable problems [17]. This led to widespread disappointment and contributed to what became known as the first "AI winter" during the 1970s—a period of reduced funding, interest, and progress in artificial intelligence research, particularly in neural networks.

The field experienced a revival in the 1980s, especially after the backpropagation algorithm was popularized in 1986 [18].This algorithm allowed for more effective training of multi-layer neural networks, fueling renewed interest in ANNs. However, even during this period of growth, the computing power of the 1980s and early 1990s did not allow for the full exploitation of ANNs potential. Consequently, the field entered another period of dormancy by the early 1990s, even though proposed models of the time bear resemblance to modern approaches [19]. The true revival of neural networks occurred in the 21st century, particularly in the 2010s, when advances in computing technology enabled the training of deep neural networks on large-scale datasets. With this increased computational power and the availability of vast amounts of data, the field of neural networks underwent a significant resurgence, becoming extremely relevant and all-pervasive [19].

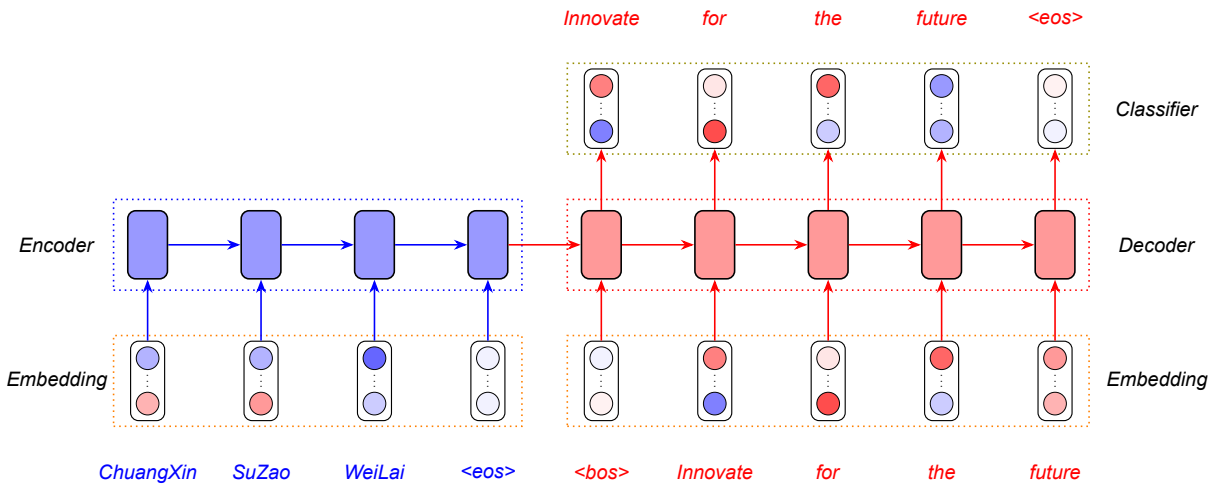


Figure 1.2: An overview of the NMT architecture, which consists of embedding layers, a classification layer, an encoder network, and a decoder network. Different colors are used to distinguish different languages (Chinese & English) [20].

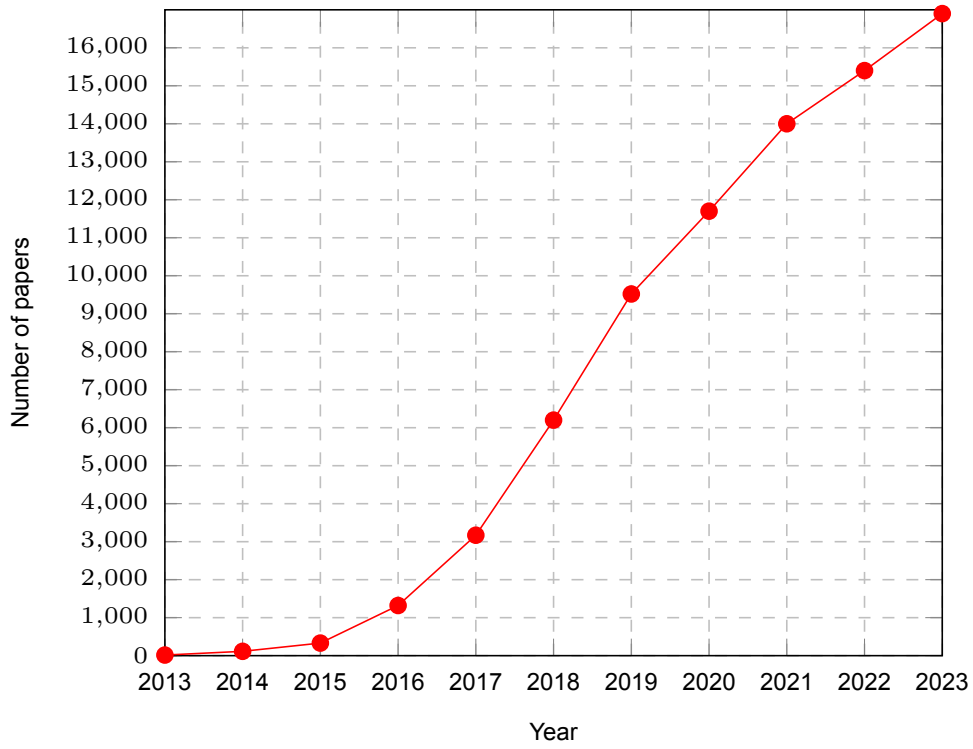


Figure 1.3: Number of papers mentioning "neural machine translation" per year according to Google Scholar [2].

Corpus	Sentences	% of OPUS
NLLB	13B	28.31
CCMatrix	11B	23.64
OpenSubtitles	8.5B	18.53
MultiCCAligned	2.2B	4.88
ParaCrawl	1.5B	3.26
DGT	1.1B	2.38
XLEnt	883M	1.92
MultiParaCrawl	789M	1.72
LinguaTools-WikiTitles	487M	1.06
CCAligned	439M	0.95

Table 1.1: Top 10 parallel corpora in the Opus collection as of September 2024. They represent 86,65% of total Opus sentences.

In the 2010s, it became apparent that neural networks were appropriate for use in various fields, easily surpassing previous techniques and scoring far better results than them. The field of machine translation was no exception, starting with modest integration of neural language models into traditional SMT systems and resulting in an explosion to scientific publi-

[2]Example Google Scholar search: `https://scholar.google.com/scholar?q=%22neural+machine+translation%22&as_ylo=2023&as_yhi=2023`

cations related to pure NMT, as reflected in *Figure 1.3*. A significant shift began when Baidu started using neural networks around 2015, with Google Translate following the next year and competitors immediately after. The change from SMT to NMT happened very rapidly, with virtually all new competititve systems using NMT by 2017, only two years after the first submission of such a system [19]. Government institutions were no exception; as of October 2024, European Union institutions use eTranslation, a "cutting-edge neural machine translation service provided by the European Commission" [3], though it faces challenges because of its limited use and requires post-translation quality inspection [21].

By using neural networks, models are trained on a significant amount of parallel data, between two or more languages. A large number can be found at the Opus project, one of the most comprehensive collections of parallel corpora [4]. Opus offers datasets in various languages extracted from a wide range of sources, including European Union or United Nations documents (Europarl, EUbookshop, JRC-Acquis, EMEA, MultiUN, ELRC-SHARE), as well as web crawled material (NLLB, CCMatrix, OpenSubtitles, ParaCrawl, Global Voices). *Table 1.1* presents the current top datasets provided by the resource.

1.3.1 Neural network architectures and attention

Figure 1.4 presents a simplified comparison of neural network architectures utilized in modern NMT systems. The first attempts used convolutional neural networks (CNNs) or recurrent neural networks (RNNs) as the encoder architecture, while the decoder used an RNN [22, 23]. However these systems performed poorly with increasing sentence length. The addition of the attention mechanism addressed this problem, producing satisfying results [24, 25]. In 2017, Vashwani et al. introduced the Transformer deep learning architecture in the groundbreaking paper *"Attention is all You Need"* [25], which demonstrates that Transformers achieve state-of-the-art performance on machine translation tasks, with significant improvements in training speed and accuracy due to the model's parallelizable structure. Each token does not depend on the previous tokens' computations in the sequence, as would be the case in an RNN or LSTM, therefore the calculations can be done independently of one another [26]. This is achieved by utilizing self-attention and multi-head attention instead. Every layer in the Transformer architecture includes a series of multi-head attention layers followed by a dense layer. Multi-head attention utilizes scaled dot-product attention to link a query with key-value pairs, represented by the equation:

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

where:

- Q is the matrix of queries,
- K is the matrix of keys,

[3] `https://commission.europa.eu/resources-partners/etranslation_en`
[4] `https://opus.nlpl.eu/`

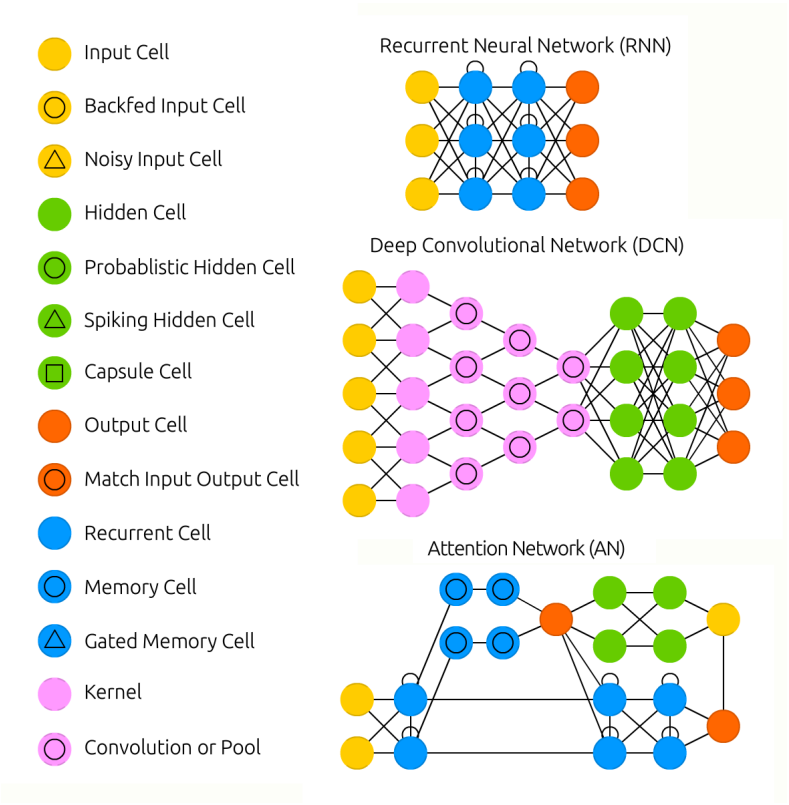


Figure 1.4: Architecture of a Recurrent Neural Network (RNN), a Convolutional Neural Network (CNN) and an attention network (in which category a Transformer belongs) [5].

- V is the matrix of values,
- dk is the dimension of the keys (or queries),

The softmax function is applied to the scaled dot product of Q and K, effectively normalizing the attention scores to create a probability distribution. This ensures that the model focuses more on relevant parts of the input data. The architecture scales well to larger datasets and more complex tasks, making it a foundation for subsequent models, not only in NMT, but also for BERT, GPT, and other large language models.

There are various attention mechanisms in transformers and other sequence models [27], three of which are depicted in *Figure 1.5* [6]. Encoder-Decoder Attention allows the decoder to focus on the entire output from the encoder. It helps the decoder focus on relevant parts of the input sequence when generating the target sequence. In Encoder Self-Attention the encoder layers apply attention to their own outputs, allowing each token in the input to attend to every other token, capturing dependencies between tokens. Lastly, Masked Decoder Self-Attention is the same as Encoder Self-Attention, but with different masking. It allows the decoder to attend only to previously generated tokens, but not future positions [28].

[5] Derived from the Neural Network Zoo: `https://www.asimovinstitute.org/neural-network-zoo/`
[6] `https://nlp.stanford.edu/seminar/details/lkaiser.pdf`

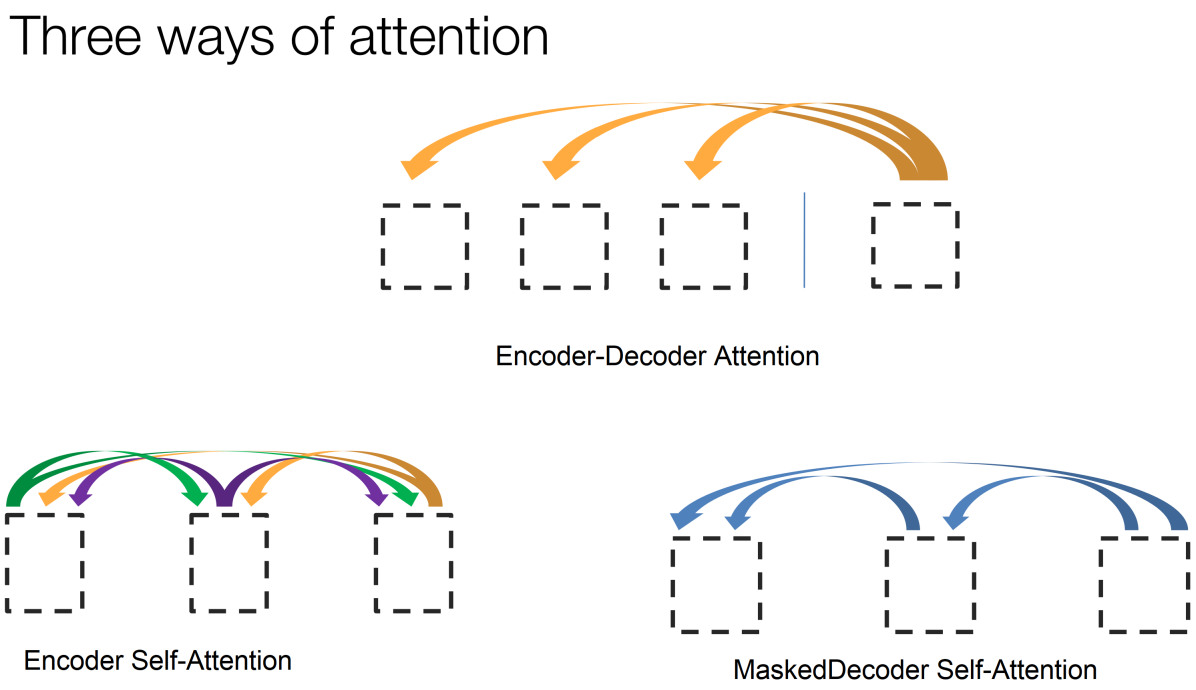


Figure 1.5: Three types (or ways) of attention.

1.3.2 Methods of evaluation

Evaluation of machine translation techniques has also been influenced by neural networks evolution, leading to the creation of new metrics, such as COMET *(Crosslingual Optimized Metric for Evaluation of Translation)*, a neural framework for training highly multilingual and adaptable MT evaluation models that can function as metrics [29]. It calculates the similarity between a machine translation output and a reference translation using token or sentence embeddings and is based on the similarity of vector representations. At the time of its presentation, COMET achieved state-of-the-art results for segment-level correlation with human judgments, and showed promising ability to better differentiate high-performing systems [29]. Unlike traditional metrics that rely on n-gram overlaps between the translated text and reference translations, COMET uses neural networks to assess translation quality. This allows COMET to better capture nuances of language, such as meaning, fluency, and adequacy, which traditional metrics might miss.

1.3.3 Techniques and new developments

Pre-training and fine-tuning are key techniques used in model training to enhance the accuracy of the output. Pre-training is the initial phase, where a model is trained on large datasets, a process that is computationally intensive and data-heavy. Most competitive neural network models undergo pre-training to achieve strong baseline performance. Fine-tuning, on the other hand, involves further training a pre-trained model on a smaller, more specific dataset, often tailored to a particular domain or task. This allows the model to gain specialized knowl-

edge and improve its performance in areas where pre-training alone may not suffice. However, fine-tuning should be applied on a case-by-case basis and is not a one-size-fits-all solution [30].

After the first wave of NMT systems, which used mostly bilingual models, multilingual models gradually became more popular. Multilingual Neural Machine Translation (MNMT) is an approach to machine translation where a single model is trained to translate between multiple languages, rather than using separate models for each language pair. The model shares parameters across languages, allowing it to learn common linguistic patterns and generalize better, especially for low-resource languages. To manage multiple languages, the model uses special tokens or embeddings to indicate the source and target languages, guiding the translation process. It also has the benefit of being able to attempt zero-shot translation, learning to translate between language pairs it has never seen in this combination during training, that being a practical example of transfer learning within neural translation models. Zero-shot translation can be improved with little additional data of the language pair in question [31].

