Fantastic Resources But Where To Find Them: Expanding ASR Systems for Less-Resourced Languages

Aleksandra Krupińska, aleksandra.krupinska.8401@student.uu.se, *Uppsala University*
Fehmi Ayberk Uçkun, uckun@kth.se, *KTH Royal Institute of Technology*
Kristóf Fischer, kristof.fischer.3430@student.uu.se, *Uppsala University*
Micaella Bruton, micaella.bruton.4363@student.uu.se, *Uppsala University*

Acknowledgement

Thank you to *Jeet Santosh Nimbhorkar* from KTH Royal Institute of Technology for being with us at the very start of the project and writing a code snippet not used in the final version of the project.

Abstract

The creation of automatic speech recognition (ASR) systems is heavily dependent upon the availability of transcribed speech data in the target language, as large amounts of this data are necessary to accurately train the model. Less-resourced languages, such as Hungarian, lack the requisite amount of freely available data which leads to the unavailability of fully functioning speech recognition systems. In an effort to counteract this, we sought to create a Hungarian ASR system without using Hungarian speech corpus, instead inserting Hungarian grapheme replacements into the outputs suggested by three trained ASR models for other languages with overlapping phoneme sets. Outputs were then weighted by character and the most probable options were returned as final output. By applying this algorithm we managed to label 10 hours of unlabelled Hungarian speech gathered from YouTube and created a corpus that can be used to train ASR models. Using this method we tested the accuracy of our labelling algorithm on Common Voice dataset without any explicit Hungarian ASR model.

1| Introduction

Languages can vary greatly in the number of resources available for them and are typically divided into two substantial modalities, spoken and written. Sometimes these two types differ in their application areas, while in other cases both play an active role in the development of a particular system. The aim of automatic speech recognition (ASR) is to process and transcribe input audio. However, in order to train systems capable of doing that, large amounts of speech data are required. The vast majority of the human population speaks only a fraction of the world's languages, with prestige languages being the main focus for most natural language processing (NLP) research. This results in a resource deficit for many languages, creating a group encompassing both languages on their way to extinction and others, not facing this threat, but lacking support in many spoken NLP applications. In such cases, developing speech technology systems can be essential, depending on the state of the language, its preservation, or encouragement of overall resource security.

When considering freely available spoken data, Hungarian is an example of a less-resourced language. While supported by some voice assistants such as Google, native speakers can still struggle to find tools that accurately allow its use as there are many other languages that are more prioritized by big tech. This is why it is important to devise a temporary solution, enabling decently accurate automatic speech recognition to assist in the promotion of more freely available, accurately performing ASR tools. One of the approaches to achieve this is to use a combination of currently existing ASR systems created for other, phonetically similar languages. In this project, we use three Wav2Vec models pre-trained for three languages: Czech, Finnish and Italian. The phoneme sets for these languages overlap with that of Hungarian, allowing for the bypass of the Hungarian ASR model for training and instead ensuring accurate output through grapheme replacements and weighted voting to decide on a singular final output.

2| Background and Related Research

2.1 | Language Resourcing

According to Besacier et al. (2014), 96% of the world's languages are spoken by only 4% of its population. With language extinction becoming an increasingly serious issue in today's globalized world, it is important to consider the actions we can undertake to contribute to the protection of human languages. Languages are an important part of the human experience and as Besacier et al. (2014) point out, even a language with 100.000 speakers can become endangered. The extinction of a language is at the same time the disappearance of a culture, and we lose a piece of what it means to be human. Unfortunately, not all languages are considered equally important in Human Language Technologies (HLT), leading to resource deficits for all but the most 'prestigious' of languages. Ensuring the proper documentation of less-resourced languages in a computer-readable format, and developing ASR systems for rapid transcription would have a significant effect on their preservation (Besacier et al., 2014). It is important to note that while a language being less- or under-resourced does not necessarily mean that it is endangered, having an adequate amount of resources absolutely strengthens its status and can help ensure its survival. Hungarian, for instance, is not considered endangered but still suffers a deficit of resources and lacks support in many NLP application areas; there is definitely room for improvement in its representation in HLT which can help to ensure it does not become endangered.

2.2 | Automatic Speech Recognition (ASR)

The task of Automatic Speech Recognition (ASR) software is to convert a speech signal into its textual representation (Besacier et al., 2014). Different types of ASR systems, such as those dealing with spelled speech, isolated speech, or continuous speech, as well as various recognized vocabulary sizes, can be distinguished. The technique used in this project is the state-of-the-art wav2vec model, which uses a multi-layer convolutional neural network to encode speech audio (Baevski et al., 2020). This solution enables a substantial reduction of the amount of labeled training data, which is especially valuable for devising the technologies for under- and less-resourced languages.

Cross-linguistic adaptation is a commonly used method in developing ASR systems when dealing with these languages. This adaptation is motivated by the fact that training a model from scratch is costly and requires access to large amounts of data, which is often not freely available for smaller languages. Tóth et al. (2008) study the cross-lingual portability of multi-layer perceptrons (MLPs) between the fundamentally different English and Hungarian. Since English is one of the most well-resourced languages with support in most, if not all, NLP applications, it makes perfect sense to use the systems developed for it and apply it to other languages. Using the Hungarian Telephone Speech Database (MTBA) as test corpus, the authors found that the performance of the multilingual configuration was in line with the monolingual Hungarian system, proving the successful portability of a tool developed for a specific language to another, fundamentally different language (Tóth et al., 2008).

Vu, Kraus, and Schultz introduced the weighing method they call multilingual A-stabil, computing confidence scores based on the output of acoustic models from more than one language (2010). We chose this method to weigh our outputs as their project was quite similar to ours; building a Czech ASR system without any transcribed data using well-trained acoustic models for Russian, Bulgarian, Polish, and Croatian. Their method uses a set of alternative hypotheses derived from each language model, computing the frequency of each output normalized by the number of alternative hypotheses. The most frequently occurring output is then selected for the final output. Using this method, they were able to achieve a word error rate (WER) of 23.6% on the development set and 22.9% on the evaluation set which performs on par with the performance of a Czech ASR trained with 23 hours of audio data with manual transcriptions (23.1% on the development set and 22.3% on the evaluation set) (Vu, Kraus & Schultz, 2010).

2.3 | Current State of Hungarian Speech Resources

It is generally said to be challenging to access reliable audio data for Hungarian; though as Mihajlik et al. (2022) report, there have been attempts of compiling such corpora. One of the resources considered was Mozilla's Common Voice project, but for the time being, the corpus consists of only 19 hours of validated speech. Another property of the collection which is potentially problematic for processing spontaneous user speech is the fact that it contains only read speech. The variety of speech data utilized in training fundamentally impacts the accuracy of the transcription. Having only read speech at one's disposal throughout the training phase is likely to yield inaccurate results while transcribing spontaneous speech, which is the core of real-life human-machine communication. Another issue with some data sets is the lack of transcriptions; transcribed data is vital during training as it is necessary to "teach" the model what to look for and how to perform analyses (Mihajlik et. al., 2022). As of 2022, the most successful Hungarian spontaneous speech recognition models result in a word error rate of 42 - 55 % (Mihajlik et. al., 2022).

3| Method

3.1 | ASR Model Selection

3 ASR models are used for our 3 chosen seed languages, Italian, Czech, and Finnish. These models were fine-tuned in their respective languages. We utilized HuggingFace (Wolf et al., 2019) library for these models. All base models are chosen as a version Wave2Vec2 (Baevski et al., 2020) structure. Firstly, the best models that are freely available in the library were chosen. It is known that adding a language model to an acoustic model increases the WER of the model by a considerable portion and these models, too, had their language models. Later, we realized that since these models have their own language models integrated into the models and these language models force the model to output a sentence available in their own respective languages, it is not possible to get close enough outputs for Hungarian data. In our case, the models without LM should be used. Next, even though the latest fine-tuned Facebook wav2vec2-base-10k-voxpopuli models are tested, we still couldn't get consistent enough output to be successful. After several trials, we found a set of good models. Their WERs on Common Voice corpus are 11.5, 9.7, and 24.6 respectively for Italian, Finnish and Czech models. Inference and WER calculation code snippets for these models can be seen in Appendix F and G.

