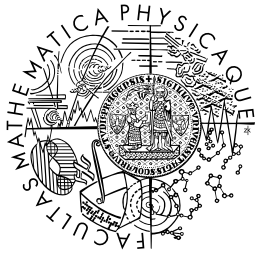

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Václav Ryšlink

Methods of Input Segmentation for Simultaneous Speech Translation

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: doc. RNDr. Ondřej Bojar, Ph.D.
Consultant: Mgr. Aleš Tamchyna, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2022

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Na vzniku této práce se přímo podílelo mnoho lidí, kterým bych rád touto cestou srdečně poděkoval. Mým oběma školitelům, pánům Ondřeji Bojarovi a Aleši Tamchynovi, za opravdu nespočet setkání a diskuzí, za komentáře k finálnímu textu a obecně za skvělý přístup a vedení. Dominikovi Macháčkovi za vytvoření ESIC datatasetu a rozuzlení několika témat. Paní Věře Klaudové za poskytnutí lingvistického pohledu na věc a za pomoc při organizaci anotace. A nakonec i všem anotátorům, bez jejichž pomoci by v této podobě tato práce nemohla vzniknout.

Kromě výše zmíněných mělo na vznik této práce také, ač nepřímý, tak velmi podstatný vliv i moje nejbližší okolí. Kamarádi ze školy a všelijakých jiných koutů, nebo kolegové z práce, kteří se mnou měli po celou dobu trpělivost.

Ze všeho nejvíc bych chtěl nakonec ale vyslovit to největší možné díky svým nejbližším – své rodině, která mě až do tohoto okamžiku dokázala dovést, a to od úplného začátku.

Title: Methods of Input Segmentation for Simultaneous Speech Translation

Author: Václav Ryšlink

Institute: Institute of Formal and Applied Linguistics

Supervisor: doc. RNDr. Ondřej Bojar, Ph.D., Institute of Formal and Applied Linguistics

Consultant: Mgr. Aleš Tamchyna, Ph.D.

Abstract: Segmentation methods are an essential part of the simultaneous machine translation process because, in the ideal case, they split the input into chunks whose translation is independent of any forthcoming context. Furthermore, the optimal splitting should also ensure that the segments with the previous characterization have minimal lengths. However, there is still no agreement about the rules that should produce such an optimal splitting. Therefore, we started with the annotation of the ESIC dataset by simulating a perfect human interpreter with an infinite amount of time and resources. Then we proposed multiple segmentation methods that we compared to each other in terms of segments' lengths, counts, and statistics of the most frequently split types of words. Apart from the segmentation methods, we also implemented and analyzed two variants of neural machine translation models – one trained solely on complete sentences and the other finetuned with partial translations. Finally, we evaluated the translation quality and delay of segments produced by splitting methods with the SLTev evaluation toolkit and discussed the effect of both machine translation models on the results.

Keywords: NLP, Simultaneous machine translation, Segmentation methods

Contents

Introduction

During simultaneous interpretation, interpreters face several challenges. At the same time, they need to pay attention to the speaker, think about the best way to express the translation and translate. Therefore, it is no wonder that they can reach their cognitive capacities during the process and, as a result, stop interpreting optimally.

On the other hand, if we consider various parallel computing techniques, machines should not suffer from bounded processing capabilities, and thus we should be able to find a way to teach them to interpret with high quality while keeping the delay as short as possible.

One of the most crucial aspects of simultaneous machine translation (SiMT) is to figure out the optimal way to segment the input. For simplicity, we assume that during the process of SiMT, a more or less standard translation model is going to be applied repeatedly to the growing input. The points in time when this translation happens will then correspond to a certain splitting of the input.

To this day, however, there is no agreement about what rules produce the most optimal splitting – i.e., producing segments of minimal lengths whose translations are independent of any forthcoming context. That is the reason why our thesis focused on the characterization and comparison of several segmentation strategies and the evaluation of their performance in the real interpretation setting. Furthermore, we also developed a strategy for producing optimal splitting by simulating a perfect interpreter with unlimited time and resources, which we then used to annotate the ESIC dataset.

Firstly, in Chapter 1, we surveyed available linguistic literature about interpretation and translation units that are very closely connected to the topic of optimal segmentation. Then, we continued by explaining key concepts for the process of SiMT, mainly covering the evaluation methodology and several segmentation strategies that have been applied in recent research (see Chapter 2).

To find out how a perfect human interpreter without time restrictions would interpret, we annotated the ESIC dataset with optimal splits and even provided Czech reference translations (see Figure 1). We described the annotation instructions and annotators' observations in Chapter 3.

In Chapter 4, we introduced several splitting methods with the description of their algorithms and ways how to utilize them both in offline and online settings. We consequently evaluated all mentioned splitting methods with the SLTev evaluation toolkit, measuring the delay and translation stability of the segments.

We also trained a neural machine translation model and tested whether we could improve the interpretation by incorporating partial translations into the training data. The results of the evaluation and the analysis of splitting methods behavior can be found in Chapter 5

Finally, in Chapter 6, we summarized all our observations from our literature survey, dataset annotation, and experiments while suggesting potential improvements and additional topics for future research.

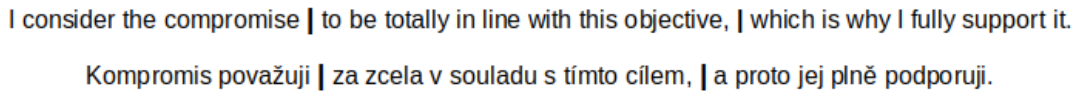

Figure 1: Example of an English sentence segmentation with the Czech translation used as a reference.

1. Linguistic theory

Before diving deeper into more technical topics, we would like to start by summarizing materials about interpretation from the linguistic perspective because it will help us to reveal important concepts for developing new segmenting algorithms.

1.1 Simultaneous interpretation

The discipline of simultaneous interpretation is highly complex and exposes interpreters to a cognitive load that can easily overwhelm human capabilities. However, before explaining what exact factors make the interpretation that difficult, we would like to first start by emphasizing the difference between the acts of translation and interpretation.

1.1.1 Comparing interpretation with translation

From an engineering point of view, the interpretation might seem like an imperfect oral translation. However, the difference is more significant in the eyes of linguistic theory. By definition, simultaneous speech interpretation is an oral conversion of a message from the source language to the target language [Barik, 1969]. Therefore, let us dissect what converting a message means.

Omitting the bidirectional influence of thoughts and language [Zlatev and Blomberg, 2015], we can use the theory of the Vauquois triangle for our explanation (see Figure 1.1). According to this theory, a pair of sentences in two different languages, stemming from the common interlingua, can be translated on three levels – on the semantic and syntactic level, or directly on words.

During standard written translation, the transfer takes place on all mentioned levels because by carefully translating word-by-word or phrase-by-phrase, we are preserving the general meaning of the sentences (semantic level), but on top of that, we also pay attention to syntactic and grammar rules to keep the target sentence as close to the source as possible. We can afford such precision because we are not pressured by any time limitations and can always work with complete information – i.e., we have the complete source text at our disposal.

Looking at simultaneous interpretation, the process and the settings are entirely different:

- Interpretation is conducted solely via speech.
- Interpreters do not have unlimited time and need to efficiently manage their limited cognitive resources for the task.
- Apart from translating comprehensibly, interpreters must interpret as frequently as possible to keep the audience engaged.

Due to the previous points, it should be clear that it is not within the human capabilities to interpret word-by-word. For those reasons, interpreters instead focus on conveying the correct meaning while figuring out the best way how to rephrase it.

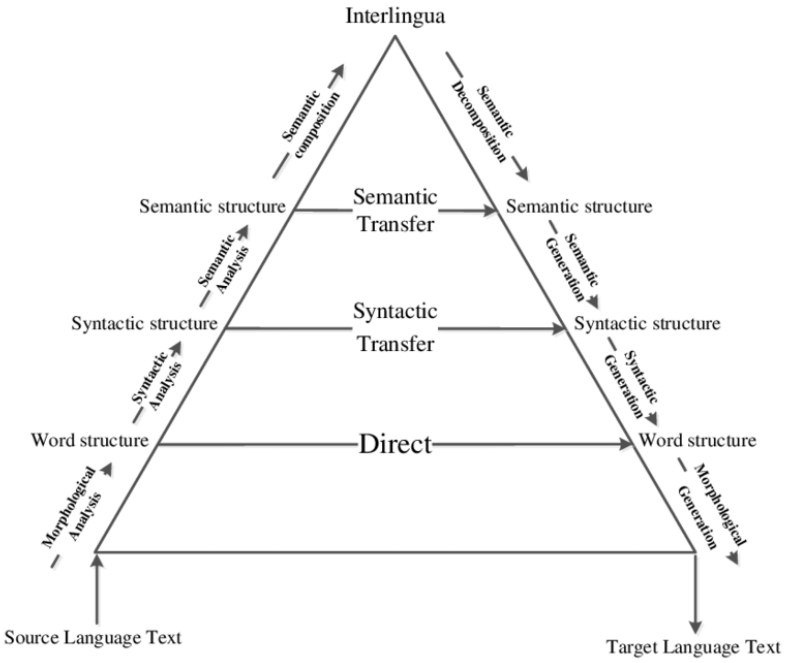

Figure 1.1: Visualization of Vauquois triangle denoting different levels of possible transfers [Albadr et al., 2018].

1.1.2 Interpretation effort model

Only a few theoretical models for simultaneous interpretation (SI) incorporate the cognitive load into their formulae – one of them is the *Effort Model of SI* by Daniel Gile [1992]. He built the model upon a principle of mental processing capabilities required for SI. He defines the task of SI with the following equation as:

$$SI = L + M + P + C \tag{1.1}$$

where variables on the right side of the equation represent different non-automatic tasks/efforts that interpreters encounter. Listening effort (L) is the process of listening to a speaker. Memory effort (M) is the short-term memory requirement to temporarily save information. P stands for the production phase – the effort to produce the interpretation; and coordination effort (C) covers the management and focus needed for orchestrating listening, memorizing, processing, and speaking.

We can similarly represent the effort's cognitive requirements by adding (R) to their abbreviations. Therefore, to interpret smoothly and successfully, we need the interpreter's total available capacity (TA) to be greater or equal to the sum of the requirements throughout the entire process.

$$TA >= LR + MR + PR + CR \tag{1.2}$$

To increase the available capacity, interpreters learn different techniques and tactics. Apart from using convenient interpretation techniques, here are the aspects that may significantly affect the interpreters' performance as well:

- Interpreters' knowledge of the discussed subject
- Speakers' dialect and speaking skills
- Noise in the room
- Interpreters' mental state

We would also like to emphasize that while a human interpretation is usually conducted via the speech-to-speech manner. Machine interpretation is more frequently used to describe the process of converting speech to text – it is typical to produce the output in the form of subtitles for conferences or lectures.

Thus, when mentioning machine interpretation, it is better to imagine a speech-to-text process if not explicitly mentioned otherwise. The extension to speech-to-speech is only a matter of a speech synthesis module [Baumann and Schlangen, 2012] that does not directly affect the quality and correctness of the interpretation.[1]

1.2 Translation units

Let us introduce this section with a thought experiment. If we wanted to create a simple translation system, we could download the translations of every word into a machine's memory. Then, we would let the machine process sentences word-by-word, locate the words in their internal dictionary, and replace them with corresponding translations. Such an approach can work to some extent for similar languages and simple sentences. However, when translating some non-trivial sentences, we can imagine the result would be most likely not very comprehensible, and even native speakers could struggle to understand it.

If we enhance our algorithm by not translating single words but pairs of words (2-grams) instead, we would probably not get a perfect translation, but the translation should have more clarity than in the previous case because two words provide an additional context that can make the translation more accurate.

We can repeat this process with 3-grams, 4-grams, ..., n-grams until n is equal to the longest sentence in our dataset, and we translate all source sentences as a whole. That way, we will always have complete information and should be able to return a perfect translation every time. So in other words, to get the best quality translation, we need to have the information about the writer's/speaker's intentions that we can only get by having similar common knowledge and a good notion of how the text/speech will continue in the nearest future.

With this idea of translating sentences by a fixed number of words, we fluently moved toward the main topic of this thesis – the *translation units* (TUs). Unfortunately, there is no single definition or description of what TUs are. Instead, there are several ways how one can grasp the topic – namely, we will look at the cognitive view, lexical view, and the view of machine translation [Thunes, 2017].

[1] Speech-to-speech approach may be still more beneficial for some application because we can preserve the original prosody and not limit ourselves only to languages with written data.

1.2.1 Cognitive translation units

Firstly, we will look at TUs as *cognitive units* during the process of the translators' activity. With that respect, TUs are viewed as units of analysis and represent chunks of sentences that translators are paying attention to.

The typical studies are conducted using Think-Aloud-Protocols (TAP), in which translators are asked to report everything that goes through their minds. In this TAP study, Malmkjær [1998] report how much translators' experience affects TUs' lengths. They discovered that language learners have TUs much shorter (primarily single words) than experience translators who perceive the text in chunks of phrases, clauses, or sentences. However, the translators themselves are almost unaware of it.

1.2.2 Lexical translation units

The second branch of TUs, called *lexical translation units*, is regarded as product-oriented. Researchers in this area are interested in translation units from the lexical point of view and do not focus on the cognitive aspect. The typical topics of interest here are the relationship between the original text and its translation, the characteristics of several different translations, and how correlated the translations are within different target languages.

However, even within this field, there is no one standard definition. Moreover, there is no agreement upon whether the translation units are within the source or target sentences. One of the earliest mentions of the term lexical TU came from Barkhudarov [1969], who defined TU as "the smallest unit of source text which has an equivalent in target text". Shuttleworth and Cowie [1997] further developed this definition as "a term used to refer to the linguistic level at which source text is recodified in the target language". Finally, Vinay et al. [1995] characterized TUs similarly as lexicological units of words that should not be translated individually.

On the other hand, for example, Malmkjær [1998] thinks about TUs as the target text parts that can be mapped onto a source text. And furthermore, some definitions refer to TUs even as a pair of source and target text segments ([Danielsson, 2001], [Malmkjaer, 2006]).

From this perspective, it is clear that there is no single correct definition of lexical TUs, but all of them are concerned with the relationship between source and target texts which is essential for TUs allocation. The observation about the importance of the source and target sentence relationship was even enhanced by Koller [1992] arguing that the TUs' size is inversely correlated with the similarity of the source and target language – i.e., the more related the languages are, the smaller TUs we should be able to make. For a very simple example, if we compare English and German, the most visible difference in sentences is that German puts verbs at the end of sentences while English tends to place them at the beginning. Because of this difference, when translating German to English, we may need to wait for the verb at the end of a German sentence to be able to start translating to English.

When referring to lexical TUs, we can also distinguish their several subtypes – depending on the strength of bonds among the words inside them. We have three different kinds in mind:

- Idioms – usually fixed expressions that is common to translate together.
- Known phrases that are bonded with weaker bonds.
- And units that are held together only because of the need to wait for additional information (e.g., when we are not sure whether we speak about her or him) or because the grammar constructs of the target language require us to wait for a particular sentence element (e.g. mentioned German to English example).

Minimal translation units

To avoid confusion, we should also introduce the term *minimal translation units* (MTUs) which can be used interchangeably with standard TUs.

Generally, when speaking about TUs, people mean the shortest possible units that are reasonable to translate. But sometimes, even a composition of several TUs can be regarded as a TU – especially when mixing up definitions of cognitive and lexical units. Therefore, by using the term MTUs, we want to signify that we mean the lexical TUs of the shortest possible length.[2]

As for the appropriate length of MTUs, there are many discussions. In this thesis, we take the view of Bennett [1994] and believe that the length of MTUs can range from individual words to much larger text chunks. Firstly, because otherwise, the task would be trivial, and secondly, the professional interpreters are able to keep pace with the speaker very precisely – even within the same sentences. Thus, the machines should be able to achieve it as well. However, some authors argue that the most appropriate translation units are clauses [Bell and Candlin, 1991], clause complexes [Barkhudarov, 1969], or sentences [Zhu, 1999].

When comparing the lengths of cognitive and lexical TUs, researchers agree that lexical TUs tend to be shorter than their cognitive counterparts [Bennett, 1994]. According to some authors, cognitive TUs may even span over whole paragraphs [Bennett, 1994], whereas lexical TUs are usually considered within sentences.

Factors influencing lengths of MTUs

As we discussed previously, one aspect that can noticeably affect the size of MTUs is the choice of source and target languages. If we take a pair of similar languages (e.g., Czech and Slovak), we will get shorter MTUs because the languages share similar sentence structures and expressions.

Because of that, one may expect the correlation that the more historically-related languages are, the shorter MTUs they will have. Nonetheless, although the previous rule holds for some cases, it does not apply to all language pairs. We can observe that, for example, with two closely related languages from the germanic language family – English and German. Despite their similar historical development, the differences in their sentence structure are causing comparably more extended TUs [Kiraly, 1992].

[2]Process-oriented linguists looking at TUs from a cognitive perspective do not use the term MTUs at all.

Another two important factors contributing to different MTUs lengths are then the experience of the translator (or interpreter) and the nature of the text. Translators with less experience reportedly translate in shorter units than their more professional colleagues [Moser-Mercer, 1997]. Similarly, more complex technical articles will have more prolonged units than news articles because of different intentions and expectations from the target audience [Lambert and Moser-Mercer, 1994].

On top of that, the interpretation can also be affected by the speech rate of the source speakers (e.g., heated debates will have a faster pace than monologue presentations). It was proven by Pio [2003], that the speed of the source speech notably affects the interpretation – especially for novices who tend to perceive it as an additional source of stress. But it can also be the opposite case. As Schlesinger [2003] reported, a higher presentation rate can reduce the strain on the professional interpreter's working memory and enhance the quality. Nevertheless, it is important to emphasize that it may apply only to professional interpreters who use different interpreting strategies and retain larger amounts of information in accessible form [Schlesinger, 2003].

1.2.3 Translation units in machine translation

Finally, the last view on TUs is from the perspective of natural language processing (NLP). Compared to the previous definitions, where the attention was usually on the source or target texts, here we consider both of them equally [Bowker, 2002].

However, we need to be aware of the fact that the topic of TUs in NLP was popular mainly because of the older rule-based MT systems, which compared to more recent statistical (SMT) and neural (NMT) machine translation systems, relied more on the linguistic theory and suffered from the problem of lexical coverage.

The closest analogy to TUs in modern statistical approaches would be the phrases in phrase-based MT systems [Koehn et al., 2003] and probably also the concept of n-grams – word sequences for which systems learn the probabilities of their occurrences. But because no linguistic analysis is involved within the n-grams, they should be rather regarded as units of processing than TUs.

For simultaneous translation, however, TU's role regains importance by enabling us to split sentences into smaller chunks that can be translated even before a speaker finishes a sentence. Therefore, in this thesis, we look at TUs as minimal translation units that we try to locate in the source texts to produce the most optimal interpretation – as short as possible but preserving a good understandability.

2. Simultaneous machine translation

In this chapter, we will continue explaining concepts and terms connected with MTUs and sentence segmentation; but with more focus on the perspective of NLP.

2.1 Machine speech translation

The first attempts to construct speech translation (ST) pipelines date to the end of the 20th century and were naturally accompanied and enabled by advances in automatic speech recognition (ASR) and machine translation (MT) research.

We will regard ST as the process of translating the source speech signal into the text of the target language. This task differs from speech-to-speech translation, where the translation is delivered via automatically synthesized speech.

2.2 Simultaneous machine translation

Simultaneous machine translation (SiMT) was initially seen as a complex task consisting of several subtasks mimicking the process of human interpreters. However, machines should have the advantage of larger memory, more computational resources, and more extended attention spans than humans. While humans need to perceive the original speech, think about the speaker's intent, and speak simultaneously, machines can have a dedicated processing unit for each task and so have the cognitive load evenly spread out. For those reasons, people may often use the term simultaneous machine translation interchangeably with simultaneous machine interpretation.

Generally speaking, the pipeline of a SiMT model consists of three main parts: ASR, segmentation and MT module. In the beginning, the sound signal enters the ASR module that transforms the sound waves into a text which is sent into the segmentation module. The segmentation module then processes the text, decides at which point the text should be split, and lets the MT system translate it into a target language.

Compared to speech translation, which is evaluated solely on the quality of produced translation, SiMT has one additional requirement. SiMT needs to translate with high quality but also with as little latency as possible so the users can track the events in real-time.

The demand for a short delay, however, causes that SiMT models no longer work with complete information because they need to be able to translate even incomplete sentences without knowing how they will finish.[1] This aspect makes out of the SiMT a multi-objective optimization problem.

[1]The uncertainty sometimes occurs even when working with complete sentences because of extra-sentential phenomena, such as coreference. In our thesis, however, we will work with an assumption that sentences provide full information about their content.

2.3 Evaluation of SiMT

As we already stated, the two most important metrics for SiMT evaluation are the translation quality and latency – we maximize the translation quality and minimize the latency. Nevertheless, optimization of both metrics at the same time is tricky. The reason is that in order to increase the translation quality, the system generally needs more information about the speaker's intent which means waiting for more context and thus increasing the latency.

To clarify the evaluation, we will explain methods for measuring translation quality and latency, outline two additional metrics influencing SiMT quality, and describe a toolkit we will use to evaluate our experiments.

2.3.1 Translation quality

The closer the machine translation is to the professional human translation, the better score it should get. That is the main idea behind measuring the translation quality. Humans are generally very capable of assessing the translation quality, but manual evaluations tend to be slow, expensive, and for some purposes, can even be regarded as subjective. Therefore, to assess the improvement of the machine translation models, researchers needed a standard metric that would reliably measure the quality of produced translations automatically and objectively for all inputs.

Nowadays, the translation quality metric of choice is the BLEU score [Papineni et al., 2002]. The most important reasons for adopting the BLEU score as a dominant MT metric were its low computational demands, easy adoption for different languages, and repeatedly reported agreement with human assessment.

The *bilingual evaluation understudy* (BLEU) compares the candidate's translation to ground truth references and consists of two main components – brevity and n-gram overlap. The first component penalizes translation candidates for short sentence lengths, and the latter checks for common $1, 2, ..., n$-grams between candidate translation and references (usually $n = 4$).

When evaluating and comparing BLEU with other results, it is important to pay attention to the BLEU parametrization and tokenization strategies, which may significantly affect the resulting values. Therefore, it is recommended to use an improved sacreBLEU score [Post, 2018] that normalizes the score by tokenizing raw inputs and own equation parameters.

On the contrary, even BLEU score is not an ideal metric for all. Especially with machines slowly approaching the human-level quality translations, researchers noticed that such a simplistic metric might be insufficient and even lead to adopting worse systems [Ma et al., 2019b]. Therefore, it has also been suggested to use other, more complex metrics such as BERTScore [Zhang et al., 2019], BLEURT [Sellam et al., 2020] or COMET [Rei et al., 2020] that are using pre-trained multilingual models to improve the evaluation.

Because in our experiments, we did not exactly use translation quality as the key metric to evaluate quality of the segments (see Section 2.3.3), we will not dive into discussing the principles behind other machine translation quality metrics. However, we believe that their adoption in the future will be another important step towards improving the performance of SiMT models.

2.3.2 Latency

Although high translation quality is considered as a key metric among all translation tasks, latency may affect the perception of interpretation to the same or even bigger extent.

As one of the first attempts to quantify latency, Cho and Esipova [2016] proposed a metric called *Average Proportion* (AP). AP measures the average number of source text tokens processed when generating partial target prediction. Nevertheless, while the metric expresses latency simply and understandably, it is not length-invariant. This means its value changes depending on the length of the source text, even with the preserved static segmenting strategy, which should score the same. Furthermore, it is not evenly distributed on the $[0, 1]$ interval since the optimal 0-wait policy generating target words right after the corresponding source words were spoken would gain a score close to 0.5 and not below that.

A general equation for measuring latency between a source sentence x and target sentence y has the following form:

$$L(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{Z(\boldsymbol{x}, \boldsymbol{y})} \sum_i C_i(\boldsymbol{x}, \boldsymbol{y}) \tag{2.1}$$

where Z is a normalization constant and C_i a cost function for each target position i. For AP, $C_i(x, y)$ directly correspond to $g(i)$ representing the number of source tokens read when generating a target word i and $Z(x, y)$ to the product of sentence lengths $|x| \times |y|$.

To compensate the discussed flaws of AP, another metric called *Average Lagging* (AL) was proposed by Ma et al. [2019a]. Its core principle is based on quantifying the lag behind the ideal policy and thus the speaker. By ideal policy, we denote the system that interprets simultaneously as the words are being said. Such a policy can be denoted as wait-0 and will have AL equal to 0. With such a definition, we can derive that, for the wait-1 policy that is only one word behind the speaker, AL will be 1. And we can even generalize for all k-wait policies for which AL will be equal to k. We can reuse the general latency Equation (2.1) and define AL as the same $L(x, y)$ with

$$C_i(x, y) = g(i) - \frac{i-1}{\tau} \tag{2.2}$$

$$Z(x, y) = \underset{i:g(i)=|\boldsymbol{x}|}{\operatorname{argmin}}(i) \tag{2.3}$$

where $\tau = |y|/|x|$ represents a normalization variable penalizing long target sentences.

Although AL was an improvement over AP, it missed one key component to be utilized for optimization – it is not differentiable. The problem is with the variable τ corresponding to an *argmin* operator that determines at what target token the summation should stop. Therefore Cherry and Foster [2019] also came up with the differentiable average lagging.

We will describe the exact form of a latency metric that we used in our experiments in Section 2.4.1.