In the last two years, the advent of generative AI, with the use of large language models (LLMs) led to investigations for their use in machine translation applications. These models differ from pre-existing NMT systems in using just a transformer's decoder, instead of an encoder-decoder architecture and are not trained in parallel datasets but try to predict the next word in a sequence that derives from a large dataset of text, predominantly in English [7]. Research is ongoing, although first insights have shown improvements in high-resource languages but neutral or negative results in under-represented languages without significant training datasets [32, 33].

1.4 Current and future challenges

Current MT systems have reached the stage where researchers are debating whether or not they can rival human translators in common linguistic pairs [34]. However, low-resource languages still have a long way to go before they reach the same state, as the performance of NMT systems correlates to the amount of provided training data [35]. Therefore, advancing machine translation of low-resource languages is of increasing interest [36]. *Figure 1.6* illustrates the various resource methods currently utilized for this effort.

Linguistic errors leading to incorrect translations are always a consideration, though the shift to NMT systems has helped in regards to this problem. However, MT systems still often struggle to understand and maintain context over long passages of text and may misinterpret or lose context when translating paragraphs or entire documents. Idiomatic expressions, slang, and cultural references are often lost or mistranslated. Words with multiple meanings (polysemy) or ambiguous phrases can be challenging to translate accurately and the correct

[7] As an example, 92.1% of the characters in the dataset used to train GPT-3 are in the English language, with only 7.9% of the characters dedicated to the other languages: `https://github.com/openai/gpt-3/blob/master/dataset_statistics/languages_by_character_count.csv`

meaning may not be chosen. Finally, specialized domains such as legal, medical, or technical fields often require specific terminology and knowledge, which general models might not possess.

Some technical challenges impact not only NMT models but also LLMs. Both systems, for instance, experience hallucinations, referring to instances when the model generates information that is not correct or coherent. These hallucinations can involve fabricating facts or details that sound plausible but are entirely incorrect or invented. In the context of NMT, these systems might produce translations that do not accurately reflect the source text [37]. Another issue observed in LLMs is model degradation when trained on generated data, leading to progressively poorer results [38]. Given the similarities between NMT and LLMs, this issue could also arise in future NMT models, especially those trained on data from online sources. Additionally, the nature of neural networks makes it difficult to trace how outputs are generated, leading to a lack of reproducibility [39]. Lastly, while advancements have been made in generating creative content, these outputs still generally fall short of those produced by human writers.

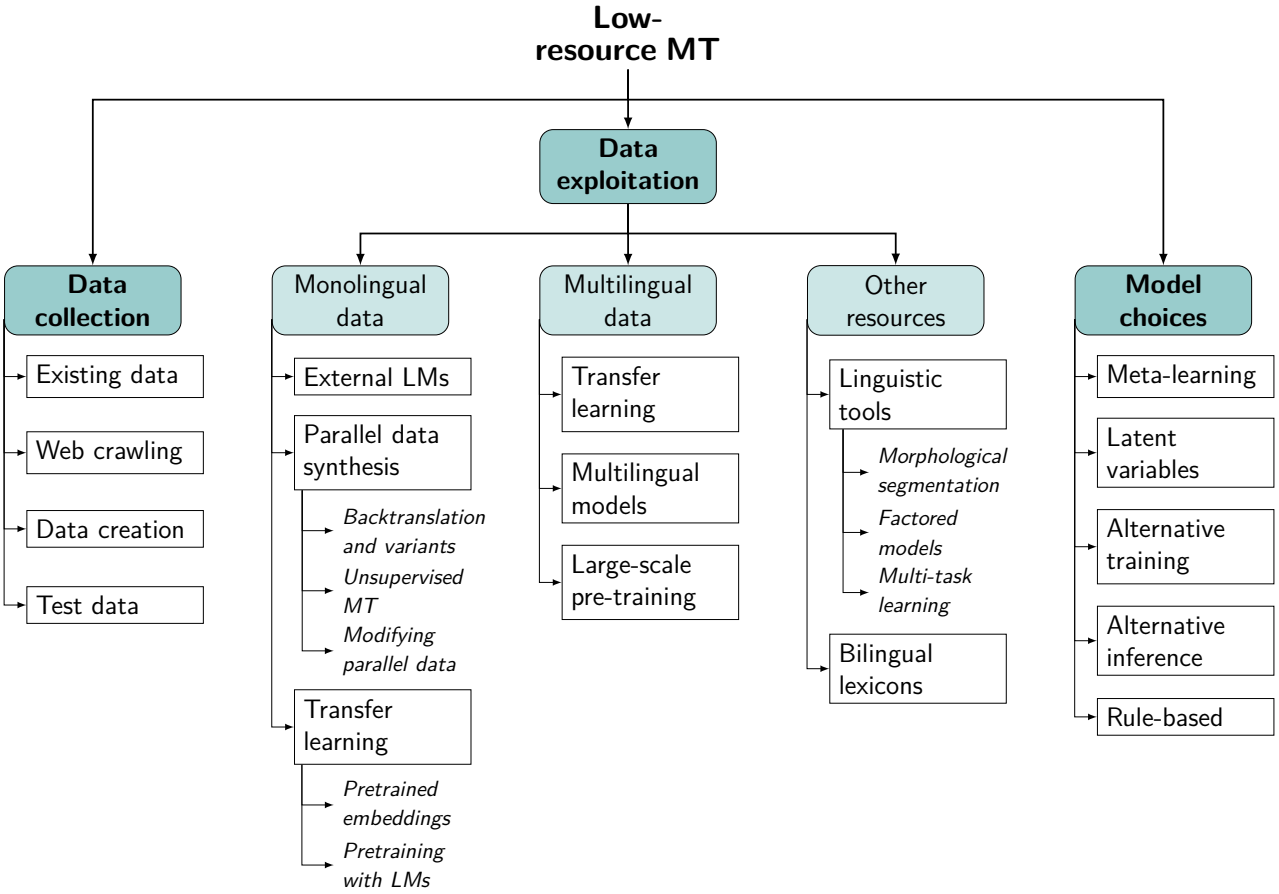


Figure 1.6: Overview of research methods for low-resource machine translation [36].

The advancement of MT has sparked ethical concerns regarding the relationship between technology and human translators. Experts in MT sometimes underestimate the importance

of human input in conjunction with automated systems. Terms like “human parity” or “super-human performance” are occasionally used, which can diminish the value of human translators. Many professional translators, concerned about their job security, are hesitant to engage in post-editing of machine-translated text, viewing it as a task that feels alienating and lacks creativity. It’s crucial to carefully consider whether MT poses a threat to the translation profession or can serve as a complementary tool, similar to other technological innovations in the past [34].

2. FOCUS OF THIS DISSERTATION AND GENERAL AIMS

The objective is to create an offline machine translation application that will be open source. It should not rely on any proprietary technologies / services and run client-side for its main purpose, while offering quality-of-life improvements compared to similar predecessors. Apart from the core mechanism of machine translation, it would be useful to offer additional functionality, such as transcribing text from images or converting speech in audio files to text. Ideally it should support as many platforms as possible and be easy to maintain in the future. Collaboration with similar projects such as project Bergamot will be pursued, to avoid reinventing the wheel.

The application should operate on CPUs. Even though high-end GPUs provide computing power that is orders of magnitude more powerful than that of CPUs, they are not ubiquitous. Furthermore, they often need manual tuning and compatible drivers to function properly, which does not offer a pleasant out-of-the-box user experience (UX).

The implemented application will be referred from now on as *Tradutorium*. The noun denoting the translator in Portuguese / Italian is *tradutor / traduttore* respectively (with similar words for the rest of the Romance languages), while the suffix *-orium* denotes the place that this particular action takes place in.

2.1 Possible advantages of a local machine translation application

2.1.1 Privacy and security

An offline machine translation application has obvious advantages in the privacy field compared to an online service, because sensitive data does not need to be transmitted over the internet, which is particularly valuable in scenarios where confidentiality is important. This was the main reason that project Bergamot was funded by the European Union's Horizon 2020 research and innovation programme [1]. Mozilla also joined the project from its origination [2] in order to add machine translation functionality to its Firefox browser, which takes a privacy-first approach, differentiating it from its commercial competitors [3]. Tradutorium aims to provide another private-friendly cross-platform option.

2.1.2 Availability and control

Because it doesn't depend on an internet connection, an offline application can be used anywhere —whether in remote areas with unreliable connectivity or during network outages. This guarantees uninterrupted access to translation services. Moreover, offline translation tools can be customized to meet specific needs by selecting appropriate models or incorporating tailored ones, enhancing accuracy and relevance based on particular requirements.

[1] `https://cordis.europa.eu/project/id/825303`

[2] `https://blog.mozilla.org/en/mozilla/local-translation-add-on-project-bergamot/`

[3] `https://www.mozilla.org/en-US/firefox/features/private/`

2.1.3 Compliance with regulations

Many industries are subject to strict regulations, such as the *General Data Protection Regulation* (GDPR) in the European Union, or the *Health Insurance Portability and Accountability Act* (HIPAA) in the United States. This is especially true in sectors like healthcare, finance, and legal, where data protection is paramount. Offline translation tools can help ensure compliance by keeping data within the organization's infrastructure and avoiding potential breaches associated with online services. Organizations can also implement additional security measures, such as encrypted local storage and restricted access, to further protect data within offline applications.

2.1.4 Lower cost of use

Online translation services typically offer free translations up to a certain word limit, with additional charges for exceeding this limit or accessing extra features. Frequent use can rapidly make these services expensive. In contrast, offline applications can handle large volumes of text without incurring extra costs. However, tt should be noted that some offline applications might have restrictions on commercial use. Fortunately, the models provided by Project Bergamot do not have such restrictions and can be used for commercial activities as well.

2.2 Existing end-user software

Before embarking on the creation of a new application, with the effort that this entails, the existing software options that are currently available were considered to prevent unnecessary duplication of effort. Those described below are high-quality options, and the need for an alternative was not because of substandard implementations but of differing needs.

The most obvious alternative is **TranslateLocally** [4], which was created by the same team that developed and maintains Bergamot Translator and uses the models created by the project. Like Tradutorium, it is mainly desktop-oriented, although a WebAssembly version is also available. It is written in C++ and uses the cross-platform Qt Framework [40].

Another open-source option is **LibreTranslate** [5]. It is based on ArgosTranslate, which uses models that have been trained using OpenNMT / CTranslate2. It is written in Python and is web-oriented, with a browser interface and API calls support. However, it is difficult to install locally because there is no executable that is distributed by the project, and one needs to clone the repo or install it in a container with Docker. It should be noted that LibreTranslate supports the parsing and translation of various formats, such as word processor documents or presentations.

[4] `https://translatelocally.com/`
[5] `https://libretranslate.com/`

2.2.1 Comparison with Tradutorium

Both of the mentioned software packages are robust and well-designed applications. However, Tradutorium sets itself apart with a different overall philosophy: it aims to integrate multiple methods of translating data, whether it be text, images, or audio. Beyond this, there are several other differences, such as platform support—Tradutorium is currently only available as a desktop application—and various engineering decisions.

For example, both Tradutorium and TranslateLocally use the Bergamot Translator, but they handle it differently. Tradutorium communicates with Bergamot Translator via shell commands, whereas TranslateLocally, based on a cursory examination of its source code, seems to use a different approach. This difference likely stems from TranslateLocally's deeper familiarity with the internals of the underlying library, knowledge that may not be immediately apparent to external developers.

3. TECHNICAL DETAILS

The following section details the technical considerations and challenges that were encountered during the planning, implementation and deployment of the application. It also offers explanations about the choice of specific technologies instead of available alternatives. It is hoped that this documented process will provide additional insights about the scope of the application and its ambitions.

3.1 .NET platform

.NET, a comprehensive and versatile software development framework created by Microsoft, is an ideal platform for an implementation like this. Originally Windows only (as .NET Framework) it was open sourced and made cross-platform in 2015. .NET offers a wide ecosystem of libraries and tools, either from a rich standard library or from the .NET community, that handle common tasks like file I/O or networking and can be easily integrated into .NET projects, making it easier for developers to create a wide range of cross-platform applications. The most popular and widely used programming language for the platform is C# (pronounced C Sharp). C# is object-oriented, requires garbage collection (which is provided by the .NET platform) and spans from high-level features such as data-oriented records to low-level features such as function pointers while offering static typing and type and memory safety as baseline capabilities [41]. Code written is compiled in the **Common Intermediate Language (CIL)**, a form of bytecode, which can then be run on any platform that's supported by the **Common Language Runtime (CLR)**, a virtual machine that can execute CIL instructions [42]. The process is similar to the execution model of the Java platform, where programs are compiled to Java bytecode and can then be run by the Java Virtual Machine (JVM) [43]. Applications that use these high-level platforms have to interact less with the platform implementations or the underlying systems and can thus focus on the main functionality.

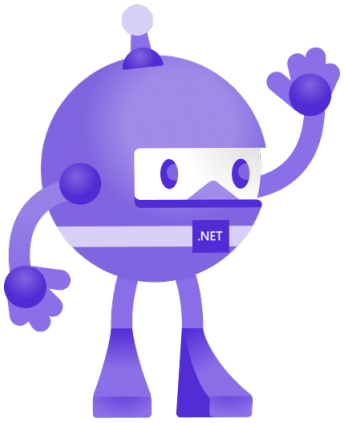


Figure 3.1: dotnet bot, the community mascot for .NET. It represents the community that comes with the .NET brand and platform.

The latest stable version at the time of development, .NET 8, was used. This version pro-

vides significant improvements to Native AOT *(Ahead-Of-Time)* compilation, a technology for compiling .NET applications into native machine code ahead of time (as is usual in other languages such as C, C++ or Go), as opposed to using JIT *(Just-In-Time)* compilation, which compiles code at runtime. Native AOT aims to improve the performance and deployment characteristics of .NET applications by generating native binaries that can be executed directly by the operating system bypassing the need for the .NET runtime environment to be installed on the target systems.

3.2 GUI framework

3.2.1 Consideration and choices

The choice of platform was not the only important decision that had to be made. Since Tradutorium is a graphical user interface (GUI) application, selecting an appropriate framework to facilitate its development was crucial. Key factors considered included cross-platform support (including mobile compatibility), extensibility, active development, and several other technical details outlined below.

Table 3.1 provides a comparison of various UI frameworks available for .NET. These frameworks can be divided into two main categories: those developed by Microsoft and those created by the community or smaller companies. Microsoft-developed frameworks are often popular due to the backing of a large corporation, which is seen as a positive indicator for their continued evolution and ongoing support. Among these, the oldest is **Windows Forms (WinForms)**, introduced with .NET 1.0, primarily serving as a wrapper for the Win32 API. **Windows Presentation Foundation (WPF)**, released with .NET 3.5, introduced features like data binding and templating and was the first of many XAML-based frameworks. XAML, or *eXtensible Application Markup Language*, is a Microsoft variant of XML used for defining UI elements in a flexible manner. Later XAML-based frameworks include **Silverlight**, **WinUI**, and finally **MAUI**, the latter being a continuation of Xamarin.Forms (mentioned below).

However, Windows Forms, WPF, and WinUI are limited to Windows. Silverlight, once a competitor to Adobe Flash, has been unsupported for years. While MAUI offers support for Windows, macOS, Android, and iOS, it does not support Linux. Generally, Microsoft frameworks prioritize Windows as the primary platform, with other operating systems being an afterthought. Additionally, Microsoft has faced criticism for spreading its resources across multiple frameworks, leading to suboptimal implementations and the potential for some to be abandoned [44].

In addition to Microsoft's offerings, there are third-party GUI libraries for .NET. **Gtk#** was initiated by the Mono project, aiming to use C# and .NET for cross-platform development, requiring a UI library. It was a wrapper for the popular GTK+ library, primarily used in Linux environments. Originally developed by Xamarin, Gtk# development stalled after Microsoft acquired Xamarin, only supporting GTK+ 2. A community fork, **GtkSharp**, added GTK+ 3 support, but development has slowed, and it doesn't support the latest GTK+ 4. Similarly,

Xamarin.Forms, also created by Xamarin, was transformed into MAUI after the Microsoft acquisition and has ceased further development.

Framework	First release	Microsoft-made	Cross-platform	Mobile support	Active develop-ment
Windows Forms	2002	Yes	Windows-only	No	Yes
Gtk#	2004	No	Yes	No	No[a]
WPF	2006	Yes	Windows-only	No	Yes
Silverlight	2007	Yes	Yes	Partial[b]	No
WinUI	2011	Yes	Windows-only	No	Yes
Xamarin.Forms	2014	No[c]	Yes	Yes	No[d]
Avalonia	2014	No	Yes	Yes	Yes
GtkSharp	2015	No	Yes	No	Yes
UNO Platform	2018	No	Yes	Yes	Yes
.NET MAUI	2022	Yes	Partial[e]	Yes	Yes

[a] Continues as GtkSharp, a community fork
[b] Supported the now deprecated SymbianOS and Windows Phone operating systems, not Android or iOS
[c] Xamarin was later acquired by Microsoft
[d] Continues as .NET MAUI
[e] Does not support Linux

Table 3.1: Available .NET UI frameworks.