3.2 | Corpora Selection and Modification

To address the issues discussed in section 2.3, we combined the 19 hours of read speech data available from the Common Voice project (Ardila et al., 2019) with nearly 11 hours of manually selected material. The corpus consists of publicly accessible subtitled YouTube videos, available under the Creative Commons license. The videos contain various speaking styles; some include carefully and accurately articulated speech read by professional voice actors, while others resemble every day, sometimes sloppy, spontaneous conversations. Another property of the material is that it varies in the number of speakers in the individual segments. If there are multiple speakers in one video, the turn-taking process is not always organized, and overlapping speech can occur.

In order to use our corpus, some data modification was needed. First of all, the videos had to be converted to .wav audio files. Then, the frequency of the data had to be downsampled to 16 kHz, which is a requirement for our Wave2Vec2 models since they are trained on data with 16 kHz sampling rate. The individual files were then compiled into one long sequence. Subsequently, a free and open-source audio editor Audacity was used to shorten silent segments and remove music segments without speech. Segments without speech were detected by setting the dB threshold level to 20 dB, meaning that everything under that level was shortened to 0.5 seconds. Music segments were then cut out manually using knowledge of the waveforms. These modifications shortened the file by approximately 1 hour. The motivation behind editing the data is that silence would not result in any output, while music would definitely generate faulty outputs. The details of the dataset can be seen in Appendix I.

3.3 | Language Model Selection

The first idea was taking the logit outputs of the three seed models and then evaluating them together with a Hungarian language model. This way we could get the best outputs that are in the shape of a proper Hungarian word. Two different Hungarian Language Models are created for this purpose. The first language model is created as a 5-gram using Europarl Bilingual Corpus (Tiedemann, 2012). This model proved to be ineffective as 5-gram is too large for our case and the corpus size is relatively small. To get a better version, a 3-gram model is created using the OSCAR (Open Super-large Crawled ALMAnaCH coRpusand) corpus (Julien et. al., 2022). To use these language models together with the logits, we need a mapping from seed language alphabets to the target language alphabet that will consider all probabilities of each letter for each character while mapping. Unfortunately, this approach proved to be impractical considering our time and manpower limitations. Language models are used to compare all possible spelling correction suggestion combinations for all words in the output and to select the most probable combination. Notebook for creating language models can be seen in Appendix H.

3.4 | Output Modification

Initial output from the models was inaccurate due to the fact that analysis was made considering another language's phoneme and grapheme systems. We first attempted to edit output by assigning Hungarian grapheme to model language logits, however, this was ultimately put aside in favor of string replacements. We then attempted to manually track necessary changes in the output by mapping model language graphemes with their Hungarian counterparts. This process was found to be too time-consuming, so we implemented a program using DiffLib to automate tracking of the necessary changes (Appendix A). This code was based on a program shared by our advisor, Jim O'Regan. The code was run separately for each model's output and changes added to dictionaries which also tracked the frequency of the changes. All frequently occurring changes were added to our string replacement dictionary. Single letter replacements were often context-dependent, so the context surrounding these changes were manually analyzed and added to the replacement dictionary as necessary.

Following string replacements, we found that while output could be accurately determined fairly regularly by our native Hungarian-speaking team member, a large amount of spelling and word boundary issues remained. We then sought to implement a Hungarian autocorrect program in an attempt to further increase the readability of model output. Our first attempt utilized rather small uni- and bi-gram dictionaries and as such while the correct output would appear in the results, it was not often the most common result (Appendix B). At this point, we decided to create our own text corpus from which to create gram dictionaries. We used nearly 18GB of web crawled Hungarian data from the OSCAR corpus and implemented a program to create our own uni-, bi-, tri-, and quad-gram probability files from which to base our autocorrect program (Appendix C). This attempt ultimately failed due to the large time and memory commitment required to fully analyze the file to create the documents. We decided to move forward with only uni- and bi-gram information and used this information in a new autocorrect program to further correct output before voting (Appendix D).

3.5 | Voting

We based our voting program on the multilingual A-stabil method, which returns the most probable output based on frequency. We chose to vote at the character level rather than the word level due to the word boundary issues that still occurred in some model outputs. Once pre-edits on all models' output were completed, those outputs were run through our voting program (Appendix E). Characters that are agreed upon by all three models are appended to the final output. In the case that two models agree and only one disagrees, the agreed-upon input is appended to the final output. In the case of a three-way tie, a random choice is made between the outputs. In the case where one of the models fails to recognize a large amount of input speech, and its output has a .5 or larger difference in length from the largest model output, that output is disregarded during voting. Once the program has reached the end of the fragment, the final output is returned.

4| Encountered Challenges

Some of these challenges are already discussed throughout the report but since the aim of the project is learning, we would like to gather all challenges we encountered, both practical and theoretical, here and our solutions to them in one place.

- **Deep Learning Framework:** At the start of the project, our own knowledge was limited to Keras Framework. The HuggingFace library works best with PyTorch. Even though they recently announced compatibility with Tensorflow, all tutorials and most of the available models were created using PyTorch. This was introduced as a challenge since learning and using a new framework comes with its own challenges.

- **Audio Data Format**: One of the first problems we encountered. In our workspace, we were using Windows 10 with PyTorch. Unfortunately, the PyAudio library uses the SoundFile library in Windows systems to manipulate audio files and SoundFile doesn't support mp3 files. This was a huge problem because we couldn't work with the most common datasets like Common Voice using the HuggingFace library. As a solution, we used an external workspace Colab at the start, and later we switched manually downloading the datasets and converting the files to the appropriate formats. We also created our own corpus in wav format.

- **Memory Problems:** This was a problem throughout the whole project. At the start, since we were not going to do any training until the end, we were confident that our computers or Colab would be enough. The first problem came with the size of the good models. Naturally, better models have more parameters and their sizes are bigger. Since we were using 3 different models to create one output, we needed high RAM and/or GPU RAM spaces. Computers crashed constantly because of out-of-memory errors. On top of being bottlenecked by models, also using hours of audio data in the programs affected the memory greatly. We were able to hold no more than 2 minutes of data while working with all models together. The solution to this problem was to work offline, meaning that working with models one by one with batches of data and saving the outputs. Later, when all outputs are available we run our mapping and voting algorithms.

- **Custom Corpus:** Since we collected our data from YouTube, there were several problems associated with it. First of all, it was not easy to find videos that fulfilled both criteria we decided on in the beginning: the videos needed to have captions, and needed to be published with the Creative Commons license. While there are a lot of videos with Hungarian speech available on YouTube, selecting these two filters clearly eliminated a very large portion of them. We still managed to collect approximately 11 hours of data. We then found out that a significant amount of this material included music segments, silences, and also just noise. All noise, music, and unnecessary silences had to be removed in order to isolate speech. Another problem was the size of the samples. Since the length of the videos vary, we had samples ranging from 1 minute to 1 hour in length. Models and our computational resources are not suitable to deal with long audios. For labeling, we found the solution by dividing the data to 30-minute batches first and then while making inference we used 10-second chunks to process the data. This made the labeling possible but we still need to post-process the data to make it suitable for training of other models. For this we decided on a $5<x<15$ second window to slice the audios and not to slice speech mid word, we used the timestamps of the labels in order to decide where to slice.