2.3.3 Flicker

Apart from the short latency, good interpreters are also distinguished by the consistency of their translations. They always wait for enough context or can start sentences in such a way so they can smoothly continue to several potential scenarios without the need to start over.

When evaluating SiMT, we can observe similar behavior. Sound systems will generate the translation segment after segment without correcting already stated previous partial translations. We will call these output adjustments as flickering – reflecting the number of words needed to be updated between two consecutive partial translation outputs.

To define the flicker more precisely, we will describe it as an *average revision count* per segment. The most complicated term in the definition is revision count RC, which is defined as

$$RC_k = \sum_{i=2}^{n_k} \left(|s_{i-1}| - |\mathrm{LCP}\left(s_{i-1}, s_i\right)| \right) \tag{2.4}$$

where s_i denotes i-th partial segment preceding the current segment k and n_k denotes the number of partial segments between complete segments $k-1$ and k (the complete segment is equivalent to a full sentence translation and partial segments to partial translations). And LCP is the abbreviation of the longest common prefix. Put simply, for each two consecutive segments in a sentence, it counts the number of words after their longest common prefix – thus, heavily penalizing if the partial segments differ from each other at the beginning.

The average is then computed in a standard way as

$$\frac{1}{K} \sum_{k=1}^{K} RC_k \tag{2.5}$$

Now, it is important to emphasize that flicker can be measured only if the system *retranslates*. It means that every time a new chunk of input is added, the system retranslates the whole partial sentence from the beginning. With this approach, the system can reevaluate previous translation mistakes caused by the lack of context and guarantee a perfect translation quality once it reaches the end of the sentence. Consequently, the BLEU score for completed sentences with the same translation model will be the same regardless of the segmentation method used. Therefore, for retranslation, flicker takes over the position of a translation quality metric [Arivazhagan et al., 2020]. On the contrary, if sentence translations are built progressively by concatenating the translation of individual segments, the flicker is constantly zero.

In our experiments, we chose to use the retranslation approach. This way, we can evaluate the flickering of partial translations, which appears to be a more reliable metric when comparing the splitting methods than the BLEU score. Retranslation additionally guarantees better translations once the end of the sentences is reached, which is something that does not apply to the other case. The only drawback of retranslation could be comparatively longer translation times.

It is also important to mention that we could afford to use retranslation because we are working in the speech-to-text mode, where it is much easier to potentially change some of the previous partial translations (e.g., transcribing

a live event). With speech-to-speech machine interpretation, it would be very complicated to change something already said. Therefore, retranslation can be applied only in text-producing settings, and the same applies to flicker measuring.

2.3.4 Visualisation

Translation quality, latency, and flicker are known metrics navigating the improvement of the SiMT field. However, recent research articles have pointed to another aspect that should be considered when optimizing SiMT. And that is the comfort of the user reading the partial outputs on the screen – covered under the term *visualization*.

They claim that even translation with good quality and short latency does not have to be perceived well if not displayed in a readable fashion. This metric naturally focuses on tasks where the final output form is text, e.g., subtitling live conferences, lectures, or political debates. Nevertheless, the theory is not restricted to only simultaneous translation and can be naturally adapted to the offline setting – most typically for subtitling movies or YouTube videos.

Ma et al. [2019a] introduced straightforward word-for-word visualization in which words appear sequentially as the machine generates them. However, it was proven that such visualization is far from ideal for humans. When people read subtitles with this technique, they encounter several problems:

- First, the emission rate of words was noticeably inconsistent since, in some passages, the tempo was too fast and, in others, too slow. Because of that, people's eyes were forced to move more frequently, which made the reading tiring.
- Secondly, people are used to perceiving text in meaningful chunks, so single-word updates require more cognitive effort.
- And lastly, there was also a problem with the unpredictable subtitle disappearance caused by reaching the width limit of the screen. Because after the limit is reached, the whole previous passage is deleted, which might be incredibly distracting in some situations.

For all those reasons, Karakanta et al. [2021] suggested a novel block visualization overcoming the flaws of the previous method. Their first attempt was to display subtitles in bigger blocks (one or two lines) which should prevent viewers from unwanted rereading [Rajendran et al., 2013] and excessive eye movement [Romero-Fresco, 2010]. This approach lowered the cognitive demands for reading the subtitles, but they discovered that it, on the other hand, increased the latency to an unbearable extent.

As a solution, Karakanta's team proposed a scrolling line visualization. In this setting, there are always two lines of subtitles displayed on the screen, and the new text appears line-by-line – moving up the previous row to the upper line. According to experiments, scrolling line visualization lowers the cognitive load when reading the subtitles while keeping bearable latency.

2.4 Evaluation toolkit

When choosing the evaluation toolkit for our experiments, we narrowed our choices to the current two most popular evaluation tools – *SLTev* [Ansari et al., 2021] and *SimulEval* [Ma et al., 2020a].

There were two critical differences between these toolkits for our use case. First, SimulEval does not support flicker evaluation, and secondly, SimulEval's architecture is designed in a client-server manner which requires users to design the code in a specific way. For those reasons, we chose SLTev as our primary evaluation tool.

2.4.1 SLTev

SLTev uses sacreBLEU [Post, 2018] for scoring the translation quality and average revision count metric for assessing interpretation flickering, as described in the above chapters (see Section 2.3.1, Section 2.3.3).

For latency calculation, SLTev proposes two different approaches similar to the average lagging principle, described in Section 2.3.2. The authors proposed that latency corresponds to the delay with which the recipient receives the message – having assigned times for all words in the reference and the proposed candidate translation, they compute the total delay as a sum of individual delays for each word.

With this common idea, they then proposed two possible approaches how to assign times to words in the reference translation.

Proportional delay

The first approach, called *proportional delay*, assigns the times to words in the reference translation using a simple heuristic.

Let us denote the original sentence S and the reference translation R and assume that we know a display time $t(w_i)$ for each word w_i of S. Finally, let s and r be a number of words of S and R, respectively. Then we will compute a display time $t(w_j^R)$ of each word w_j^R of R as

$$t(w_j^R) = t(w_0) + j \times sr \tag{2.6}$$

When we have assigned times to all words in the reference translation, we can finally compute the delay by summing up the time differences between corresponding words in the candidate translation C whose words w_k^C have the display time $t(w_k^C)$ precomputed from the provided data.

To explain the calculation on a real example (see Table 2.1), let us have an English source sentence with display times assigned to each word in centiseconds. Following the Equation (2.6), we computed the word display times for the German reference translation and added a candidate translation with word times recorded from the progressive translation.[2]

[2]The words *"unser"* and *"Unternehmen"* have assigned later times than the last word *"vorstellen"* because they did appear in the partial translations laters.

Original English sentence						
We	would	like	to	introduce	our	company
782	805	827	846	919	961	1062

Reference German Translation						
Wir	würden	gern	unser	Unternehmen	vorstellen	
786	812	836	895	954	1062	

Candidate German translation						
Wir	möchten	unser	Unternehmen	vorstellen		
800	870	1200	1200	910		

Table 2.1: Examples of input sentences for delay calculation with times assigned to each words in centiseconds.

Now, we have everything we need and can proceed to calculate the proportional delay of the reference and candidate translations as

$$(800 - 786) + (1200 - 895) + (1200 - 954) + 0 = 565 \tag{2.7}$$

In the summation, the words *"würden"* and *"gern"* are not included because they did not appear in the candidate translation. The delay of the word *"vorstellen"* is then clipped to 0 because it appeared earlier in the candidate translation than in the reference.

Delay calculation with alignments

The second approach does not use such an evenly distributed assignment of times for their reference translation but utilizes automatic word alignment (SLTev uses MGIZA) that will assign the reference words the times of source words they are aligned to.

In our experiments, we decided to use the first approach – not risking the possibility of flaws in the word alignment.

2.5 Sentence segmentation

Now, it is the time to describe more rigorously what segmentation methods really are and show some examples of methods that are frequently used these days. But even before that, let us explain the type of the SiMT pipeline architecture we used for our experiments because it also affected the types of segmentation methods we were able to try.

2.5.1 Cascade and direct SiMT architectures

SiMT models can be divided into two main categories. Historically, the standard way to create a SiMT system was by creating a pipeline connecting an ASR with a MT module and placing a sentence segmentation logic between them. This type of architecture is called a *cascade model*.

Despite its known drawbacks, namely the error propagation and slower execution, this kind of pipeline enables practitioners to utilize high-quality audio transcriptions and parallel corpora to train working MT systems that are easy to run and maintain.

On the contrary, *direct* or *end-to-end* SiMT systems may benefit from single neural network-like architecture that takes audio signals of a source speech as an input and directly outputs written or spoken translation in a target language. With this approach, errors made by ASR do no longer propagate to the MT module, and there is no additional delay caused by handling the data across two modules. Moreover, training one system as a whole should be theoretically more robust than training two separate ones. There is, however, still not enough data for end-to-end systems because aligning audio in the source language with the text data in the target language is significantly more complex than obtaining simple transcripts and parallel translations.

Comparing both types of models in terms of quality, there have been several studies proclaiming cascade [Inaguma et al., 2020] or end-to-end ([Indurthi et al., 2020], [Pino et al., 2019]) systems as superior. However, without unified benchmarking data, it is impossible to answer the question confidently. This claim is also supported by a recent study by Bentivogli et al. [2021] – revealing that results obtained from the best candidates of both categories are, to this date, on par, and the subtle difference observed in their behavior is beyond the scope of human distinction.

Primarily, we are mentioning this distinction because, for our purposes, we needed to be able to effortlessly switch between the splitting methods and therefore needed to use the cascade architecture.

2.5.2 Segmentation methods

When using the terms from the previous chapter (see Section 1.2.2), searching for the optimal segmentation method corresponds to searching for MTUs. To rephrase it, each segmentation method defines translation units according to different rules, and we are looking for such a set of rules that will produce MTUs and yield comprehensible interpretation with a minimal possible delay.

In reinforcement learning, we can describe segmentation strategies as a learned policy performing two distinct actions – *read* and *write* [Gu et al., 2017]. The input data can be either in a text format as a stream of words or a raw audio signal cut into passages of the same size. The system should then select the read action whenever its inner state has low confidence about the speaker's intent and needs additional context – thus, requesting the following word (or audio segment). And in the opposite situation, when there is a high probability that the following tokens will not affect the translation of the current segment, the system should select the write action and output the current partial translation.

The general goal when learning the policy is always to maximize the translation quality and minimize the delay. Nevertheless, because optimizing quality leads to increased delay and vice versa, it depends on particular requirements of how long delay can be tolerated. This statement, however, does not contradict the fact that for each choice of acceptable delay, there exists an optimal solution that preserves the demanded latency while still translating with high quality.

2.5.3 Categories of segmentation methods

The segmentation strategies can fall into two main categories of decision policies. The first one, denoted as *fixed*, is more widely used and characterized by simple rules focusing only on the number of processed tokens. Because of the ease of its implementation, fixed policies gained popularity, especially in the end-to-end interpretation models. On the other hand, *adaptive policies* are more complex, and the actual context and preceding words influence their segmentation decisions.

Fixed policies

Starting with fixed policies, the widest usage seizes a simple but very effective strategy called wait-k policy [Ma et al., 2019a]. It is based on the simple assumption that most translation units do not exceed size k. Therefore it is sufficient to wait for the first k words at the beginning of the speech and then alternate between reading one source word and writing one translation word.

With little adjustments, the same techniques could also be applied to audio signals that can be segmented into constant chunks [Ma et al., 2020b] or by word boundaries [Ren et al., 2020]. Despite all advantages of the wait-k strategy, people, however, argue that human-like interpreting machines must deploy adaptive segmentation strategies that can produce chunks of different lengths depending on the available context.

Adaptive policies

An interesting intermediate step from wait-k strategies is combining different values of k. It can be implemented so that the system decides an optimal k depending on the translation confidence or by training an NMT system while sampling from several values of k [Elbayad et al., 2020].

Kano et al. [2021] segment input sentences utilizing the part of speech prediction of oncoming sentence constituents. They work on the English-Japan language pair where interpreters face the problem of two different language structures. While English puts verbs right after the subject (SVO), in Japanese, verbs come right at the end of a sentence (SOV). Thus, they created simple rules to keep the object-verb parts in the same chunks.

Another interesting approach is training a predictive model based on a dataset with optimal segmentation. But the critical question is how to define and where to obtain the golden data. In their paper, Zhang et al. [2020], split source sentences into meaningful units (corresponding to MTUs from our previous definitions) using the NMT system and proclaimed them as the golden data. They first translated the source sentences as a whole and then proceeded monotonically word by word while checking when the partial translation made up for a prefix of the complete translation. And at these points, they place the boundaries of meaningful units. To enhance the quality of meaningful units, they retrained their NMT model on translations generated with a new prefix-attention method.

Source segmentation can also be directly predicted from word embeddings and complemented with the acoustic information [Iranzo-Sánchez et al., 2021]. More specifically, they created the acoustic word vectors from three values: duration

of the previous silence, duration of the current word, and duration of the ensuing silence.

Because in our experiments, we wanted to compare as many different methods as possible, we utilized less complex splitting strategies, which, however, can be very easily explained and understood in terms of the type of splits they do.

3. Annotation process

Apart from standard parallel corpora with professional translations, the community has also gathered transcripts from live interpretation. However, this kind of data is still not ideal for deducing ideal speech segmentation for interpretation – either by mapping the corresponding sentence parts manually or, for example, using the alignment method (see Section 4.4). The reason is that the interpreters might have the partial interpretation prepared in mind and ready to present, but because of other external factors, they needed to postpone the speaking. Thus, the ideal segmentation might occur in their head but not in transcripts.

Another problem lies within the nature of interpretation itself. Because it is not based on the word-for-word translation, the only thing that interpreters transfer reliably is the meaning and speakers' intentions. Therefore, the correspondence between words or sentences of the original speech and the interpretation is harder to find than with direct translations.

Despite these obstacles, we proceeded to the task and tried to create a manually annotated dataset with optimal sentence segmentation for interpretation. To our knowledge, all research papers so far have been using BLEU score, latency and potentially flicker as metrics of how good the splitting in simultaneous translation is. However, no dataset could explain what the optimal values of each of the metrics are. Is it better to favor BLEU or latency? To what extent? Or is it dependent on the situation and type of the conversation? We asked ourselves those questions but did not know the answers. Therefore, we decided to construct a dataset with an optimal splitting that could reveal what an optimal segmentation strategy would look like if there were a perfect interpreter with an infinite amount of time for thinking and complete information about the sentences, including the not yet uttered parts.

3.1 ESIC dataset

For the annotation and also other experiments in this thesis, we used the Europarl Simultaneous Interpreting Corpus (ESIC) dataset [Macháček et al., 2021] that contains 10 hours of recordings and manually-checked transcripts of European Parliament speeches in English with Czech and German interpretations.

The dataset is divided into 370 speech folders, each containing data for a particular interpretation of an English speaker. In each speech folder, there are several files corresponding to different formats of transcripts – with different markers, timestamps, or level of quality. There are audio and video recordings from the European Parliament as well.

In this thesis, we were actively working with files called *en.OSt.man.orto.txt* that contain the orthographical form of the English speech without special tags, disfluencies, and with sentences separated on individual lines. For the time information, we used *en.OSt.man.orto+ts.txt* files that copy the previous files' format but also display a start and end time for each word.

3.2 Annotation methodology

When thinking about an annotation methodology, we had two main ideas. The first idea was to take the original English sentences and Czech interpretations from the dataset and let annotators find the word-to-word or rather phrase-to-phrase correspondence. Although this approach would definitely make sense and would not be that time-demanding, it has one severe drawback we have already discussed – human interpreters may have the right partial translations in their minds much earlier than they say them aloud because they need to listen and interpret simultaneously. Machines do not suffer from the lack of memory or processing power, so we would not be able to get the optimal splitting strategy from those annotations.

Keeping this fact in mind, we decided that we needed to be more precise and have a better way of how to get the splitting resulting in ideal partial translations. As a result, we completely omitted the available Czech and German interpretations from the ESIC and focused only on the original English sentences. We formulated our goal for the annotators in the following way:

Insert a special splitting symbol ("@") into the English sentences so that whenever the translators will reach the splitting symbol, they can translate the partial sentence leading up to that symbol without worrying that some further information hidden in the forthcoming text will alter their partial translation.

More specifically, we let annotators read full transcripts of each English sentence and find the optimal splits inside them. By optimal, we mean the shortest possible text chunks that will be translated the same with or without knowing how the sentence will continue.

With this approach, an interesting question arises that an arbitrary sentence part can potentially have an unlimited number of possible continuations. Therefore, it would be difficult to iterate over all possibilities, and the annotation would be hard to make. So we prompted annotators to use their judgment and willingness to risk when deciding about splits. As a result, each annotator might have a slightly different splitting strategy. Nevertheless, it should always be optimal from their point of view – splitting as soon as meaningful partial translations arise.

Lastly, we needed to emphasize to annotators the language that should be used as the reference translation – because, as we mentioned in Section 1.2.2, optimal splits depend on the target language. Based on the availability of Czech native speakers, we naturally opted for Czech as our target language. Ideally, we would also want to use German and even other target languages, so we would be able to compare how much the splits varied. However, due to the time and resource restrictions, we could annotate each sentence only with the Czech target reference and only once. The same applies for having the same sentences split by multiple people with the same target language to get an inter-annotator agreement.

After a few initial iterations, we asked annotators to additionally supply the splitting with the reference translations that are split into the same number of

chunks as their English counterparts. This requirement made the annotation more time-consuming but provided valuable data we could use in our experiments.

3.3 Technical details of the annotation

When creating the annotations, we used *Google Drive* with *Google Documents* as our default environment (see Figure 3.1). We chose Google environment mainly because of its simple user interface and general familiarity. Since our task required only the insertion of splitting marks into prepared texts and providing corresponding translations, a simple text editor served our purpose very well. As for the splitting marks, we used the character "@" for its good visibility and because it was not contained in the ESIC dataset.

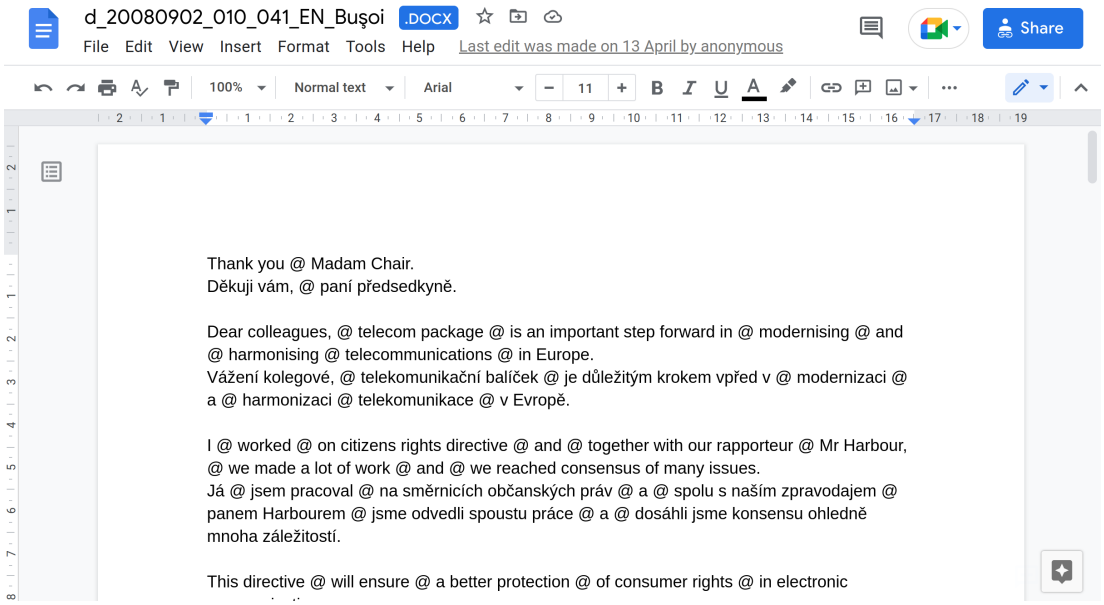

Figure 3.1: Screenshot of a Google Document with an annotated speech folder from the ESIC dataset by one of our annotators. We included some additional examples from the annotation in Figure A.4.

For convenience, we randomly split the whole ESIC datasets into 25 packages – each containing 14 or 15 text files with English transcripts divided into sentences on each line.

In total, we managed to work with six annotators – all studying translation or interpretation in the bachelor's or master's programmes at *Charles University, Faculty of Arts*. We initially had a 30-minute introductory call with each of them explaining the work's purpose and discussing the annotators' manual we sent beforehand.

Annotators successively reserved packages one at a time so that only one person could work on one package. They worked depending on their time availability, and thus, each of them managed to complete different amounts of packages (see Table 3.1). We needed to annotate the last two remaining packages ourselves to finish the annotation in time.

	# of annotated speech folders	# of annotated sentences
Annotator 1	45	684
Annotator 2	89	1334
Annotator 3	73	1106
Annotator 4	73	1077
Annotator 5	45	577
Annotator 6	30	459
Annotator 7	15	249

Table 3.1: Table showing number of folders and sentences that annotators managed to complete.

There were between 178 and 275 sentences in each annotation package, and it took approximately 5 to 7 hours to finish one. A significant portion of that time caused the need to write down the reference translations – splitting alone would take less time but may not be that precise.

During annotating, annotators were allowed to use publicly available automatic translation tools (e.g., Google Translator or LINDAT) to look up unknown words and the Internet to search for abbreviations or complicated technical terms. Therefore, we expect some reference translations might be adopted from a translation tool and then only manually modified. The annotators should, however, always check that such translations yield the optimal splitting conditions.

To simulate the real settings as closely as possible, annotators were also allowed to use interpreting techniques if they knew any. As a result, the reference translations do not need to be exact translations of the original English sentences – for example, *"Government of Côte d'Ivoire"* can be translated as *"tamní vláda" (the local government)* if the meaning is clear from the preceding context.

After the annotation was finished, we checked whether the annotation format was preserved and corrected any undesired errors. Very frequently, we encountered two types of mistakes. Firstly, we needed to separate the splitting symbol from the neighboring words, and secondly, we had to frequently double-check if the number of splits in the original English sentences corresponded to the number of splits in the target Czech translations.

Missing spaces before and after the splitting symbols could be corrected very quickly in an automated way by checking the first and last characters of words and potentially separating them if they were equal to the splitting symbol. On the other hand, fixing a different number of splitting symbols in original and target sentences was more challenging because there is no simple way to correct that automatically. Therefore, every time we detected sentence pairs with a different number of splits, we needed to manually search the sentences and add or remove splitting symbols to even out the splits. From our experience, these mistakes were not caused by the inability to find the correct splitting but by lack of attention because it is very easy to get distracted when doing such uniform tasks, especially for an extended period of time.

Apart from annotations, we also took notes about various typing errors in the dataset that will be corrected and incorporated into the version of the ESIC dataset.

The manual annotation of optimal segments and reference translations has

received funding from the grant 19-26934X (NEUREM3) of the Czech Science Foundation and took four months to complete, starting in February 2022 and ending by the end of May 2022.[1]

3.4 Feedback from annotators

During the whole time of the dataset creation, we collected feedback from our annotators and got our own experience when annotating as well. Here are the most interesting points we uncovered when creating optimal splits using the Czech translations as a reference:

- Splitting sentences into meaningful stand-alone chunks feels challenging, but one can get better and faster at the task with the increasing experience. Especially when dealing with texts from the same source (e.g., political debates), one can quickly learn the similar patterns
- When working on the sentence level, one can sometimes get confused with the pronouns that can refer to nouns occurring in the previous sentences or will become evident only after reading the future sentences. We did not consider this as a problem since the annotators always had the whole text at their disposal, and the number of situations when a pronoun was referencing, for example, a commonly known person or a person in the room, was insignificant.
- The longer chunks were often caused by the need to wait for a noun located at the end of the sentence. Because otherwise, it would be necessary to guess some endings of previous words, such as adjectives which in the Czech language have different endings based on the gender of an associated noun.
- Some prepositions (e.g., *"for"*, *"at"*) and conjunctions (e.g., *"and"*, *"or"*) can be, in some cases split into individual segments, but at the same time, in other sentences, the very same prepositions/conjunctions need to be put in more extended segments to find out how they should be translated.

  We noticed it as well, especially with conjunctions. When, for example, conjunction *"and"* appeared between two clauses, we always put it into a stand-alone segment. However, when it was connecting two adjectives, we waited for a noun to figure out their endings.
- The length of segments depends not only on the used words but also on the length of sentences. Longer sentences with more complex structures usually lead to longer segments. Similarly, the speaker's level of English knowledge might have the same effect.

  Among the recordings in the dataset, there were both native and non-native English speakers. The annotators noticed that non-native speakers use less predictable sentence constructs that lead to longer segments.

[1] Link to the annotation is included within the attachment to this thesis.

- In the Czech language, there is a tendency to put essential pieces of information at the end of the sentence. In English, there is not so much emphasis on that. In technical terms, this aspect is connected to the topic-focus articulation, which also impacts the segmentation.

  Let us look at the following English sentence:

  "They speak French in Quebec as well."

  In broken Czech, we could translate it in the same order as:

  "Oni mluví francouzky v Quebecu také."

  which would allow us to make the shortest possible segments. However, the translation sounds robotic, and a good translator would never translate it that way. With an intention to emphasize the word *"french"*[2], the proper translation would probably be:

  "V Quebecu také mluví francouzky."

  So firstly, we did not translate the word *"They"* at all because, in Czech, people usually omit pronouns at the beginning of the sentence and express them only with the suitable form of the verb. And secondly, we completely changed the order of the sentence, which sounds correct but requires us to wait until the end of the sentence to start translating – therefore, the whole sentence would be one long segment.