This leaves **Avalonia** and **Uno Platform** as strong contenders. Both support Windows, macOS, Linux, Android, iOS, and WebAssembly. Avalonia is often considered a spiritual successor to WPF, offering cross-platform support while fixing many bugs and introducing new features. It is not based on WPF's codebase, as Avalonia's development began before WPF was open-sourced. On the other hand, Uno Platform extends the Windows-only WinUI to work across multiple platforms. Avalonia is generally seen as more consistent, extensible, and future-proof, with the .NET community often favoring it in discussions and comparisons between the two [44].

3.2.2 Avalonia framework

When .NET was open sourced in 2014, with plans to become cross-platform and not Windows-only as before, the need for a cross-platform GUI framework that would be native for .NET surfaced. The obvious paradigm was to create something that would follow in the footsteps of other existing frameworks for the platform, most notably WPF. The main benefit of XAML

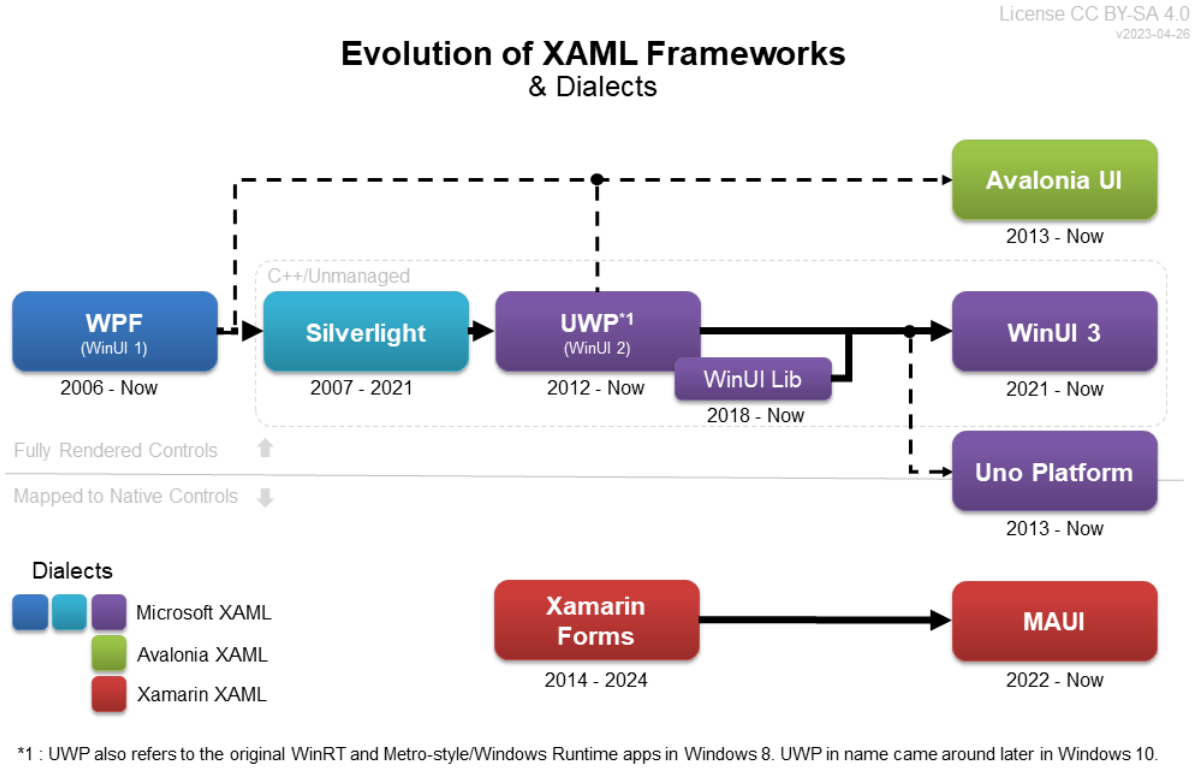


Figure 3.2: Evolution of XAML frameworks [45].

is that the UI code is separated from the logic of the application, in contrast with previous approaches, like Windows Forms, where the two were intermingled.

Figure 3.3: Avalonia project logo.

Since WPF is considered a spiritual predecessor, and its internal codename before the first public release was Avalon, the name Avalonia was chosen, symbolizing its influence. Both frameworks use their own set of custom controls rather than adopting the native widgets provided by the underlying operating system. The main difference is that WPF is built on DirectX, which is limited to Windows, while Avalonia relies on SkiaSharp, a .NET binding to the Skia graphics library, for rendering its widgets. Both approaches support hardware-accelerated rendering, providing better performance and smoother graphics. Rendering the Avalonia controls with Skia guarantees consistent behavior across all platforms, while utilizing native controls may exhibit different results based on the platform's implementation.

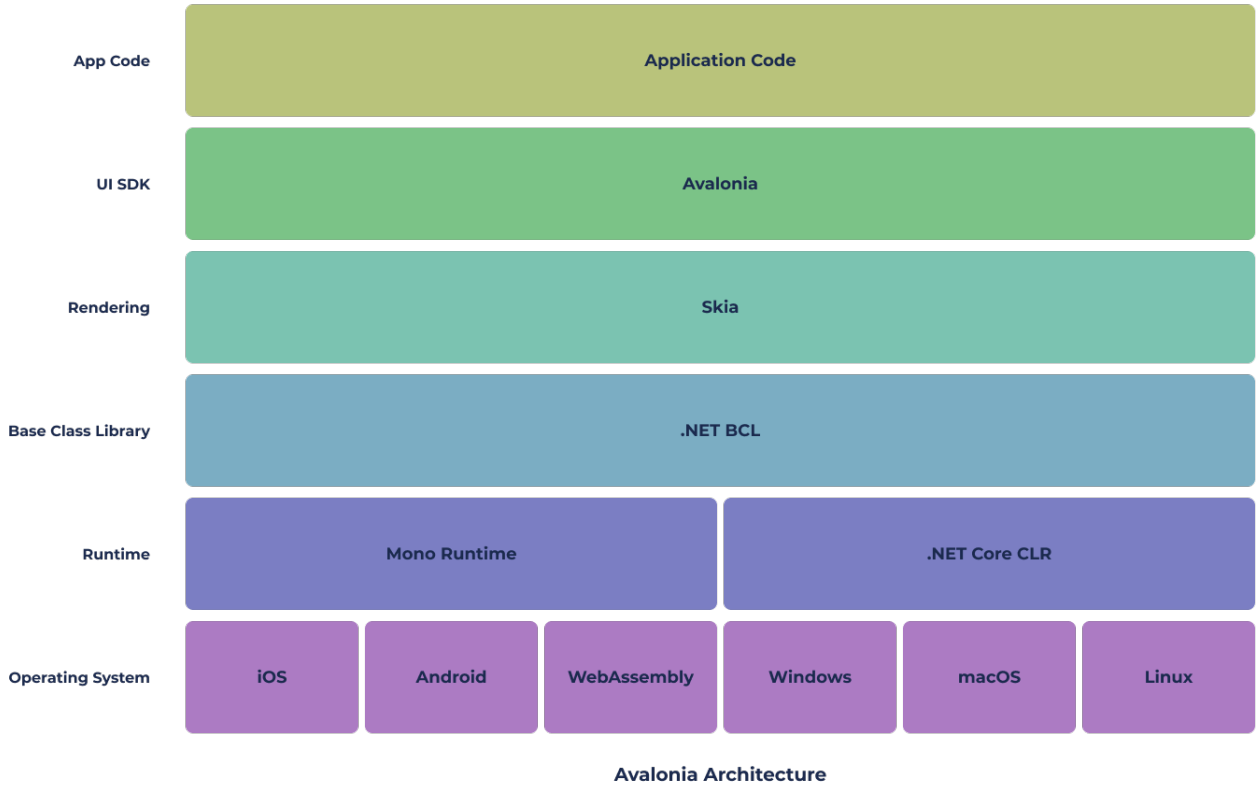


Figure 3.4: Avalonia architecture. Note that the rendering layer uses Skia everywhere, instead of each platform's native controls.

Through the years, Avalonia went from just a preliminary prototype to a production-ready state and is now being used by various open source projects or companies, one of which is Jetbrains, a well-known corporation producing popular integrated development environments (IDEs) aimed at developers. In 2023, version 11 was released, which is considered the first version with a stable API and introduced support for mobile platforms like Android and iOS. Outside the main Avalonia product, the team responsible created Avalonia XPF, which can execute Windows-only WPF applications on other operating systems and is aimed mainly at companies with existing codebases. Even though this is a different project from the main Avalonia UI framework, it funds Avalonia development, while showcasing the team's technical ingenuity and long-term plans that appear to lead to continued prosperity and active development.

3.3 Development tools

The package manager used by the .NET ecosystem is called Nuget. It simplifies the process of managing dependencies in .NET applications, handling the installation and updates of various libraries, including Avalonia, NTextcat, Tesseract, Whisper.net and many others. Unfortunately, Bergamot Translator cannot be used through a .NET package manager (since no bindings exist for this specific ecosystem) and requires a separate compilation process

before it can be used, which is described in a later section.

For Windows development, Visual Studio 2022 on Windows 10 was utilized. Although mainly known as an IDE, it offers a comprehensive suite of tools specifically designed for C++ development. These include a C++ compiler, an integrated debugger, CMake support, various build systems, and static analyzers. Together, they provide everything needed to build, run, and compile software like Bergamot Translator. However, Visual Studio is not available on other operating systems (a version that existed for macOS was recently retired). To address this, Jetbrains Rider, a cross-platform IDE for .NET development was used on Debian 12 (*"Bookworm"*). Rider 2023.3 EAP (which stands for Early Access Program), a prerelease version, was used since only that version supported .NET 8 at the time of early development.

3.4 Website and repository

The domain `tradutorium.org` has been registered and can be used as the official home of the project. The code is hosted on a Git repository, located at Gitlab [1], but also mirrored at Github for easier access and redundancy. The models used are included in another Git repository [2]. Pull requests and issues can be submitted at the Gitlab instance. The reason Gitlab was chosen for bug tracking is its support for self-hosting the core software and officially supporting export or import of the issues that a repository contains [3]. Github, in contrast, is a Software as a Service (SaaS) application. Owing to the federated nature of the Git protocol, the hosted code or commits do not present a problem in the case of migration to a different service. However, secondary metadata that is contained in the repository, like the bug tracker, is more complex to transfer to an alternative service.

The project is made available under the Mozilla Public License (MPL), which is a weak copyleft license also used by projects like Mozilla Firefox and LibreOffice and is not expected to cause license incompatibilities with the underlying dependencies. Conveniently, it also happens to be the license used by Bergamot Translator, due to its close association with the Mozilla Foundation.

3.5 Supported platforms

The technologies used (.NET and Avalonia) are cross-platform, can be run on desktop (Windows, Mac, Linux) or mobile (Android, iOS) operating systems, and even support browser environments through WebAssembly compilation. The same applies to computer architectures, as .NET runs at least on the 32-bit and 64-bit versions of the x86 and ARM instruction sets, with RISC-V being under development. In theory, it should be possible to easily create versions for the aforementioned platforms.

[1] Gitlab code repository: `https://gitlab.com/Spyros3000/tradutorium`
[2] Models repository: `https://gitlab.com/Spyros3000/tradutorium-models`
[3] `https://docs.gitlab.com/ee/user/project/issues/csv_export.html`

The truth, as usual, is more complex. Since Tradutorium relies on a multitude of other libraries, whether for offline machine translations or audio and image transcription, if these are not developed in a programming language that compiles to the CLR and instead utilize *unmanaged code* they will need to be compiled or cross-compiled for specific instruction sets (for example, Windows x86-64 or Linux ARM64).

Because of that complication, some platforms that are easily accessible have higher priority. Therefore, focus has been given mostly to Windows x86-64 and Linux x86-64.

3.6 Implementation details

3.6.1 Internal language code conversion

The libraries discussed in the next section each use different coding systems for the languages they support. Bergamot follows the ISO 639-1 standard, which represents languages with two-letter codes. In contrast, NTextCat and Tesseract use the ISO 639-3 standard, featuring three-letter codes to encompass all known natural languages. Wiktionary takes a different approach, using the more readable English names for languages instead of any coding standard. Since text needs to be passed between these libraries, converters were developed to handle the differences. A CSV *(Comma-Separated Values)* file was created, which is loaded at the start of the application, and is then converted into a .NET object. Whenever a conversion is needed (e.g., between different standards or formats), a function is called to search this .NET object for the appropriate language name, returning the corresponding code or name in the required standard.

Two-letter language code (ISO 639-1)	Three-letter language code (ISO 639-3)	English Wiktionary name
af	afr	Afrikaans
am	amh	Amharic
ar	ara	Arabic
as	asm	Assamese
az	aze	Azerbaijani

Table 3.2: Extract from the different language codes that are needed in the application.

3.7 Deployment

3.7.1 Available deployment methods

Traditionally, .NET applications were deployed as bytecode in the CIL, with the assumption that the .NET Framework or .NET runtime was already installed on the system. When the .NET framework was first introduced in 2002, disk space was limited, and internet speeds

were slow by today's standards. A framework that only needed to be downloaded and installed once, and could then run any application built for it, was an appealing solution [46]. This approach mirrored the deployment strategy of Sun Microsystems' Java platform at the time.

While this deployment method is still available today, it is gradually being phased out in favor of more modern alternatives. With ample disk space and fast internet speeds no longer a concern, the need for a separate runtime installation has become less appealing, especially as users find it unintuitive and cumbersome. Consequently, recent versions of .NET allow for self-contained application deployment, targeting specific runtime environments. These applications use ahead-of-time (AOT) compilation, in contrast to the traditional just-in-time (JIT) compilation employed when executing CIL with a compatible runtime.

The first method, known as **ReadyToRun**, could be described as a middle ground toward full AOT compilation. It reduces the workload of the just-in-time (JIT) compiler by replacing as much intermediate code as possible with precompiled native code. This approach improves startup times but increases application size due to the inclusion of both intermediate and native code. Additionally, the necessary runtime components for executing the intermediate code are bundled with the application. ReadyToRun is a welcome deployment method when performance is critical [47], especially in cases where the more advanced method described below is not feasible.

The second method, **Native AOT**, fully compiles the entire application to native code ahead of time, as can be inferred by the name. This eliminates the need for a runtime, resulting in faster startup times, reduced memory usage, and smaller application sizes compared to those deployed with ReadyToRun. However, not all .NET applications can take advantage of this performance boost. Features that rely on runtime behaviors, such as reflection, dynamic code generation, or dynamic loading, cannot be precompiled because the compiler lacks the necessary runtime information to handle them [48].

3.7.2 Native AOT: challenges and solutions

Starting with version 11, the Avalonia framework removed from its codebase dependencies on .NET internal features that were incompatible with Native AOT, allowing applications built with Avalonia to use this deployment method [50]. However, another issue arose. As of .NET 7 (released in November 2022), Native AOT was primarily designed for console applications. The current stable release, .NET 8 (November 2023), expanded support to a broader range of applications, making it a viable option for this project.