- **Language Models:** These models were one of the core problems we had. The first problem was associated with the size of these models. We tried to work with a language model created from a small corpus but it didn't work well. When we tried to use a bigger corpus, the number of sentences processed was too high to handle in a tight time schedule. The corpora needed to be preprocessed in order to remove unrelated (not a word) characters from the sentences. After the processing, we tried to create a trigram language model together with unigram, bigram, trigram, and quadgram frequency files in order to use them for spell checking. There were more than 6 million samples and 2 million words but even though we waited for many hours, we didn't have time to use all the available data. Lastly, the problem was that all successful enough ASR models were using language models. We needed to use good enough models without a language model for our purposes. This situation resulted in using a Czech model with 24% WER.

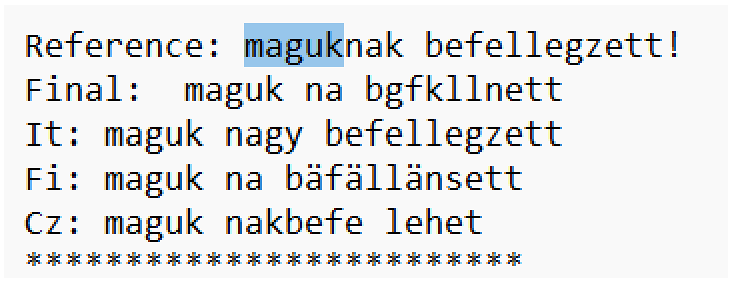
- **Non-Standardized Outputs and Changing Language Models:** Each time we tested a new language model, the outputs and therefore, the changes necessary to be made to convert the output to proper Hungarian graphemes, were altered. So with each new model, however many hours had been spent on mapping the grapheme changes were essentially thrown out as wasted time. The time waste problem was solved by writing a program utilizing DiffLib to semi-automate the change mapping, though a certain amount of hours was still required in order to map changes as accurately as possible. Another issue with mapping the output changes was that the output was not regular in the way it was expected. Though in theory the languages have overlapping phonemes that match to their own respective graphemes, the production of phones can affect how the ASR system "hears" and transcribes it. Initial dictionaries based solely on phoneme/grapheme matches were mostly unusable. This problem was solved by tracking the contexts surrounding necessary changes, looking at how the surrounding sounds affect output, and mapping changes as needed based on the surrounding context.

5| Results

The labeling as the result of the voting has achieved WER 99.02% and CER: 68.79% on the subset of Common Voice dataset. The labeling without voting, meaning that only using spell checking and language model has achieved WER 96.22% and CER 48.20% using only the Finnish model while Finnish raw output has WER 98.74% and CER 52.26%. These results show that alphabet mapping + spell checking + language model for combinations provide advantages but so little that we are not sure if it's worth the effort. Also getting a worse score after the voting means that we assigned wrong probabilities to languages.

If we analyze the unsuccessful results we get, we can make some suggestions. Firstly, continued effort, perhaps with tri- and quad-gram word dictionaries and character level bi-, tri-, and quad-gram dictionaries could help to return better final output. Moreover, the application of the ideal logit mapping could achieve better results. Most importantly, being able to have better models with lower WER score without a language model and better rules to map seed languages should improve the success rate.

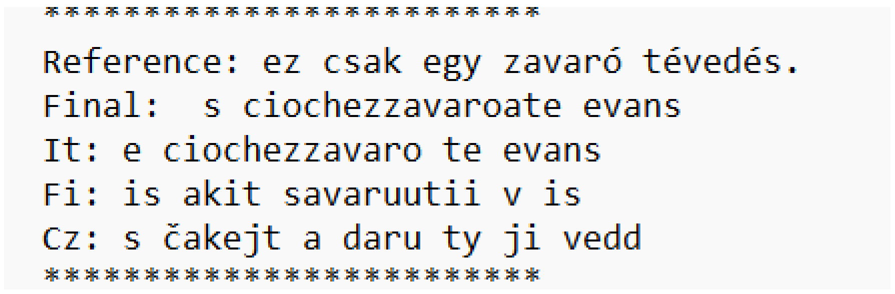
Instances of reference being close to the final output:



```
Reference: maguknak befellegzett!
Final:  maguk na bgfkllnett
It: maguk nagy befellegzett
Fi: maguk na bäfällänsett
Cz: maguk nakbefe lehet
*************************
```

The above example is one of the few sentences where the output is somewhat close to the reference. Word boundaries are almost in place, and most of the characters are predicted correctly.

Instances of reference being far from the final output:



```
*************************
Reference: ez csak egy zavaró tévedés.
Final:  s ciochezzavaroate evans
It: e ciochezzavaro te evans
Fi: is akit savaruutii v is
Cz: s čakejt a daru ty ji vedd
*************************
```

The above example is an illustration of an instance where the prediction is far from the reference. Most letters are unfortunately predicted incorrectly, and word boundaries are also out of place.

6| Conclusion

Overall, this experience gave us a better understanding of the work and processes necessary to build a fully functioning ASR system, as well as gave us the opportunity to build our own corpora and experience data collection first-hand and the difficulties and dilemmas that can occur during this process. We have left this project with a greater understanding and respect for the concepts first presented to us in lectures, as well as a healthy amount of patience for the currently existing ASR and speech dialogue systems that we encounter in our everyday life - so much of the work that goes into these systems is generally unrecognized by the general population and it was quite humbling to work on such a project ourselves.

References

Mihajlik, Péter & Balog, András & Gráczi, T. & Kohári, Anna & Tarján, Balázs & Mády, K.. (2022). BEA-Base: A Benchmark for ASR of Spontaneous Hungarian.

Audacity Team (2021). Audacity(R): Free Audio Editor and Recorder [Computer application]. Version 3.0.0 retrieved March 17th 2021 from https://audacityteam.org/.

J. Abadji, P. Ortiz Suarez, L. Romary, and B. B. Sagot, “Towards a Cleaner Document-Oriented Multilingual Crawled Corpus,” Jan. 2022. doi: 10. 48550/arxiv.2201.06642. [Online]. Available: https://arxiv.org/abs/2201.06642v1.

R. Ardila, M. Branson, K. Davis, et al., “Common Voice: A Massively Multilingual Speech Corpus,” LREC 2020 - 12th International Conference on Language Resources and Evaluation, Conference Proceedings, pp. 4218–4222, Dec. 2019. doi: 10.48550/arxiv.1912.06670. [Online]. Available:https://arxiv.org/abs/1912.06670v2.

T. Wolf, L. Debut, V. Sanh, et al., “HuggingFace’s Transformers: State-of-the-art Natural Language Processing,” arXiv preprint arXiv:1910.03771, Oct. 2019. doi: 10.48550/arxiv.1910.03771. [Online]. Available: https://arxiv.org/abs/1910.03771v5.

A. Baevski, H. Zhou, A. Mohamed, and M. Auli, “wav2vec 2.0: A Frame-work for Self-Supervised Learning of Speech Representations,” Advances in Neural Information Processing Systems, vol. 2020-December, Jun. 2020, issn: 10495258. doi: 10.48550/arxiv.2006.11477. [Online]. Available:https://arxiv.org/abs/2006.11477v3.