  When splitting up sentences into segments, one will, therefore, encounter situations like this very often, and it is up to the annotator to find the right balance, whether to wait for the end of the sentence and translate it in the correct order or start translating earlier but risk that the final translation will sound robotic.

- Lastly, there are many ways to translate a sentence, but only some lead to optimal splitting. That is a bit different from the previous point because all translations may be correct and fluent, but only some of them produce the shortest (i.e., optimal) segmentation.

  Put differently, even an ideal interpreter would be facing the issue whether to focus more on simultaneity (more frequent splitting), or on fluency which is an aspect of the target language.

[2]Depending on the context, the aim may also be to emphasize the word *"Quebec"*. In that case, the right translation would rather be *"Francouzsky se mluví i v Quebecu"*.

4. Splitting methods

In this section, we would like to describe and explain the algorithms that we used for the segmentation. We will also provide their different variants and possible enhancements alongside the description. To each method's description, we will also include in **bold** the abbreviation that we will use in the subsequent parts of the thesis. For example, instead of writing about a segmenting method that splits after each word, we will denote it as *every word splitting*.

For most methods, we also included a pseudo-algorithm that takes a sentence divided into words as an input and returns a list of words' indices, after which a segmentation method will put the splitting symbol.

4.1 Control segmentation strategies

To put extreme values into perspective, we needed to cover in our analysis the two control methods:

- Splitting after every word **(every word splitting)**
- And splitting only at the end of sentences **(full sentence splitting)**

The description should be self-explanatory. With every word splitting, we attempt to create a new partial translation after each new word of the sentence. And on the opposite side, with full sentence splitting, we split only when the sentences are finished.

4.2 Splitting after punctuation characters

Apart from control splits, another straightforward segmentation method is splitting based on the currently processed word. Since human language is complex and each word can have a different meaning depending on the context, it is not obvious that there is a method that could spot the right moments for splitting only based on the current word without an additional context.

However, there is a category of words that could yield reliable results – punctuation (e.g., dots, commas, or question marks). Because the function of punctuation in the text is to split sentences into clauses and other standalone parts, it is only natural to use them as indicators for interpreting **(punctuation splitting)**. They cannot be heard and are visible only in a text format. Therefore, we need an ASR with reliable punctuation prediction for this method to work in real interpretation settings. Nevertheless, having the speech preprocessed into a text, the algorithm is very straightforward and does not need any lookahead. After analyzing punctuation marks in the ESIC dataset, we decided to set the rule to split whenever the current word ends with *“.”*, *“,”*, *“;”*, *“?”*, *“!”* or *“:”*.

Algorithm 1: Punctuation splitting

Input: sentence_words, punctuation_chars
Output: splits

```
splits = []
for (i,word) in enumerate(sentence_words) do
    for punctuation in punctuation_chars do
        if word.endswith(punctuation) then
            splits.append(i)
            break
```

Naturally, one could extend this method to conjunctions as well. Because, similarly to commas, they split sentences into clauses. However, we decided not to include them in our analysis since coordination is a complex topic, and the conjunctions can, for example, connect two adjectives describing a noun that follows right after. Therefore, when splitting only based on the information about the currently processed word, it would not be able to distinguish among those several subcases. This is a particular case for translating English to Czech and might be irrelevant for other language pairs.

4.3 Segmentation based on silence

Before explaining the **silence splitting**, we would like to describe a new visualization tool that helped us formulate it.

4.3.1 Tool for visualization of interpretation

When we first started working on the topic of speech segmentation, we did not have a clear way how to represent interpretation with both the text and time information. That is why we developed a visualization tool that helped us understand how professional interpreters work with the language and how well they can follow the speaker.

When analyzing the start and end times of words in the dataset, we noticed two groups of neighboring words – the ones that follow each other without a break and those that do not. In other words, the end time of the previous word may be equal to the start time of the following word, or there might be a moment of silence.

Now is a good moment to remind ourselves about how the times of words were obtained in the first place. In the ESIC dataset, words' start and end times were generated automatically with the *forced alignment* [Kisler et al., 2017] – not manually. Thus, they might not always be reliable and correct all the time (see Figure 4.1). We verified that assumption by analyzing the audio signals of recordings with the *pydub* library. We first converted the original *ogg* files to the *mp3* format and then extracted the magnitudes of audio waves. After that, we detected silent audio ranges by comparing their magnitudes to a silence-threshold value.

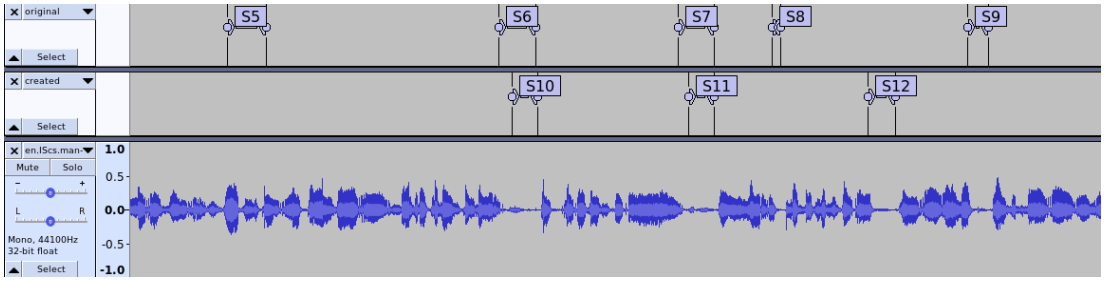

Figure 4.1: Screenshot from Audacity displaying an audio recording from the ESIC dataset in the bottom track with two sets of intervals of silence above it. In the first row, the silent ranges were detected using the word times from ESIC data, and in the other one, the silent ranges were detected by our silence threshold algorithm.

Nevertheless, let us return to the visualization tool. The initial thought was to place sentences word by word on a plot where the x-axis would represent a time when the words were said (choosing between start or end times).

To enhance the previous idea, we also needed to utilize the y-axis. First, we turned around the plot and repositioned the time to the y-axis, going from top to bottom. Then, we used the information about the silent parts between words so that every time the following word did not have the same start time as the end time of the previous word; we shifted it for $+\delta$ to the right. This way, we can see in what parts of the sentence the speaker made pauses or the interpreter waited for additional context. The words can be shifted to the right multiple times, but when a word has the same start time as the previous word's end time, it is shifted back to the left by $-\delta$. So it can gradually get back to the initial x-position but not beyond that (see Figure 4.2).

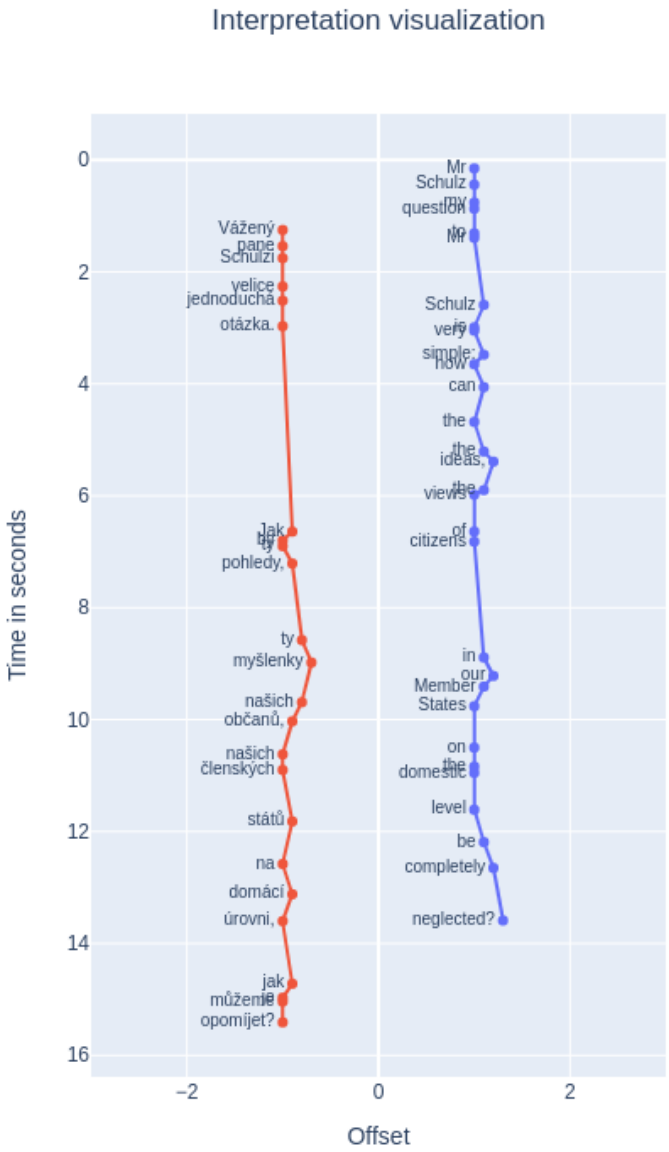

Figure 4.2: Image of our visualization tool displaying how well the Czech interpreter could follow the English speaker. The y-axis is running top-to-bottom with the words, the x-axis left-to-right indicates when there was a pause between the words.

We believe that this representation can help linguists to visualize the flow of interpretation better and, with some modifications, can be potentially even used as a basis for a metric of the interpreter's performance – such as maximum deviation from the initial x-position, most extended sequence without a pause, the total number of pauses and more.

The visualization can additionally be enhanced with automatic word alignment when displaying both original speeches and their interpretation in one plot (see Figure 4.3). This way, we can see the correspondence between words in both utterances, which might help to quickly get oriented during the analysis.

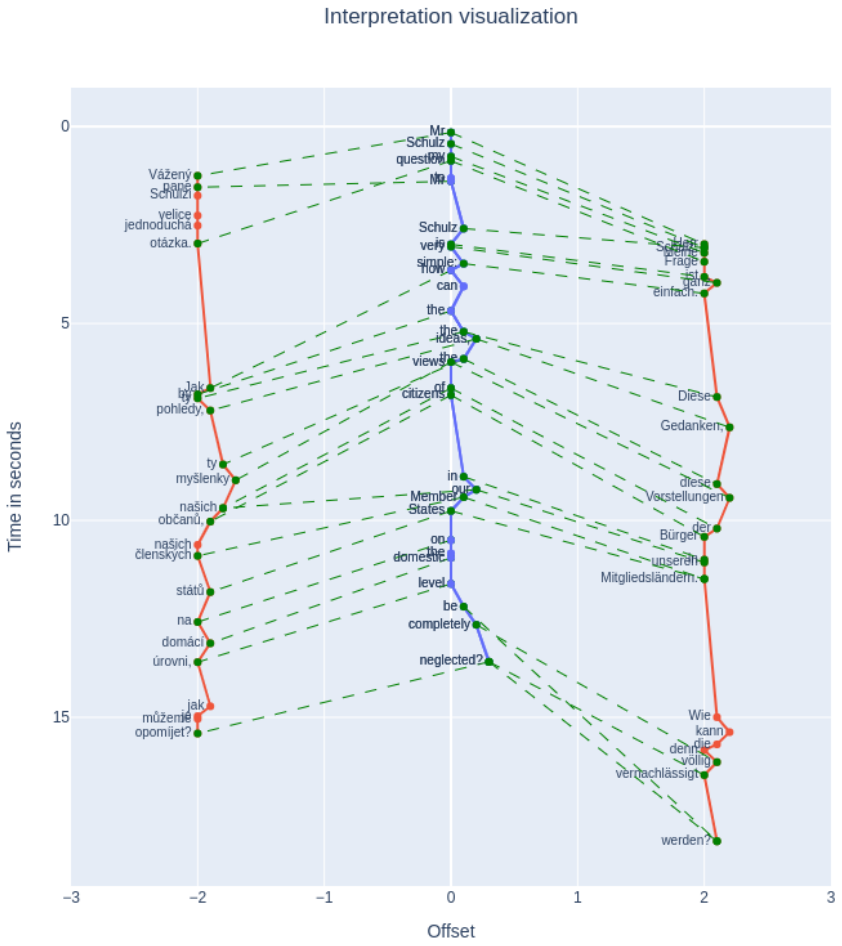

Figure 4.3: Graphical visualization of the Czech (left) and German (right) interpretation supplemented with the word alignment from the awesome-align tool in green, see Section 4.4.2 below.

4.3.2 Splitting method

After creating the visualization tool, we knew we should also try to utilize the information about intervals of silence between words for segmentation in the following way – split in the silent moments.

Algorithm 2: Silence splitting

```
Input: sentence_words_with_times
Output: splits
splits = []
for (i,word) in enumerate(sentence_words_with_times) do
    if i == 0 then
        previous_word = word
        continue
    if previous_word.end_time != word.start_time then
        splits.append(i-1)
    previous_word = word
```

To validate whether our idea is promising, we verified that after 86.99% of sentences from ESIC, there is a silent moment before the start of another one. This observation assures us about the possibility of our hypothesis that many speakers make pauses after finishing meaningful pieces of information, even though they usually read their speeches. Out of 91133 pairs of words in the dataset, the silent spaces were detected between 21932 of them.

4.3.3 Possible intonation extension

When experimenting, we were also looking at the possibility of splitting sentences based on the speaker's intonation using, e.g., the *crepe* library [Kim et al., 2018]. Eventually, we did not include it in our final experiments because it was very complicated to set the correct values and thresholds for splitting. However, we believe this approach could also effectively reveal the right places for splitting since the information is available straight from the audio and does not require reaching the end of sentences.

4.4 Word alignment splitting

Word alignment is a well-known task that is part of several NLP applications. It found meaningful applications mainly in the past when it was an essential component of statistical machine translation models. However, even today, word alignment is sometimes used in more advanced NMT models (e.g., for generating attention priors or decoding) and many other machine translation-related disciplines.

Generally, word alignment is usually performed on the sentence level to find the matching between two groups of words, typically from two different languages, that have a common semantical meaning. In the ideal case, each word would have only one counterpart, but because languages can follow different grammar rules and have a distinct vocabulary, the pairing may be one-to-many, many-to-many, or some words can even be left without a pairing.

For sentence segmentation, we will use the word alignment to construct an algorithm that will separate words into standalone chunks so that no pairing from the word alignment will cross the borders of those chunks. This technique has already been used, for example, by Xu et al. [2005].

4.4.1 Splitting method

The algorithm works with two parallel texts – from the source and target language, that are firstly aligned with an arbitrary word alignment tool.

Some tools may return two sets of alignments – *sure* and *possible* alignments. The sure set contains alignments of words that are each other's counterparts, whereas possible alignments cover those connections that are more ambiguous. For example, when aligning the sentence

"I have been living in Czechia for 10 years."

with the Czech translation

"V Česku již žiji 10 let."

The pair *("living", "žiji")* would be categorized as a sure alignment, while pairs *("has", "žiji")* and *("been", "žiji")* as possible alignments.

For segmentation, we recommend using the union of sure and possible alignments because covering all possible alignments can reveal those less apparent connections that can prevent the algorithm from splitting too early with an insufficient amount of information.

When we have the alignments, we can start splitting. The main idea is to use the word alignments to split the sentences into such parts whose word alignments do not cross. At the same time, these segmented parts need to be minimal. Therefore, we need to find for each source word the maximal index of a word from the target sentence to which the current word or any preceding word is aligned. Then we iterate through the list of maximally aligned indices, and when we find a word whose maximal alignment is equal to or higher than for any other preceding word after the last splitting, we make a new split. We summarized the process in the following pseudo-algorithm **(alignment splitting**).

Algorithm 3: Alignment splitting

Input: word_alignments, sentence_words
Output: splits

```
max_aligned_is= find_maximal_alignment_indices(word_alignments)
current_i = 0
splits = []
while True do
    current_max_alignment = max_aligned_is[current_i]
    split_after = current_i
    for (j, max_alignment) in enumerate(max_aligned_is[current_i+1:]) do
        if max_alignment ≤ current_max_alignment and
          max_aligned_is[current_i+1: j] ≤ current_max_alignment then
            split_after = j
            break
    splits.append(split_after)
    current_i = split_after + 1
    if current_i ≥ len(sentence_words) then
        break
```

4.4.2 Word alignment tools

For our main analysis, we used two different word alignment tools to see how much they would differ in their segmentation performance.

The first tool we used was a statistical aligner based on the famous IBM models – *fast-align* [Dyer et al., 2013]. It is an unsupervised aligner that needs to be trained for every language pair separately (the choice of source and target language matters). Thus, we first trained the model on English-Czech sentence pairs of the EUROPARL dataset [Koehn, 2005] and then aligned English sentences from the ESIC with the Czech translations that annotators prepared as part of our annotation process.[1]

Similarly, we also extracted alignments from the *awesome-align* [Dou and Neubig, 2021], which extracts word alignments from multilingual BERT embeddings. We used its default model without any additional finetuning.

4.5 Segmentation based on phrase chunking

According to Karakanta et al. [2021], another critical aspect of segmentation during machine interpretation is how much standalone the segments are. In other words, it is better to wait a bit longer but output segments that make sense on their own. When thinking about these criteria, we started to work with the idea that we should also try to do splits based on *phrase chunking*.

Because of the similar names, phrase chunking can get confused with the sentence segmentation that this thesis is focused on. Thus, when speaking about segmentation or chunking in general, we mean segmentation for interpretation purposes. And on the other hand, when we mean the process of extracting sentence constituents such as noun or verb phrases, we will always denote it as phrase chunking.

Phrase chunking is a process of splitting sentence into meaningful units. We can look at this task as a sequence tagging problem where we assign each token with a label of a chunk to which it belongs.[2]

It is also worth mentioning that phrase chunking originates in sentence parse trees that consecutively divide sentences into smaller parts until only individual words remain. Therefore, the process of labeling a sentence with phrase chunks is similar to finding the right level of depth for each group of words in a syntactic tree to label them in a disjunctive manner. Naturally, this can be a source of some ambiguity in the labels, especially when, e.g., deciding whether the group of neighboring words belongs to one or several chunks of the same type – from our experience, this is very typical for noun phrases. Therefore, different models can predict particular phrase chunks differently.

For our experiments, we chose an NLP framework called *flair* [Akbik et al., 2019] – gathering several state-of-the-art sequence labeling models for tasks such

[1] We did not use the original Czech sentences from the ESIC because they came from the interpretation, and as a result, the number of sentences was different from the English texts. If we really needed to use the ESIC interpretation data, we could perform the word alignment on the document level. That, however, would be less precise.

[2] Concrete implementation of tagging such as Inside-Outside-Beginning notation or other types of labeling is always dependent on the particular tool.

as named entity recognition, part-of-speech tagging, or phrase chunking. Concretely, their phrase chunking model is based on *flair embeddings* using the bidirectional LSTM-CRF layer. We included a list of all phrase chunks tags in the Appendix (see Table A.2)

4.5.1 Splitting method

Compared to the alignment splitting, where one needs to have complete sentences of both the source and target texts to split up the sentence, with **phrase chunk splitting**, we can work directly with both complete and partial sentences.

With complete sentences, the splitting algorithm is reasonably simple. We let the model tag words with phrase chunk labels and then, according to our own rules, make a split between certain types of chunks. The splitting rules might be in three different forms:

- Split between two types of phrase chunks
- Split after a certain type of a phrase chunk
- Split before a certain type of a phrase chunk

The only problem might be the words assigned with no phrase chunk label. Nevertheless, since these are usually single words between regularly labeled phrase chunks, connecting them to the preceding phrase chunk is the best solution. If they would be connected to the following phrase chunk, there might be a risk that those seemingly unimportant words may change the translation of the preceding chunk. So prolonging the preceding chunk is a safer option.

Algorithm 4: Phrase chunk splitting

```
Input: sentence_words, split_rules
Output: splits
sentence_phrase_chunks = predict_phrase_chunks(sentence_words)
for (i,phrase_chunk) in enumerate(sentence_phrase_chunks) do
    if i == 0 then
        previous_phrase_chunk = phrase_chunk
        continue
    if (previous_phrase_chunk.label, phrase_chunk.label) in split_rules then
        splits.append(phrase_chunk.last_word_i)
    previous_phrase_chunk = phrase_chunk
```

4.5.2 Sequential approach

In the real interpretation setting, methods need to be able to predict segments also with partial sentences. Therefore, we proceeded very similarly when working with progressively growing partial sentences.

While the phrase chunking model has no problem predicting tags for unfinished sentences, the tags may change depending on how big context we provide. The only requirement is to have at least one-word lookahead. That means predicting the split before the last known word. With the very same principle, however,

we can have a two, three, or n-word lookahead which may make the tags more stable.

In the algorithm, we first predict the chunks inside the partial sentence and then check whether the last word is in the same chunk as the word before it. If the last word is in a different phrase chunk and the pair of phrase chunks corresponds to some of our splitting criteria, we make a split. This way, we can iterate through the whole sentence and predict the splits for all words while always working with only a partial sentence.

Algorithm 5: Sequential phrase chunk splitting

```
   Input: sentence_words, lookahead, split_rules
   Output: splits
 1 assert lookahead > 0
 2 partial_sentence = []
 3 current_word_i = lookahead
 4 for (i,word) in enumerate(sentence_words) do
 5     partial_sentence.append(word)
 6     if i < lookahead then
 7         continue
 8     partial_p_chs = predict_phrase_chunks(sentence_words)
 9     current_word_p_ch = get_word_p_ch(partial_p_chs, current_word_i)
10     next_word_p_ch = get_word_p_ch(partial_p_chs, current_word_i + 1)
11     if current_word_p_ch.index != next_word_p_ch.index then
12         if (current_word_p_ch.label, next_word_p_ch.label) in split_rules
            then
13             splits.append(current_word_i)
14     current_word_i += 1
```

4.5.3 Problems with different tokenizers

Here, we would also like to mention a generally important issue that arises when using different NLP tools for segmenting methods. Commonly, most of the NLP tools come with their own tokenization preprocessing – a way how to split sentences into elementary units. The most basic example of tokenization is to divide sentences into words split by spaces, but other types of tokenization are usually more granular, for example, regarding a verb *"doesn't"* as two tokens *"does"* and *"n't"*.

In those cases, we must allow all our segmentation methods to make splits at the very same places because otherwise, it would be hard to compare them. In our case, we decided that splitting can occur only between two words divided by space because we instructed our annotators to do it that way. Therefore, for every segmentation method, we have to check whether some splits are not present in the middle of words. And if they are, always shift them to the first space after a particular word.

4.6 Segmentation based on POS tags

Similar principles described for phrase chunking also apply to splitting sentences based on the part-of-speech (POS) tags **(POS splitting)**. The only difference is that during the process of POS tagging, we assign POS labels to each token in a sentence, not to groups of words as phrase chunking does.

With the same approach as with phrase chunks, we can split up both complete and partial sentences by defining the same three types of rules:

- Split between certain two types of POS tags
- Split after a certain type of POS tag
- Split before a certain type of POS tag

Additionally, and this is again very similar to phrase chunking, we need to pay close attention to what type of tokenization the POS tagger uses. Because in this case, one word can be split into more tokens and so have multiple POS tags. Therefore, when splitting according to our rules, we did not check for equality of POS tags but whether a word contains at least one of the POS tags from a splitting rule.

As a POS tagging tool, we chose to use Stanza [Qi et al., 2020], which covered the POS tags with reasonable granularity (see the whole list of tags in Table A.1). The logic behind the algorithm is very similar to phrase chunk splitting. For the reference, see Section 4.5.1.

4.7 Splitting with machine translation

When we instructed annotators about how to make golden splits in the data, we told them to split the sentences at such places so that no additional information in the sentence would affect the partial translation. In other words, we looked for sentence prefixes whose translation is a prefix of the translation of the complete sentence. The good thing is that a very similar process can also be simulated with a machine translation model **(translation splitting)**.

It is sufficient to translate the whole sentence at first and then consequently translate all its prefixes while checking whether their translation is a prefix of the complete translation. If it is, then the split sign is placed after the last word of the original prefix. Similarly to the alignment splitting, this method requires access to a complete sentence to be able to split.

Algorithm 6: Translation splitting

Input: sentence_words
Output: splits

```
sentence_translation = translate(sentence_words)
partial_sentence = []
for (i,word) in enumerate(sentence_words) do
    partial_sentence.append(word)
    partial_translation = translate(partial_sentence)
    if sentence_translation.starts_with(partial_translation) then
        splits.append(i)
```

4.7.1 Machine translation theory

Since translation is an essential topic for sentence segmentation evaluation and also for the previous method, we provide a short introduction to the topic of neural machine translation (NMT). In the past, state-of-the-art models for machine translation were statistical models. However, with increasing data and computational resources, the Transformer-based neural models has now become the number one choice for translation.

The basic principle of generating a sentence in a target language from an original sentence X can be described with the following equation:

$$p(Y \mid X) = \prod_{t=1}^{|Y|} P\left(y_t \mid X, y_{<t}\right) \tag{4.1}$$

where $X = x_1x_2x_3...x_n$ represents the original sentence with n input tokens and $Y = y_1y_2y_3...y_m$ stands for the target sentence with m tokens.

Nowadays, almost all NMT models utilize the encoder-decoder framework [Cho et al., 2014] – consisting of four main components:

- Embedding layer
- Encoder
- Decoder
- Classification layer

The input text is first tokenized into a list of tokens which are mapped via the embedding layer into a continuous space of vectors $x \in \mathbb{R}^d$. After that, the vectors are sent to the encoder and decoder layers, whose outputs are then projected back to the space of words with the classification layer (see Figure 4.4).