When Avalonia was combined with Native AOT in this project, the transition was largely smooth, as expected. The framework itself posed no issues, however a separate issue arose related to how JSON serialization was handled in the project. Specifically, a custom JSON converter had been implemented to process input from Wiktionary (detailed in *section 4.5*). This converter was designed to parse and handle the structured data from Wiktionary en-

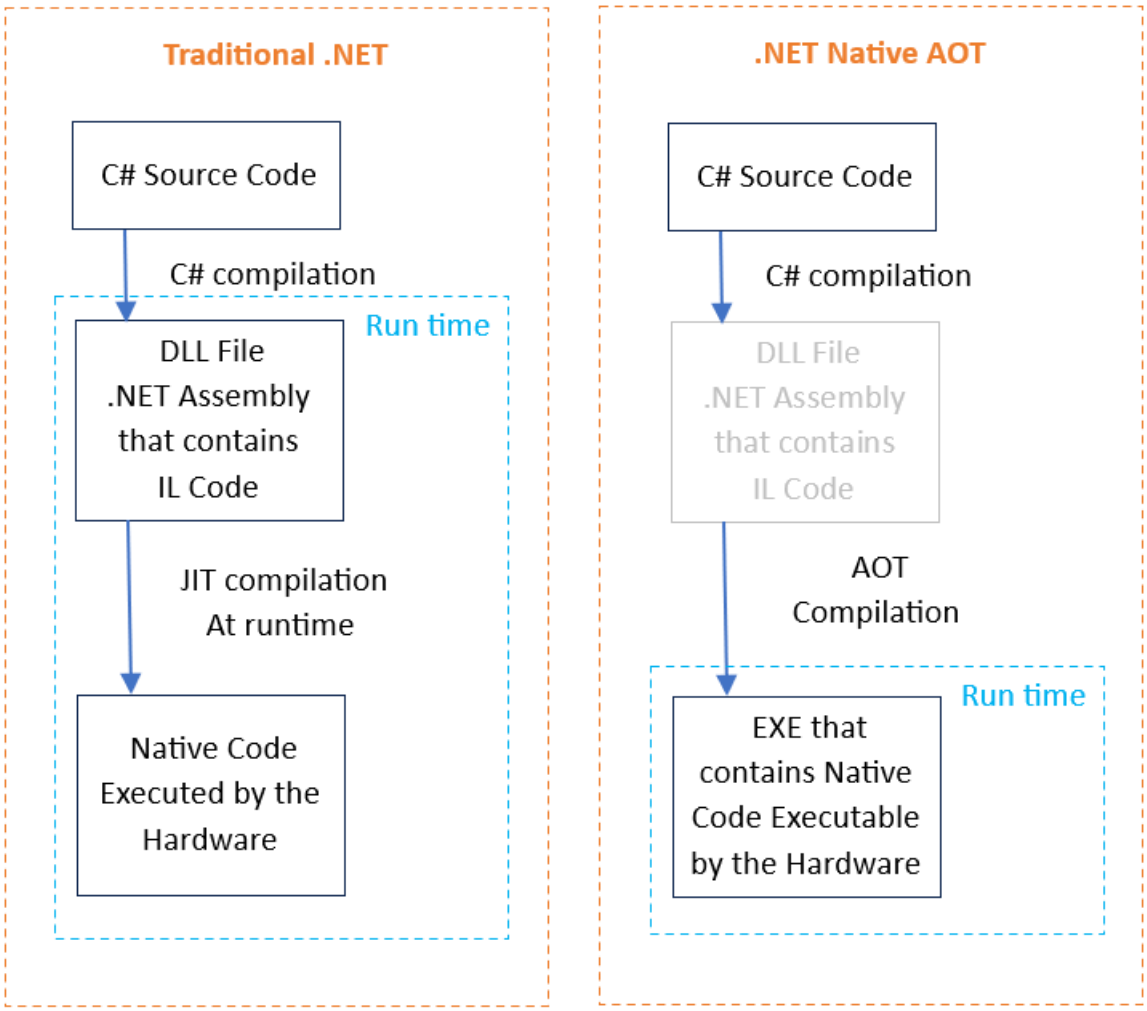


Figure 3.5: Differences between Just-In-Time and Ahead-Of-Time compilation [49].

tries, transforming it into usable objects within the application.

In .NET 8, reflection-based serialization was disabled by default in the .NET JSON serialization library (`System.Text.Json`). Reflection, which is a feature that allows dynamic access to metadata at runtime, is inherently incompatible with the goals of Native AOT. Native AOT aims for static compilation and optimizes executables by eliminating unused code paths and features, so reflection, which relies on runtime inspection and code generation, conflicts with these optimizations [51]. As a result, the custom JSON converter, which had previously relied on this feature, stopped working after the switch to Native AOT. This failure led to the following error message:

System.InvalidOperationException: Reflection-based serialization has been disabled for this application. Either use the source generator APIs or explicitly configure the 'JsonSerializerOptions. TypeInfoResolver' property.

The recommended solution of using source generation for JSON serialization was utilized. Source generation is a compile-time process that generates code specifically tailored to handle JSON (de)serialization for the required types, rather than relying on runtime reflection. Implementing the source generation APIs involved modifying the JSON converter to work with the new `JsonSerializerOptions` and using the `JsonSourceGenerationOptionsAttribute` to pre-generate the necessary serialization logic during compile time. This method provided both performance improvements and Native AOT compatibility. The required adjustments were implemented without significant effort, thanks to the detailed .NET documentation [4].

This results in an executable with a size of roughly 35MB. It's important to note that for Tradutorium to function correctly, the folder containing the executable must also include additional files or directories, such as libraries, models, or configuration files. *Table 3.3* lists the files and folders included in the Windows version, along with an explanation of their purpose.

Filename	Notes
audio_transcription_models	Models for audio transcription
config	Application configuration files
language_codes	CSV with language codes
language_models	Models for machine translation
nonbreaking_prefixes	Prefixes for Bergamot models
ntextcat_models	Models for language detection
runtimes	
tesseract_models	Models for image transcription
x64	
av_libglesv2.dll	
bergamot.exe	Bergamot translator
libHarfBuzzSharp.dll	
libSkiaSharp.dll	
pcre2-8.dll	Bergamot translator dependency
Tradutorium.AudioTranscription.exe	
Tradutorium.ImageTranscription.exe	
Tradutorium.Main.exe	Main application

Table 3.3: Files (in black) and folders (in blue) included in the Windows version of Tradutorium.

[4] `https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/source-generation`

4. FEATURES

In this section, the functionality offered by the application, Tradutorium, is described in detail. In addition, *Appendix I* and *Appendix II* attempt to visualize the technical and functional aspects of the application. Most features can run on a standard recent computer without significant resource needs. Exceptions, such as audio transcription, which may require more robust computing power, are addressed in the accompanying text or relevant tables.

As previously indicated, the project and its source code are available on a GitLab repository, at `https://gitlab.com/Spyros3000/Tradutorium`. The domain `tradutorium.org` is also registered and may serve as the project's future homepage, though it is currently inactive.

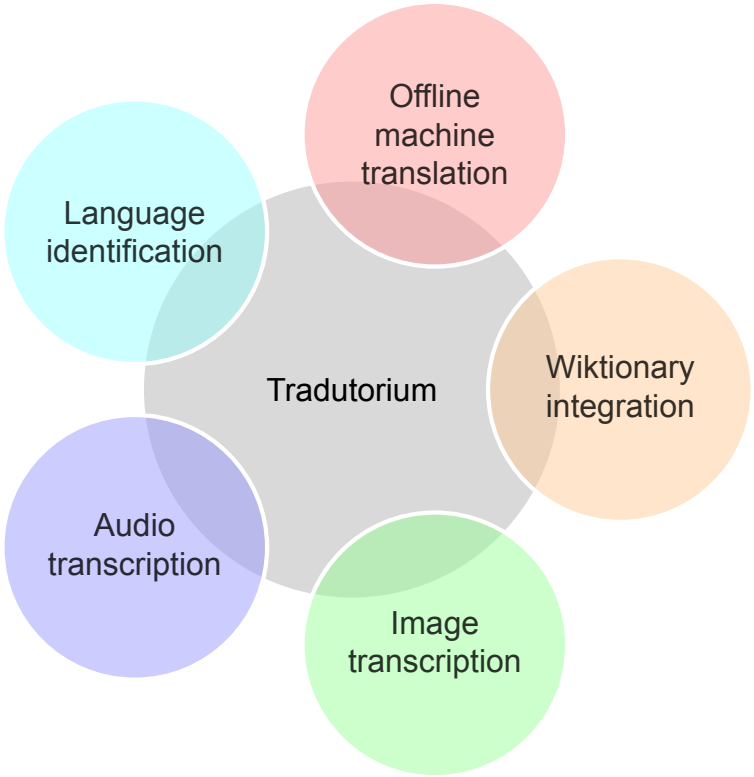


Figure 4.1: Current Tradutorium features.

4.1 Language identification

Language detection is the process of automatically identifying the language in which a given text is written. In this specific case it is the first step in the machine translation workflow. Once the language is detected, the system can select the appropriate translation model, instead of requiring users to select manually the input language.

The input text should be at least a few sentences long to accurately identify the correct language. If the text is too short, there's a higher chance of incorrect language detection. For instance, phrases like "La casa", "Una cosa", "Un momento", "Una mano" have the same meaning in both Italian and Spanish. Similarly, pairs such as "Mio padre/Mi padre", "Il problema/El problema", "Il vino/El vino" differ only in the articles. Other examples like "Una domanda/Una demanda", "La strada/La estrada", "La musica/La música" or "Buona fortuna/Buena fortuna" can easily be confused. Similar overlaps exist between these languages and Portuguese or Romanian, as they all belong to the Romance language family.

Technical details

NTextCat [a] is the language detection system that was used in Tradutorium. It is a C# implementation of the TextCat library by Ivan Akcheurov (originally in Perl, implementation of a popular text categorization algorithm). TextCat's primary purpose is to determine the language of a given text sample (language identification) and is seemingly the most common software for this purpose (Wikimedia uses a PHP port of TextCat). It is an implementation of the Cavnar and Trenkle "N-Gram-Based Text Categorization" technique by Gertjan van Noord [52].

The language models provided by NTextCat have been sourced from Wikipedia, with three options available. The first model, `Core14.profile.xml`, supports only 14 relatively popular languages. The second one, `Wiki82.profile.xml`, covers 82 languages based on Wikipedia editions, and the third, `Wiki280.profile.xml`, supports 280 languages from various Wikipedia editions. Tradutorium defaults to using `Wiki82.profile.xml`. However, since Wikipedia editions do not always match real-world languages, at least one precaution was necessary. If "simple english" is detected (referring to the Simple English Wikipedia, which uses a simplified English vocabulary), an automatic redirection to "english" is performed.

[a] `https://github.com/ivanakcheurov/ntextcat`

4.2 Offline machine translation (Project Bergamot)

Figure 4.2: Bergamot project logo.

The Bergamot project is a collaborative initiative involving several academic teams focused on machine translation, along with the Mozilla Foundation. Together, they aim to develop an efficient system for offline machine translation. The project shares contributors with the Marian NMT effort and relies on that system for training its language models. However, the Bergamot models are designed to prioritize compactness and low computational overhead to ensure they run smoothly on standard hardware. They utilize quantization techniques in order to achieve this goal, converting the continuous, high-precision values typically used

in model parameters into lower-precision representations. This reduces the storage requirements and speeds up inference while minimizing the loss in model accuracy, enabling the model to run efficiently on less powerful hardware [53]. The final models are only 15-20MB in size while still maintaining competitive BLEU scores.

Available models	
Bulgarian → English (bgen)	English → Bulgarian (enbg)
Czech → English (csen)	English → Czech (encs)
German → English (deen)	English → German (ende)
Persian → English (faen)	English → Persian (enfa)
French → English (fren)	English → French (enfr)
Icelandic → English (isen)	
Italian → English (iten)	English → Italian (enit)
Norwegian (Bokmål) → English (nben)	
Dutch → English (nlen)	English → Dutch (ennl)
Norwegian (Nynorsk) → English (nnen)	
Polish → English (plen)	English → Polish (enpl)
Portuguese → English (pten)	English → Portuguese (enpt)
Russian → English (ruen)	English → Russian (enru)
Ukrainian → English (uken)	English → Ukrainian (enuk)

Table 4.1: Pre-trained language pairs provided by project Bergamot as of September 2023.

Perhaps the most important piece of software produced by the project is Bergamot Translator, a “cross platform C++ library focusing on optimized machine translation on the consumer-grade device” that uses language models produced by the project. Even though the main purpose of the project is to facilitate website translation that happens locally, through the Firefox browser, Bergamot Translator can be compiled for many computing platforms.

Technical details

Use in browsers is possible through Emscripten, which allows C or C++ applications and libraries to be precompiled and run efficiently there. This is achieved by utilizing LLVM, an open-source suite of compiler technologies, along with Clang, its associated C and C++ compiler.

Tradutorium relies on Bergamot Translator for translation between languages, which is the main purpose of the application. The software needs to be pre-compiled in order for Tradutorium to later use the produced executable. In lieu of an extensive tutorial from other sources, *section 5* offers a thorough description of the compilation process with the hope of being useful to interested consumers of Bergamot Translator.

4.3 Audio transcription (Whisper) and translation

Speech recognition (also known as automatic speech recognition (ASR) or speech-to-text) is the field developing methods that enable computers and other devices to recognize spoken language, converting it into text or executing commands based on the recognized speech. It involves receiving audio input, analyzing the speech, and identifying the words and phrases spoken by the user. The reverse process is called speech synthesis or text-to-speech.

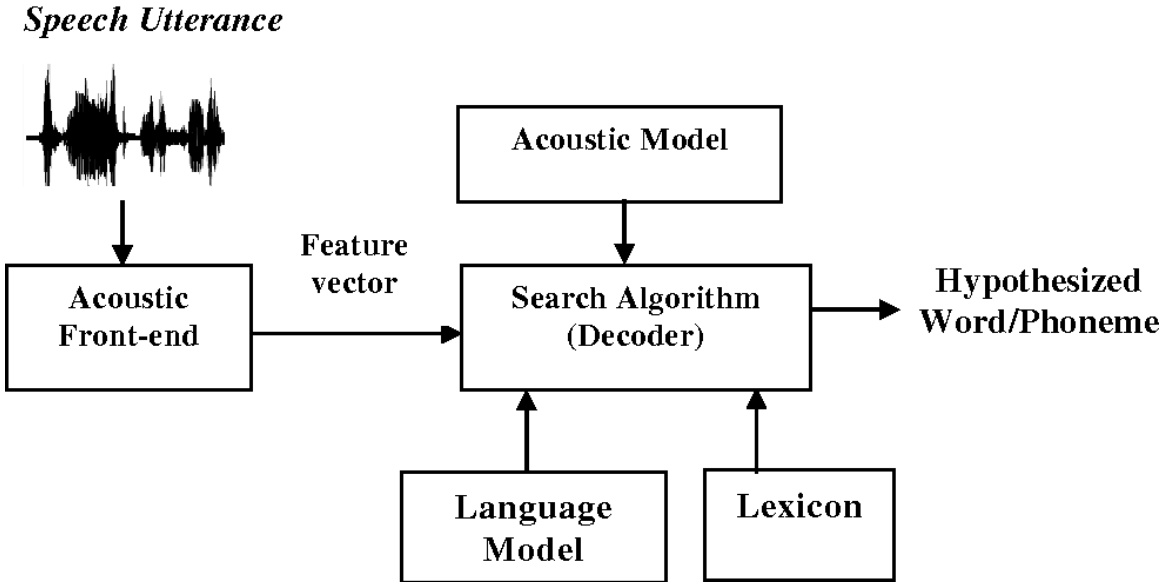


Figure 4.3: A typical speech recognition architecture [54].

Integration of speech recognition within Tradutorium means that users can input pre-recorded audio of human speech, which is converted to text. The text is then used to generate a translation into a pre-selected target language. The original input text should not be deleted immediately, in order for the user to be able to select a different target language and see a new translation.

Technical details

The ecosystem of open-source audio transcription libraries was less mature in late 2022. The most advanced open source speech-to-text library at the time was Coqui STT [a], based on (the now abandoned) Mozilla DeepSpeech [b]. However since then, OpenAI has released a new audio transcription library, called Whisper, that can be used freely by anyone. Whisper is built on a large-scale transformer model, trained on a vast amount of multilingual and multitask supervised data. The latter category involves training on multiple related tasks simultaneously, with each task having its own labeled dataset, but sharing common underlying structures or features. This extensive training allows it to generalize well across different tasks and languages,

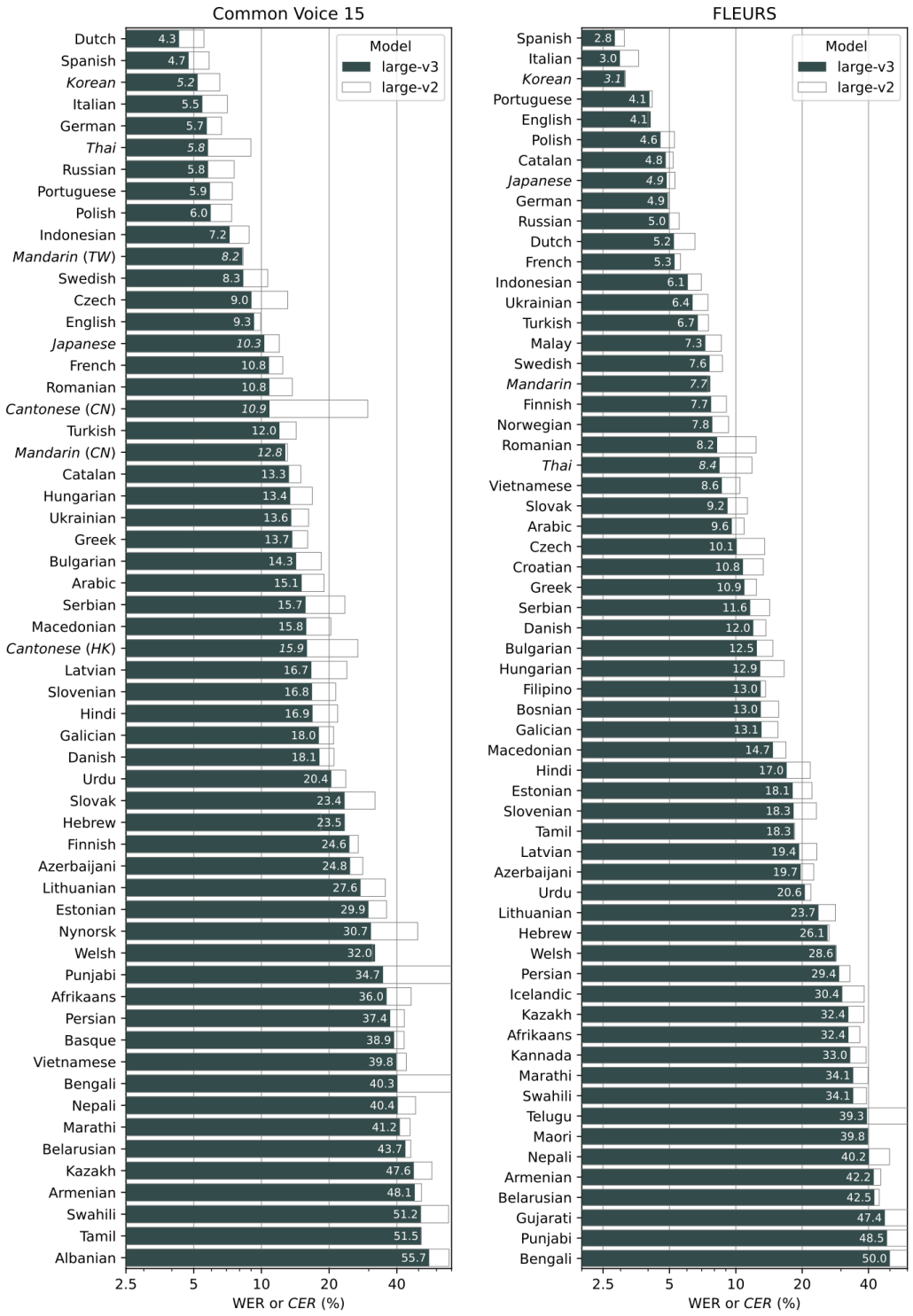


Figure 4.4: Languages supported by Whisper models (as of November 2023) and their WERs (word error rates) or CER (character error rates, shown in Italic) evaluated on the Common Voice 15 and Fleurs datasets.

achieving high accuracy in transcribing speech, especially in controlled environments.