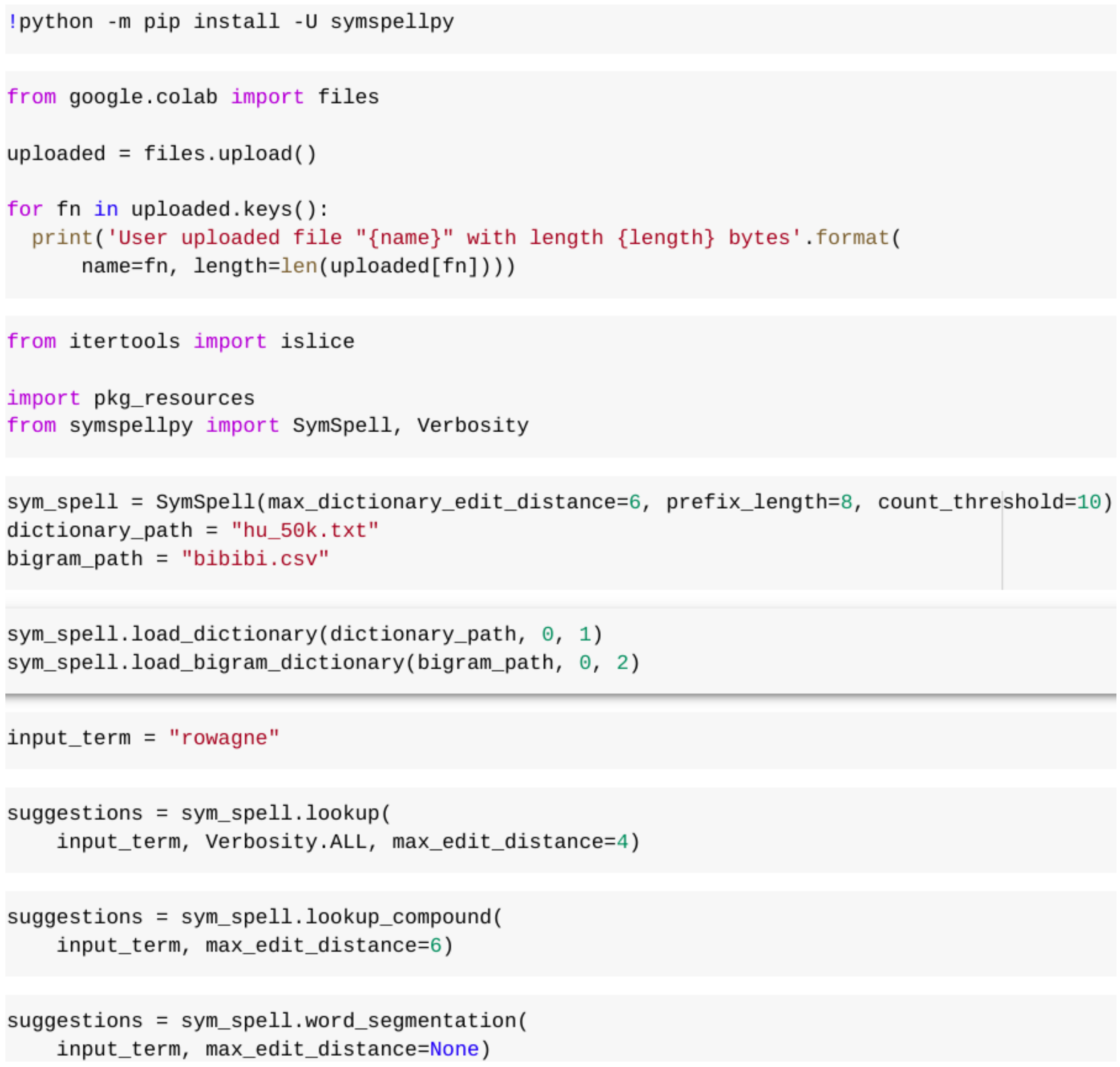
J. Tiedemann, Parallel Data, Tools and Interfaces in OPUS - ACL Anthology. [Online]. Available: https://aclanthology.org/L12-1246/

Appendix

Appendix A. DiffLib program to track necessary changes in output from model language to Hungarian

```python
from difflib import SequenceMatcher
def print_replacements(texta, textb):
    sm = SequenceMatcher(a=texta, b=textb)
    for op, a_start, a_end, b_start, b_end in sm.get_opcodes():
        frag_a = texta[a_start:a_end]
        frag_b = textb[b_start:b_end]
        a_pre = a_start - 1 if a_start > 0 else 0
        b_pre = b_start - 1 if b_start > 0 else 0
        a_post = a_end + 1 if a_end < (len(texta) - 1) else a_end
        b_post = b_end + 1 if b_end < (len(textb) - 1) else b_end
        if op == "equal":
            continue
        elif op == "delete":
            if frag_a == " ":
                continue
            elif len(frag_a) == len(frag_b):
              fi_changes[(texta[a_pre:a_post], textb[b_pre:b_post])] += 1
            else:
              fi_changes[(texta[a_pre:a_post], textb[b_pre:b_post])] += 1
        elif op == "insert":
            if frag_b == " ":
                continue
            elif len(frag_a) == len(frag_b):
              fi_changes[(texta[a_pre:a_post], textb[b_pre:b_post])] += 1
            else:
              fi_changes[(texta[a_pre:a_post], textb[b_pre:b_post])] += 1
        elif op == "replace":
          if len(frag_a) == len(frag_b):
            fi_changes[((texta[a_start-2:a_end+2]), frag_a, frag_b, (textb[b_start-2:b_end+2]))] += 1
          else:
            fi_changes[((texta[a_start-2:a_end+2]), frag_a, frag_b, (textb[b_start-2:b_end+2]))] += 1
```

Appendix B First Autocorrect Notebook



```
!python -m pip install -U symspellpy
```

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

```
from itertools import islice

import pkg_resources
from symspellpy import SymSpell, Verbosity
```

```
sym_spell = SymSpell(max_dictionary_edit_distance=6, prefix_length=8, count_threshold=10)
dictionary_path = "hu_50k.txt"
bigram_path = "bibibi.csv"
```

```
sym_spell.load_dictionary(dictionary_path, 0, 1)
sym_spell.load_bigram_dictionary(bigram_path, 0, 2)
```

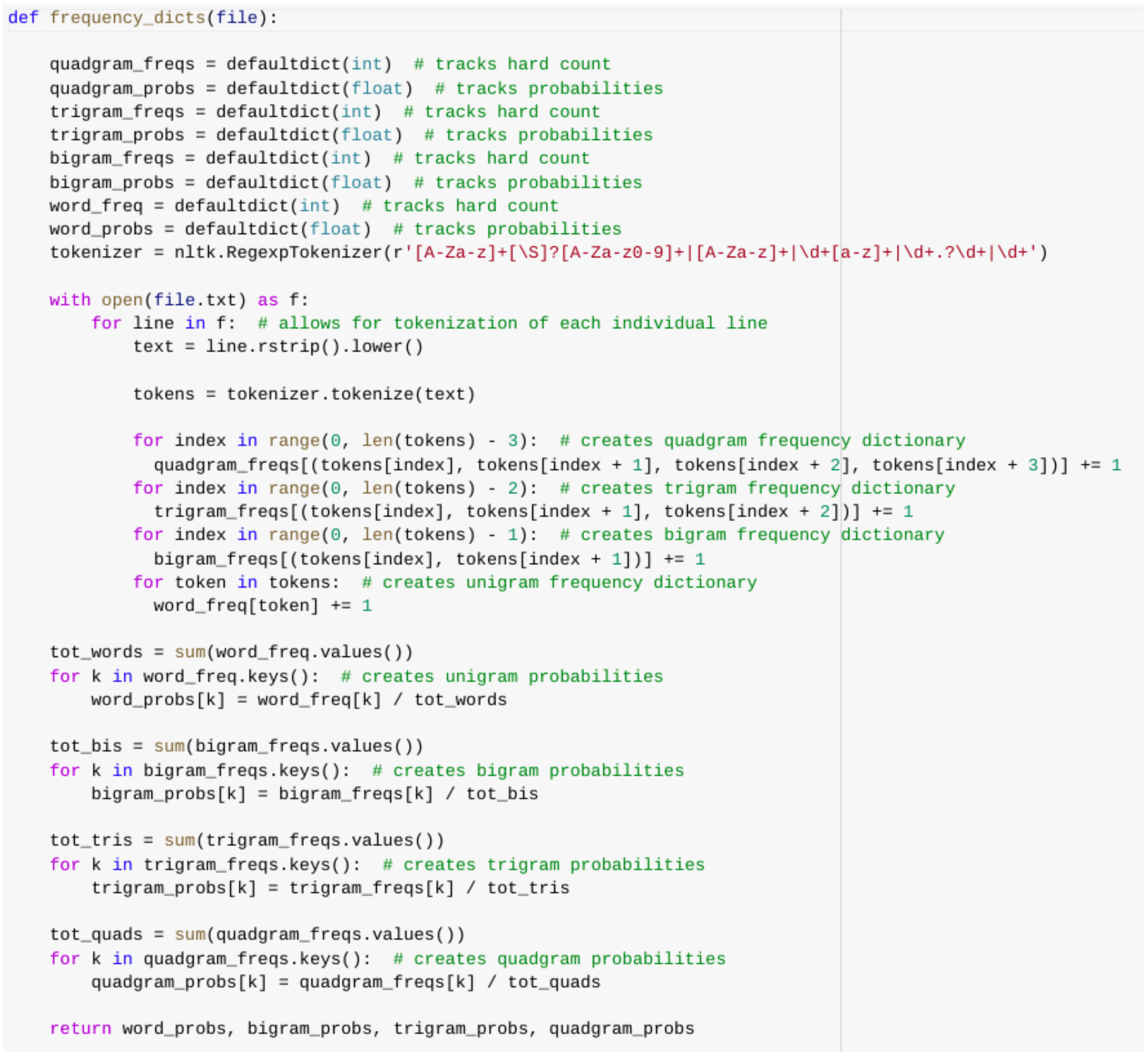
```
input_term = "rowagne"
```

```
suggestions = sym_spell.lookup(
    input_term, Verbosity.ALL, max_edit_distance=4)
```

```
suggestions = sym_spell.lookup_compound(
    input_term, max_edit_distance=6)
```