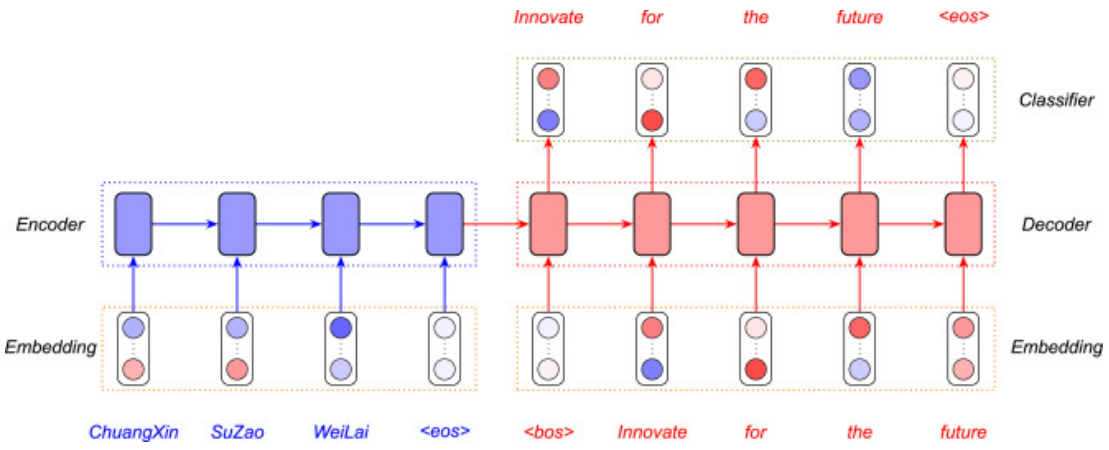

Figure 4.4: Overview of a general NMT architecture with the embedding, encoder, decoder and classification layer [Tan et al., 2020].

As we wrote in the beginning, the number one choice for the encoder and decoder is the Transformer architecture [Vaswani et al., 2017] which formerly replaced the recurrent neural networks by achieving state-of-the-results on various language pairs as well as other NLP tasks.

4.7.2 NMT training

Typically, the NMT models are trained on whole sentences, and thus, their performance on the unfinished sentences might be impaired – the translations make less sense, or the model tries to translate them as finished sentences by forcing the punctuation marks at the end. We wanted to explore this topic more, and therefore apart from the standard NMT model trained on complete sentences, we also incorporated into our experiments one additional model finetuned on partial sentences.

Baseline NMT

To train the NMT models, we used the *HuggingFace* library [Wolf et al., 2019]. As the original speeches in the ESIC dataset are in English and because we managed to annotate our dataset only with Czech reference translations, we needed to train our models in only one direction – from English to Czech.

We trained our baseline translation model by finetuning the *Marian*-based [Junczys-Dowmunt et al., 2018] opus-mt-en-cs model from *Helsinki-NLP* group with data from the English-to-Czech subset of the WMT16 dataset containing 997240 training parallel sentence pairs.

We run the finetuning for one epoch with a learning-rate set to 2e-5 and the weight-decay of 0.01 using the Huggingface's *Seq2SeqTrainer*.

Finetuned NMT

We created the second model by repeating the finetuning with the same settings but this time with partial sentences. However, before proceeding to training, we first needed to obtain the training data. We reused the same WMT16 dataset and randomly sampled 30000 sentence pairs. Using our baseline translation model, we then split each English sentence using the translation splitting method (see Section 4.7).

When we collected splitted English and Czech sentences, we had to decide what kind of splits to use. Because in a sentence split into n chunks, $n-1$ possible partial sentences can be used. Selecting only one split would not sufficiently utilize the data, but on the other hand, using all splits in every sentence could lead to overfitting for longer sentences with more splits.

Therefore, we randomly sampled at most three partial sentences from each sentence. This means that we selected all partial prefixes for all sentences with four or fewer segments, and for sentences with more segments, we randomly picked three. Through this process, we generated 76787 training sentence split pairs.

Finally, we complemented the partial sentence pairs with the same number of complete sentences sampled randomly from the rest of the WMT16 dataset to avoid catastrophic forgetting [Kirkpatrick et al., 2017]. However, we are aware this topic is really complex, and many different strategies for training an NMT system to handle prefixes could potentially be explored.

When applied to the same sentence, the concrete behavior of both translation models can be seen in Figure A.5.

4.8 Segmentation from the annotation

The last type of segmentation we used in the experiments is the one manually created by annotators **(golden splitting)**. As mentioned before (see Section 3.2), their instruction was to create as many splits as possible so that nothing that comes after a split would alter the way how they would translate the partial sentence leading up to the splitting sign.

4.9 From offline to online segmentation

As mentioned in sections above, some methods require information that is not generally available during the inference time (e.g., reference translations or word alignments). Therefore, in order to also present realistic results, we needed to adapt those methods to the online setting.

By an *offline* environment, we mean the situation where we have complete sentences and potentially even their translations with the word alignment. Conversely, an *online* environment should simulate the real interpretation conditions where the methods process the sentences word-by-word and need to decide about splitting even before a sentence ends. In this regard, we can divide our splitting methods into three groups:

- Methods working in offline and online environments with consistent performance.
- Methods that can work online, but the quality of splitting is determined by how many words ahead they can see.
- And lastly, methods that can work only in offline settings and therefore can do splits only when the whole sentences are available.

In the first group, there are four of our splitting methods – splitting based on silence, punctuation, and the two control methods. These methods process sentences word-by-word and and at any given moment can decide whether there should be a split after the last word or not. With control methods, the splitting conditions are trivial. In the case of punctuation, it is only necessary to check whether the current word ends with an appropriate character. And when we split depending on silence, though it may change based on the inner working of different ASR models, the information on whether a silent part follows a current word or not should be provided with very little latency.

In the second group, there is the splitting based on phrase chunks and POS tags. For both of these methods, there is a need to have at least one-word lookahead to decide the split – because the algorithms need to know the classification of the one word ahead to check their splitting criteria.[3] It is, however, important to notice that the more lookahead the algorithms have, the more precise their chunk and POS classifications should be because of the additional context (see Figure 4.5).

[3]Omitting the possibility that the POS splitting would contain only rules in the form "split after" a particular POS tag

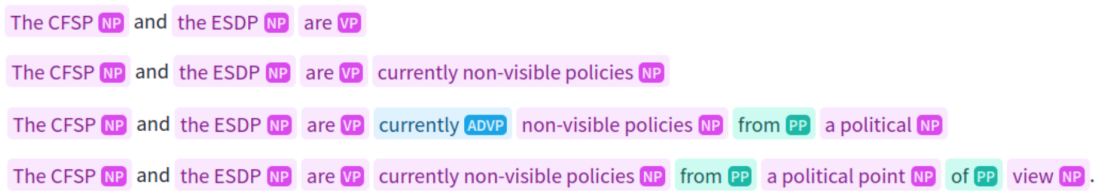

Figure 4.5: Changes of phrase chunk prediction with a progressively growing sentence.

The remaining group (alignment, translation, and golden data splitting) differs from all the previous methods because there is no way how they can be deployed in online settings. They will always require to see at least the end of the current sentence to make some reasonable split predictions. Therefore, for those methods, we needed to train a statistical model that learns to predict the splits even for partial sentences.

4.9.1 Training of statistical splitting models

To train the models, we again decided to utilize the *flair* library [Akbik et al., 2019] that also contains pre-trained word embeddings and pipelines for different machine learning tasks.

For our purpose, we chose to use *binary classification* models that will predict whether there should be a splitting sign after the n-th word from the end of the current partial sentence or not. In this case, different values of n correspond to different lookaheads. But before actual training, we needed to conveniently create train, dev, and test datasets. For this purpose, we use the ESIC dataset in which we first split sentences using all three methods with the knowledge of complete sentences.

Datasets creation process

Totally, there are 5486 sentences in the ESIC dataset, which are divided into 370 speech folders. By default, the speech folders are already divided into train, dev, and test folders, but the test folder was too big for our experiments. Thus, we needed to redistribute the speech folders to keep only some of the speech folders in dev and test sets and let the rest be transferred to the training set.

We randomly iterated through the speech folders in the original dev set and cumulatively counted the sentences in them. After reaching more than 400 sentences, we stopped and transferred the rest of the speech folders to the training set. We repeated the same procedures also with the test set, where we kept at least 1000 sentences. Following this process, we ended up with 4069 sentences in the training set, 407 in the dev set, and 1010 sentences for testing.

When training binary classification models, one would ideally like to have the same number of positive and negative examples so that the model does not learn to predict only based on the data distribution. Therefore, we needed to make sure we have a very close number of partial sentences marked for splitting and those that should not be split. We approached this problem similarly to training

the finetuned NMT model. We took at most three positive and three negative samples from each sentence with particular lookahead.

With positive samples, the algorithm was more straightforward. If a sentence had more than four splits, then we randomly chose three of them. If less than four, we use all their segments except the last one. Then we took all those partial sentences and added a particular number of lookahead words — if there were enough additional words. If not, we would throw the sample away.

For negative samples, we also took at most three random segments, but additionally, we needed to select one random position inside of them to represent the negative sample – except for the last word that would represent a positive sample. And again, we would skip any partial sentences that would not have enough words for a particular lookahead. Moreover, we would also skip a sample if the randomly chosen segment had only one word.

This way, we totally obtained 16157 sentences for the train set, 1616 sentences for the dev set, and 3928 sentences for the test set.

Classifiers architecture

We opted for the default Transformer-based flair model *TextClassifier* for binary classification using *distilbert-base-uncased* embeddings [Sanh et al., 2019]. We used the same architecture and settings for all split types and lookaheads. The learning rate was set to 5.0e-5, mini-batch size to 4, and the training took at most ten epochs.

The binary classifiers and NMT models were trained within the *Google Colab* environment using the *Tesla P100-PCIE-16GB* graphics card. Links to all training logs are part of the source code attached to the thesis.

5. Results

Finally, in this chapter, we will show our experiments' results. We will start with observations from the analysis of our annotated dataset, continue with characteristics of our splitting methods and end with the comparison of SLTev results.

5.1 Analysis of the golden dataset

Having collected data from all annotators, we first analyzed how long and how often annotators split sentences, what kind of words were usually split up, and how these numbers differ among individual annotators. Based on those findings, we then concretely described how we constructed the splitting methods based on phrase chunking and POS tags.

From 86017 possible places where the split symbols could be placed, annotators decided to split sentences in 20692 cases. The following two histograms show the distributions of segment lengths and segment counts in the annotations.[1]

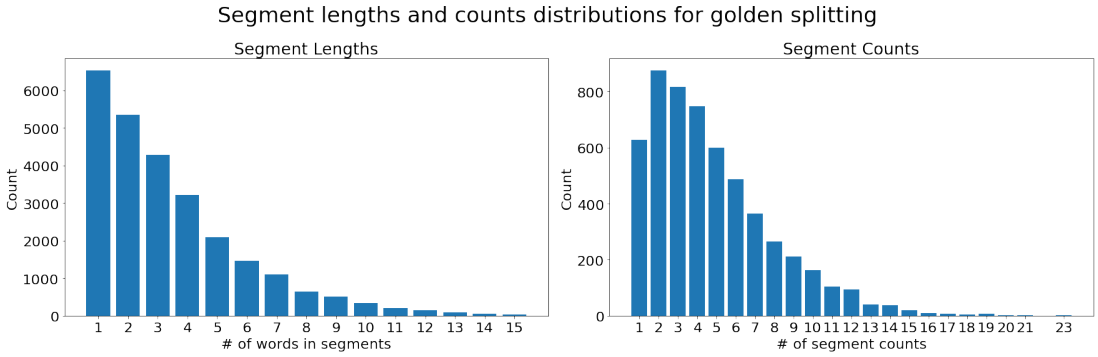

Figure 5.1: Distribution of segment lengths and counts for golden splitting method applied to the whole ESIC dataset.

From Figure 5.1, we can see that annotators split sentences the most frequently into two segments, while the segments have the most frequently only one word. At first, these statistics may seem not legitimate because one would expect that with so many segments of one-word length, more sentences should be split into a higher number of segments. Therefore we plot the distribution of the number of words in sentences for potential clarification (see Figure 5.2).

Nevertheless, the sentence length distribution looked according to our expectations. The only interesting aspect was the two peaks in two-word and four-word sentences, mainly caused by the frequent usage of phrases *"Thank you."* and *"Thank you so much."* in our dataset.

[1] In our experiments, we may switch from working with segments to working with splitting symbols. When working with splitting symbols, we excluded the implicit splitting symbols at the end of every sentence from our statstics

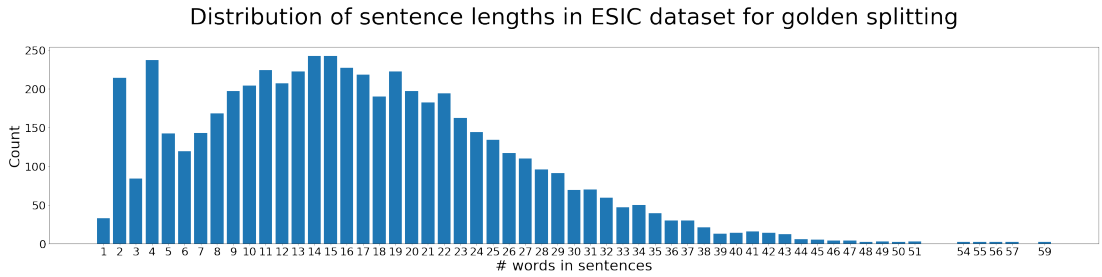

Figure 5.2: Distribution of sentence lengths in the ESIC dataset.

And so, a skewed sentence length distribution was not the cause. The explanation is different. Although there is a lower number of sentences split into, e.g., two segments than six segments, the total contribution to the length distribution is more significant for the six-segment sentences. Because when multiplying the number of segments with the number of sentences they are part of, one gets a more significant number for sentences segmented into multiple splits. And these are exactly the sentences that contribute to the length distribution with many segments of small lengths. Now we will describe our analysis of the possible criteria that annotators may have used for splitting.

Looking at single words was insufficient because the further context might be at least as important as the words right next to the splitting symbol. Therefore, to get a better perspective on annotators' splitting behavior, we analyzed whether the splits were made inside or outside of the phrase chunks and what were the POS tags of words on the beginnings and ends of segments (see Figure 5.3).

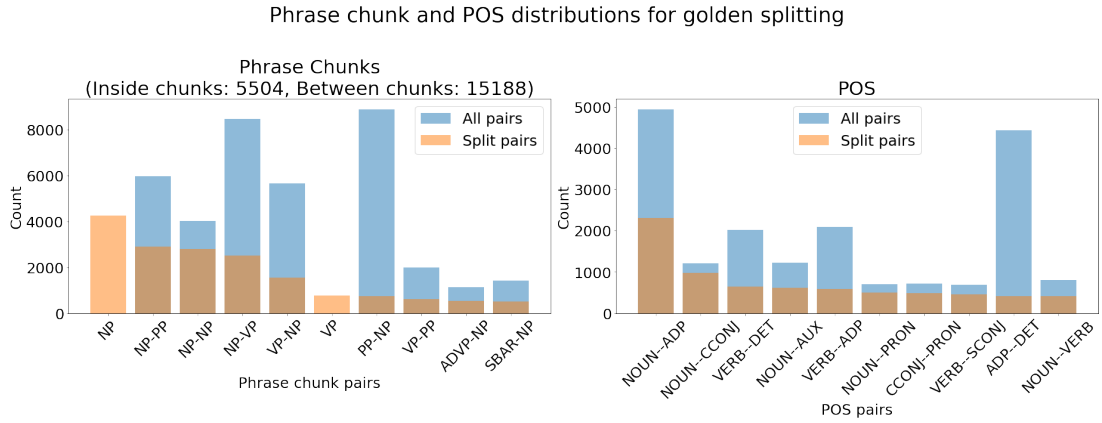

Figure 5.3: Ten most frequently split phrase chunk and POS pairs for the golden splitting.

We especially consider phrase chunks as an essential factor for splitting as they naturally divide sentences into meaningful chunks that bear meaning on their own and thus are easy to read at once – fulfilling the condition for promising interpretation segments [Karakanta et al., 2021].

Out of 20692 splitting marks that the annotators made, only 5504 were placed inside phrase chunks. And from those, more than 4000 were placed inside noun phrases – we suspect that many of those splits inside *NPs* can be attributed to

coordination which might or might not be split by the phrase chunking model. Should we consider annotators as a reliable source of perfect segmentation, this observation significantly reduces the possibilities of where the segmentation for frequent and reliable interpretation can occur.

When looking at individual ratios of pairs that were split and their total counts (pair split ratio),[2] we can see that apart from *NP-NP* pair, no other pair of phrase chunks with a higher number of splits has the ratio particularly high. This means that even though phrase chunks might be good indicators for potential places to split, there is not a set of phrase chunk pairs that would guarantee the split occurrence. Nevertheless, among the most ten frequent phrase chunks, there are only two of them that do not involve noun phrases.

Graph of POS tag pairs confirmed the previously stated observation about nouns' importance for splitting when five out of ten most often split POS pairs contained nouns. Needless to say, nouns are by far the most frequent POS, so their high counts are not really unexpected. Nevertheless, we still believe that spaces around certain nouns might be a good potential place for splitting since, for example, obviously inconvenient pair *ADP-NOUN* was not covered among the most frequent ones.

5.1.1 Rules for phrase chunk and POS splitting methods

After the previous observations, we wanted to try to formulate splitting methods that would always split between certain pairs of phrase chunks or POS tags. Therefore, to obtain the right phrase chunk/POS pairs for splitting, we sorted the pairs according to the pair split ratio (see Figure 5.4) and incorporated into the splitting rules for those pairs with at least 100 splits and whose ratio surpassed 50 %. We then summarized the rules in 5.1.

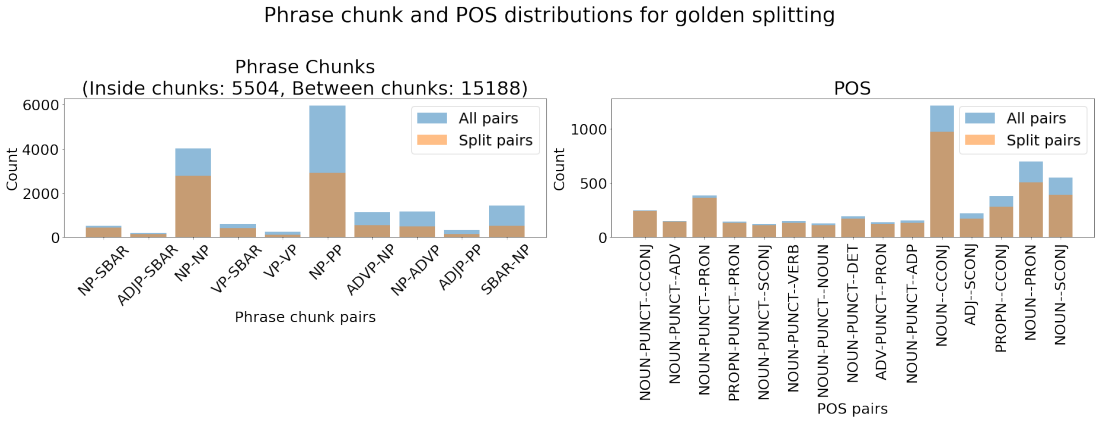

Figure 5.4: Graphs of phrase chunk and POS pairs with the highest pair split ratio.

To get better notion about exactly what types of words the rules are making splits, we put together examples of representatives sentences for most of the rules in Table A.3 and Table A.4.

[2]We did not show the ratio for inside splits because it would be too misleading to present all spaces inside a particular phrase chunk as the total count.

	Chunk phrase rules
Between	(NP, SBAR), (NP, NP), (VP,VP), (ADVP, NP), (ADJP,SBAR), (VP,SBAR), (NP,PP)

	POS rules
Between	(NOUN,CCONJ), (ADJ,SCONJ), (PROPN,CCONJ), (NOUN,PRON), (NOUN,SCONJ), (CCONJ,DET), (CCONJ,PRON), (CCONJ,NOUN-PUNCT), (VERB,SCONJ), (CCONJ,VERB), (VERB,CCONJ), (CCONJ,ADP), (CCONJ,ADV), (CCONJ,NOUN), (PROPN,VERB), (NOUN,PART), (NOUN,VERB), (PROPN,AUX), (NOUN,AUX), (PROPN,ADP), (NOUN,ADP)
After	PUNCT

Table 5.1: Table showing segmentation rules for the phrase chunk and POS splitting that we derived from the annotation data.

5.1.2 Differences between individual annotators

With seven different annotators working on the dataset, we also wanted to determine whether there were some differences among their splitting strategies – in terms of segment lengths, counts, chunk phrases, and POS tags.

This time, we started with the analysis of phrase chunk and POS tags, and generally, we did not observe any significant differences. All annotators had very similar distributions of the most frequent split pairs (both phrase chunks and POS tags); even the inside/between ratio stayed very close to 75%. To keep the subgraphs visibile, we needed to divide them into two following figures (see Figures 5.5, 5.6).

As for segment lengths and counts (see Figure 5.7), the most visible difference was between *Annotators* 4, 5, and the rest. *Annotators* 4 and 5 were the only ones that did not have a typical descending trend in the segment length distribution. This fact also affected their segment count distributions with a relatively small number of sentences split into more than six segments. *Annotator* 4 was also the one that has the lowest ratio of made splits and the total number of annotated sentences – 2338 split symbols in 1077 sentences.

Despite these small differences, we can still assume that all annotators understood the instructions and annotated with similar rules in mind. Moreover, some slight differences might even be caused by the distribution of sentences assigned to a particular annotator.

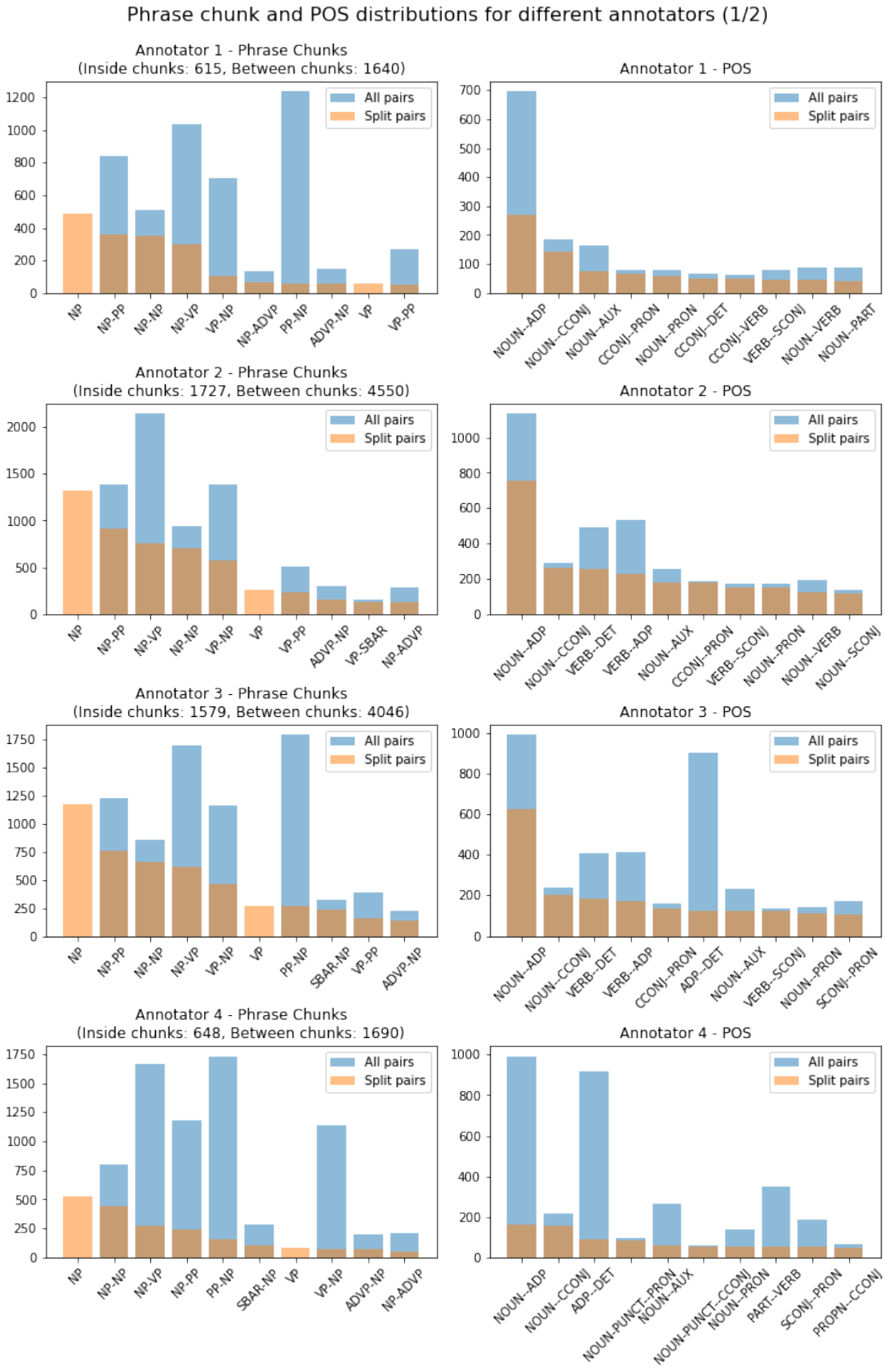

Figure 5.5: Ten most frequent phrase chunk and POS pairs split by individual annotators (1/2).