Whisper takes an audio file or real-time audio stream as input. The audio is then processed into features that represent the sound waves' frequency and amplitude. As mentioned previously, a transformer model is used to encode the extracted audio features. This encoding process converts the raw audio into a more abstract representation that captures the nuances of speech, including phonemes and intonation patterns. The encoded features are then decoded into text using a language model that predicts the most likely sequence of words in the multiple languages supported by Whisper. The final output is a transcription of the spoken language in text format that can be afterwards be translated by other means [55]. Whisper models can also provide machine translation for that output, however this feature was not utilized since the functionality is already provided by the MT-focused Bergamot models.

Whisper is designed to run on graphics processing units (GPUs) to take advantage of their performance capabilities. However, Georgi Gerganov reimplemented the library as Whisper.cpp, enabling it to run on standard computer processors (CPUs) [c]. Additionally, Sandro Hanea developed .NET bindings [d] for Gerganov's version, facilitating integration with .NET-based applications. These bindings were utilized in Tradutorium to transcribe audio.

[a] https://github.com/coqui-ai/stt
[b] https://venturebeat.com/business/mozilla-winds-down-deepspeech-development/
[c] https://github.com/ggerganov/whisper.cpp
[d] https://github.com/sandrohanea/whisper.net

Model	Parameters	Size	Memory usage
tiny / tiny.en	39 M	75 MB	∼273 MB
base / base.en	74 M	142 MB	∼388 MB
small / small.en	244 M	466 MB	∼852 MB
medium / medium.en	769 M	1.5 GB	∼2.1 GB
large-v1 / large	1550 M	2.9 GB	∼3.9 GB

Table 4.2: Available Whisper.cpp models. As the list progresses the output quality increases, however the same is true for the memory usage and time needed.

4.4 Image optical character recognition (Tesseract) and translation

In the same way that Tradutorium handles audio transcription for translation, image transcription has also been integrated. The resulting text can then be translated via the application's built-in machine translation models. This is possible through a process called *Optical character recognition* (OCR).

OCR refers to the problem of converting text that is contained in images into machine-encoded text. The input may consist of typed, handwritten or printed text from various sources. The quality of the transcription depends on the readability of the input image, the variety of languages, fonts and styles in which text can be written, the complex rules of languages and other factors [56]. It is important in order to automate mass editing or processing of analog documents or texts, that would not be possible by human manpower alone. It also significantly helps visually impaired users, who can then use accessibility software with the machine-encoded text. In the context of machine translation, it can be considered as a first step, with the second being the translation of the recognized text in the target language.

The idea of OCR dates back to the 19th century. The first primitive devices, designed to help the blind appeared in the early 20th century, with systems that correspond to the modern meaning of the term being available around the mid of the century, interpreting Morse codes. In later decades the field found increasingly wide support in the industry, reading passports and price tags, before finally becoming available to consumers [56].

The process of OCR can be examined by analyzing the multiple stages that are needed for the creation of a resulting output. They are usually the following [56]:

- **Pre-processing**: enhancing the quality of the image
- **Segmentation**: separating the image into characters
- **Feature extraction**: extracting features from the image
- **Classification**: categorizing characters into classes
- **Post-processing**: improving the accuracy of the results

The classification step of OCR until recently used traditional classifiers such as Bayesian classifier, decision tree classifier or nearest neighborhood classifiers [56]. Nowadays, state-of-the-art systems utilize neural networks, trained on large datasets, to recognize written text. The results are superior compared to previous approaches.

Technical details

The most advanced and popular open-source OCR system available for everyone to use is Tesseract. Originally developed by Hewlett-Packard labs in the 1980s, it was later open sourced [57]. Following the latest technological advances, version 4 added an LSTM-based engine. Its quality is lower than that of closed-source competitors [58], however it can be considered more than adequate in the majority of cases, as evidenced by a recent testimonial by the Internet Archive team [59]. Accuracy for common languages using the Latin script is significantly higher than languages with different scripts [58].

A .NET wrapper for the library exists, created by Charles Weld [a]. It supports the pre-trained neural models for more than 100 languages that are available from the official Tesseract project [b]. It is also cross-platform, although the Linux version of the wrapper must detect a couple of dependencies in the system in order to run as intended [c].

[a] https://github.com/charlesw/tesseract/
[b] https://github.com/tesseract-ocr/tessdata/
[c] https://github.com/charlesw/tesseract/issues/503

4.5 Wiktionary integration

Figure 4.5: Logo of the English Wiktionary.

Wiktionary is “a collaborative project to produce a free-content multilingual dictionary, which aims to describe all words of all languages using definitions and descriptions in English.” It is managed by the Wikimedia Foundation as its sister project, Wikipedia, and aims to provide definitions, translations, pronunciations, etymology, and other linguistic information for words in various languages. It has grown to be one of the most substantial linguistic resources on the Internet. It is available in more than 190 languages of varying wealth.

Tradutorium aims to integrate with the most plentiful version, the English Wiktionary. The purpose of this integration is to enable easy lookup of definitions within the input text, with the first found definition being displayed. This feature may be incorporated into other areas of the application later. It should be noted that the English Wiktionary includes definitions for words in multiple languages, not only English. The same is true for the other available versions. For example, the Greek Wiktionary offers definitions of words in English, French and so forth.

Technical details

An official API is available for the English Wiktionary [a] and it was used in order to get the definition of a selected word. The data are returned in JSON format, as an object that contains key/value pairs. Therefore, they cannot be automatically deserialized by the `System.Text.Json` library that is included with .NET. A JSON converter needed to be implemented in order to recognize the structure of the provided JSON file and parse the definitions. After the process of deserialization, the converted object can be easily used by Tradutorium, in order to display at least the first definition of a word, with a hyperlink to the relevant Wiktionary webpage in case the user wants to consult other probable definitions.

The definitions are stored in a list of .NET objects that were created specifically for this purpose. All the languages with definitions of the searched word are available in that list. Tradutorium then presents the first available definition from the language that the input was detected to be in.

The Wiktionary definition contains markup, some of which is unique to that service. In a second step, the definition is converted to valid HTML that can be displayed by an HTML control. This step replaces the Wiktionary markup with hardcoded values or URLs that correspond to the English Wiktionary website. As the Avalonia TextBlock cannot display HTML text, the Avalonia HTML renderer was used [b].

[a] `https://en.wiktionary.org/api/rest_v1/`
[b] `https://github.com/AvaloniaUI/Avalonia.HtmlRenderer`

4.6 Pivot languages

The language pairs that have been trained as part of project Bergamot are currently English-oriented, as can be seen in *Table 4.1*. This means that one of the input or output languages is English, therefore it is not possible to translate directly between two other languages, e.g. French and Italian. However, since in this specific example the French-English, English-French, Italian-English and English-Italian language pairs exist, another method can be used, which includes two translation steps. It uses an intermediate (pivot) language (which in this case is the English language) as the output language of the first step. A visualization of the concept can be seen in *Figure 4.6*. Afterwards, it translates again, from the pivot language to the requested language. In the previous example, this means that instead of translating directly from French to Italian, it uses the following path: French $\rightarrow$ English $\rightarrow$ Italian.

The double translation process can introduce errors and compound inaccuracies, leading to a lower overall translation quality compared to direct translation. Each translation step may introduce its own set of errors, and nuances of meaning can be lost or distorted. Practical experience, though, shows that these problems are underrepresented in languages that show

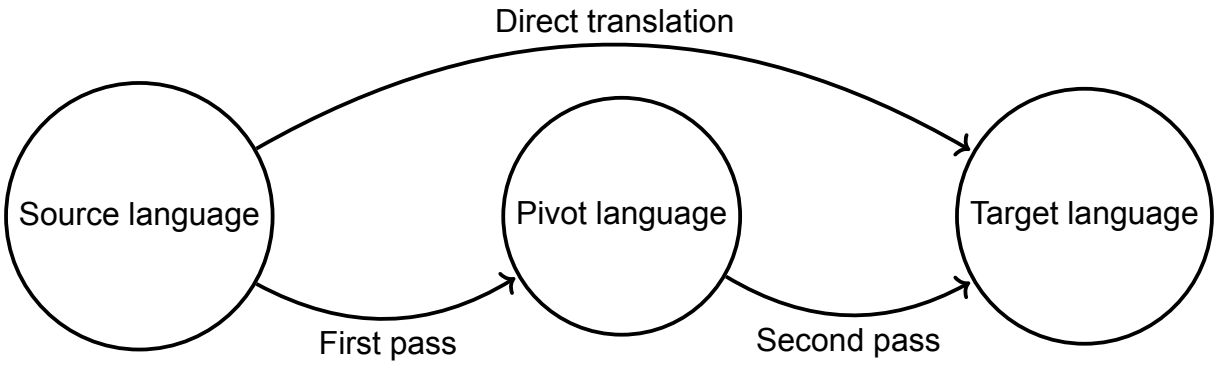


Figure 4.6: Direct translation between source and target language versus translation through a pivot language.

linguistic similarities and that indirect translation is a worthwhile technique, not to be underestimated. It also makes the system much more scalable: in the possible case of translating between hundreds of languages, one would need to train thousands of direct language pairs. However, by using a pivot language, one only needs to train a number of models that is equal to the number of supported languages multiplied by two (since translations from and to the target language need to be supported).

4.7 Application localization

Although it is not fundamental to the application goals, the Tradutorium UI is available in multiple languages. Four translations are currently provided (English, Greek, Italian and French), and more can be added if willing translators are found. Currently, the Tradutorium UI is fairly simple, and the supported languages are part of the Indo-European family, sharing similar traits. However, if the application is to be localized into a wider range of languages with varying characteristics, particular care should be taken to address common localization challenges. These include supporting Right-to-Left (RTL) languages, handling complex plural forms, managing gendered languages, and adapting the UI layout to accommodate longer or shorter text strings.

Filename	Language	Notes
`Resources.resx`	English (en-US)	Default, fallback language.
`Resources.el-GR.resx`	Greek	
`Resources.fr-FR.resx`	French	
`Resources.it-IT.resx`	Italian	

Table 4.3: Translations in Tradutorium and the corresponding `.resx` files.

Technical details

The code contains no hardcoded strings but descriptive identifiers (IDs), which serve as placeholders. During the runtime, the corresponding translations for these IDs are loaded and shown. In .NET applications, localized resources are stored in .resx files. These are XML files used to store key-value pairs, where keys represent the text or UI element, and values are the translations [60]. Each localization originates as a simple text file, which is then converted to the .resx format by the `resgen` tool [a]. Each .resx file corresponds to a specific language and culture, as can be seen in *Table 4.3*.

The application's UI language can be changed via the Settings menu, and the change takes effect after restarting the application. The selected language is stored in the `lang.txt` file located in the configuration directory, which is checked during each application launch. The `Culture` (also known as locale outside of .NET) is then set to match the selected language, and the `ResourceManager` class in .NET loads the corresponding localized resources.

[a] `https://learn.microsoft.com/en-us/dotnet/framework/tools/resgen-exe-resource-file-generator`

5. BERGAMOT TRANSLATION COMPILATION PROCESS

Tradutorium relies on Bergamot Translator for translation between languages, which is the main purpose of the application. The software needs to be pre-compiled in order for Tradutorium to later use the produced executable. The compilation process is described in detail in the hopes of being useful to interested consumers of Bergamot Translator, in lieu of a non-existent extensive tutorial in other sources.

5.1 Compiling on Unix

5.1.1 Required dependencies

Compiling Bergamot Translator on Unix systems is designed to be a straightforward process, especially given that Unix-based environments are often considered the primary platform for this project. However, before compiling, it's necessary to install the required dependencies, such as libpcre2-8-0 (*Perl Compatible Regular Expressions*, a regular expression library) and OpenBLAS (a highly optimized *Basic Linear Algebra Subprograms*, or BLAS, library). This is a simple process thanks to the wide availability of software in the official Debian repositories. OpenBLAS was chosen over Intel MKL (*Math Kernel Library*) for this reason, as MKL (which is used in the Windows version, described below) requires a more involved setup.

However, the BLAS library can be switched later if needed by manually installing Intel MKL and specifying its installation path before starting the compilation process. This might be necessary in cases where performance issues arise. Several comparisons have reported noticeably better performance with Intel MKL compared to OpenBLAS [1], including those involving TranslateLocally which uses Bergamot Translator internally [2].

5.1.2 Compilation overview

As expected, the compilation process proceeded without any complications by following the instructions provided in the repository. The executed commands effectively cloned the repository, set up the build environment, and compiled the project, leading to a successful build with no issues encountered.

```
git clone --recursive https://github.com/browsermt/bergamot-translator.git

mkdir build-native
cd build-native
cmake ../
make -j2
```

[1] https://phabricator.wikimedia.org/T247245

[2] https://github.com/XapaJIaMnu/translateLocally/blob/3cbe86d622e93b8a19c1e28f2ff86e1e62b9ba2e/README.md#openblas

5.1.3 Challenges and solutions

During compilation of the software on a Linux system, an unexpected problem was encountered, because of a software bug that exists in a core system component.

The Linux compilation was tested in Debian 12 *"Bookworm"*, the latest stable release of that Linux distribution. Unfortunately, this operating system bundles GCC 12.2 as the default compiler. This version of GCC is hit by a bug that detects false positives [3], and prevents compilation when warnings are treated as errors. This affects intgemm, a core dependency of Bergamot Translator. Even though a patch was submitted to intgemm [4], the problem persists. The available workarounds were to either downgrade to GCC 11, upgrade to a newer version of GCC (either GCC 12.3 or GCC 13) or to add the argument `-Wno-uninitialized; -Wno-maybe-uninitialized;` to appropriate places in the `bergamot-translator/3rd_party/marian-dev/CMakeLists.txt` config file, in order to turn off the relevant warnings. The third option was chosen.

5.2 Compiling on Windows

5.2.1 Required dependencies

The compilation process on Windows required significantly more effort compared to the somewhat straightforward Unix compile process. First of all, the following prerequisites needed to be installed in the system before the compilation process could be started:

1. *Visual Studio*, with the "C++ desktop development" component selected,
2. *Git for Windows*,
3. *Intel MKL* (*OpenBLAS* can also be used as an alternative, although it was not tried on Windows),
4. *vcpkg*, a package manager for C and C++ development. It should be installed in a filepath that does not contains whitespace and it will be used to install the boost and pcre2 dependencies.