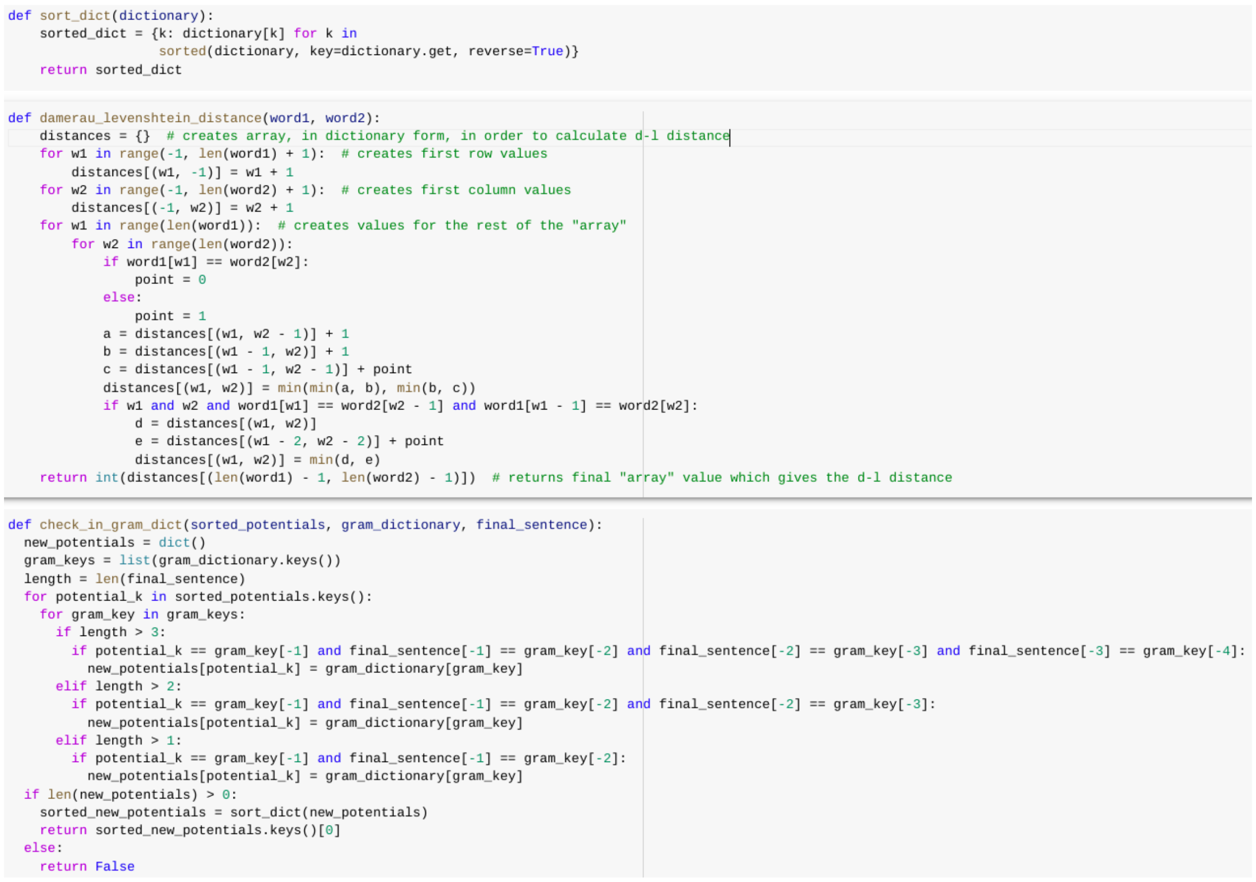
```
suggestions = sym_spell.word_segmentation(
    input_term, max_edit_distance=None)
```

Appendix C Program to create own word n-gram files



```python
def frequency_dicts(file):

    quadgram_freqs = defaultdict(int)  # tracks hard count
    quadgram_probs = defaultdict(float)  # tracks probabilities
    trigram_freqs = defaultdict(int)  # tracks hard count
    trigram_probs = defaultdict(float)  # tracks probabilities
    bigram_freqs = defaultdict(int)  # tracks hard count
    bigram_probs = defaultdict(float)  # tracks probabilities
    word_freq = defaultdict(int)  # tracks hard count
    word_probs = defaultdict(float)  # tracks probabilities
    tokenizer = nltk.RegexpTokenizer(r'[A-Za-z]+[\S]?[A-Za-z0-9]+|[A-Za-z]+|\d+[a-z]+|\d+.?\d+|\d+')

    with open(file.txt) as f:
        for line in f:  # allows for tokenization of each individual line
            text = line.rstrip().lower()

            tokens = tokenizer.tokenize(text)

            for index in range(0, len(tokens) - 3):  # creates quadgram frequency dictionary
              quadgram_freqs[(tokens[index], tokens[index + 1], tokens[index + 2], tokens[index + 3])] += 1
            for index in range(0, len(tokens) - 2):  # creates trigram frequency dictionary
              trigram_freqs[(tokens[index], tokens[index + 1], tokens[index + 2])] += 1
            for index in range(0, len(tokens) - 1):  # creates bigram frequency dictionary
              bigram_freqs[(tokens[index], tokens[index + 1])] += 1
            for token in tokens:  # creates unigram frequency dictionary
              word_freq[token] += 1

    tot_words = sum(word_freq.values())
    for k in word_freq.keys():  # creates unigram probabilities
        word_probs[k] = word_freq[k] / tot_words

    tot_bis = sum(bigram_freqs.values())
    for k in bigram_freqs.keys():  # creates bigram probabilities
        bigram_probs[k] = bigram_freqs[k] / tot_bis

    tot_tris = sum(trigram_freqs.values())
    for k in trigram_freqs.keys():  # creates trigram probabilities
        trigram_probs[k] = trigram_freqs[k] / tot_tris

    tot_quads = sum(quadgram_freqs.values())
    for k in quadgram_freqs.keys():  # creates quadgram probabilities
        quadgram_probs[k] = quadgram_freqs[k] / tot_quads

    return word_probs, bigram_probs, trigram_probs, quadgram_probs
```