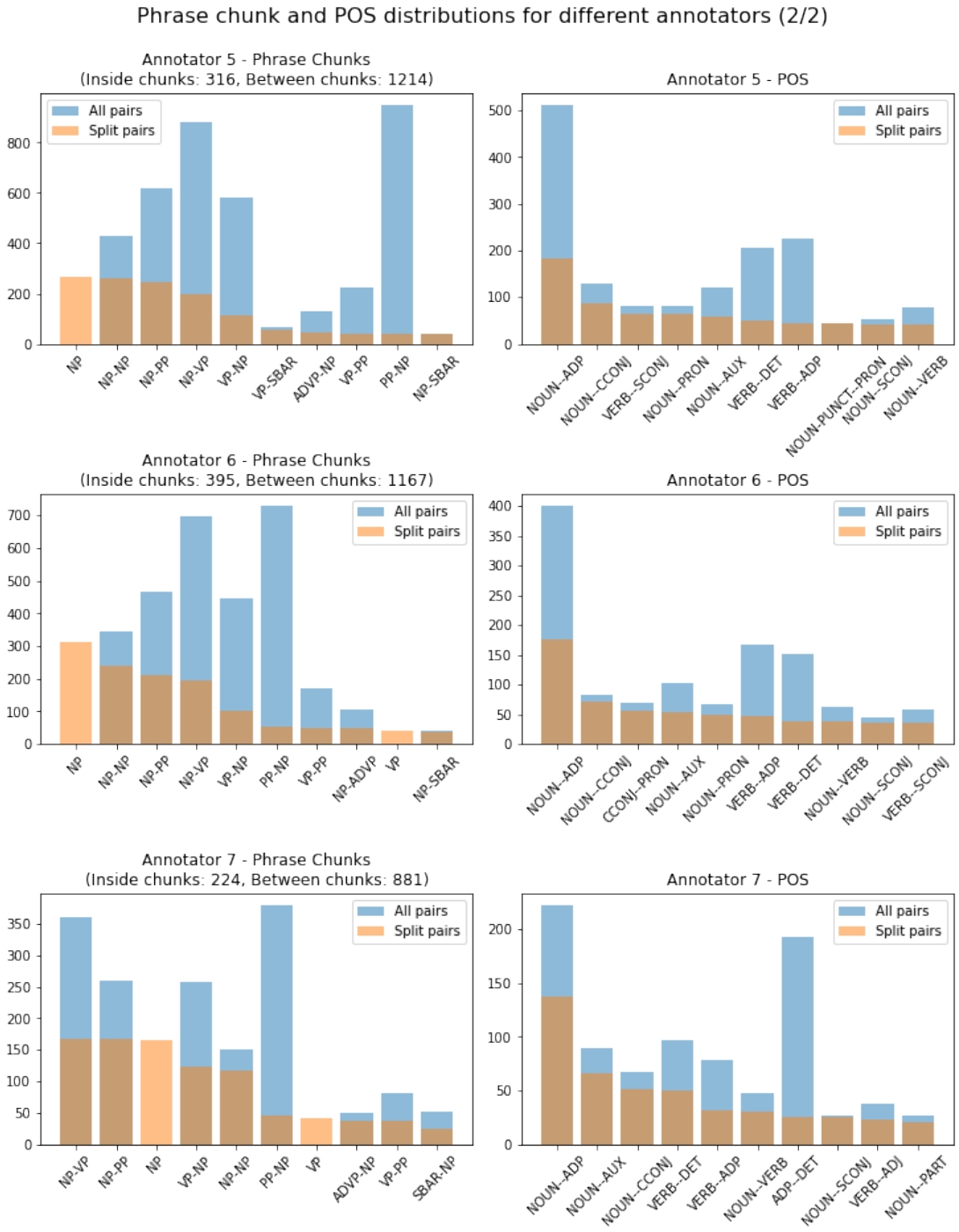

Figure 5.6: Ten most frequent phrase chunk and POS pairs split by individual annotators (2/2).

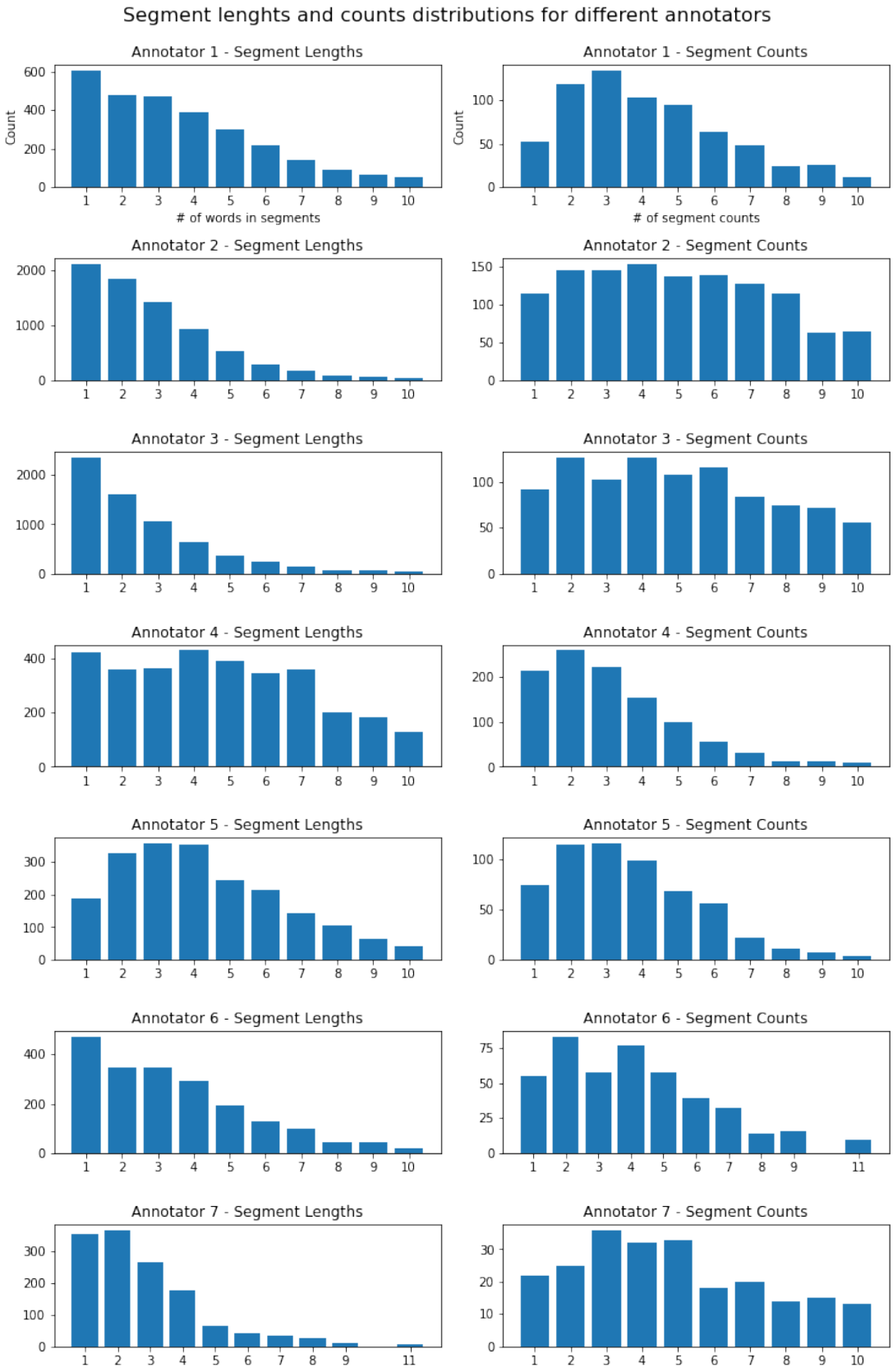

Figure 5.7: Distributions of segment lengths and counts for different annotators.

5.2 Analysis of splitting methods

Similarly to golden splits, we analyzed the other splitting methods as well. As for the total number of splits (see Table 5.2), annotators totally placed 20692 splitting symbols. Considering their splitting as optimal, splitting methods with more splits should generally be regarded as more aggressive – trading lower latency for worse translation quality. And on the other hand, methods with a lower number of splits should be characterized as more patient for a reasonable amount of context.

From all splitting methods, only the alignment-based splitting made more splits than annotators. On the contrary, the punctuation splitting had the lowest number of placed splitting symbols. Looking at the two translation splitting methods, the higher number of splits was made by the NMT model finetuned on the partial sentences. This observation supports the assumption that the finetuning had a positive effect on the model because there were more partial translations that constituted a prefix of the complete sentence translation.

Splitting method	**# of splitting symbols**
Punctuation	4 269
Phrase chunk	12 654
Baseline translation	16 061
Silence	17 485
POS	20 300
Finetuned translation	20 384
Golden	20 692
Alignment fast-align	45 707
Alignment awesome-align	48 604
Every word	86 017

Table 5.2: Table summarizing numbers of splitting symbols placed by each segmentation method.

We also checked the ratio of splits inside or between phrase chunks (see Table 5.3). In the dataset, there were 86017 possible places to place the splitting character, out of which 40663 were inside phrase chunks and 45354 between them. Naturally, the chunk splitting method had the highest share of splits on the edges of phrase chunks, followed by punctuation and POS segmentation. Both alignment methods apparently did not respect the phrase chunks when splitting and split more in favor of "inside splits" than even the every word splitting.

Splitting method	# of inside splits	# of between splits
Punctuation	980	3 289
Phrase chunk	0	12 654
Baseline translation	5 797	10 264
Silence	7 246	10 239
POS	5 336	14 964
Finetuned translation	7618	12 766
Golden	5 504	15 188
Alignment fast-align	22 529	23 178
Alignment awesome-align	25 651	22 953
Every word	40 663	45 354

Table 5.3: Table showing counts of splits by each segmentation method that were placed inside a phrase chunk or between two different phrase chunks.

When analyzing splitting methods' results based on phrase chunk pairs, we noticed that all methods split very frequently inside or after noun phrases, very similarly to our golden data. However, the order of the ten most frequent phrase chunk pairs differed for each splitting method. Looking at the same graphs but for POS tags similarly revealed high numbers of splits for nouns and generally different distributions.

As for segment lengths and counts analysis, the graphs followed the expected trend. The more segments of lower lengths, the more evenly distributed segment counts. Furthermore, we also noticed that splitting based on phrase chunks, in particular, had the lowest number of one-word segments to two-word and three-word segments.

The detailed graphs can be viewed in the Appendix (see Figure A.1, Figure A.2, Figure A.3) – as well as an example of splitting differences when all splitting methods were applied to the same sentence (see Table A.5).

5.2.1 Comparison of splitting method in pairs

So far, we have seen how splitting methods compare to each other in terms of segment lengths, counts and how they respect or ignore different phrase chunks and POS pairs. From these pieces of information, we understood how the methods behave, but still, it did not really reveal whether the methods split at the same or different places.

To find out, we compared the methods to each other, counted the number of splits they shared, and converted it to percentages as a ratio of common splitting symbols to the union of all their placed splitting symbols. The results are presented in the following graph (see Figure 5.8).

Each cell in the heatmap represents the ratio of splits of a method on the y-axis that are shared with the method on the x-axis. Because this relationship is not symmetrical, the value of the cell in the coordinates (x, y) is not generally equal to the value of a cell (y, x). We also ordered the methods according to the number of splits they made, so the methods that split the least are on top and left, respectively.

Apart from the diagonal, the only cell with 100% agreement is at the intersection of punctuation and POS splitting methods. A very high agreement is also

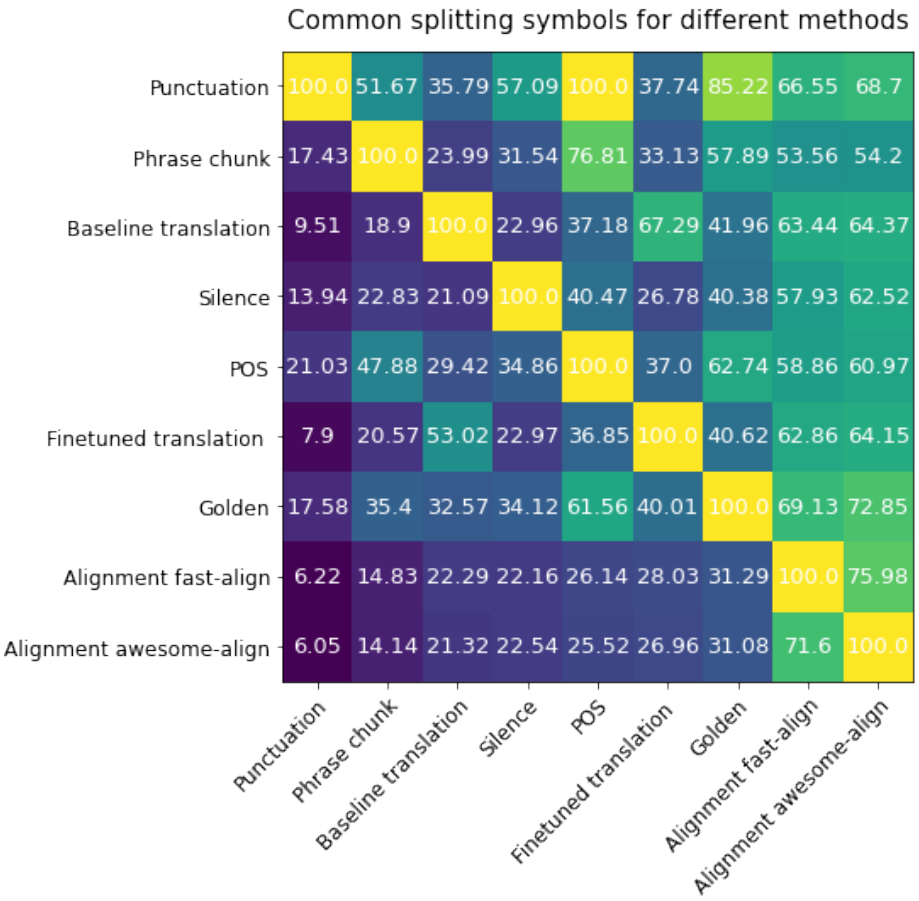

Figure 5.8: Heatmap displaying common splitting symbols for all pairs of splitting methods used on the whole ESIC dataset.

between the punctuation method and golden data – approving that punctuation characters are good indicators for sound splits. The only time splitting after punctuation might cause trouble is when the commas connect several adjectives, and the noun that the adjectives are bounded to is still not known – because otherwise, one would need to guess the correct adjectives' ending. That, however, is a special case for interpreting from English to Czech that other language pairs do not need to share.

Splitting based on our phrase chunk rules had the most common splits with the POS splitting but not vice versa. That agrees with our assumption that the POS splitting is more granular – allowing splitting in more subcases.

Moving on to the translation splitting methods, we can see that the baseline version had the most common splits with the finetuned one. But still, the baseline version is not a real subset of the finetuned version because the finetuned translation model did not register almost a third of all the baseline translation splits.

The last methods we wanted to mention in this regard are the alignment methods. Because their frequency of splitting was significantly the highest, it is no surprise that other methods always share at least half of the splits with them.

Theory of granular and true unique splits

If we flipped the values in the previous Figure 5.8 – taking the remainder to 100%, we would get the percentages of splits that are unique to one of the methods. However, having that number does not give us any additional information, so we came up with two categories of unique splits that could reveal something more. We called them the *granular unique splits* and *true unique splits*. Let us define them more rigorously.

Let two splitting methods A and B split a sentence S with splitting symbols $A_S = \{a_i, a_j, ...\}$ and $B_S = \{b_x, b_y, ...\}$ where a_i denotes a split after the i-th word of S made by the splitting method A – the same notion applies for the method B as well.

Then, we will call a splitting symbol $a_i \in A_S$ a *granular unique* symbol of the method A if $b_i \notin B_S$ and there exist two splitting symbols after p-th and q-th word of S such that $a_p, a_q \in A_S$ and $b_p, b_q \in B_S$, while $p < i < q$ and there is no splitting symbol $b_m \in B$ such that $p < m < q$.

If a_i is a unique splitting symbol of the method A, that means $b_i \notin B_S$, and the previous definition does not apply, we will call the splitting symbol a_i a *true unique* symbol of the method A.

Explained more simply, a splitting symbol a_i of the method A is called *granular unique* if it is located inside two "border" splitting symbols that are shared between both splitting methods ($a_p, a_q \in A_s; b_p, b_q \in B_s$), and additionally, there is no splitting symbol $b_m \in B_s$ that is also located inside the same pair of "border" splitting symbols. To clarify the concept even more, an example of the classification of unique symbols can be seen in Figure 5.9.

| Is this a position, | or | are these anomalies, | discussed | in | Council? |
| Is this | a | position, | or are these | anomalies, | discussed | in | Council? |

| A first-reading position | was adopted | by European Parliament | on 14 November | last year. |
| A first-reading position | was | adopted | by European Parliament on | 14 | November | last | year. |

Figure 5.9: Examples of two sentences, split by two different splitting methods, with true unique splitting symbols in red, granular unique splitting symbols in green and common splitting symbols in grey color.

To correctly classify the splitting symbols at the beginnings and ends of the sentences, it is necessary to place implicit splitting symbols before the first word and after the last word of every sentence. These implicit symbols, however, should never be covered in statistics. The relation of unique granular symbols is again not symmetric and can only be defined when comparing two splitting methods against each other.

Unique split analysis

Having unique symbols split into those two categories, we can more easily find out if one splitting method is a subset of another – that means they share the splits, but one method is more aggressive and splits even those segments that the other method kept as a whole.

For the visualization, we used the same approach as we did with the common splits. We counted the granular unique splits across the whole ESIC dataset for each pair of splitting methods and then displayed it within a heatmap as a ratio of granular unique splits to all unique splits (see Figure 5.10).

The value in the coordinates (x, y) then expresses the percentual share of granular unique splits of the splitting method y when compared with the method x.

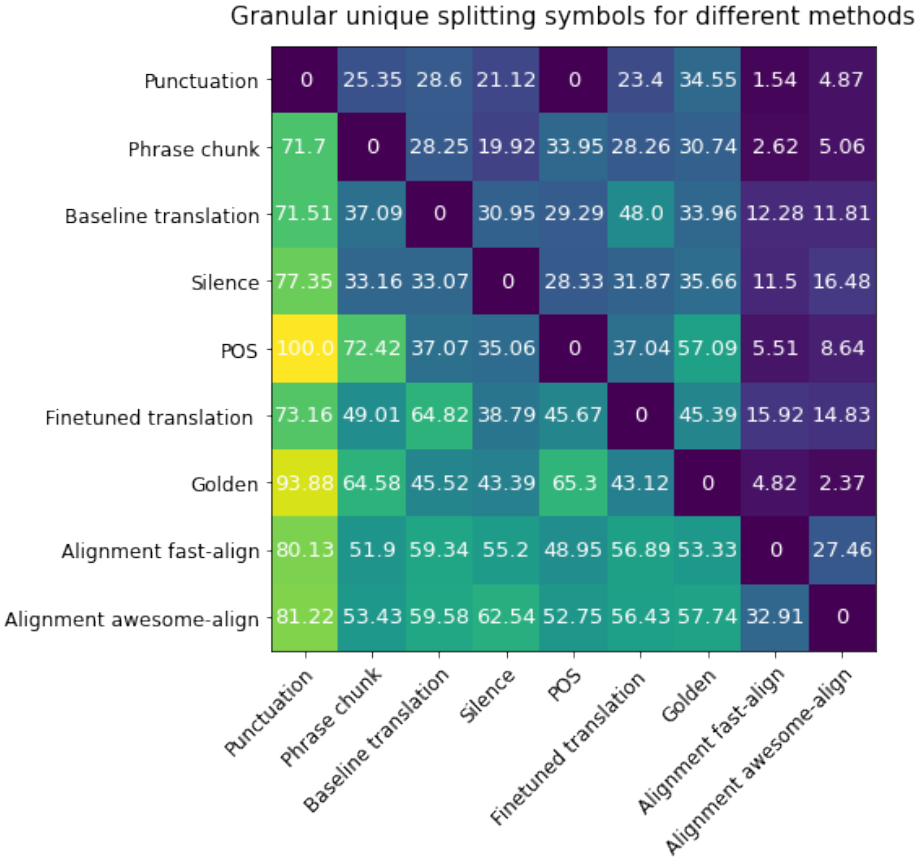

Figure 5.10: Heatmap displaying granular unique splits for all pairs of splitting methods used on the whole ESIC dataset.

The methods were once again ordered by the total number of splits. Therefore, there is generally a bigger potential to see a more significant ratio of granular splits in the left lower triangle under the diagonal. Because if a method splits less frequently, then there is a smaller chance that a particular sentence will have a unique split and share the necessary border splits with a more frequently splitting method. That was proved by the fact that there is only one cell with above 50% value on the right side.

According to our expectations, the most significant proportion of granular unique splits has the punctuation method column. Furthermore, punctuation splits achieve a 100% score with the POS method since the POS method covers all of its splits (see Figure 5.8). Golden data have a very high proportion with the punctuation method as well.

Other than that, a higher proportion of granular unique splits is present with the pair of POS and phrase chunking methods; and between the translation

splittings. That is a good showcase that the more similar methods, the more granular unique splits there should be. The other method pairs did not score that high, including even the alignment splitting, where the probability of granular unique splits is much higher since they split significantly more often.

5.2.2 More detailed view on the alignment splitting

Since alignment splitting methods have been quite popular among other researchers, we decided to look at how much the splitting will change depending on the chosen alignment tool, translation, and language.

For our analysis, we added two additional popular alignment tools: *eflomal* [Östling and Tiedemann, 2016] and *sim-align* [Jalili Sabet et al., 2020]. Similarly to fast-align, we first trained eflomal on the EUROPARL dataset [Koehn, 2005] and symmetrized its result with the same *grow-diag-final* heuristic. For sim-align, we used the default settings and gathered results from the *mwmf* matching method.[3]

For additional translations, we used API for the LINDAT translation model [Popel et al., 2020] and translated our original English version of ESIC into Czech and German. We opted for an existing MT model to make the conditions for both Czech and German equal – not risking training one model better than the other. Let us first use these different translations for the comparison of the alignment splitting in terms of split counts.

	LINDAT Czech	**Annotation Czech**	**LINDAT German**
eflomal	51 169	49 010	41 127
awesome-align	50 245	48 504	37 209
fast-align	47 789	45 707	40 360
sim-align	34 928	31 654	25 803

Table 5.4: Table showing number of splits made by the alignment splitting methods with different translation tools and target languages.

Only by looking at Table 5.4, we can notice differences among the alignment tools. Even for the same reference translation, the two alignment methods can differ up to 17356 splits in the case of eflomal and sim-align.

With English-Czech alignments, we would expect our golden translations to produce more splits because the annotators were instructed to write down optimal translations that will produce the most optimal splits – i.e., no other translations should produce more granular splits. However, according to this data, it is the other way around. On average, alignment methods working with LINDAT translations made 2389 more splits than the same methods using our translations from the annotation. We assume this fact is mainly caused by the imperfection of the alignment splitting method and because MT models may output more monotone and literal translations, that can be aligned more easily. Thus, we are still convinced our golden translations should be better suited for faster interpreting.

Comparing the split counts for Czech and German translations, the alignment methods aligning the German translations made on average 9908 splits less than

[3]We summarized the exact procedures within the *README.txt* in the attached source code.

the same alignment methods working with Czech translations – suggesting that German sentences may contain more longer-distance relations among the words than their Czech equivalents. Although we do not consider alignment splitting methods very reliable, we regard this difference as significant enough to confirm the hypothesis that a target language influences the size of the optimal sentence splits for interpretation.

The last thing we noticed while comparing alignment methods' split counts was that the alignment methods kept the same order in terms of the number of splits across different target translations. We spotted the only exception for the German translations, where awesome-align and fast-align flipped their positions.

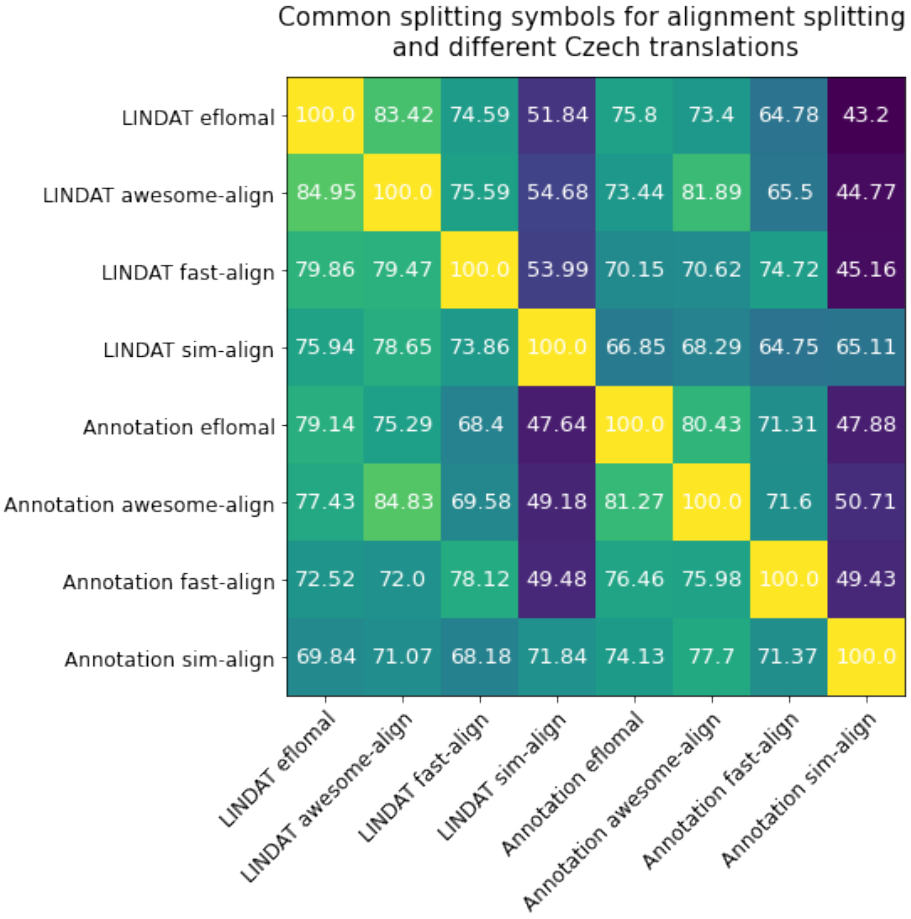

Figure 5.11: Heatmap showing percentual ratio of common splits among alignment methods with different Czech reference translations – automatically translated by LINDAT and manually by annotators.