The installation process is easier in Unix environments, with the dependencies being available just by running a few commands. In the Windows world each necessary software must be downloaded and installed separately. vcpkg should be downloaded from its Git repository and then compiled. Only then can the boost and pcre2 dependencies be installed:

```
git clone https://github.com/Microsoft/vcpkg.git
.\vcpkg\bootstrap-vcpkg.bat
vcpkg install boost:x64-windows
```

[3] `https://gcc.gnu.org/bugzilla/show_bug.cgi?id=105593`
[4] `https://github.com/kpu/intgemm/issues/101`

```
vcpkg install pcre2:x64-windows
```

To ensure that the system can locate PCRE2 on Windows, the directory where the library is installed must be added to the system's PATH environment variable. The PATH variable tells Windows where to look for executable files and libraries, so any tools or programs that rely on PCRE2 will be able to find and use it from the command line. On the test system, the full file path looked like this:

```
C:\vcpkg\installed\x64-windows\bin
```

5.2.2 Compilation overview

To download the Bergamot Translator code from the official GitHub repository, you will need to execute a series of commands that resemble typical Unix command-line processes. These commands will help you clone the repository to your local machine, allowing you to access the source code and make any necessary modifications or enhancements.

```
git clone --recursive https://github.com/browsermt/bergamot-translator.git
mkdir build-native
cd build-native
```

Then, the `x86_x64 Cross Tools Command Prompt for VS 2019 / 2022` should be launched. The standard Windows command line is inadequate to continue the compilation process, because the Visual Studio installation that exists in the system should be used. This allows the command line to access the various installed Visual Studio components related to C++ development. However, this step alone is not sufficient. After initializing the command prompt, another script needs to be executed, this time focusing on the various dependencies for the Bergamot Translator. This script will determine the installation filepath of these dependencies or download them as needed. The `CheckOrInstallDeps.bat` script [5] from Marian NMT can be used for this purpose. It should be copied in the `bergamot-translator` directory (which was created automatically when the repository was cloned) and then modified to include the paths for the dependencies, such as Intel MKL.

Finally, the following commands should produce a `bergamot.exe` that can be used by Tradutorium. The second command configures the project using CMake, specifying that it should generate build files specifically for the NMake build system, which is the standard make utility for Microsoft development environments. Using NMake is necessary for Windows-based development, particularly when working with Microsoft Visual Studio tools. The project is to be built in Release mode, for use by end users.

```
cd <directory_filepath>\bergamot-translator\build-native
cmake ../ -G "NMake Makefiles" -DCMAKE_BUILD_TYPE=Release
```

[5] `https://github.com/marian-nmt/marian/blob/master/vs/CheckOrInstallDeps.bat`

As a side note, in order for Bergamot Translator to run on end-user systems where the above compilation process has not been followed, the `pcre2.dll` library should be copied in the same folder as `bergamot.exe`, since it is a dependency that could not be compiled into the final executable file. If this dependency is not detected in the runtime, `bergamot.exe` cannot be executed. This specific file can be found in the `vcpkg` installation path.

6. MODEL TRAINING FOR LANGUAGE MODEL

An English-to-Greek machine translation language model was created, in the same manner that existing Bergamot models have been produced. Bergamot does not offer a pre-trained model for this language pair (as of September 2024) and it could fill this need. It also allowed to get a deeper look at the insights of the project and the challenges the creation of a model entails.

The necessary instructions for training a Bergamot-compatible language model can be found in the project's repository [61]. The provided scripts include tailored configurations in order to achieve the desired results. For additional guidance, the MarianNMT documentation was also consulted when necessary. Supplementary information was found on the personal web-pages of Bergamot Translator developers [62].

The model was trained on an Nvidia TITAN Xp, with 12GB of VRAM, graciously provided by the Institute for Language and Speech Processing (ILSP). Marian v1.11.0 was used during training [63]. Hardware constraints meant that the corpus of texts was smaller than other tries. The Europarl dataset [64] and the DGT 2018 dataset [65] were combined and used, as can be seen in *Table 6.1*. Validation sets included a part of Europarl that was not used when training. To evaluate, the WMT datasets could not be used as in the examples, since they do not include Greek corpora as part of their language pairs. Instead, a Europarl test set (with unseen data that were not used during the training) was used as a dataset with the same genre of training data and the devtest of the Flores200 dataset as data of different genres without significant overlap with the type of training data [67]. Sacrebleu was utilized to generate the BLEU and Chrf scores.

Dataset	Number of Lines	Greek sentences	English sentences
Europarl	1,229,000	1,428,828	1,343,274
DGT 2018	4,900,254	8,082,676	7,832,718
Total	**6,129,254**	**9,511,504**	**9,175,992**

Table 6.1: Training dataset, Europarl and DGT 2018.

The main stages of the model creation included:

- Data preparation from parallel corpora in English and Greek,
- Training of the teacher language model,
- Word alignment / knowledge distillation from teacher to student,
- Training of the student language model,
- Quantization of the student model to achieve a shrunk-down model,

In the context of machine translation, and also in terminology used by MarianNMT and the Bergamot project, teacher-student models (a process also known as knowledge distillation) are a powerful technique to improve efficiency without compromising translation quality [68]. The teacher-student framework enables training a smaller, more efficient model (student) by utilizing the knowledge learned by a larger, more complex model (teacher). This approach is particularly useful in resource-constrained environments. The student models can be more easily executed on a CPU instead of a GPU that offers more performance, at the cost of higher prices and not wide availability.

The terminology used refers to a "teacher model" as the original model trained by Marian NMT and "student model" as the final, quantized model and whose hardware requirements are significantly lower's than those of the teacher. In this chapter, when a filename is mentioned it is a subdirectory to the `https://github.com/browsermt/students/tree/819f41962c8867859f8e511dcba9b63fc997db82/train-student` repository, which includes the scripts needed to train a Bergamot model.

6.1 Teacher training

After the parallel corpora preparation the training of the teacher model began. As is usual, it is based on a transformer architecture which achieves competitive results in the last decade. The training process took about 9 days to complete until 45 epochs were reached. The model with the best performance on the validation set was selected as the final teacher model, after comparing with the existing highest score at each checkpoint. The full size of the produced teacher model is 357MB.

The generated model was a transformer and the most important parameters used during the training process are shown in *Table 6.2*. The full parameters used during training can be found in *Appendix III*.

Parameter	Value	Explanation
enc-depth	6	Number of encoder layers (s2s).
dec-depth	6	Number of decoder layers (s2s).
dropout-rnn	0.2	Scaling dropout along rnn layers and time.
dropout-src	0.1	Dropout source words.
dropout-trg	0.1	Dropout target words.

Table 6.2: Principal parameters used during the training of the teacher model.

6.2 Word alignment

This step generates the word alignment and lexical shortlists from the teacher model that will be used during the training of the student model. The so-called teacher model's soft targets are the probability distributions over the possible translations produced by the teacher model,

which not only include the correct translation but also reflect the probability of alternative translations. The soft targets provide richer information about the teacher model's decision-making process and inform the student model not only what the correct translation is but also what alternatives are likely and how confident the teacher is in its prediction. In contrast, hard targets include only the correct translation and ignore all other possibilities.
The `alignment/install.sh` installed the `fast_align` and `extract-lex` tools needed for this step and then the `alignment/generate-alignment-and-shortlist.sh` script generated the word alignment and lexical shortlists.

6.3 Student training

A student model is a compressed version of a larger, more powerful teacher model. It is trained to replicate the behavior of the teacher model, which has been trained on a large dataset and achieves high accuracy but at the cost of computational complexity. The student model learns not only from the raw data but also from the teacher's predictions (often referred to as soft targets) during knowledge distillation.

Following the instructions of the Bergamot project, the student model was fine-tuned by emulating 8bit GEMM *(General Matrix Multiplication)* during training during the training, in order to prepare for the following quantization step and achieve better results. A model without finetuning was also prepared in order to compare results and determine the significance of this step. More details are available at the `finetune` subdirectory.

The student model is trained using both the original training data and the word alignments and lexical shortlists distilled by the teacher model. The script that was modified can be found at `models/student.base/train.sh`. In the same directory an evaluation script exists (`eval.sh`) which was not used. Instead Sacrebleu was utilized directly with the custom datasets mentioned earlier.

The full parameters used during training can be found in *Appendix IV*. The student model is more compact, as it has a size of only 163 MB.

6.4 Quantization

Quantization in model training is an optimization technique used to reduce the computational and memory requirements of a machine learning model by representing its weights and activations with lower precision. It is especially useful in student models that are distilled from larger teacher models, as it enables further compression while preserving much of the accuracy learned during distillation. The process involves converting the model's parameters from high-precision formats, like 32-bit floating-point (FP32), to lower-precision formats, such as 16-bit floating-point (FP16), 8-bit integers (INT8), or even lower, significantly decreasing the memory footprint and computational cost but also the size of the model [30, 69, 70]. This step is essential if everyday hardware has to be supported. The lack of quantization can

lead to higher precision computations, which may require more memory and computational power, usually available only to expensive graphic processing units [69].

The quantization process compares competitively with the full models, at a fraction of their size. There is by design a marginal accuracy loss, however advances in quantization help to mitigate it and the benefits continue to surpass the drawbacks of the method [69]. For high-stakes applications requiring exact precision, mixed-precision quantization is used, a method where different layers or parts of a model are quantized to varying bit-widths. It helps retain accuracy by preserving higher precision in sensitive and efficient layers, and only apply low-precision quantization to insensitive and inefficient layers [70].

As mentioned, during the training process, quantization was simulated to reduce the BLEU hit. This approach allows the model to adjust and compensate for the reduced precision while it is being trained. The opposite would be Post-Training Quantization (PTQ), where quantization is applied as a post-processing step and a slight drop in accuracy may occur [69, 70].

The intgemm library [1] is used for quantization of Bergamot models to 8-bit integers. Unfortunately the quantization step could not be completed because of a segmentation fault during the execution of the last command. The fault produced an error message similar to an existing issue in the Firefox Translations Training repository [2]. Marian 11.2 was tried to see if the error can be bypassed, without success.

6.5 Results

The evaluation on the Europarl test set yielded acceptable results, as expected, given that the models were trained on a similar dataset. Although the test set was not included in the training data, the vocabulary often overlaps. This overlap is significant because models typically rely on familiar words and phrases to generate accurate translations. Consequently, the training allows the models to develop a robust understanding of the language and context likely to appear in the Europarl corpus.

	BLEU	**ChrF**
Teacher model	28.6	51.7
Student model	20.4	41.1

Table 6.3: BLEU and ChrF results of the trained models on the Europarl test set.

However, in a dataset that contains significantly different data, such as Flores200, the scores dropped significantly. This indicates that the models struggle to generalize beyond their training contexts. The Europarl and DGT datasets do not represent the full linguistic diversity or contextual variety found in real-world applications. The Flores200 dataset may contain specific phrases or genre-specific jargon from the 842 web articles it was sourced. Therefore, the

[1] https://github.com/kpu/intgemm/

[2] https://github.com/mozilla/firefox-translations-training/issues/450

model, trained mostly on formal instances of speech or notation, has not encountered before the informal way of writing that is commonly used in web content and cannot comprehend it in order to translate.

	BLEU	ChrF
Teacher model	14.4	36.0
Student model	3.0	18.0

Table 6.4: BLEU and ChrF results of the trained models on the Flores test set.

Finally, a student model was created without using the finetuning command that is recommended during the latter's training process. This approach allowed for a direct comparison between the two training methodologies. As anticipated, the evaluation scores for the non-finetuned student model were similar to those of the finetuned model, but slightly lower. This decrease can be attributed to the lack of additional refinement that finetuning provides, which helps optimize the model's performance on specific tasks or datasets.

	BLEU	ChrF
Student model (Europarl)	19.8	40.5
Student model (Flores)	2.2	16.4

Table 6.5: BLEU and ChrF results of the trained student models without the finetuning step.

In NMT, models sometimes default to generating empty outputs when they encounter challenging phrases or contexts that lack sufficient training data. This behavior can stem from different explanations, the most common being that the correct-length translations are outscored by empty sentences due to label smoothing or the high-frequency of End-of-Sentence (EoS) tokens. This leads the model to avoid errors by outputting nothing instead [71]. This was also encountered here, both in the teacher and the student models.

A comparison was made with other models trained by project Bergamot and used in Tradutorium. These models have been quantized, perform significantly better and show that the process can generate solid results. Reasons for the low performance of the experimental en$\rightarrow$el model can be attributed to insufficient training data, since it was trained on a significantly smaller amount of sentences, whereas the Bergamot / Mozilla models are trained with vast datasets, such as Opus [66]. Other probable reasons were the use of parameters during the training process as well as the domain / genre mismatch, because the training data was mainly of legal nature, while the evaluation used a more generic dataset.

	BLEU	ChrF
bg $\rightarrow$ en model	37.9	63.6
fr $\rightarrow$ en model	41.9	65.6

Table 6.6: BLEU and ChrF results of other pre-trained models on the Flores test set.

7. FUTURE DIRECTIONS

7.1 Addition of more Bergamot-compatible models and automation of the necessary steps

After Firefox introduced its offline translation capabilities through Project Bergamot, the development of additional translation models accelerated. A new repository was created to host these models [1], and the current list can be found in *Table 7.1*. Furthermore, some language pairs have been enhanced, with updated versions replacing the older ones.

These new models can be integrated into Tradutorium. However, the integration process is currently manual and involves several undocumented steps to ensure the program can locate the models in specific file paths. Given the increasing number of models, this approach is time-consuming and error-prone. Ideally, this process should be automated to minimize human involvement. Additionally, it's important to manage model versions carefully to avoid using outdated ones when newer versions are available.

Available models	
Bosnian → English (bsen)	
Catalan → English (caen)	English → Catalan (enca)
Croatian → English (hren)	English → Croatian (enhr)
Danish → English (daen)	English → Danish (enda)
Estonian → English (eten)	English → Estonian (enet)
Finnish → English (fien)	English → Finnish (enfi)
Greek → English (elen)	
Hungarian → English (huen)	English → Hungarian (enhu)
Indonesian → English (iden)	
Latvian (Lettish) → English (lven)	English → Latvian (Lettish) (enlv)
Lithuanian → English (lten)	English → Lithuanian (enlt)
Maltese → English (mten)	
Romanian → English (roen)	
Serbian → English (sren)	
Slovak → English (sken)	English → Slovak (ensk)
Slovenian → English (slen)	English → Slovenian (ensl)
Spanish → English (esen)	English → Spanish (enes)
Turkish → English (tren)	English → Turkish (entr)
Vietnamese → English (vien)	

Table 7.1: Language pairs provided by Firefox translations as of September 2024, either in production or in development.

[1] `https://github.com/mozilla/firefox-translations-models`

7.2 Dynamic download of models

The ability to download a version of the application with no or minimal models included and then downloading the necessary models on demand could be useful. In this case, a reliable and permanent web service is essential, as the application's functionality would be compromised if the provided link becomes unavailable. The `tradutorium.org` domain has already been registered and could potentially be used for this purpose.

Currently, the only models in Tradutorium that can be dynamically downloaded are the Whisper GGML models [2], used for audio transcription. These models are supported by a custom downloader implemented by the whisper.cpp/whisper.net projects and are hosted on Huggingface, which is a stable platform unlikely to shut down.

Mozilla also supports neural machine translation in the Firefox browser, with dynamic loading of Bergamot project models. Initially available as a Web extension [3], this functionality was integrated into the core browser in September 2023 (version 118), allowing language pairs to be downloaded on demand. Later versions of Firefox implemented various improvements to the translation functionality.

7.3 Audio recording, before the transcription

Allowing users to record audio directly within the application and then providing options to transcribe and translate it would be a desirable feature. Currently, the audio must be recorded separately and then imported into Tradutorium. Although there is a cross-platform API for sound devices (OpenAL [4]) and a .NET library (OpenTK [5]) that offers OpenAL bindings among other functionalities, implementing this feature proved to be complex.

To address this, a user interface should be designed that allows users to select the correct input audio device from the available options. The recording should continue until the user sends a stop signal, such as clicking a "Stop" button. However, OpenTK has recently re-architected the programming classes it offers, making the previous methods for handling audio capture somewhat obsolete. As a result, a new, less-documented approach will be needed to solve the audio capture problem.

7.4 Support and testing of more desktop computing platforms

As mentioned previously, Tradutorium depends on many underlying libraries. Most of them need to be compiled for the various system architectures that are supported. In practice, this means that due to the extreme prevalence of the 64-bit x86 architecture, the application has

[2] Georgi Gerganov Machine Learning models

[3] `https://addons.mozilla.org/en-US/firefox/addon/firefox-translations/`

[4] `https://www.openal.org/`

[5] `https://opentk.net/`

been tested only on x86-64 Windows and x86-64 Linux targets.

The following targets therefore should be targeted and tested thoroughly: Windows ARM64, Linux ARM64, macOS x86-64, macOS ARM64. Many more CPU architectures are available, but they may not be in common use or are not supported at the time by .NET (like RISC-V, which is under development). At the current moment, only 64-bit need to be supported, since 128-bit architectures are nonexistent and 32-bit architectures have low and dwindling demand, mainly because of their inability to handle the large amount of memory used by most systems.