Appendix D Second AutoCorrect Program



```python
def sort_dict(dictionary):
    sorted_dict = {k: dictionary[k] for k in
                   sorted(dictionary, key=dictionary.get, reverse=True)}
    return sorted_dict
```

```python
def damerau_levenshtein_distance(word1, word2):
    distances = {}  # creates array, in dictionary form, in order to calculate d-l distance
    for w1 in range(-1, len(word1) + 1):  # creates first row values
        distances[(w1, -1)] = w1 + 1
    for w2 in range(-1, len(word2) + 1):  # creates first column values
        distances[(-1, w2)] = w2 + 1
    for w1 in range(len(word1)):  # creates values for the rest of the "array"
        for w2 in range(len(word2)):
            if word1[w1] == word2[w2]:
                point = 0
            else:
                point = 1
            a = distances[(w1, w2 - 1)] + 1
            b = distances[(w1 - 1, w2)] + 1
            c = distances[(w1 - 1, w2 - 1)] + point
            distances[(w1, w2)] = min(min(a, b), min(b, c))
            if w1 and w2 and word1[w1] == word2[w2 - 1] and word1[w1 - 1] == word2[w2]:
                d = distances[(w1, w2)]
                e = distances[(w1 - 2, w2 - 2)] + point
                distances[(w1, w2)] = min(d, e)
    return int(distances[(len(word1) - 1, len(word2) - 1)])  # returns final "array" value which gives the d-l distance
```

```python
def check_in_gram_dict(sorted_potentials, gram_dictionary, final_sentence):
  new_potentials = dict()
  gram_keys = list(gram_dictionary.keys())
  length = len(final_sentence)
  for potential_k in sorted_potentials.keys():
    for gram_key in gram_keys:
      if length > 3:
        if potential_k == gram_key[-1] and final_sentence[-1] == gram_key[-2] and final_sentence[-2] == gram_key[-3] and final_sentence[-3] == gram_key[-4]:
          new_potentials[potential_k] = gram_dictionary[gram_key]
      elif length > 2:
        if potential_k == gram_key[-1] and final_sentence[-1] == gram_key[-2] and final_sentence[-2] == gram_key[-3]:
          new_potentials[potential_k] = gram_dictionary[gram_key]
      elif length > 1:
        if potential_k == gram_key[-1] and final_sentence[-1] == gram_key[-2]:
          new_potentials[potential_k] = gram_dictionary[gram_key]
  if len(new_potentials) > 0:
    sorted_new_potentials = sort_dict(new_potentials)
    return sorted_new_potentials.keys()[0]
  else:
    return False
```

Appendix E Voting Algorithm (multi-page)

```
from difflib import SequenceMatcher
import random
```

```
def sort_dict(dictionary):
    """Sorts a dictionary by key in reverse order
    (largest to smallest, reverse alphabetical, etc.)"""

    sorted_dict = {k: dictionary[k] for k in
                   sorted(dictionary, key=dictionary.get, reverse=True)}

    return sorted_dict
```

```
from collections import defaultdict

def choice_dict(choices):
  count_dict = defaultdict(int)

  for item in choices:
    if isinstance(item, str):
      count_dict[item] += 1
    else:
      for c in item:
        count_dict[c] += 1

  return count_dict
```

```
def make_choice(the_dict):
  sorted_count = sort_dict(the_dict)
  keys = list(sorted_count.keys())
  values = list(sorted_count.values())

  if len(values) > 1:
    for i, v in enumerate(values):
      if i == 0:
        if v == values[i+1]:
          return str(random.choice((keys[0], keys[1]))) #currently returns random choice from the tied keys
        else:
          return keys[0]
  else:
    return keys[0]
```