Displaying common splits of alignment methods for golden annotators' and LINDAT translations (see Figure 5.11) divided our heatmap into four smaller squares. The most interesting sections of the heatmap are in the upper-right and lower-left corners because there are the intersections of the same alignment tools for different reference translations. Except for the sim-align, the highest agreement in both of these squares are between the same alignment methods on the diagonal. From those results, sim-align splitting appears as the most different from the other.

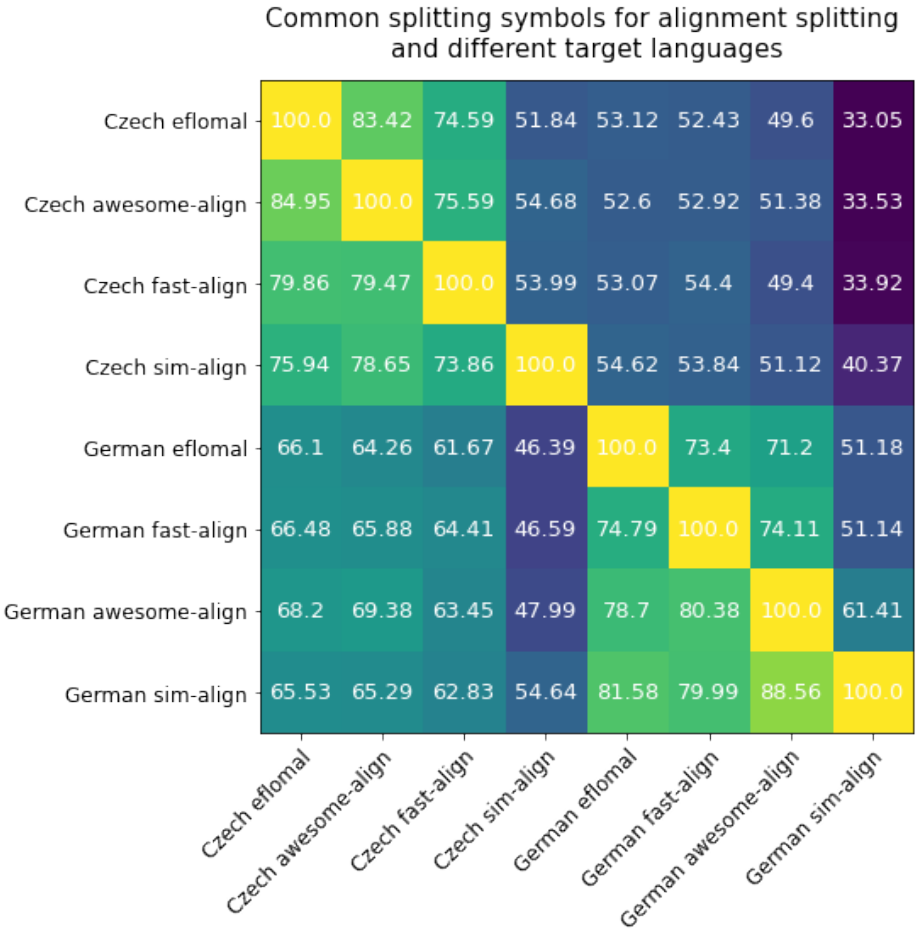

Figure 5.12: Heatmap showing percentual ratio of common splits among alignment methods with different reference languages – Czech and German.

On the other hand, when comparing Czech and German reference translations (see Figure 5.12), we did not observe previously mentioned high numbers on the diagonals. Nevertheless, we can see here that there are more common splits among alignment methods applied to the same target language than when comparing across languages.

With this section, we conclude our comparison of splitting methods. Even though it is rather complicated to get a better understanding of splitting when looking at larger data, we tried to visualize it in more views to deeply understand how the splitting methods work and differ from each other.

5.3 From offline to online methods

As mentioned in Section 4.9, some segmentation methods can work very well with complete sentences, but at the same time, they cannot be applied to partial sentences. Therefore in this section, we will explain how we adjusted those methods to work in the online setting as well.

5.3.1 Phrase chunking and POS splitting

We start with the two methods that did not need bigger changes – phrase chunk and POS splitting. Let us work with the assumption that a sentence split according to our rules with phrase chunks or POS tags predicted with the knowledge of a complete sentence can be regarded as golden data. Then, we want to determine how accurate the splitting will be when working only with partial sentences.

From this point, we will call the methods processing sentences word-by-word with a lookahead as *sequential methods* and the methods predicting splits with the whole sentences as *oracle methods*.

Sequential versions of phrase chunk and POS splitting need at least one-word lookahead (see Section 4.9). Because of that, we analyzed the sequential methods starting with the lookahead of one word up to five words which seemed to us as a still reasonable delay for real-time interpreting.

During the segmentation, when processing sentences with lookaheads, we assumed we could know if we reached the end of a sentence. Therefore for a lookahead of n words, the last segment of each sentence will contain at least $n+1$ words. Because when the end of a sentence is reached with a lookahead, it is faster to output the rest of the sentence at once than trying to split it into more segments. That is why we did not compare our sequential methods with the oracle versions with implicit lookahead of zero words but with their adjusted lookahead versions that copy the same behavior when reaching the end of sentences. Otherwise, the sequential methods might appear less accurate than they really are.

	Prescision	**Recall**	**Precision(0)**	**Recall(0)**
		Phrase chunk		
Seq 1	96.03	74.17	96.03	73.19
Seq 2	97.36	96.08	97.36	88.48
Seq 3	97.54	97.25	97.54	82.18
Seq 4	98.15	97.78	98.15	76.79
Seq 5	98.58	98.3	98.58	71.69
		POS		
Seq 1	98.66	84.97	98.66	83.59
Seq 2	99.34	96.72	99.34	90.16
Seq 3	99.65	98.84	99.65	85.4
Seq 4	99.8	99.22	99.8	79.56
Seq 5	99.89	99.42	99.89	73.78

Table 5.5: Table showing values of precision and recall for sequential versions of phrase chunk and POS methods. The first two columns contain scores obtained by comparing the sequential methods with their oracle counterparts with the same lookahead; the last two columns have the values of precision and recall coming from the comparison of the sequential methods with a corresponding oracle method with zero lookahead.

Finally, we want to emphasize that when evaluating phrase chunk/POS splitting, we do not measure how precise their tagging is but how accurate the splits arising from the tagging are. When a tagger works with incomplete sentences, it can predict a different tag than when knowing the whole sentence. But if the

wrong tag splits or does not split the partial sentence the same way as the correct tag would, then it makes no difference for our purposes, and we even cannot spot that.

In Table 5.5, we can see that even though for phrase chunk one-word lookahead the recall is 74, 17%, starting from two-word lookahead, the recall jumped over 96%. With the precision, the sequential methods got above 96% even for one word lookahead. With POS splittings, the data are very similar. Achieving almost perfect precision with all sequential methods and again from two-word lookahead, getting the recall over 96%.

We are aware, that the choice of particular splitting criteria may cause these results, but generally, with a lookahead of two words, there is very little difference between sequential and oracle methods. Thus, we can conclude that the sequential methods of phrase chunk and POS splitting can be very effectively used in online settings, almost copying their oracle counterparts.

5.3.2 Training machine learning models

For the other methods that cannot be directly used for sequential processing, we needed to train binary classifiers that learned to split the partial sentences from the oracle splits. We used the same range of lookaheads as in the previous section, and trained a separate classifier for each splitting method with a particular lookahead.

Let us first look at the statistics we collected from the model training to see how the models performed. For each model, we got the F-score for both classes – SPLIT and NO-SPLIT.[4] F-score is a harmonic mean of precision and recall computed as

$$F = 2 \times \frac{precision \times recall}{precision + recall} \tag{5.1}$$

Naïve evaluation

Figure 5.13 shows that the most accurate classifiers were trained when predicting golden splits from our annotators. Translation and alignment models performed a bit worse, with alignment models slightly better at classifying SPLIT classes and translation models at NO-SPLIT classes.

When looking at the particular models from one method, we can observe a trend that models with lower lookahead scored better than models looking further into a sentence. We expected this correlation to be the other way around because the more lookahead and context a model has, the easier it should be to decide about the split. In our opinion, the main cause of the problem was the small amount of data. Because with bigger lookaheads, the end of partial sentences can cover even multiple new segments that can potentially confuse the classifier about what it should really predict. Furthermore, the prediction might be difficult because, despite the constant lookahead across the whole datasets, the intended breaking points were not marked explicitly.

[4]To get the F-score for the whole dataset, we would only need to take an average of F-scores for individual classes.

We are not sure why the baseline translation with the four-word lookahead happened to be an outlier. However, it is very plausible that it was caused by the mentioned combination of insufficient data and a larger lookahead.

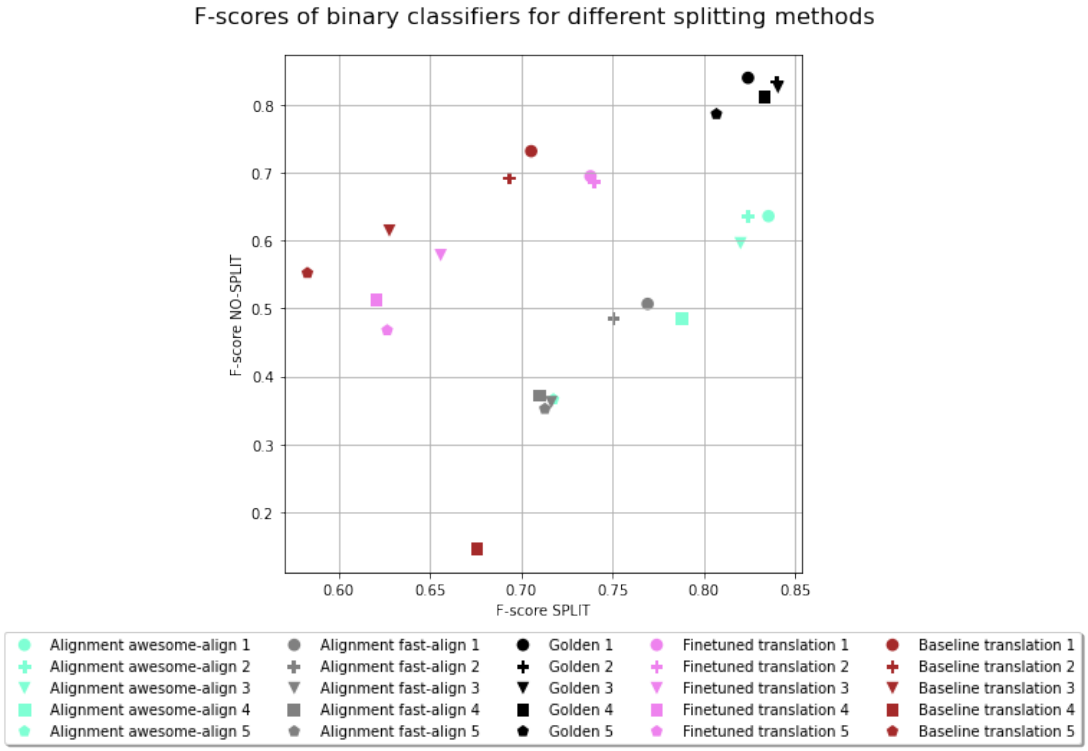

Figure 5.13: Summary of F-scores for SPLIT and NO-SPLIT classes of all of our trained binary classifiers trained with different lookaheads.

Realistic evaluation

A constructed test set from 1000 split sentences gave us a rough estimation of how the trained models perform. Nevertheless, the reality might be different. To not overfit, we used only up to three samples of each sentence in all sets (train, dev, test). But when interpreting, a classifier needs to decide about a particular partial sentence after every word in the sentence (subtracting the lookahead). So even though the results from training looked very promising, we rather split all test sentences with the sequential methods and compared the splits with their oracle versions. This is the only way to see how accurate our sequential splitting will be in real interpretation settings.

With this adjustment, the results are very different (see Table 5.6). While keeping the recall relatively high, the precision dropped very significantly – especially for the translation methods, where the precision fell below 50%. That means that the sequential methods made far more splits than they were expected to make. We believe this problem could be fixed with more data because, as we wrote previously, we took only up to three samples from each sentence, which might very easily not cover all partial sentence lengths and contexts sufficiently. Therefore, there is definitely still a need for further exploration that we will discuss in Section 6.3.

	Precision	Recall	Precision(0)	Recall(0)
		Golden splitting		
Seq 1	51.06	85.02	51.06	83.66
Seq 2	51.98	86.21	51.98	81.08
Seq 3	51.17	86.54	51.17	76.05
Seq 4	50.95	85.52	50.95	69.53
Seq 5	48.42	82.56	48.42	61.66
		Baseline translation		
Seq 1	31.44	70.16	31.44	66.68
Seq 2	30.84	70.16	30.84	63.19
Seq 3	25.68	60.91	25.68	51.48
Seq 4	19.08	94.49	19.08	74.77
Seq 5	23.4	58.4	23.4	42.93
		Finetuned translation		
Seq 1	36.48	75.09	36.48	71.37
Seq 2	36.97	75.73	36.97	68.19
Seq 3	32.01	67.38	32.01	57.19
Seq 4	29.18	63.1	29.18	50.16
Seq 5	29.26	66.86	29.26	49.62
		Alignment fast-align		
Seq 1	57.49	77.02	57.49	70.05
Seq 2	57.12	76.65	57.12	64.85
Seq 3	52.98	76.96	52.98	60.65
Seq 4	52.26	75.3	52.26	54.86
Seq 5	50.82	75.18	50.82	50.38
		Alignment awesome-align		
Seq 1	66.1	87.68	66.1	81.46
Seq 2	67.65	85.29	67.65	74.2
Seq 3	65.18	86.54	65.18	70.54
Seq 4	61.39	84.32	61.39	63.83
Seq 5	57.66	75.32	57.66	52.67

Table 5.6: Table showing values of precision and recall for sequential versions of splitting methods to which we needed to train a binary classifier in order to use them in online settings. First two columns contain scores that were obtained by comparing the sequential methods with their oracle counterparts with the same lookahead; and the last two columns have the values of precision and recall coming from the comparison of the sequential methods with a corresponding oracle method with zero lookahead.

The last question is why the translation splits had a worse recall than the rest. We assume that they are hard to predict based on their inner principle. Because for translation splitting, we are trying to teach classifiers to predict when our manually trained and finetuned translation models will output a partial translation that would be a prefix of a complete translation. So it is definitely possible that the rules for splitting under such conditions seemed to a classifier more random than when trying to learn the splitting criteria for golden data and alignment splitting based on less complicated rules.

5.4 SLTev results

Finally, we can gather everything we have built so far and evaluate all splitting methods with SLTev to see the results for both the baseline translation model and its finetuned version when applied on partial sentences. For the SLTev evaluation, we used the same 1000 sentences that we already used as a test set during the evaluation of sequential splitting methods.

As we mentioned in Section 2.4.1, although SLTev does output all three metrics for evaluation of machine interpretation – BLEU score, delay, and flicker, we could not use the BLEU score in our settings. The main issue is that SLTev evaluates the BLEU score only with complete sentences, which means that all our methods scored the same. To use the BLEU score, we would need to abandon our retranslation process and grow the partial translations by segments. Then, in the end, the complete sentences would be different for each method, but the flicker would be zero.

However, we argue that for the evaluation of splitting methods, flicker with the retranslation approach may be an even more representative measure of splits' quality than BLEU when growing translations by segments. Because if the segments are short, then the translation module starts to translate earlier and is more prone to changing the partial translations, thus increasing the flicker. That is why flicker reflects the quality of segments and is a supplementary metric to the delay score that, on the other hand, favors shorter segments over the longer ones.

For completeness, when evaluating BLEU on complete sentences concatenated to two documents, methods with the baseline MT model scored 40.011 and with the finetuned one 40.219. Such a small difference means that even though the finetuned MT model behaved differently than the baseline version when working with partial sentences (see Figure A.5), they do not differ much with complete sentences. Thus, the finetuning had the intended effect.

5.4.1 SLTev requirements

For simultaneous translation evaluation, SLTev needs three files:

- A text file containing golden translations.
- A text file with progressively growing speaker's sentences with a particular display time for each new segment (or rather word).

- And lastly, a text file with the progressively growing interpretation of the original speaker with particular display times.

Preparing the control file with golden translations was straightforward. We only needed to extract golden translations from annotators and concatenate them. With the other two files, we needed to provide also the time information, so it was more complicated.

Because there is information about when each word appears in the ESIC dataset, we could grow the original sentences word by word using data straight from the dataset (see Figure 5.14). Given SLTev's requirements, however, we needed to adjust the times so that all test sentences appear as one long speech (and not several speeches, always starting from the time 0). So whenever we reached the end of one speech, we added the end time of its last word to the start time of the first word of the following sentence.[5] This way, we ensured that each speech's relative shifts were preserved.

```
P 3419 3438 This
P 3419 3450 This is
P 3419 3456 This is a
P 3419 3476 This is a very
P 3419 3515 This is a very simple
P 3419 3564 This is a very simple question,
P 3419 3579 This is a very simple question, I
C 3419 3620 This is a very simple question, I think.
```

Figure 5.14: Example of a progressively growing English sentence from a text file used for SLTev evaluation. The first letter marks whether a certain text is a partial (P) or a complete (C) sentence and the following two timestamps in centiseconds represent the start time of the whole sentence and the end time of the last word.

With the third file, we were required to concatenate the times as well, but we also needed to decide how to set the display time for translations in the first place. Because we did not optimize the splitting methods' running times and efficiency, we did not include the time spent deciding and splitting to the translation times. Thus, we set the display time of each partial translation equal to the end time of the last English word processed before deciding to split, including the potential lookahead (see Figure 5.15).

[5] We also tried to evaluate the splitting methods without this additional process of concatenating the times. And the results of all methods were completely the same in terms of flicker. For the delay, there was a shift for all values. So that the individual values of delay changed, but the relative positions of all the methods stayed the same.

```
P 3438 3419 3438 Tento
P 3450 3419 3450 To je
P 3476 3419 3476 To je velmi
P 3564 3419 3564 To je velmi jednoduchá otázka,
C 3620 3419 3620 Myslím, že je to velmi jednoduchá otázka.
```

Figure 5.15: Example of a progressively growing Czech candidate translation from a text file used for SLTev evaluation. The first letter at each lines denotes whether it is a partial (P) or a complete (C) sentence. The first number is then the display timestamp of the last processed English word (including the lookahead). The last two numbers correspond to start and end times of the partial/complete English sentence that was processed when deciding about the split.

5.4.2 SLTev evaluation

We started the SLTev evaluation with the oracle methods with one-word lookahead. We did not use the zero-word lookahead because this way, we could compare the oracle methods with the sequential ones that need to start with at least one-word lookahead because of the phrase chunk and POS splitting methods.

SLTev results for oracle methods

The first thing that caught our attention in Figure 5.16) was that apart from the whole sentence segmentation, the translation splits also achieved zero flicker when using a particular MT model. That was a good check of the correctness of our implementations.

Except for these two methods, complete sentence and translation splitting, all other methods achieved lower flicker with the finetuned MT model than with the baseline version trained only on complete sentences.

We found it very interesting that doing more splits generally does not guarantee lower delay. For example, both translation methods made more splits than the phrase chunk splitting and still had a more significant delay. The explanation with high probability lies within the type of splits the translation methods make – if we look closely at the data (see Figure A.3) we can see that the translation splitting made many splits of length one. Therefore, we examined the split sentences in a more detailed way and discovered that for translation splitting, it is very common to not do any splits in the first half of the sentence and then do the splits after each word towards the end. Thus, the delay is very small for words towards the end of sentences, but the waiting period is extended for the words at the beginning. We speculate that this phenomenon of initial long chunks is caused by the nature of MT systems that can at the beginning be very unsure about the exact wording, but when a sentence reaches its end, the translations might get more stable. However, it does not necessarily mean that the semantic meaning of initial partial translations is wrong; it can only be expressed differently.

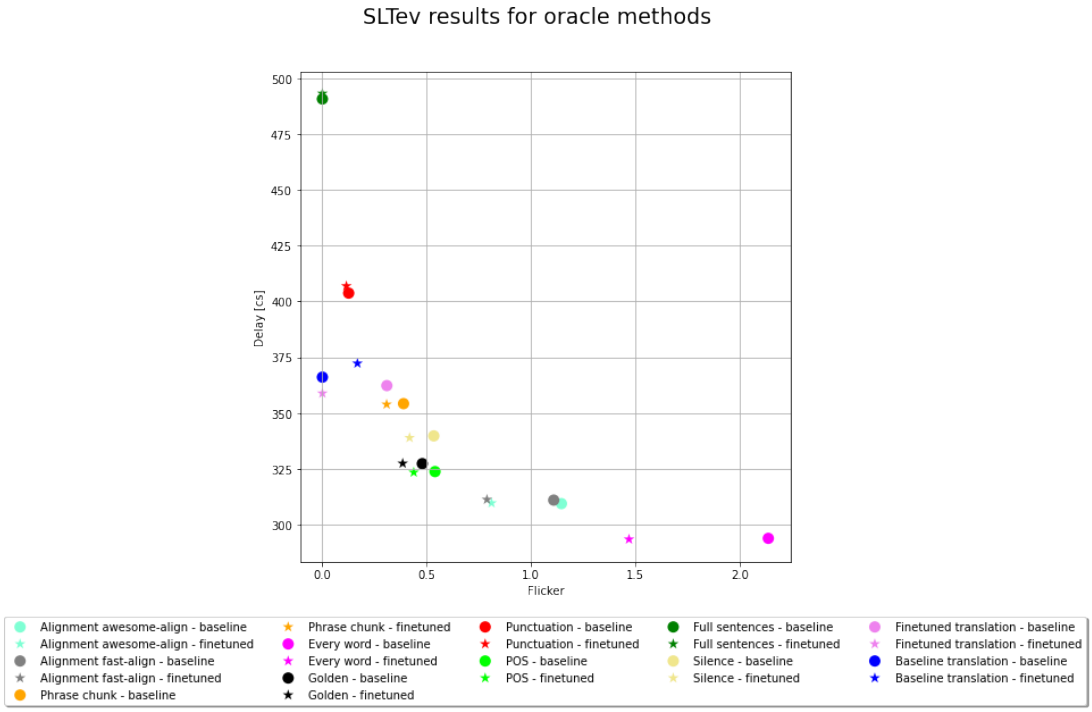

Figure 5.16: SLTev results for all oracle splitting methods with baseline and finetuned translation models.

Looking at the graph from a multi-objective optimization perspective, all points except the silence splitting and particular translation methods make up a Pareto front. This means that no method is better at both delay and flicker than all the other methods (with the same translation model). We consider this finding surprising because we did not try to construct the methods in any special way.

Nevertheless, when evaluating splitting methods with SLTev, the methods may seem more similar to each other than when comparing their common and unique splits (see Figure 5.8, Figure 5.10). That obviously raises an immediate question, what metric should be taken more into account. Does that mean that despite the differences in split positions, the methods tend to have more similar scores of delay and flicker? Or the metrics of delay and flicker are not that precise to pinpoint important differences?

We believe that the methodology of splitting comparison is far from ideal, and adopting more comparison aspects may be one of the solutions. The other one would be enhancing the existing metrics of delay and translation quality scores because, as we, for example, saw in our example of delay calculation (see Section 2.4.1), excluding words that did not appear in the reference from the summation is not ideal. Therefore, developing new SiMT evaluation tools with more advanced delay and translation quality metrics might also significantly help to measure models' performance in a better way.

SLTev results for sequential methods

Continuing the narrative from the previous section, we will now briefly comment on the results of the sequential methods as well – shown in Figures 5.17 and 5.18.

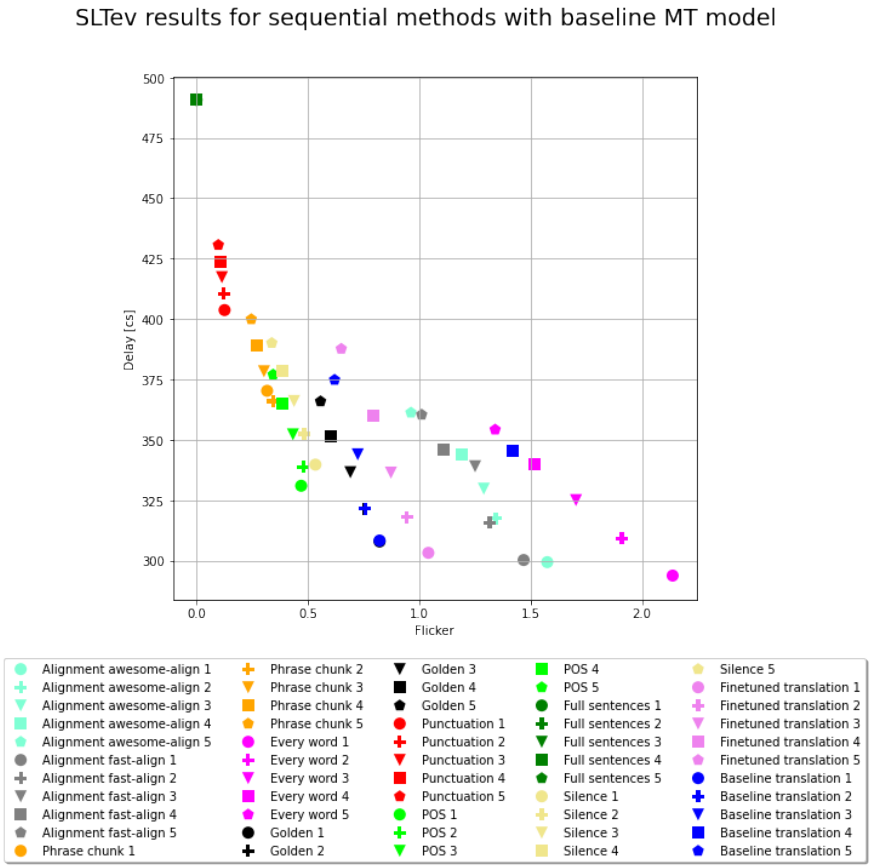

Figure 5.17: SLTev results for all sequential splitting methods used with our baseline translation model.