Platforms	Platform priority	Type of platform
x86-64 Windows	Tier 1	Desktop
x86-64 Linux	Tier 2	Desktop
x86-64 macOS	Tier 2	Desktop
ARM64 Windows	Tier 2	Desktop
ARM64 Linux	Tier 2	Desktop
ARM64 macOS	Tier 2	Desktop
ARM64 Android	Tier 3	Mobile
ARM64 iOS	Tier 3	Mobile
WebAssembly	Tier 3	Web, general
RISC-V Linux	Tier Unknown	Desktop

Table 7.2: Platforms that Tradutorium aims to support. Tier 1 platforms have been tested and work out of the box. Tier 2 platforms should work without major changes but have not been tried. Tier 3 support requires significant additions to the application, such as a mobile UI and ways to communicate with the underlying dependencies. Finally, Tier Unknown platforms are not yet supported by the .NET runtime and will be evaluated when this changes.

7.5 Mobile version

A mobile version would be highly beneficial, as the need for accessible machine translation often arises when traveling or in outdoor settings. While this is achievable, it requires extensive reimplementation to redesign the user interfaces to be mobile-friendly. Additionally, there may be significant challenges with the underlying libraries or mechanisms. For instance, currently, Tradutorium interacts with Bergamot Translator via a shell console that processes commands and outputs translations. While shell consoles are typical on desktop platforms, they are not as prevalent on mobile operating systems, which are more restricted and locked down.

7.6 WebAssembly version

A WebAssembly version would be beneficial for running the application on various browsers and browser-only platforms, such as Chromebooks. However, it's important to distinguish

between two different operational modes. An offline web application has access to all necessary program files, libraries, and models stored locally, allowing for instant loading. In contrast, an online web application relies on remote systems to load most of its data, making it dependent on network latency and speed.

In the online mode, only the essential files needed for the application's basic operation should be loaded initially, with additional resources requested as needed. Some features may be more limited or even unavailable in this mode. For example, while it is feasible to load the base Whisper model for audio transcription (140MB), doing so may take some time. However, loading the medium model, which is much larger at 1.8GB, might be impractical or impossible in certain situations.

7.7 Multiple translation engines and language pairs

Right now, translation is possible using the linguistic pairs that have been prepared as part of project Bergamot and trained with Marian. Through the process of quantization they have been extremely minimized and they are optimal for use in applications, because of their size and performance. There is nothing inherent though, that prevents Tradutorium from using other NMT engines and corresponding models, assuming they can be integrated without unexpected problems of course. The performance and size in most cases is not as optimized as the Bergamot models, but their use would be decided by the user, who would allow or not the use of specific models depending on their performance, license or supported languages. Possible targets for integration are:

- Pre-trained language pairs of varying size (larger than Bergamot's) are provided by the ArgosTranslate project for several languages [6]. The license details are unclear, although the models have been trained on open datasets and are already used by LibreTranslate.
- A multilingual model called NLLB-200 *(No Language Left Behind)* [67]. It was created by Meta and supports roughly 200 languages, many of them not well supported by existing tools [7]. It should be noted however that, due to the model's license, it can only be used in non-commercial settings. *Table 7.3* presents a comparison among the available versions of this model.
- Decoder-only transformer-based large language models (LLMs) that may offer better results in machine translation compared to previous approaches.

All the aforementioned models can be presumably used by the CTranslate2 library, if the latter is integrated, similarly to Bergamot Translator. The project claims compatibility with these specific model types [8].

[6] `https://www.argosopentech.com/argospm/index/`
[7] `https://ai.meta.com/blog/nllb-200-high-quality-machine-translation/`
[8] `https://github.com/OpenNMT/CTranslate2`

Model	Model type	Params	Size
NLLB-200-Distilled	Dense	600M	∼4.4 GB
NLLB-200-Distilled	Dense	1.3B	∼8.6 GB
NLLB-200	Dense	1.3B	∼8.6 GB
NLLB-200	Dense	3.3B	∼20.6 GB
NLLB-200	MoE [9]	54.5B	∼404 GB

Table 7.3: Available versions of the NLLB-200 multilingual model. As the list progresses the output quality increases, however the same is true for the memory usage and time needed.

7.8 Code refactorings

The codebase could be enhanced by re-architecting it to follow the MVVM (Model-View-ViewModel) design pattern, utilizing the MVVM Community Toolkit [72]. This toolkit, widely used in the .NET ecosystem, supports .NET 8 and Native AOT as of late 2024 [9]. Implementing MVVM helps separate the application's core logic from its user interface, making the code more maintainable. Additionally, because the core functions are independent of the UI, this approach allows for easier transitions between different UI technologies, such as switching between desktop and mobile modes.

Researching and trying alternative methods to integrate the Bergamot Translator directly, without relying on a command shell, could also facilitate porting Tradutorium to additional platforms. While the performance gain from this approach may be minimal, it could simplify the process and make the application more versatile.

[8] Mixture of Experts model. Should achieve the same quality as its dense counterpart much faster during pretraining.

[9] Native AOT support is available since version 8.3: `https://devblogs.microsoft.com/dotnet/announcing-the-dotnet-community-toolkit-830/`

8. CONCLUSIONS

The development of this thesis combined practical and research interests, encompassing both application programming and model training.

The programming aspect (Tradutorium) operated at a higher level, and is indirectly linked to NMT. It “stood on the shoulders of giants” by reusing existing models and libraries developed by numerous contributors, all of whom are appropriately credited in the main text. The intended purpose was to utilize these available resources in order to create an end-user application that would be useful to users that are not necessarily involved with the academic community. Although the outcome is preliminary and requires further refinement to reach a professional standard, the functionality described (offline NMT, audio transcription, OCR, language identification, and integration with Wiktionary resources) has been successfully implemented and can serve as a foundation for future iterations without major obstacles.

One significant observation during the development was the need for automation to streamline the release of new versions, avoiding the manual and time-consuming setup of dependencies. Additionally, refactoring the codebase can strengthen the project’s structure, making future expansions more manageable. These are discussed in the “Future Directions” section and will be the first areas that I will focus on when further development proceeds.

The model training followed the instructions provided by the Bergamot project, primarily as an experiment to better understand the mechanics of creating an NMT language model. The goal was not to compete with larger, automated efforts that typically dedicate significant computing power and employ comprehensive corpora to achieve superior translation scores. As is evident by the empirical findings after the model creation, this turned out to be infeasible, due to limited resources. Instead, this project focused more on exploring and documenting the intricacies that were encountered and the results that the process achieved. The challenges and details that required specific attention contributed to a deeper understanding of the current NMT pipeline and research. These insights may prove beneficial to readers who wish to undertake similar endeavors.

From a personal perspective, the programming component of this project was more straightforward and easy to complete, due to prior academic and professional experience. In contrast, training an NMT model was “terra incognita”, requiring thorough research at many steps that would be considered straightforward but presented unanticipated barriers. These challenges had to be overcome before proceeding, although they sometimes turned into learning opportunities.

Throughout this journey, Dr. Tambouratzis provided invaluable assistance, offering detailed responses and explanations whenever necessary. His feedback was crucial; without it, this thesis would fall short in depth and fail to meet the required academic standards.

ABBREVATIONS - ACRONYMS

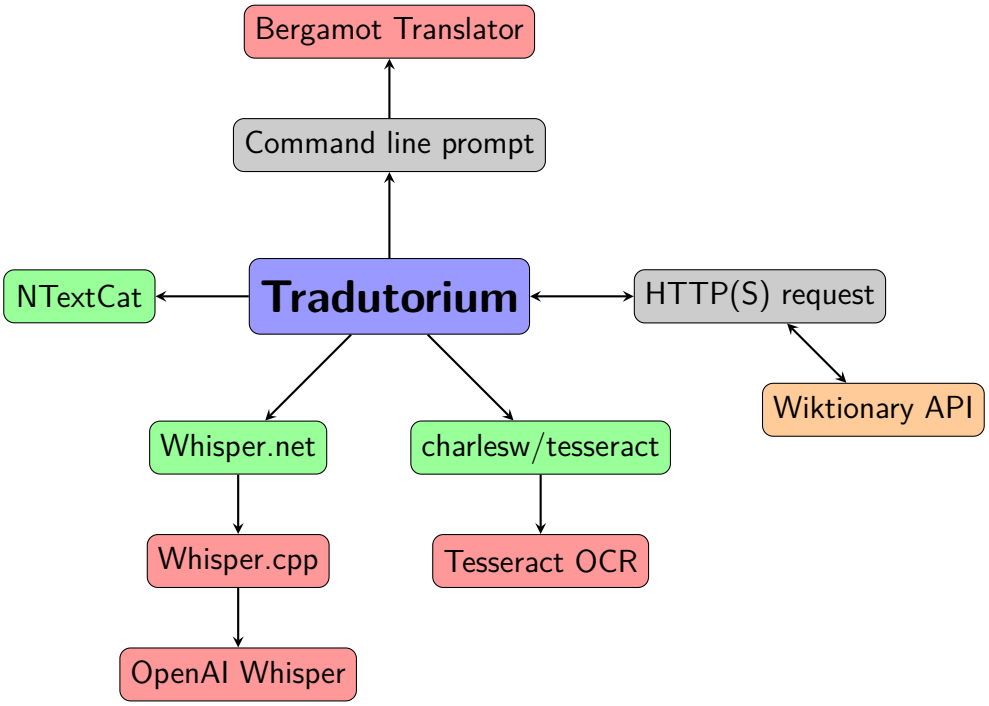
AI	Artificial Intelligence
ANN	Artificial Neural Network
ALPAC	Automatic Language Processing Advisory Committee
AOT	Ahead-of-Time
API	Application Programming Interface
ASR	Automatic Speech Recognition
BLAS	Basic Linear Algebra Subprograms
BLEU	BiLingual Evaluation Understudy
chrF	CHaRacter-level F-score
CIL	Common Intermediate Language
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CNN	Convolutional Neural Network
COMET	Crosslingual Optimized Metric for Evaluation of Translation
CPU	Central Processing Unit
CSV	Comma-Separated Values
EAP	Early Access Program
EBMT	Example-based machine translation
EoS	End-of-Sentence
FOSS	Free Open Source Software
GEMM	General Matrix Multiplication
GDPR	General Data Protection Regulation
GGML	Georgi Gerganov Machine Learning
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HIPAA	Health Insurance Portability and Accountability Act
HTML	HyperText Markup Language
ID	Identifier
IDE	Integrated Development Environment
Intel MKL	Intel Math Kernel Library
JIT	Just-in-Time
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LLM	Large Language Model
LSTM	Long Short-Term Memory
METEOR	Metric for Evaluation of Translation with Explicit ORdering
MNMT	Multilingual Neural Machine Translation
MoE	Mixture of Experts

MPL	Mozilla Public License
MVVM	Model-View-ViewModel
NLLB	No Language Left Behind
NMT	Neural Machine Translation
OCR	Optical Character Recognition
OpenAL	Open Audio Library
OpenTK	Open Toolkit
PCRE	Perl Compatible Regular Expressions
PTQ	Post-Training Quantization
RISC	Reduced Instruction Set Computer
RNN	Recurrent Neural Network
RTL	Right-to-Left
SaaS	Software as a Service
SMT	Statistical Machine Translation
UI	User Interface
UX	User Experience
VS	Visual Studio
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

APPENDIX I

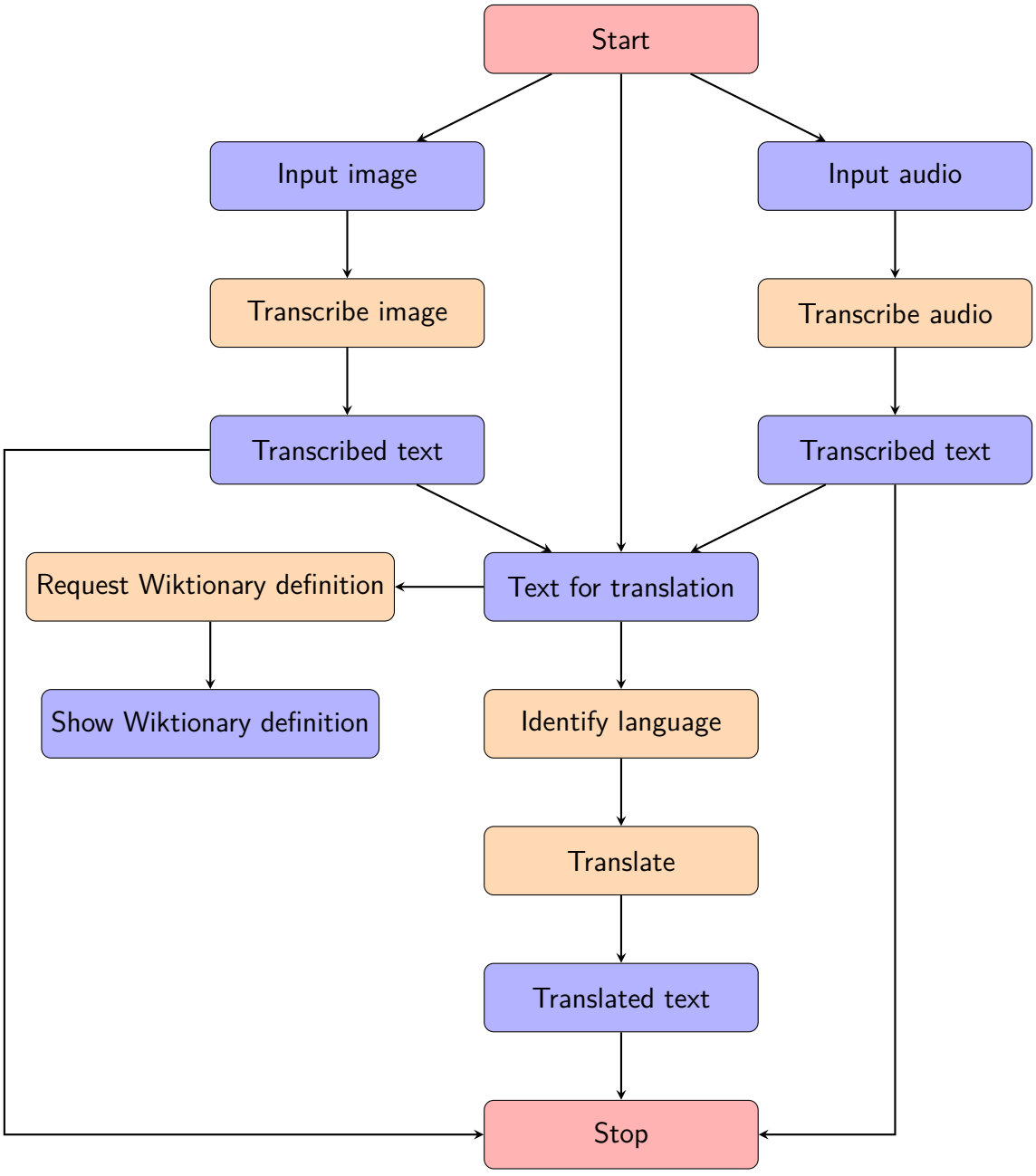
Visual representation of the libraries utilized to create Tradutorium:

- Tradutorium
- .NET library or wrapper *(managed code)*
- native library or executable *(unmanaged code)*
- system or network utility
- Internet resource



APPENDIX II

Tradutorium flow diagram:



APPENDIX III

Teacher model training command:

```
marian \
--model enel/transf_6layr_8hds_sed0_valid.log_model.npz \
--type transformer \
--train-sets corp_europarl+dgt.en corp_europarl+dgt.el \
--vocabs enel/vocab.enel.spm enel/vocab.enel.spm \
--sentencepiece-options \
--normalization_rule_name=nfkc \
--sentencepiece-alphas 0.2 0 \
--dim-vocabs 32000 32000 \
--disp-freq 100 \
--mini-batch-fit \
--workspace 7000 \
--layer-normalization \
--exponential-smoothing \
--dropout-rnn 0.2 \
--dropout-src 0.1 \
--dropout-trg 0.1 \
--valid-metrics cross-entropy \
--valid-sets europarl-v7_test.en europarl-v7_test.el \
--transformer-heads 8 \
--dim-emb 512 \
--valid-freq 10000 \
--beam-size 6 \
--normalize=0.6 \
--early-stopping 5 \
--cost-type=ce-mean-words \
--max-length 200 \
--save-freq 100000 \
--keep-best \
--log enel/transf_6layr_24hds_train_sed0.log \
--valid-log enel/transf_6layr_8hds_valid_sed0.log \
--enc-depth 6 \
--dec-depth 6 \
--learn-rate 0.0001 \
--lr-warmup 8000 \
--lr-decay-inv-sqrt 8000 \
--lr-report \
--seed 0 \
--label-smoothing 0.1
```

APPENDIX IV

Student model training command:

```
marian \
--model model.npz -c student.base.yml \
--train-sets corpus.{$SRC,$TRG}.gz -T ./tmp --shuffle-in-ram \
--guided-alignment corpus.aln.gz \
--vocabs vocab.spm vocab.spm --dim-vocabs 32000 32000 \
--max-length 200 \
--exponential-smoothing \
--mini-batch-fit -w 9000 --mini-batch 1000 --maxi-batch 1000 \
--devices $GPUS --sync-sgd --optimizer-delay 2 \
--learn-rate 0.0003 --lr-report --lr-warmup 16000 \
--lr-decay-inv-sqrt 32000 --cost-type ce-mean-words \
--optimizer-params 0.9 0.98 1e-09 --clip-norm 0 \
--valid-freq 5000 --save-freq 5000 --disp-freq 1000 --disp-first 10 \
--valid-metrics bleu-detok ce-mean-words \
--valid-sets devset.{\$SRC,\$TRG} --valid-translation-output devset.out \
--quiet-translation \
--valid-mini-batch 64 --beam-size 1 --normalize 1 \
--early-stopping 20 \
--overwrite --keep-best \
--log train.log --valid-log valid.log
```

REFERENCES

[1] Poibeau, Thierry (2017). *Machine translation*. MIT Press.