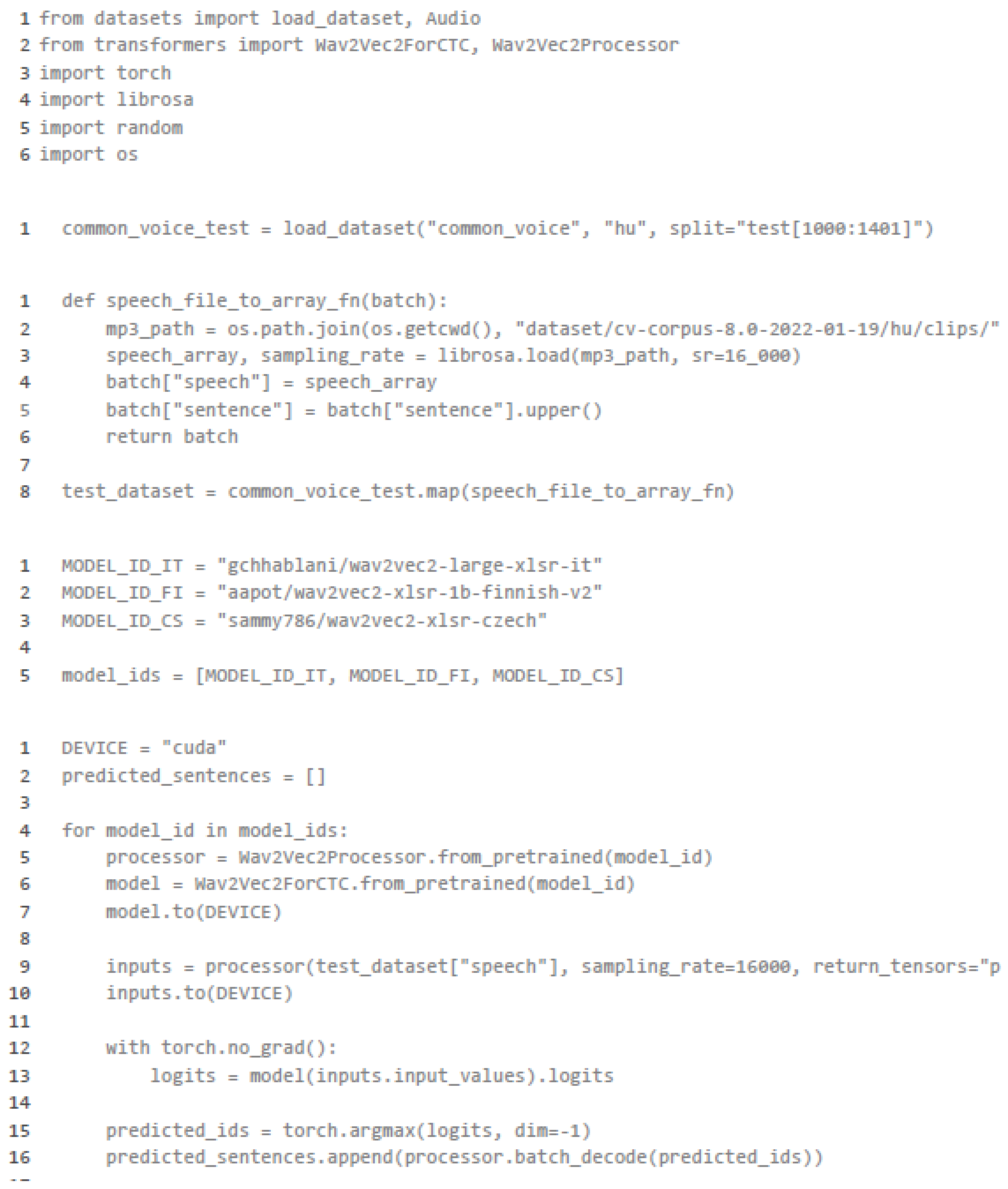
```
def sentence_processing(it_sent, fi_sent, cz_sent=None, reference=None): #takes strings as input, only req 2 langs #USE THIS ONE
  l12_sent = []

  final_output = ' '

  if cz_sent: #when 3 usable sents
    it_fi = SequenceMatcher(None, it_sent, fi_sent)
    fi_cz = SequenceMatcher(None, fi_sent, cz_sent)
    cz_it = SequenceMatcher(None, cz_sent, it_sent)

    it_fi_sent = []
    fi_cz_sent = []
    cz_it_sent = []

    for op, a_start, a_end, b_start, b_end in it_fi.get_opcodes():
      frag_it = {it_sent[a_start:a_end]}
      frag_fi = {fi_sent[b_start:b_end]}

      if op == "equal":
        for item in frag_it:
          it_fi_sent.append(item)
      elif op == "replace":
        for i in frag_it:
          for j in frag_fi:
            it_fi_sent.append((i,j))

    for op, a_start, a_end, b_start, b_end in fi_cz.get_opcodes():
      frag_fi = {fi_sent[a_start:a_end]}
      frag_cz = {cz_sent[b_start:b_end]}

      if op == "equal":
        for item in frag_fi:
          fi_cz_sent += item
      elif op == "replace":
        for i in frag_fi:
          for j in frag_cz:
            fi_cz_sent.append((i,j))

    for op, a_start, a_end, b_start, b_end in cz_it.get_opcodes():
      frag_cz = {cz_sent[a_start:a_end]}
      frag_it = {it_sent[b_start:b_end]}

      if op == "equal":
        for item in frag_cz:
          cz_it_sent += item
      elif op == "replace":
        for i in frag_cz:
```

```
    for i in frag_cz:
      for j in frag_it:
        cz_it_sent.append((i,j))

for i, item in enumerate(zip(it_fi_sent, fi_cz_sent, cz_it_sent)):
  it_fi_item = item[0]
  fi_cz_item = item[1]
  cz_it_item = item[2]


  if isinstance(it_fi_item, str) and isinstance(
    fi_cz_item, str) and isinstance(cz_it_item, str): #when all agree
      final_output += it_fi_item


  elif isinstance(it_fi_item, tuple) and isinstance(
      fi_cz_item, tuple) and isinstance(cz_it_item, tuple): #when none agree
      it_choice = it_fi_item[0]
      fi_choice = fi_cz_item[0]
      cz_choice = cz_it_item[0]

      c_dict = choice_dict([it_choice, fi_choice, cz_choice])
      final_choice = make_choice(c_dict)

      final_output += final_choice

  elif isinstance(it_fi_item, tuple) and isinstance(fi_cz_item, tuple):
    it_choice = it_fi_item[0]
    fi_choice = fi_cz_item[0]
    cz_choice = cz_it_item

    c_dict = choice_dict([it_choice, fi_choice, cz_choice])
    final_choice = make_choice(c_dict)

    final_output += final_choice

  elif isinstance(fi_cz_item, tuple) and isinstance(cz_it_item, tuple):
    it_choice = it_fi_item
    fi_choice = fi_cz_item[0]
    cz_choice = cz_it_item[0]

    c_dict = choice_dict([it_choice, fi_choice, cz_choice])
    final_choice = make_choice(c_dict)

    final_output += final_choice

  elif isinstance(cz_it_item, tuple) and isinstance(it_fi_item, tuple):
    it_choice = it_fi_item[0]
```

```
      it_choice = it_fi_item[0]
      fi_choice = fi_cz_item
      cz_choice = cz_it_item[0]

      c_dict = choice_dict([it_choice, fi_choice, cz_choice])
      final_choice = make_choice(c_dict)

      final_output += final_choice

    elif isinstance(cz_it_item, tuple):
      it_choice = it_fi_item
      fi_choice = fi_cz_item
      cz_choice = cz_it_item[0]

      c_dict = choice_dict([it_choice, fi_choice, cz_choice])
      final_choice = make_choice(c_dict)

      final_output += final_choice

    elif isinstance(fi_cz_item, tuple):
      it_choice = it_fi_item
      fi_choice = fi_cz_item[0]
      cz_choice = cz_it_item

      c_dict = choice_dict([it_choice, fi_choice, cz_choice])
      final_choice = make_choice(c_dict)

      final_output += final_choice

    elif isinstance(it_fi_item, tuple):
      it_choice = it_fi_item[0]
      fi_choice = fi_cz_item
      cz_choice = cz_it_item

      c_dict = choice_dict([it_choice, fi_choice, cz_choice])
      final_choice = make_choice(c_dict)

      final_output += final_choice

else: #only 2 usabe sents
    it_fi = SequenceMatcher(None, it_sent, fi_sent)
    it_fi_sent = []

    for op, a_start, a_end, b_start, b_end in it_fi.get_opcodes():
      frag_it = {it_sent[a_start:a_end]}
      frag_fi = {fi_sent[b_start:b_end]}

      if op == "equal":
```

```
      if op == "equal":
        for item in frag_it:
          final_output += item

      elif op == "replace":
        for i in frag_it:
          for j in frag_fi:
            final_output += str(random.choice([i, j]))


return final_output
```

Appendix F Inference Snippet



```
from datasets import load_dataset, Audio
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor
import torch
import librosa
import random
import os
```

```
common_voice_test = load_dataset("common_voice", "hu", split="test[1000:1401]")
```

```
def speech_file_to_array_fn(batch):
    mp3_path = os.path.join(os.getcwd(), "dataset/cv-corpus-8.0-2022-01-19/hu/clips/"
    speech_array, sampling_rate = librosa.load(mp3_path, sr=16_000)
    batch["speech"] = speech_array
    batch["sentence"] = batch["sentence"].upper()
    return batch

test_dataset = common_voice_test.map(speech_file_to_array_fn)
```

```
MODEL_ID_IT = "gchhablani/wav2vec2-large-xlsr-it"
MODEL_ID_FI = "aapot/wav2vec2-xlsr-1b-finnish-v2"
MODEL_ID_CS = "sammy786/wav2vec2-xlsr-czech"

model_ids = [MODEL_ID_IT, MODEL_ID_FI, MODEL_ID_CS]
```

```
DEVICE = "cuda"
predicted_sentences = []

for model_id in model_ids:
    processor = Wav2Vec2Processor.from_pretrained(model_id)
    model = Wav2Vec2ForCTC.from_pretrained(model_id)
    model.to(DEVICE)

    inputs = processor(test_dataset["speech"], sampling_rate=16000, return_tensors="p
    inputs.to(DEVICE)

    with torch.no_grad():
        logits = model(inputs.input_values).logits

    predicted_ids = torch.argmax(logits, dim=-1)
    predicted_sentences.append(processor.batch_decode(predicted_ids))
```

Appendix G Error Calculation Snippet

```
from datasets import load_metric
```

```
wer = load_metric("wer")
cer = load_metric("cer")
```

```
wer_score = 100 * wer.compute(predictions=all_final, references=final_ref)
cer_score = 100 * cer.compute(predictions=all_final, references=final_ref)
```

```
print(f"WER: {wer_score}%")
print(f"CER: {cer_score:.2f}%")
```

Appendix H Language Model Creation Notebook

```
!pip install datasets
!pip install transformers
```

```
from datasets import load_dataset

dataset = load_dataset("oscar", language="hu", split="train")
```

```
chars_to_ignore_regex = [",", "?", "¿", ".", "!", "¡", ";", "；", ":", '""', "%", '"'
                         "؟", "،", "।", "॥", "«", "»", "„", "“", "”", "「", "」", "‘", "’",
                         "{", "}", "=", "`", "_", "+", "<", ">", "…", "–", "°", "´", "ʾ", "
                         "、", "└─", "─┐", "·", "~", "﹏", "，", "｛", "｝", "（", "）", "［'
                         "『", "』", "〝", "〟", "⟨", "⟩", "〜", "：", "！", "？", "♪", "؛", "
```

```
import re

def extract_text(batch):
  text = batch["text"]
  batch["text"] = re.sub(chars_to_ignore_regex, "", text.lower())
  return batch
```

```
dataset = dataset.map(extract_text, remove_columns="id")
```

```
with open("text.txt", "w") as file:
  file.write(" ".join(dataset["text"]))
```

```
!sudo apt install build-essential cmake libboost-system-dev libboost-thread-dev libbo
```