First of all, it is important to remind ourselves that the sequential versions of control methods, punctuation, and silence splitting are entirely identical to their oracle counterparts, and the difference among their scores for different lookaheads was caused only by the different behavior towards the end of sentences.

Meanwhile, other methods, especially those with trained binary classifiers, can have more irregular score patterns with different lookaheads. For them, we can observe generally higher flicker and lower latency compared to their oracle versions because the sequential versions split more often. We noted the most significant score change with the translation splitting methods, which recorded more than two times higher flicker than their oracle counterparts. Except for the translation splitting, all other methods remained positioned relative to each other in the same way.

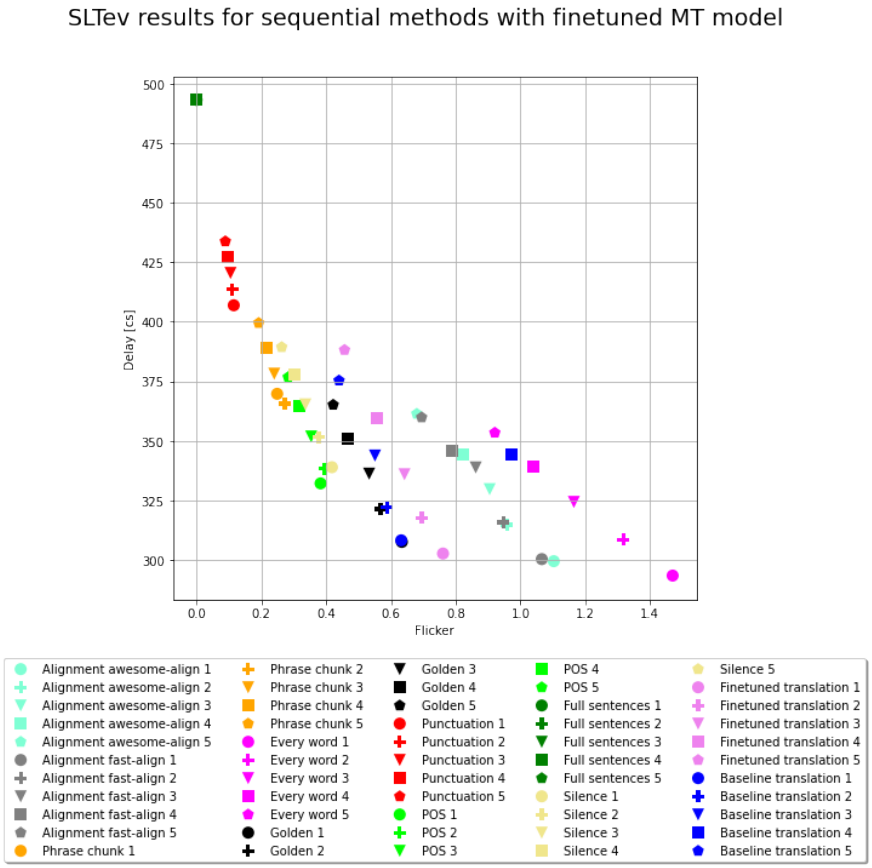

Figure 5.18: SLTev results for all sequential splitting methods used with with our finetuned translation model.

And finally, when comparing the scores from the baseline and finetuned translation, we observed the same shift as with the oracle methods that the methods with finetuned MT model achieved smaller flicker while keeping almost the same delay. The delay theoretically should not change at all. But because the delay is computed for only the words that are present in the candidate and reference translation, it is possible to record some minor changes.

The increased lookahead caused the best flicker improvement for every word splitting and both alignment methods – these were the methods that split the most frequently, and prohibiting the segmentation by the end of sentences did not allow them to place more excessive splitting marks. For other methods, the flicker improvement was not that significant, and we generally can observe that the less splits a method makes, the less flicker improvement the increased lookahead will have.

All methods (except the full sentence splitting) with increased lookahead also recorded increased delay. And again, the biggest change is visible with more aggressively splitting methods.

Generally, when comparing different values of lookahead, it is complicated to determine what values would work the best in the real interpretation setting because, especially with trained binary classifiers, we did not achieve great results in terms of precision and recall. However, we can see that, for example, for phrase chunk and POS splitting, which were able to mimic the oracle versions reliably, some lookaheads appear as really suboptimal (e.g., POS 2 and Phrase chunk 3).

Nevertheless, we can speculate from our results that higher lookaheads might not be that effective in enhancing the translation stability because, especially for less-frequent splitting methods, the flicker improvement is exchanged for a more significant delay increase.

As a result, the Pareto front for sequential methods is made out of versions with one-word lookahead; this would suggest that even though the increased lookahead may stabilize the translation, it would still be a suboptimal solution to some different splitting methods with smaller lookahead.

6. Discussion

To conclude the thesis, in this section, we will summarize our thoughts about simultaneous machine interpretation, list our findings, and suggest ideas for improvements and future work.

6.1 Translation units

Although there is already a good amount of research dealing with the topic of simultaneous machine interpretation, we did not find any article discussing the relationship between human and machine interpreting.

So despite the progress with sophisticated end-to-end architectures, we still do not know whether humans and machines interpret similarly or whether they work with completely different MTUs. In order to find out, we had to define the concept of (minimal) translation units. However, that proved difficult due to still ongoing discussions about what the translation units really are and because everybody defines them according to their needs. Therefore, in Chapter 1, we firstly covered all possible views on TUs to understand the topic better.

From the research, it is clear that for different requirements, there really needs to be multiple definitions of TUs. But despite small variations, such as whether the TUs should be on the speaker's side, interpreter's side, or even both; all TUs' definitions described a similar topic but emphasized its different aspects. We also noticed a visible difference between how research communities of linguists and NLP engineers view the topic of interpretation. For interpreters, interpreting means something completely different than translation because of all the necessary strategies to stay synchronized with a speaker. On the other hand, NLP practitioners often perceive interpretation more as an imperfect translation; therefore, there is no real need to distinguish between simultaneous machine translation and interpretation from their point of view.

Getting back to TUs, the only NLP article elaborating more on their definition we found, was by Zhang et al. [2020], where they defined TUs as meaningful units by characterizing their two important properties:

- Meaningful units need to be short to reduce latency.
- The translation of meaningful units should not be affected by the forthcoming words.

These two characteristics agree with our instructions to the annotators that they should split sentences as often as possible (to preserve short lengths and reduce latency) but only when their partial translation will not change based on the following information.

6.2 Dataset annotation

Our aim with the annotated dataset was thus clear: to connect the two worlds of NLP engineers and linguist theorists and try to teach machines to interpret the same way that an ideal human interpreter would.

Since related work on this topic is limited, we first needed to think about what information we wanted to acquire and how to define the task properly. We finally ended up asking annotators to provide both the splitting annotation and the reference translation to show that the presented splitting is really possible. With this approach, we managed to gather unique data that can serve for further research both to linguists and engineers. At the same time, we hope our instructions will be improved in the future as some of the open questions still need to be resolved.

Most importantly, we would like to see a better clarification of how the annotators should solve the situations when there is a clear space for splitting (the following information does not change the partial translation), but there is a high possibility that in the real settings, when not knowing how a sentence will continue, an interpreter cannot be sure about the translation (e.g., whether the English verb will be positive or negative).

We also did not manage to properly check whether all annotators approached the task the same way. We did evaluate their splitting using our analysis of segment counts, lengths, and phrase chunk/POS pairs, but because we managed to annotate each sentence only once, we could not truly measure the annotators' agreement in terms of splitting and reference translations.

Lastly, it would be beneficial to make a questionnaire for annotators to find out how they felt about the task and their strategies. We tried to collect the feedback along the way but did not make it formal. For example, asking directly how frequently annotators used a machine translation to get the reference translations.

Despite all of the mentioned drawbacks, we believe that we set a good starting position for future research that can not only continue applying the annotation to other datasets but also improve our own annotations by including more annotators or creating the same annotations with different target languages.

6.3 Splitting methods

When we needed to choose the splitting methods for our analysis, we were influenced by the following criteria:

- Because we wanted to test several methods and compare them against each other, we could not choose methods that would take too much time to implement.
- We did not want to choose such methods that would not reveal much about their inner working because of the black-box nature.
- We needed to be able to incorporate the splitting method into our working pipeline.
- We wanted to make the methods as diverse as possible.

For those reasons, we opted rather for less complex but more explanatory methods that can be relatively quickly implemented and reveal how much complexity the optimal segmentation really needs. Nevertheless, we ended up with 11 different methods that we can divide into following categories.

- Control methods: splitting after each word and at the end of sentences served to view the extremes
- Alignment methods: because alignment segmentation is quite popular in the research, we decided to analyze them to see how good and precise they really are, even in terms of different word alignment tools
- Golden annotation: splitting based on our annotation, regarded as the optimal one
- Translation methods: splitting methods copying the process of our annotation but with trained NMT systems on complete and partial sentences
- Punctuation splitting: a relatively simple method that could bring good results with very little effort
- Silence splitting: the only splitting method not working directly with the text but the acoustic information
- Phrase chunk and POS splitting: methods based on known concepts of phrase chunking and part-of-speech tagging with rules derived from the analysis of our annotation

Additionally, when implementing the splitting methods, we also designed a new way to visualize simultaneous translation by combining the word alignment with the acoustic and time information from the dataset. In the future, we would like to release this visualization tool as a standalone Python package that could be easily deployed and used for interpretation analysis.

At the same time, we are aware that there is potentially a lot of space for improvement and additional research that we, unfortunately, due to the time and resource limitations, did not manage to complete.

First of all, during our analysis, we were working with the assumption that our data came from the real settings. However, although we were displaying the partial sentences word by word, the text was manually corrected and did not come straight from an ASR module. That means we did not have to handle the various mistakes that the ASR may make. Therefore, we are not really sure how our methods' performance would change if the input text was not preprocessed.

Secondly, due to the number of methods, we were not able to spend more time on improving the models for sequential versions of splitting methods that needed to train a binary classifier. We are sure that choosing different architecture or emphasizing the place of splitting with a unique character would noticeably improve the performance when comparing the methods with different lookaheads. The same also applies to the construction of the train and test datasets, where we put only up to three positive and negative samples from each sentence. It would be definitely worth exploring the effect of presenting the training algorithm with more samples of each sentence and observing at what point it would overfit.

And thirdly, it would also be interesting to see whether binary classification is the best option for the task in the first place. In our opinion, it should be the most straightforward and fastest variant, but we can also imagine that for the same task, one could use a sequence tagger as well – each time, labeling words or

partial sentences with tags denoting whether a split should or should not follow. In the end, we would probably use only the tag of the currently processed word, but we could potentially reveal more about the types of words causing the changes of tags by reexamining the tags of previous words.

6.4 Analysis of the results

Creating the most advanced splitting method was never a real goal of this thesis. Our main aim was to summarize available knowledge about TUs from linguists, try to apply it when constructing and analyzing our automated splitting methods, and reveal what might be the important aspects affecting the splitting. Furthermore, comparing our results with other research is rather complicated as it is not only the type of dataset affecting the final delay and translation quality score but also the used translation model and the strategy of interpreting (retranslation vs. growing partial translations by segments).

As one of the most important aspects of this thesis, we consider our approach for analysis and visualization of splitting methods' behavior. Reporting the statistics about segment counts and lengths, supplemented with the most frequent phrase chunk/POS tags at both sides of the splits, should provide good information about what kinds of splits the methods make. In this regard, we consider our observation about the between and inside phrase chunk split ratio for our golden annotations as crucial. Because it clearly shows that meaningful splits almost all the time find themselves at the borders of the phrase chunks. This observation could be used when developing more advanced splitting methods by effectively reducing the number of possible spaces where the split can occur.

In addition to the splitting analysis, we also came up with a new approach for comparing splitting methods among each other by introducing the definition of granular unique splits. A notion that can be very helpful when exploring whether one splitting method is a subset of another method. As the opposite of granular unique splits, we defined true unique splits whose higher number represents the state when two methods split according to different criteria.

We also wanted to analyze more thoroughly how much will the alignment splitting methods change depending on the used alignment tool, different translations, and different languages. We were very surprised to find out that even though the alignment algorithm approaches the sentence splitting in a similar manner as humans do, the alignment splitting algorithms made significantly more splits than the annotators. Moreover, when comparing the alignment methods working with translations from our annotators and from LINDAT, more splits were placed with the general LINDAT translations. That theoretically should not be possible because the annotators were instructed to produce translations achieving the most optimal splitting, which also means the shortest splits possible. But as we noted, the machine translation may favor shorter splitting because of its monotone and word-by-word translations. For those reasons, we do not recommend using alignment splitting for machine interpretation systems because it may split excessively and unreliably.

Before concluding, we need to discuss our trained NMT models as well. In the beginning, we were not sure whether a finetuning on partial sentences would have a good or bad effect on the translation performance for interpretation. Therefore,

we were glad to see that not only the finetuned MT model produced more splits when used for splitting – i.e., it was more consistent with the progressive translations, but also achieved smaller flicker when used within the SLTev evaluation.

Similarly, as for the training of binary classifiers, there is also big space left for exploring optimal architecture and data preparation that can improve translation splitting and flickering. It would also be beneficial to, for example, explore what effect the lookahead has not only for splitting but also for translation. Because in our experiments, we provided the lookahead-extended partial sentences only to the splitting module, and the translation got the partial sentence without the additional words. Therefore, there is room for training NMT models with additional words – similarly as they are trained when implementing wait-k splitting methods.

But not only that, we could even include in our translations some previous sentences that could potentially bear an important context for translating partial sentences. To move one step closer to the level of human interpretation, we should, however, completely omit training NMTs on parallel translations and use the real interpretation data instead. As Zhao et al. [2021] suggests, there could be a big difference between those approaches since interpretation requires not only correct translation but sometimes also a clever way how to express the intents of the speaker with an indirect description.

Regarding the results of the SLTev evaluation, we were not able to replicate the behavior of some oracle methods with their sequential ones – namely the methods trained binary classifiers. Despite that, we can get a very good idea of how the sequential methods will perform when improving the models by looking at the scores of their oracle variants.

Because we used retranslation, we could not properly measure the BLEU score of our splitting methods. Since SLTev measures BLEU only for complete sentences, all our methods scored the same. The only difference appeared when we were using the baseline or finetuned MT model. Therefore, it would be definitely a good check to evaluate our splitting methods while translating only the new segments and see how correlated the BLEU and flicker scores are.

Conclusion

This thesis studied the output quality, latency, and flicker when a translation system is translating a growing input step by step. Even though there are still many potential ways how we could improve our experiments or supplement them with additional data, we believe that we lay a solid foundation for future research in the field of SiMT.

We would especially highlight our contribution to formulating a definition of optimal human interpreting and putting together a set of instructions for the annotation process according to the definition (see Chapter 3). With our golden data, we were then able to analyze between what kinds of words annotators tend to split most often and formulate two new splitting methods based on that (see Section 5.1.1)

Furthermore, although these two methods did not perfectly mimic the human splitting, we were able to make an important observation about the golden data – more than 75% of all golden splits were placed between two phrase chunks (not inside one) which significantly reduces the search space for any other splitting method (see Section 5.1).

When comparing the splitting methods to each other, in Section 5.2.1, we also introduced a new way to express a relation that one splitting method is a subset of another. We achieved that by defining two classes of splits made only by one of the two splitting methods in the comparison.

Additionally, in Section 5.2.2, we analyzed the behavior of splitting methods based on the word alignment – using different word alignment tools, target translations, and languages. Because of the excessive splitting and noticeable difference among the variants, we regard alignment splitting methods as unreliable and do not recommend their use in practice.

Aside from splitting, we found a good use case for word alignment in combination with acoustic and time information for the visualization of interpretation that can help professional interpreters analyze their performance (see Section 4.3.1).

In the end, we prepared two NMT systems that we incorporated into our evaluation. The baseline model trained solely on complete sentences, and its finetuned version retrained on partial sentences. For all splitting methods, using the finetuned MT model led to better translation stability – thus, showing a promise to be a key component to optimize when deploying the SiMT systems.

Finally, we compared all the splitting methods in terms of translation stability and latency; and suggested a more thorough analysis of different segmentation methods' behavior (see Section 5.4).

Bibliography

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. FLAIR: An easy-to-use framework for state-of-the-art NLP. pages 54–59, 2019.

Musatafa Albadr, Sabrina Tiun, and Fahad Al-Dhief. Evaluation of machine translation systems and related procedures. *Journal of Engineering and Applied Sciences*, 13:3961–3972, 06 2018.

Ebrahim Ansari, Ondřej Bojar, Barry Haddow, and Mohammad Mahmoudi. SLTEV: Comprehensive evaluation of spoken language translation. pages 71–79, April 2021. doi: 10.18653/v1/2021.eacl-demos.9. URL `https://aclanthology.org/2021.eacl-demos.9`.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, and George Foster. Re-translation versus streaming for simultaneous translation. pages 220–227, July 2020. doi: 10.18653/v1/2020.iwslt-1.27. URL `https://aclanthology.org/2020.iwslt-1.27`.

Henri Charles Barik. *A study of simultaneous interpretation*, pages 1–25. The University of North Carolina at Chapel Hill, 1969.

Leonid Barkhudarov. *Urovni yazykovoy iyerarkhii i perevod [Levels of language hierarchy and translation]*, pages 3–12. 1969.

Timo Baumann and David Schlangen. INPRO_iSS: A component for just-in-time incremental speech synthesis. pages 103–108, July 2012. URL `https://aclanthology.org/P12-3018`.

Roger T Bell and Christopher Candlin. *Translation and translating: Theory and practice*, volume 298. Longman London, 1991.

Paul Bennett. The translation unit in human and machine. *Babel*, 40(1):12–20, 1994. ISSN 0521-9744. doi: https://doi.org/10.1075/babel.40.1.03ben. URL `https://www.jbe-platform.com/content/journals/10.1075/babel.40.1.03ben`.

Luisa Bentivogli, Mauro Cettolo, Marco Gaido, Alina Karakanta, Alberto Martinelli, Matteo Negri, and Marco Turchi. Cascade versus direct speech translation: Do the differences still make a difference? pages 2873–2887, 01 2021. doi: 10.18653/v1/2021.acl-long.224.

Lynne Bowker. *Computer-aided translation technology: A practical introduction*. University of Ottawa Press, 2002.

Colin Cherry and George Foster. Thinking slow about latency evaluation for simultaneous machine translation. 2019. doi: 10.48550/ARXIV.1906.00048. URL `https://arxiv.org/abs/1906.00048`.

Kyunghyun Cho and Masha Esipova. Can neural machine translation do simultaneous translation? 2016. doi: 10.48550/ARXIV.1606.02012. URL https://arxiv.org/abs/1606.02012.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Pernilla Danielsson. The automatic identification of meaningful units in language. 2001.

Zi-Yi Dou and Graham Neubig. Word alignment by fine-tuning embeddings on parallel corpora. 2021. doi: 10.48550/ARXIV.2101.08231. URL https://arxiv.org/abs/2101.08231.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. A simple, fast, and effective reparameterization of ibm model 2. 2013.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. Efficient wait-k models for simultaneous machine translation. 2020. doi: 10.48550/ARXIV.2005.08595. URL https://arxiv.org/abs/2005.08595.

Daniel Gile. *Basic theoretical components in interpreter and translator training*, pages 185–194. John Benjamins, 1992. URL https://www.jbe-platform.com/content/books/9789027285898-z.56.29gil.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. Learning to translate in real-time with neural machine translation. pages 1053–1062, April 2017. URL https://aclanthology.org/E17-1099.

Hirofumi Inaguma, Shun Kiyono, Kevin Duh, Shigeki Karita, Nelson Yalta, Tomoki Hayashi, and Shinji Watanabe. ESPnet-ST: All-in-one speech translation toolkit. pages 302–311, July 2020. doi: 10.18653/v1/2020.acl-demos.34. URL https://aclanthology.org/2020.acl-demos.34.

Sathish Indurthi, Houjeung Han, Nikhil Kumar Lakumarapu, Beomseok Lee, Insoo Chung, Sangha Kim, and Chanwoo Kim. End-end speech-to-text translation with modality agnostic meta-learning. pages 7904–7908, 2020. doi: 10.1109/ICASSP40776.2020.9054759.

Javier Iranzo-Sánchez, Javier Jorge, Pau Baquero-Arnal, Joan Albert Silvestre-Cerdà, Adrià Giménez, Jorge Civera, Albert Sanchis, and Alfons Juan. Streaming cascade-based speech translation leveraged by a direct segmentation model. *Neural Networks*, 142:303–315, 2021. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2021.05.013. URL https://www.sciencedirect.com/science/article/pii/S0893608021002057.

Masoud Jalili Sabet, Philipp Dufter, François Yvon, and Hinrich Schütze. SimAlign: High quality word alignments without parallel training data using static and contextualized embeddings. pages 1627–1643, November 2020. URL https://www.aclweb.org/anthology/2020.findings-emnlp.147.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. Marian: Fast neural machine translation in C++. pages 116–121, July 2018. doi: 10.18653/v1/P18-4020. URL https://aclanthology.org/P18-4020.

Yasumasa Kano, Katsuhito Sudoh, and Satoshi Nakamura. Simultaneous neural machine translation with constituent label prediction. pages 1124–1134, November 2021. URL https://aclanthology.org/2021.wmt-1.120.

Alina Karakanta, Sara Papi, Matteo Negri, and Marco Turchi. Simultaneous speech translation for live subtitling: from delay to display. 2021. doi: 10.48550/ARXIV.2107.08807. URL https://arxiv.org/abs/2107.08807.

Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. Crepe: A convolutional representation for pitch estimation. 2018. doi: 10.48550/ARXIV.1802.06182. URL https://arxiv.org/abs/1802.06182.

Don Kiraly. Lörscher, wolfgang. 1991. translation performance, translation process, and translation strategies: A psycholinguistic investigation. *Target. International Journal of Translation Studies*, 4(1):126–129, 1992. ISSN 0924-1884. doi: https://doi.org/10.1075/target.4.1.15kir. URL https://www.jbe-platform.com/content/journals/10.1075/target.4.1.15kir.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Thomas Kisler, Uwe Reichel, and Florian Schiel. Multilingual processing of speech via web services. *Computer Speech Language*, 45:326–347, 2017. ISSN 0885-2308. doi: https://doi.org/10.1016/j.csl.2017.01.005. URL https://www.sciencedirect.com/science/article/pii/S0885230816302418.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. pages 79–86, September 13-15 2005. URL https://aclanthology.org/2005.mtsummit-papers.11.

Philipp Koehn, Franz J. Och, and Daniel Marcu. Statistical phrase-based translation. pages 127–133, 2003. URL https://aclanthology.org/N03-1017.

Werner Koller. *Einführung in die Übersetzungswissenschaft*, page 343 S. UTB für Wissenschaft : Uni-Taschenbücher ; 819 : Linguistik. Quelle Meyer, Heidelberg [u.a.], 4., völlig neu bearb. aufl. edition, 1992. ISBN 3-8252-0819-2 and 3-494-02192-9 and 978-3-8252-0819-6 and 978-3-494-02192-8. Literaturverz. S. 301 - 328.

Sylvie Lambert and Barbara Moser-Mercer. *Bridging the gap: Empirical research in simultaneous interpretation*, volume 3, page 255. John Benjamins Publishing, 1994.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. Stacl: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. pages 3025–3036, 01 2019a. doi: 10.18653/v1/P19-1289.

Qingsong Ma, Johnny Wei, Ondřej Bojar, and Yvette Graham. Results of the WMT19 metrics shared task: Segment-level and strong MT systems pose big challenges. pages 62–90, August 2019b. doi: 10.18653/v1/W19-5302. URL https://aclanthology.org/W19-5302.

Xutai Ma, Mohammad Javad Dousti, Changhan Wang, Jiatao Gu, and Juan Pino. SIMULEVAL: An evaluation toolkit for simultaneous translation. pages 144–150, October 2020a. doi: 10.18653/v1/2020.emnlp-demos.19. URL https://aclanthology.org/2020.emnlp-demos.19.

Xutai Ma, Juan Pino, and Philipp Koehn. Simulmt to simulst: Adapting simultaneous text translation to end-to-end simultaneous speech translation. 2020b. doi: 10.48550/ARXIV.2011.02048. URL https://arxiv.org/abs/2011.02048.

Dominik Macháček, Matúš Žilinec, and Ondřej Bojar. Lost in interpreting: Speech translation from source or interpreter? 2021. doi: 10.48550/ARXIV.2106.09343. URL https://arxiv.org/abs/2106.09343.