[2] Hutchins, John (2014). *The history of machine translation in a nutshell*. Unpublished *(https://web.archive.org/web/20200817105925/http://www.mt-archive.info/10/Hutchins-2014.pdf)*

[3] Rozentsveig, V. Yu. (1958). *The Work on Machine Translation in the Soviet Union* Fourth International Congress of Slavicists Reports. First Moscow State Pedagogical Institute of Foreign Languages, Moscow, USSR *(https://aclanthology.org/www.mt-archive.info/MT-1958-Rozentsveig.pdf)*

[4] Hutchins, John (1996). *ALPAC : The (in)famous report*. MT News International, no. 14, June 1996, pp. 9-12. *(https://aclanthology.org/www.mt-archive.info/90/MTNI-1996-Hutchins.pdf)*

[5] Hutchins, John (2006). *Machine translation: a concise history*. Journal of Translation Studies, vol.13, nos.1-2 (2010). Special issue: The teaching of computer-aided translation *(https://web.archive.org/web/20071021224238/http://www.hutchinsweb.me.uk/CUHK-2006.pdf)*

[6] Brown, Peter, et al. (1993). *The Mathematics of Statistical Machine Translation: Parameter Estimation*. Computational Linguistics, 19(2):263–311. *(https://aclanthology.org/J93-2003/)*

[7] Koehn, Philipp, et al. (2007). *Moses: Open source toolkit for statistical machine translation*. Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics. *(https://aclanthology.org/P07-2045/)*

[8] Duarte, Tiago, et al. (2014). *Speech Recognition for Voice-Based Machine Translation*. IEEE. 31. 26-31. 10.1109/MS.2014.14. *(https://www.researchgate.net/publication/260526163_Speech_Recognition_for_Voice-Based_Machine_Translation)*

[9] Papineni, Kishore, et al. (2002). *Bleu: a Method for Automatic Evaluation of Machine Translation.* Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics. *(https://aclanthology.org/P02-1040/)*

[10] Doddington, George (2002). *Automatic evaluation of machine translation quality using n-gram co-occurrence statistics.* HLT '02: Proceedings of the Second International Conference on Human Language Technology Research, pages 138–145. *(https://aclanthology.org/www.mt-archive.info/HLT-2002-Doddington.pdf)*

[11] Koehn, Philipp (2009). *Statistical machine translation*. Cambridge University Press.

[12] Way, Andy (2018). *Quality expectations of machine translation*. arXiv preprint arXiv:1803.08409. *(https://arxiv.org/abs/1803.08409)*

[13] Banerjee, Satanjeev and Lavie, Alon (2005). *METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments*. Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics. *(https://aclanthology.org/W05-0909/)*

[14] Popović, Maja (2015). *chrF: character n-gram F-score for automatic MT evaluation.* Proceedings of the Tenth Workshop on Statistical Machine Translation, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics. *(https://aclanthology.org/W15-3049/)*

[15] McCulloch, Warren Sturgis and Pitts, Walter (1943). *A Logical Calculus of the Ideas Immanent in Nervous Activity* Bulletin of Mathematical Biophysics Vol 5, pp 115–133.

[16] Rosenblatt, Frank (1957). *The Perceptron—a perceiving and recognizing automaton*. Report 85-460-1. Cornell Aeronautical Laboratory. *(https://websites.umass.edu/brain-wars/1957-the-birth-of-cognitive-science/the-perceptron-a-perceiving-and-recognizing-automaton/)*

[17] Minsky, Marvin, and Papert, Seymour (1988). *Perceptrons: An Introduction to Computational Geometry.* MIT Press. ISBN: 0-262-53477-0

[18] Rumelhart, David, Hinton, Geoffrey and Williams, Ronald (1986). *Learning representations by back-propagating errors*. Nature. 323 (6088): 533–536. *(https://www.cs.toronto.edu/~hinton/absps/naturebp.pdf)*

[19] Koehn, Philipp (2020). *Neural machine translation*. Cambridge University Press.

[20] Tan, Zhixing, et al. (2020). *Neural Machine Translation: A Review of Methods, Resources, and Tools*. arXiv preprint arXiv:2012.15515. *(https://arxiv.org/abs/2012.15515)*

[21] Virtapuro, Miko, et al. (2023). *Overview on Machine Translation Services*. Nordic Council of Ministers *(https://www.norden.org/en/publication/overview-machine-translation-services)*

[22] Kalchbrenner, Nal and Blunsom, Phil (2013). *Recurrent Continuous Translation Models.* Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1700–1709, Seattle, Washington, USA. Association for Computational Linguistics. *(https://aclanthology.org/D13-1176/)*

[23] Sutskever, Ilya, et al. (2014). *Sequence to sequence learning with neural networks.* Proceedings of the NIPS Conference, page 3104–3112. NIPS. *(https://papers.nips.*

cc/paper_files/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html)

[24] Bahdanau, Dzmitry, et al. (2014). *Neural machine translation by jointly learning to align and translate*. arXiv preprint arXiv:1409.0473. *(https://arxiv.org/abs/1409.0473)*

[25] Vaswani, Ashish, et al. (2017). *Attention is all you need.* Advances in Neural Information Processing Systems, volume 30, pages 5998–6008. *(https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html)*

[26] Alammar, Jay (2018). (Retrieved 27/10/2024) *The illustrated transformer.* Jay Alammar–Visualizing Machine Learning One Concept at a Time. *https://jalammar.github.io/illustrated-transformer/*

[27] Zhaoyang, Niu, et al. (2021). *A review on the attention mechanism of deep learning.* Neurocomputing 452, 48-62.

[28] Levine, Sergey (2021). (Retrieved 14/09/2024). *Designing, Visualizing and Understanding Deep Neural Networks*. CS W182 / 282A at UC Berkeley. *(https://cs182sp21.github.io/static/discussions/dis7-sol.pdf)*

[29] Rei, Ricardo, et al. (2020). *COMET: A Neural Framework for MT Evaluation.* Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 2685–2702, Online. Association for Computational Linguistics. *(https://aclanthology.org/2020.emnlp-main.213/)*

[30] Zhao, Wayne Xin, et al. (2023). *A Survey of Large Language Models*. arXiv preprint arXiv:2303.18223. *(https://arxiv.org/abs/2303.18223)*

[31] Johnson, Melvin, et al. (2017). *Google's multilingual neural machine translation system: Enabling zero-shot translation.* Transactions of the Association for Computational Linguistics 5: 339-351 *(https://aclanthology.org/Q17-1024/)*

[32] Hendy, Amr, et al. (2023). *How Good Are GPT Models at Machine Translation? A Comprehensive Evaluation*. arXiv preprint arXiv:2302.09210. *(https://arxiv.org/abs/2302.09210)*

[33] Kocmi, Tom, et al. (2023). *Findings of the 2023 Conference on Machine Translation (WMT23): LLMs Are Here but Not Quite There Yet.* Proceedings of the Eighth Conference on Machine Translation, pages 1–42, Singapore. Association for Computational Linguistics. *(https://aclanthology.org/2023.wmt-1.1/)*

[34] Poibeau, Thierry (2022). *On "Human Parity" and "Super Human Performance" in Machine Translation Evaluation*. Proceedings of the Thirteenth Language Resources and Evaluation Conference, pages 6018–6023, Marseille, France. European Language Resources Association. *(https://aclanthology.org/2022.lrec-1.647/)*

[35] Koehn, Philipp and Knowles, Rebecca (2017). *Six Challenges for Neural Machine Translation*. arXiv:1706.03872. *(https://arxiv.org/abs/1706.03872)*

[36] Haddow, Barry, et al. (2022). *Survey of Low-Resource Machine Translation*. arXiv preprint arXiv:2109.00486. *(https://arxiv.org/abs/2109.00486)*

[37] Guerreiro, Nuno M., et al. (2023). *Looking for a Needle in a Haystack: A Comprehensive Study of Hallucinations in Neural Machine Translation*. Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, pages 1059–1075, Dubrovnik, Croatia. Association for Computational Linguistics. *(https://aclanthology.org/2023.eacl-main.75/)*

[38] Shumailov, Ilia, et al. (2023). *The Curse of Recursion: Training on Generated Data Makes Models Forget*. arXiv preprint arXiv:2305.17493. *(https://arxiv.org/abs/2305.17493)*

[39] Kaddour, Jean, et al. (2023). *Challenges and Applications of Large Language Models*. arXiv preprint arXiv:2307.10169. *(https://arxiv.org/abs/2307.10169)*

[40] Bogoychev, Nikolay, et al. (2021). *TranslateLocally: Blazing-fast translation running on the local CPU*. arXiv preprint arXiv:2109.10194 *(https://arxiv.org/abs/2109.10194)*

[41] .NET Team (2023). (Retrieved 19/08/2023). What is .NET, and why should you choose it? *(https://devblogs.microsoft.com/dotnet/why-dotnet/)*

[42] Microsoft Learn (2023). (Retrieved 19/08/2023). Common Language Runtime (CLR) overview *(https://learn.microsoft.com/en-us/dotnet/standard/clr)*

[43] Singer, Jeremy (2003). *JVM versus CLR: a comparative study*. PPPJ (Vol. 3, pp. 167-169). *(https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b57aaf581e043fb63c56ebd662720190e3121220)*

[44] Polyak, Nick (2023). (Retrieved 14/10/2023). Multiplatform XAML/C# Miracle Package: Avalonia. Comparing Avalonia to WinUI based Solutions. CodeProject. *(https://www.codeproject.com/Articles/5366945/Multiplatform-XAML-Csharp-Miracle-Package-Avalonia)*

[45] robloo (2024). (Retrieved 14/08/2024). *XAML Framework Comparison*. Github. *(https://github.com/robloo/PublicDocs/blob/master/XAMLFrameworkComparison.md)*

[46] Spolsky, Joel (2002). (Retrieved 09/09/2023). Our .NET Strategy *(https://www.joelonsoftware.com/2002/04/11/our-net-strategy/)*

[47] Microsoft Learn (2023). (Retrieved 19/08/2023). ReadyToRun Compilation *(https://learn.microsoft.com/en-us/dotnet/core/deploying/ready-to-run)*

[48] Microsoft Learn (2023). (Retrieved 19/08/2023). Native AOT deployment *(https://learn.microsoft.com/en-us/dotnet/core/deploying/native-aot/)*

[49] NDepend (2024). (Retrieved 14/08/2024). *.NET Native AOT Explained*. NDepend official blog. *(https://blog.ndepend.com/net-native-aot-explained/)*

[50] Avalonia UI (2023). (Retrieved 14/10/2023). *Welcome to the New Era of App Development: Introducing Avalonia v11*. Avalonia UI official blog. *(https://avaloniaui.net/Blog/welcome-to-the-new-era-of-app-development-introducing-avalonia-v11,a8907121-eae9-4a16-aca8-2432e1dac13a)*

[51] Tsarpalis, Eirik (2023). (Retrieved 07/10/2023) What's new in System.Text.Json in .NET 8 *(https://devblogs.microsoft.com/dotnet/system-text-json-in-dotnet-8/)*

[52] Cavnar, William B., and John M. Trenkle. (1994). *N-gram-based text categorization.* Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval. Vol. 161175. *(https://www.let.rug.nl/vannoord/TextCat/textcat.pdf)*

[53] Heafield, Kenneth, et al. (2019). *Deliverable 5.1: Translation software with initial CPU optimizations*. Horizon 2020 Research and Innovation Action Grant Agreement No. 825303 *(https://browser.mt/assets/D5.1.pdf)*

[54] Karpagavalli, S., and Edy Chandra (2016). *A review on automatic speech recognition architecture and approaches.* International Journal of Signal Processing, Image Processing and Pattern Recognition 9.4: 393-404. *(https://web.archive.org/web/20160804142314/http://www.sersc.org/journals/IJSIP/vol9_no4/34.pdf)*

[55] Radford, Alec, et al. (2022). *Robust Speech Recognition via Large-Scale Weak Supervision*. OpenAI. *(https://cdn.openai.com/papers/whisper.pdf)*

[56] Islam, Noman (2017). *A survey on optical character recognition system*. arXiv preprint arXiv:1710.05703. *(https://arxiv.org/abs/1710.05703)*

[57] Smith, Ray (2007). *An overview of the Tesseract OCR engine.* Ninth international conference on document analysis and recognition (ICDAR 2007). Vol. 2. IEEE. *(https://research.google/pubs/pub33418/)*

[58] Hegghammer, Thomas (2022). *OCR with Tesseract, Amazon Textract, and Google Document AI: a benchmarking experiment.* Journal of Computational Social Science 5.1: 861-882. *(https://link.springer.com/article/10.1007/s42001-021-00149-1)*

[59] Kahle, Brewster (2020). (Retrieved 16/09/2023). *FOSS wins again: Free and Open Source Communities comes through on 19th Century Newspapers (and Books and Periodicals...)* Internet Archive Blogs. *(https://blog.archive.org/2020/11/23/foss-wins-again-free-and-open-source-communities-comes-through-on-19th-century-newspapers-and-books-and-periodicals/)*

[60] Microsoft Learn (2023). (Retrieved 14/10/2023). Localization in .NET *(https://learn.microsoft.com/en-us/dotnet/core/extensions/localization)*

[61] Bergamot project contributors (2023). (Retrieved 09/09/2023). Training speed-optimized student NMT models *(https://github.com/browsermt/students/blob/819f41962c8867859f8e511dcba9b63fc997db82/train-student/README.md)*

[62] Bogoychev, Nikolay (2021). (Retrieved 29/09/2024) Efficient machine translation *(https://nbogoychev.com/efficient-machine-translation/)*

[63] Junczys-Dowmunt, Marcin, et al. (2018). *Marian: Fast neural machine translation in C++.* arXiv preprint arXiv:1804.00344 *(https://arxiv.org/abs/1804.00344)*

[64] Koehn, Philipp (2005). *Europarl: A Parallel Corpus for Statistical Machine Translation*. Proceedings of Machine Translation Summit X: Papers, pages 79–86, Phuket, Thailand. *(https://aclanthology.org/2005.mtsummit-papers.11/)*

[65] Steinberger, Ralf, et al. (2012). *DGT-TM: A freely Available Translation Memory in 22 Languages*. Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12), pages 454–459, Istanbul, Turkey. European Language Resources Association (ELRA). *(https://aclanthology.org/L12-1481/)*

[66] Tiedemann, Jörg and Thottingal, Santhosh (2020). *OPUS-MT – Building open translation services for the World*. Proceedings of the 22nd Annual Conference of the European Association for Machine Translation, pages 479–480, Lisboa, Portugal. European Association for Machine Translation. *(https://aclanthology.org/2020.eamt-1.61/)*

[67] NLLB Team, et al. (2022). *No Language Left Behind: Scaling Human-Centered Machine Translation*. arXiv preprint arXiv:2207.04672. *(https://arxiv.org/abs/2207.04672)*

[68] Kim, Young Jin, et al. (2019). *From research to production and back: Ludicrously fast neural machine translation.* Proceedings of the 3rd Workshop on Neural Generation and Translation *(https://aclanthology.org/D19-5632/)*

[69] Nagel, Markus, et al. (2021). *A white paper on neural network quantization.* arXiv preprint arXiv:2106.08295 *(https://arxiv.org/abs/2106.08295)*

[70] Gholami, Amir, et al. (2021). *A survey of quantization methods for efficient neural network inference* arXiv preprint arXiv:2103.13630. *(https://arxiv.org/abs/2103.13630)*

[71] Shi, Xing, et al. (2020). *Why neural machine translation prefers empty outputs*. arXiv preprint arXiv:2012.13454 *(https://arxiv.org/abs/2012.13454)*

[72] Microsoft Learn (2023). (Retrieved 19/08/2023). Introduction to the MVVM Toolkit - .NET Community Toolkit *(https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/)*