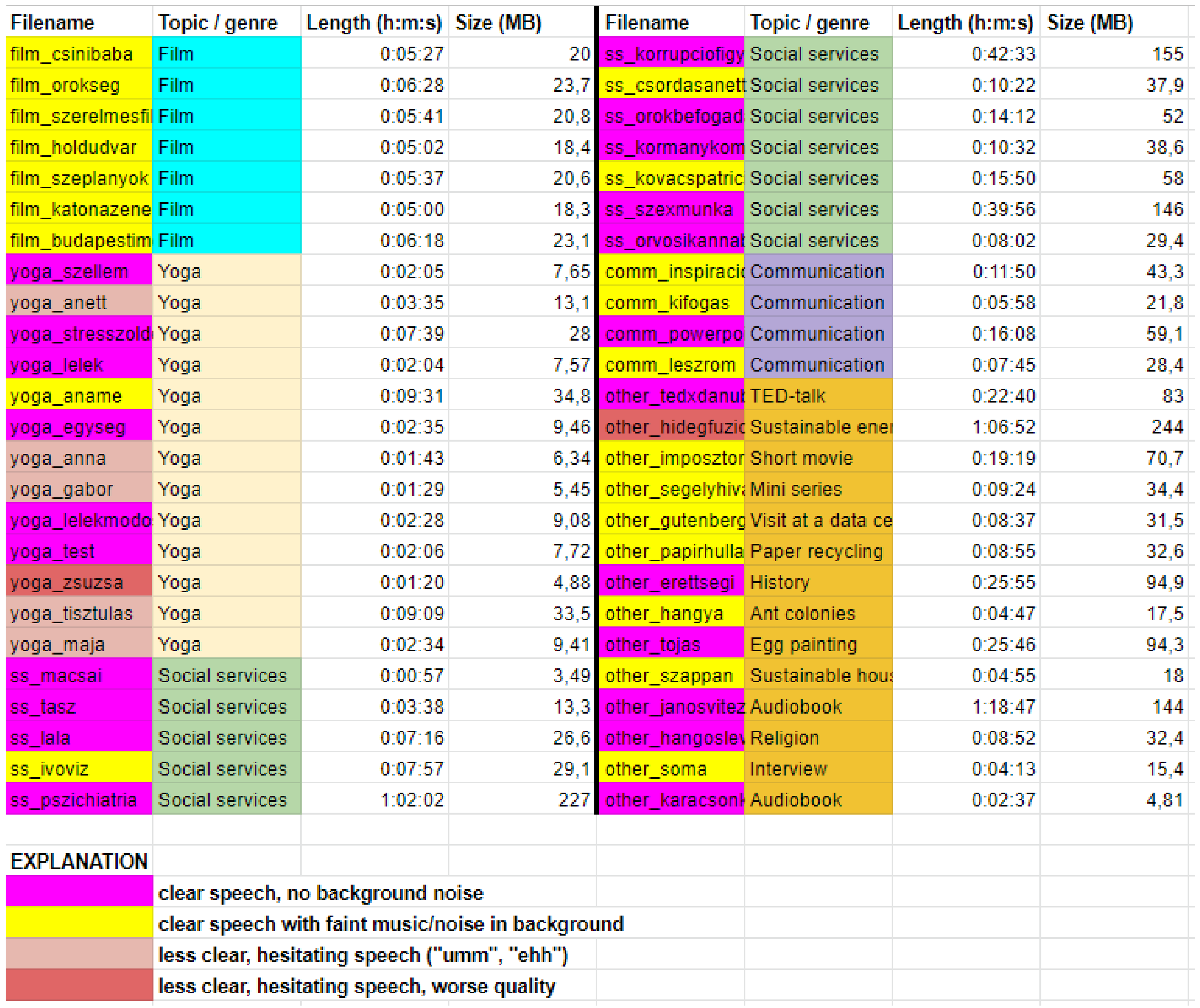
```
!wget -O - https://kheafield.com/code/kenlm.tar.gz | tar xz
```

```
!mkdir kenlm/build && cd kenlm/build && cmake .. && make -j2
```

```
!kenlm/build/bin/lmplz -o 3 <"text.txt" > "3gram_hu.arpa"
```

```
with open("3gram_hu.arpa", "r") as read_file, open("3gram_correct_hu.arpa", "w") as wr
  has_added_eos = False
  for line in read_file:
    if not has_added_eos and "ngram 1=" in line:
      count=line.strip().split("=")[-1]
      write_file.write(line.replace(f"{count}", f"{int(count)+1}"))
    elif not has_added_eos and "<s>" in line:
      write_file.write(line)
      write_file.write(line.replace("<s>", "</s>"))
      has_added_eos = True
    else:
      write_file.write(line)
```

Appendix I Our YouTube-corpus



Filename	Topic / genre	Length (h:m:s)	Size (MB)	Filename	Topic / genre	Length (h:m:s)	Size (MB)
film_csinibaba	Film	0:05:27	20	ss_korrupciofigy	Social services	0:42:33	155
film_orokseg	Film	0:06:28	23,7	ss_csordasanett	Social services	0:10:22	37,9
film_szerelmesfil	Film	0:05:41	20,8	ss_orokbefogad	Social services	0:14:12	52
film_holdudvar	Film	0:05:02	18,4	ss_kormanykom	Social services	0:10:32	38,6
film_szeplanyok	Film	0:05:37	20,6	ss_kovacspatric	Social services	0:15:50	58
film_katonazene	Film	0:05:00	18,3	ss_szexmunka	Social services	0:39:56	146
film_budapestim	Film	0:06:18	23,1	ss_orvosikannat	Social services	0:08:02	29,4
yoga_szellem	Yoga	0:02:05	7,65	comm_inspiracio	Communication	0:11:50	43,3
yoga_anett	Yoga	0:03:35	13,1	comm_kifogas	Communication	0:05:58	21,8
yoga_stresszold	Yoga	0:07:39	28	comm_powerpo	Communication	0:16:08	59,1
yoga_lelek	Yoga	0:02:04	7,57	comm_leszrom	Communication	0:07:45	28,4
yoga_aname	Yoga	0:09:31	34,8	other_tedxdanub	TED-talk	0:22:40	83
yoga_egyseg	Yoga	0:02:35	9,46	other_hidegfuzic	Sustainable ener	1:06:52	244
yoga_anna	Yoga	0:01:43	6,34	other_imposztor	Short movie	0:19:19	70,7
yoga_gabor	Yoga	0:01:29	5,45	other_segelyhiva	Mini series	0:09:24	34,4
yoga_lelekmodo	Yoga	0:02:28	9,08	other_gutenberg	Visit at a data ce	0:08:37	31,5
yoga_test	Yoga	0:02:06	7,72	other_papirhulla	Paper recycling	0:08:55	32,6
yoga_zsuzsa	Yoga	0:01:20	4,88	other_erettsegi	History	0:25:55	94,9
yoga_tisztulas	Yoga	0:09:09	33,5	other_hangya	Ant colonies	0:04:47	17,5
yoga_maja	Yoga	0:02:34	9,41	other_tojas	Egg painting	0:25:46	94,3
ss_macsai	Social services	0:00:57	3,49	other_szappan	Sustainable hous	0:04:55	18
ss_tasz	Social services	0:03:38	13,3	other_janosvitez	Audiobook	1:18:47	144
ss_lala	Social services	0:07:16	26,6	other_hangoslev	Religion	0:08:52	32,4
ss_ivoviz	Social services	0:07:57	29,1	other_soma	Interview	0:04:13	15,4
ss_pszichiatria	Social services	1:02:02	227	other_karacsonl	Audiobook	0:02:37	4,81

EXPLANATION	
🟪	clear speech, no background noise
🟨	clear speech with faint music/noise in background
🟧	less clear, hesitating speech ("umm", "ehh")
🟥	less clear, hesitating speech, worse quality