Kirsten Malmkjaer. *Translation Units*, pages 92–93. 12 2006. ISBN 9780080448541. doi: 10.1016/B0-08-044854-2/00491-0.

Kirsten Malmkjær. *Unit of translation*, pages 286–287. Routledge, 1998.

Barbara Moser-Mercer. The expert-novice paradigm in interpreting research. *Translationsdidaktik: Grundfragen der Übersetzungswissenschaft*, pages 255–261, 1997.

Robert Östling and Jörg Tiedemann. Efficient word alignment with Markov Chain Monte Carlo. *Prague Bulletin of Mathematical Linguistics*, 106:125–146, October 2016. URL http://ufal.mff.cuni.cz/pbml/106/art-ostling-tiedemann.pdf.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. pages 311–318, July 2002. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040.

Juan Pino, Liezl Puzon, Jiatao Gu, Xutai Ma, Arya D. McCarthy, and Deepak Gopinath. Harnessing indirect training data for end-to-end automatic speech translation: Tricks of the trade. 2019. doi: 10.48550/ARXIV.1909.06515. URL https://arxiv.org/abs/1909.06515.

Sonia Pio. The relation between st delivery rate and quality in simultaneous interpretation. 2003.

Martin Popel, Markéta Tomková, Jakub Tomek, Łukasz Kaiser, Jakob Uszkoreit, Ondrej Bojar, and Z. Žabokrtský. Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals. *Nature Communications*, 11, 2020.

Matt Post. A call for clarity in reporting bleu scores. pages 186–191, 01 2018. doi: 10.18653/v1/W18-6319.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. 2020. URL `https://nlp.stanford.edu/pubs/qi2020stanza.pdf`.

Dhevi J. Rajendran, Andrew T. Duchowski, Pilar Orero, Juan Martínez, and Pablo Romero-Fresco. Effects of text chunking on subtitling: A quantitative and qualitative examination. *Perspectives*, 21(1):5–21, 2013. doi: 10.1080/0907676X.2012.722651. URL `https://doi.org/10.1080/0907676X.2012.722651`.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. COMET: A neural framework for MT evaluation. pages 2685–2702, November 2020. doi: 10.18653/v1/2020.emnlp-main.213. URL `https://aclanthology.org/2020.emnlp-main.213`.

Yi Ren, Jinglin Liu, Xu Tan, Chen Zhang, Tao Qin, Zhou Zhao, and Tie-Yan Liu. SimulSpeech: End-to-end simultaneous speech to text translation. pages 3787–3796, July 2020. doi: 10.18653/v1/2020.acl-main.350. URL `https://aclanthology.org/2020.acl-main.350`.

Pablo Romero-Fresco. *Standing on quicksand: Hearing viewers' comprehension and reading patterns of respoken subtitles for the news*, pages 175–195. 01 2010. ISBN 978-90-420-3180-7. doi: 10.1163/9789042031814_014.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

Miriam Schlesinger. Effects of presentation rate on working memory in simultaneous interpreting. 2003.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. BLEURT: Learning robust metrics for text generation. pages 7881–7892, July 2020. doi: 10.18653/v1/2020.acl-main.704. URL `https://aclanthology.org/2020.acl-main.704`.

M. Shuttleworth and M. Cowie. *Dictionary of Translation Studies*. St. Jerome Pub., 1997. ISBN 9781900650038. URL `https://books.google.cz/books?id=hEtiAAAAMAAJ`.

Zhixing Tan, Shuo Wang, Zonghan Yang, Gang Chen, Xuancheng Huang, Maosong Sun, and Yang Liu. Neural machine translation: A review of methods, resources, and tools. *AI Open*, 1:5–21, 2020. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2020.11.001. URL `https://www.sciencedirect.com/science/article/pii/S2666651020300024`.

Martha Thunes. The concept of 'translation unit' revisited. *Bergen Language and Linguistics Studies*, 8, 11 2017. doi: 10.15845/bells.v8i1.1331.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. doi: 10.48550/ARXIV.1706.03762. URL https://arxiv.org/abs/1706.03762.

J.P. Vinay, J. Darbelnet, J.C. Sager, and M.J. Hamel. *Comparative Stylistics of French and English: A Methodology for Translation*. Benjamins translation library. John Benjamins Publishing Company, 1995. ISBN 9789027216106. URL https://books.google.cz/books?id=I06D-6gU45sC.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. 2019. doi: 10.48550/ARXIV.1910.03771. URL https://arxiv.org/abs/1910.03771.

Jia Xu, Richard Zens, and Hermann Ney. Sentence segmentation using ibm word alignment model 1. 2005.

Ruiqing Zhang, Chuanqiang Zhang, Zhongjun He, Hua Wu, and Haifeng Wang. Learning adaptive segmentation policy for simultaneous translation. pages 2280–2289, November 2020. doi: 10.18653/v1/2020.emnlp-main.178. URL https://aclanthology.org/2020.emnlp-main.178.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. 2019. doi: 10.48550/ARXIV.1904.09675. URL https://arxiv.org/abs/1904.09675.

Jinming Zhao, Philip Arthur, Gholamreza Haffari, Trevor Cohn, and Ehsan Shareghi. It is not as good as you think! evaluating simultaneous machine translation on interpretation data. pages 6707–6715, November 2021. doi: 10.18653/v1/2021.emnlp-main.537. URL https://aclanthology.org/2021.emnlp-main.537.

Chunshen Zhu. Ut once more: the sentence as the key functional unit of translation: the sentence as the key functional unit of translation. 1999.

Jordan Zlatev and Johan Blomberg. Language may indeed influence thought. *Frontiers in Psychology*, 6, 2015. ISSN 1664-1078. doi: 10.3389/fpsyg.2015.01631. URL https://www.frontiersin.org/articles/10.3389/fpsyg.2015.01631.

List of Figures

List of Tables

List of Abbreviations

AL Average lagging

AP Average proportion

ASR Automatic speech recognition

BLEU Bilingual evaluation understudy

ESIC Europarl Simultaneous Interpreting Corpus

MT Machine translation

NLP Natural language processing

NMT Neural machine translation

POS Part of speech

SI Simultaneous interpretation

SiMT Simultaneous machine translation

SMT Statistical machine translation

SOV Subject-object-verb

ST Speech translation

SVO Subject-verb-object

TAP Think aloud protocol

TU Translation unit

A. Appendix

In this section, we included some figures that may help readers to clarify better the topics we wrote about but, at the same time, were not that crucial for the main ideas of the thesis.

A.1 Phrase chunk and POS splitting

Tag	Meaning
ADJ	adjective
ADP	adposition
ADV	adverb
AUX	auxiliary
CCONJ	coordinating conjunction
DET	determiner
INTJ	interjection
NOUN	noun
PART	particle
PRON	pronoun
PROPN	proper noun
PUNCT	punctuation
SCONJ	subordinating conjunction
SYM	symbol
VERB	verb
X	other

Table A.1: Table showing all POS tags that can be predicted by the Stanza model.

Tag	Meaning
ADJP	adjectival
ADVP	adverbial
CONJP	conjunction
INTJ	interjection
LST	list maker
NP	noun phrase
PP	prepositional
PRT	particle
SBAR	subordinate clause
VP	verb phrase

Table A.2: Table showing all phrase chunk tags that can be predicted by the flair model.

NP-NP And secondly, (ADVP | NP) Europe was outfoxed by China and India, (NP |) and (NP | NP) the US joined in.
ADVP-NP Thank you very much (ADVP | NP) Mr President.
NP-PP Thank you (NP | NP) Mr President, (NP | VP) thank you (NP | PP) for your patience (NP |) and (NP |) indulgence.
ADJP-SBAR We (NP | PP) as members of the Parliament (NP | VP) have a duty (NP | VP) to make sure (ADJP | SBAR) that Europe's policies reflect the will (NP |) of citizens.
VP-VP The European dream (NP | VP) is (VP | VP) to retire (VP | PP) at the French Riviera (NP | ADVP) as soon as possible.
NP-SBAR And (PP |) given the fact (NP | SBAR) that we are now looking at (PP | NP) climate change legislation, (NP | NP) that is very serious.
VP-SBAR And (NP |) I think (VP | SBAR) that (SBAR | NP) more should be done in this area.

Table A.3: Table showing a typical split for each phrase chunk pair that we used as a rule in the phrase chunk splitting method. By a typical split, we mean a sentence containing the phrase chunk pair and a pair of words that were most commonly seen when the particular phrase chunks happened to be next to each other.

NOUN-ADP I would like (VERB | PART) to underline (VERB | SCONJ) that (SCONJ | DET) the implementation (NOUN | ADP) of the regulation (NOUN | AUX) is (AUX | ADV) crucially important.
ADJ-SCONJ We (PRON | SCONJ) as members of the Parliament (PROPN | VERB) have a duty (NOUN | PART) to make sure (ADJ | SCONJ) that Europe's policies reflect the will (NOUN | ADP) of citizens.
NOUN-PRON This is the gist of the report (NOUN | PRON) that we are debating tonight.
PROPN-VERB And (CCONJ | PRON) we believe (VERB | SCONJ) that the EU (PROPN | VERB) has to be also honest (ADJ | ADP) on its own side.
NOUN-AUX This report (NOUN | AUX) is part (NOUN | ADP) of the simplification package (NOUN | ADP) of the company law.
PROPN-ADP Anyhow, the Commission (PROPN | VERB) leaves flexibility for Member States (PROPN | ADP) in terms of (ADP | DET) this additional publication requirements.
CCONJ-DET And secondly, (ADV-PUNCT | PROPN) Europe was outfoxed by China and India, (PROPN-PUNCT | CCONJ) and (CCONJ | DET) the US joined in.

Table A.4: Table showing a typical split for some POS pairs that we used as a rule in the POS splitting method. By a typical split, we mean a sentence containing the POS pair and a pair of words that were most commonly seen when the particular POS tags happened to be next to each other.

A.2 Further analysis of splitting methods

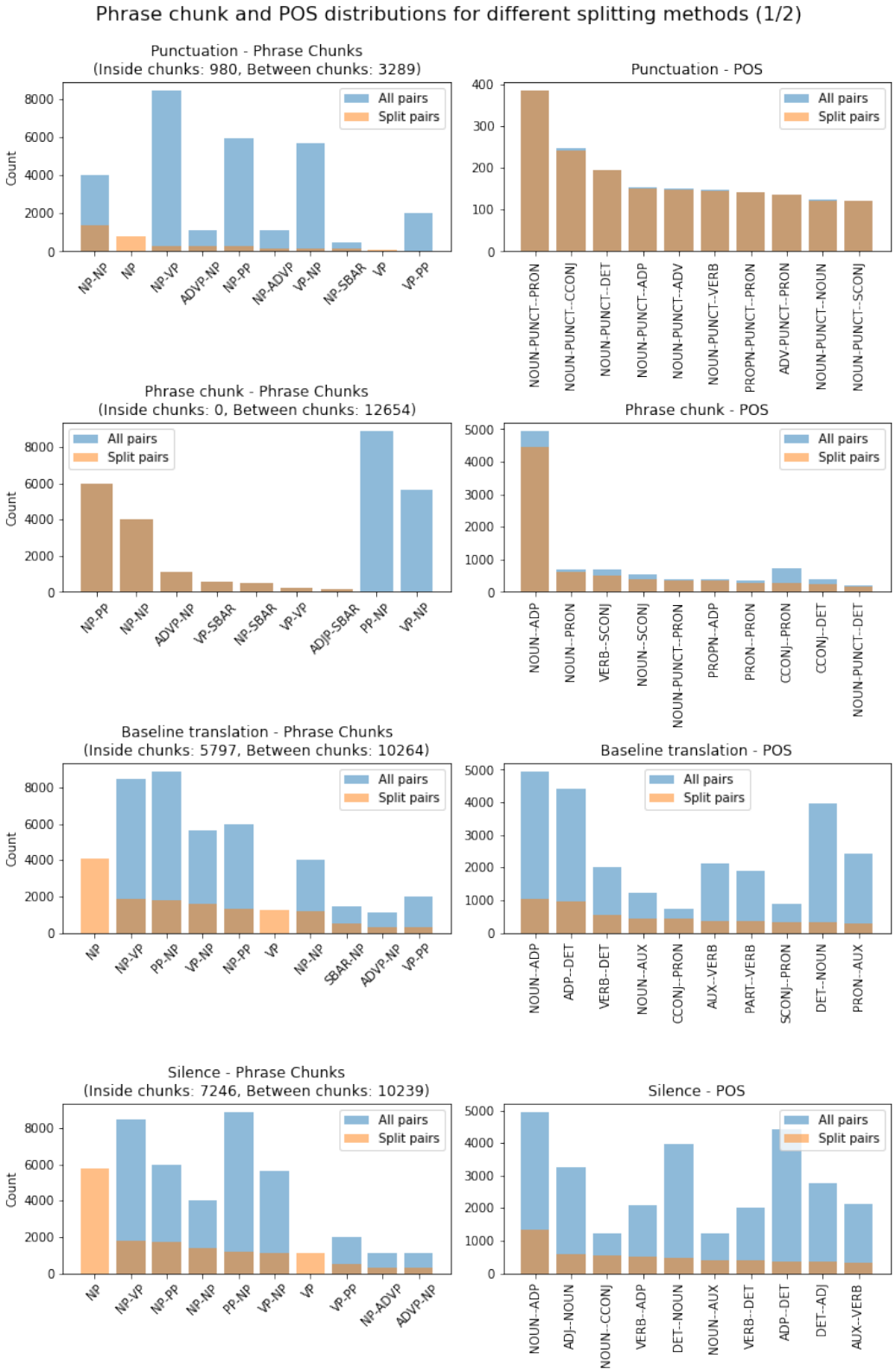

Figure A.1: Ten most frequent phrase chunk and POS pairs split by different segmentation methods (1/2).

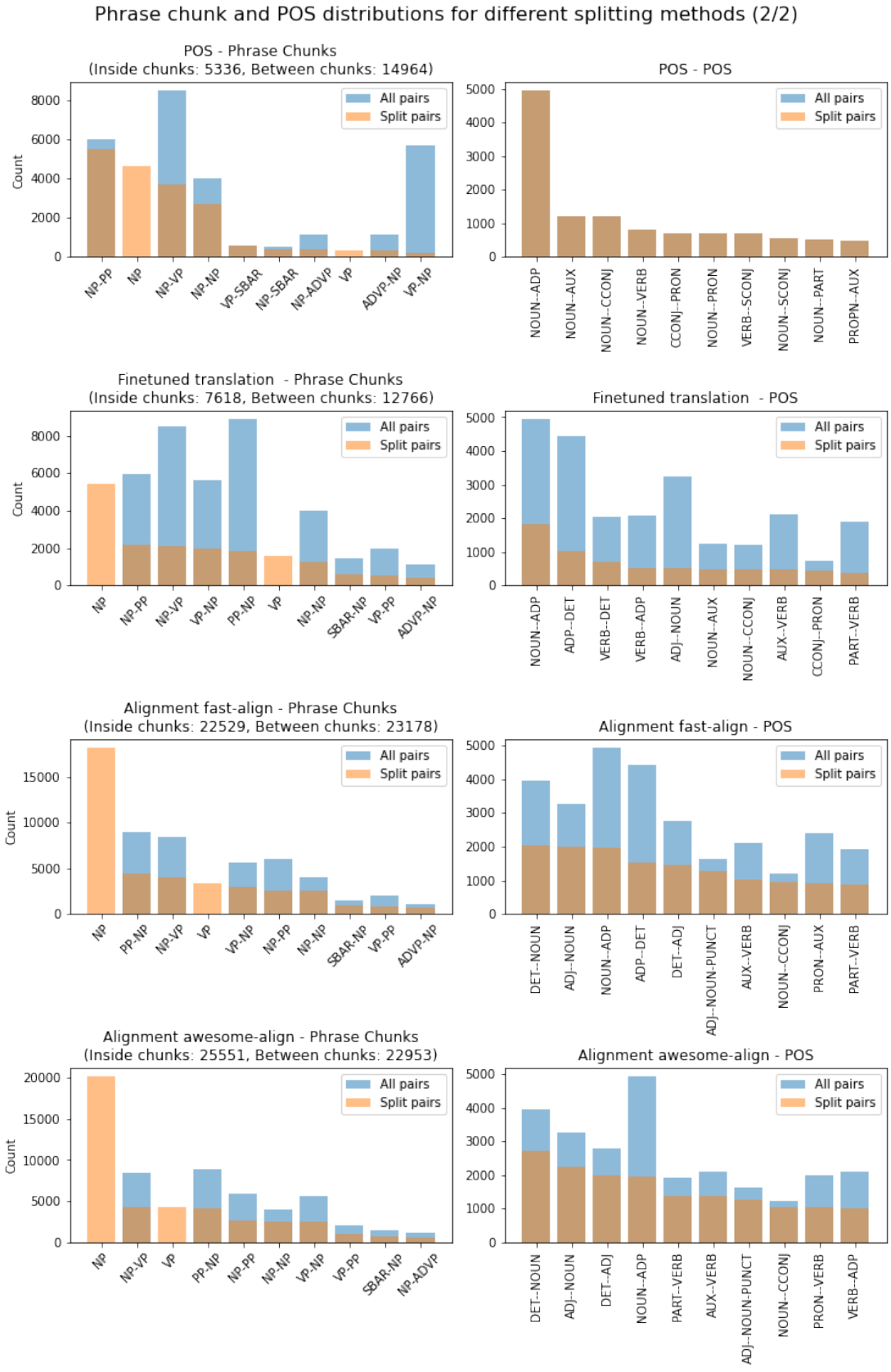

Figure A.2: Ten most frequent phrase chunk and POS pairs split by different segmentation methods (2/2).

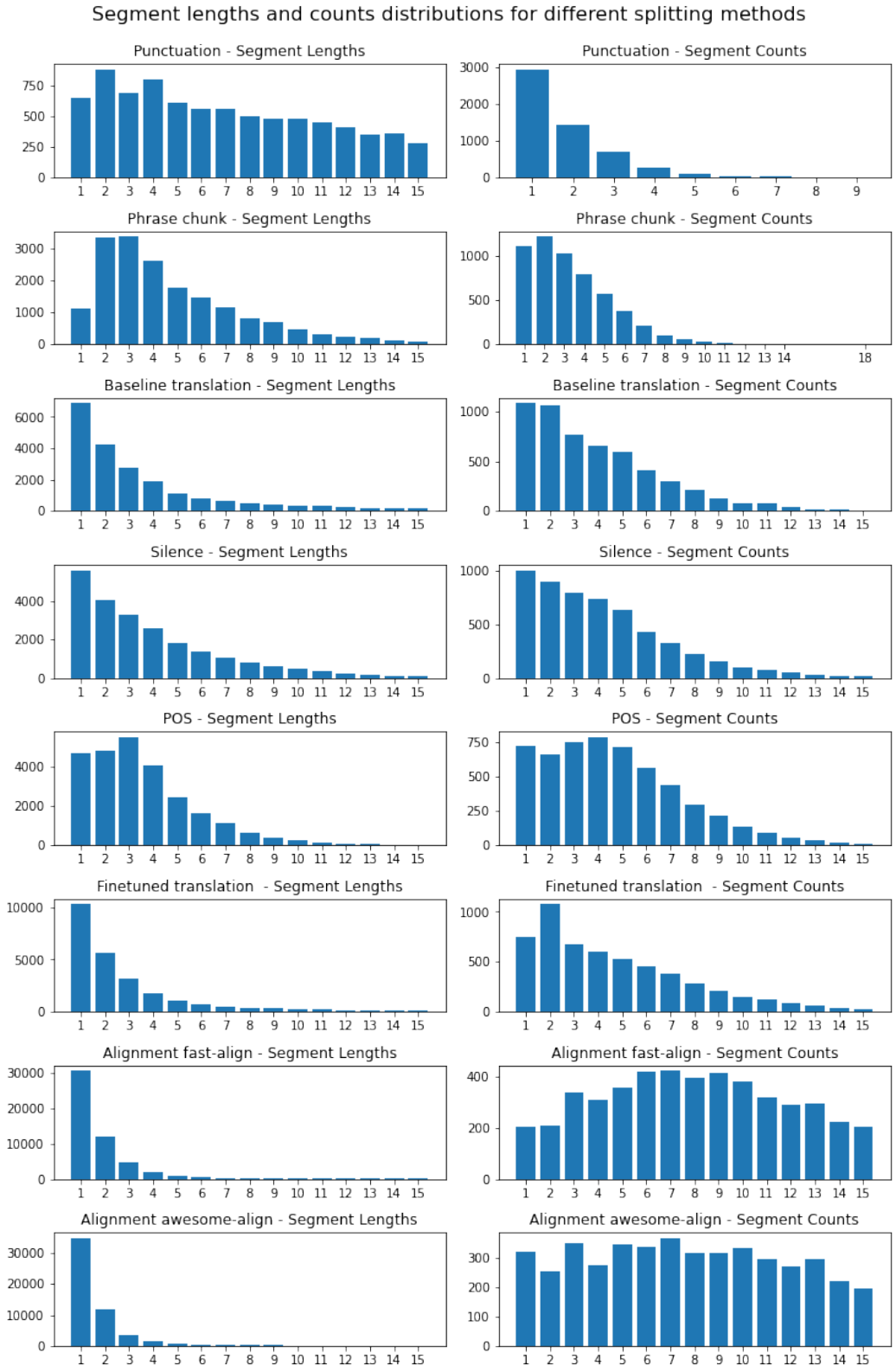

Figure A.3: Distributions of segment lengths and counts for different splitting methods.

A.3 Differences among splitting methods and MT models

But there's one thing @ that you even mention @ in your paper @ that you presented to us.
Ale je tu jedna věc, @ kterou dokonce zmiňujete @ ve své práci, @ kterou jste nám představil.

It's about the own financial resources @ of the European Union.
Jde o vlastní finanční zdroje @ Evropské unie.

But you do not mention @ where they should come from.
Ale nezmiňujete, @ odkud by měly pocházet.

Unfortunately @ I did not get an answer from you, @ hopefully now.
Bohužel @ jsem od vás nedostal odpověď, @ doufejme nyní.

What about a financial transaction tax?
A co daň z finančních transakcí?

No, @ and @ Mistral is no use @ because it cannot approach these landlocked countries.
Ne, @ a @ Mistral je k ničemu, @ protože se nemůže přiblížit k těmto vnitrozemským zemím.

Figure A.4: Few examples of sentences split by annotators and their reference translations split into the same number of chunks.

Punctuation Second, | the way to eliminate tax fraud is not a reduction of the competition but strict elimination of tax exemptions.
Phrase chunks Second, | the way to eliminate tax fraud is not a reduction | of the competition but | strict elimination | of tax exemptions.
Baseline translation Second, | the way to eliminate tax fraud is not a reduction of the competition but strict elimination of tax exemptions.
Silence Second, | the way to eliminate | tax fraud | is not a reduction | of the competition | but | strict | elimination | of | tax | exemptions.
POS Second, | the way | to eliminate tax fraud | is not a reduction | of the competition | but strict elimination | of tax exemptions.
Finetuned translation Second, | the way to eliminate | tax fraud is not | a reduction | of the competition but | strict | elimination | of tax exemptions.
Golden Second, | the way to eliminate tax fraud | is not a reduction of the competition | but | strict elimination | of tax exemptions.
Alignment fast-align Second, | the way to | eliminate | tax | fraud | is not | a reduction | of the competition | but | strict | elimination of | tax | exemptions.
Alignment awesome-align Second, the | way | to | eliminate | tax | fraud | is not | a | reduction of the | competition | but | strict | elimination of | tax | exemptions.

Table A.5: Table summarizing different behavior of all splitting methods when applied to the same sentence.

-Think
(B) Přemýšlej
(F) Přemýšlejte

-Think about
(B) Zamyslete se nad
(F) Přemýšlejte o

-Think about cars,
(B) Zamysleme se nad auty,
(F) Přemýšlejte o autech,

-Think about cars, this
(B) Přemýšlejte o autech, o tom.
(F) Zamyslete se nad auty, to

-Think about cars, this is
(B) Zamyslete se nad auty, to je
(F) Přemýšlejte o autech, to je

-Think about cars, this is something
(B) Zamyslete se nad auty, to je něco.
(F) Přemýšlejte o autech, to je něco

-Think about cars, this is something Russian
(B) Zamyslete se nad auty, to je něco ruského
(F) Přemýšlejte o autech, to je něco ruského

-Think about cars, this is something Russian industry
(B) Přemýšlejte o automobilech, to je něco, co ruský průmysl
(F) Přemýšlejte o autech, to je něco, co ruský průmysl

-Think about cars, this is something Russian industry is
(B) Přemýšlejte o automobilech, to je něco, co ruský průmysl je
(F) Přemýšlejte o autech, to je něco, co ruský průmysl je

-Think about cars, this is something Russian industry is also
(B) Přemýšlejte o automobilech, to je něco, co ruský průmysl je také
(F) Zamyslete se nad automobily, to je něco, co ruský průmysl je také

-Think about cars, this is something Russian industry is also struggling
(B) Přemýšlejte o automobilech, to je něco, co ruský průmysl také zápasí
(F) Zamyslete se nad automobily, to je něco, co ruský průmysl také zápasí

-Think about cars, this is something Russian industry is also struggling with.
(B) Zamysleme se nad auty, je to něco, s čím ruský průmysl také zápasí.
(F) Zamyslete se nad auty, je to také něco, s čím ruský průmysl zápasí.

Figure A.5: Partial translations of an English sentence with results from our baseline (B) and finetuned (F) NMT models.