DOI: https://doi.org/10.14232/wpcl.2022.7.1

Transcribe: a Web-Based Linguistic Transcription Tool

Ludwig Maximilian Breuer, Arnold Graf, Tahel Singer and Markus Pluschkovits

1. Introduction and Overview

Transcribe is a transcription application currently being developed in the context of the task-cluster E of the Special Research Programme (FWF F 60) "German in Austria. Variation – Contact – Perception" (abb. as SFB DiOE in the following). The present paper aims to describe the principles of its development in the context of the SFB DiOE, its solutions for audio processing and visualization, and its use-case as a transcription tool for linguistics in general and linguistic research groups in particular. We conclude with a practical application of automatic translation of eye-dialect transcripts of Viennese to orthographic standard German transcripts, utilizing the automatic tokenization and multiple tiers of Transcribe.

Transcribe was developed as an offline-first web app. The main advantages of web-based applications are obvious: web applications are compatible with all operating systems that are able to run modern browsers, and thereby eliminate the issue of porting software. The distribution of web-based applications is extremely simple in comparison to traditional desktop software – instead of going through an installation process, the users simply navigate to a URL in their browser, which can easily be bookmarked. Updates on the app are transparent and do not require any action from the user.

Transcribe was developed with linguistic research groups in mind, and with the goal to support and enhance their workflow with a variety of functions. In contrast to many other common applications, Transcribe utilizes a client-server-architecture, which enables a link to a central back-end. This central back-end can host all the relevant (linguistic) research data, as is the case in the SFB DiOE (see DiOE 2020: DioeDB).

However, Transcribe is also designed to support smaller-scale projects, where a centralized server storing the research data is either unfeasible or unnecessary. For this reason (and to increase resilience against high latency and network issues), Transcribe was conceived as an offline-first application (see Linklater et al. 2018: 260), meaning that it can also be used locally, while still retaining almost all of its functions. Transcribe can therefore either be used as a stand-alone tool, or as a part of a toolkit.

Some of the basic tenants of Transcribe can be summarized by the concepts of user-centered design, collaboration, and flexibility. The goal of the development is to combine the smooth interactivity of modern, native desktop apps with the benefits of browser-based web applications. To facilitate this, some technical improvements had to be made, which shall be showcased in the following paragraphs.

The development of Transcribe is recorded in a public GitHub repository (see DiOE 2020: Transcribe). The aim is to disseminate the code as widely as possible and adhere to basic concepts of the Open-Science movement (see European Commission 2016: 33).

Nicole Palliwoda (ed.): *Data Processing and Visualization in Variational Linguistics/Dialectology*. Working Papers in Corpus Linguistics and Digital Technologies: Analyses and methodology. Vol.7. 2022.

2. User Interface

2.1. Principles of Transcribe's UI-Design

The intended use of Transcribe is within both a professional and scientific setting, the userbase is therefore conceived as any such people that are concerned with linguistic data in these contexts. They use the tool repeatedly, for longer periods of time, and utilize its functions parallel or sequential during different steps of their workflow. For this reason, the goalpost for Transcribe's UI was what has been termed the sophisticated user (see Debasmita and Ardhendu 2015: 130) and their requirements. In the spirit of user-centered design, the development process has been accompanied by periodic meetings between the developers and users of Transcribe in order to identify issues and improve the UI iteratively (Chammas, Quaresma and Mont'Alvão 2015: 5398).

Despite the orientation towards sophisticated users, Transcribe aims to provide an intuitive user interface for first-timers. This was enabled by following some core tenets of the philosophy of Interaction Design (see Debasmita and Ardhendu: 131–134 and Nielsen and Molich 1990: 251pp). Among them are the following:

- The principle of familiarity (see Raskin 1994: 17): If possible, Transcribe does not introduce new or formerly unknown terms, icons, or means of interacting with the program. Transcribe relies on established shortcuts, either known from other transcription software (e.g. Exmaralda), or familiar from the operating systems of current desktop computers. For example, Transcribe enables users to select multiple elements by keeping the shift or control key pressed, it offers context-sensitive menus, and supports pinch-to-zoom to zoom in or out of the waveform-visualization, which are all features users usually know from their operating system.
- The principle of discoverability: The bulk of Transcribe's features can be found in the submenu actions, which also highlights their respective keyboard shortcuts. More specialized functions appear automatically in those contexts where they are actually needed – e.g. a pop-up on-screen keyboard for IPA characters in the tier dedicated for phonetic transcription, or in the search bar.
- The principle of consistency (see Nielsen 1999: 2): Visual elements which resemble each other should have similar functions and support similar interactions. A coherent design scheme, as was utilized for Transcribe, provides an implicit system of rules of interaction for both the users and the designers.
- The principle of safety of interaction: in order to facilitate fluid interaction with an application, especially those that are concerned with the entry of data, mechanisms that prevent user errors (or enable the users to undo them painlessly) must be put in place. The application should be what is called a "relaxed environment," which does not require constant, intense focus (Debasmita and Ardhendu 2015: 132). To help create such a relaxed environment, Transcribe uses a generous undo-redo-system, locally saving up to 1500 individual operations, displaying them visually in a dedicated history, and allowing for selective undoing of them. Undoing and redoing individual operations can be done by using the standard keyboard shortcut of the end-user's device, which hopefully makes users intuitively default to them. Additionally, Transcribe periodically saves the latest version of a current transcript in a local cache as long as the transcript is being worked on, which means that it can be recovered even after a system crash.

2.2. Organization and Implementation of the UI

As in almost all subdisciplines of software development, developing a graphical UI employs the principle of isolation of functional units (cf. Krasner/Pope 1988: 26). While the benefits of this approach can be assumed to be widely known, the following summarizes them succinctly:

> Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit (Krasner and Pope 1988: 26).

The UI of Transcribe is being developed with the Vue.js framework in accordance with the principles stated above. For most of its interactive control elements, it employs John Leider's Vuetify (see Leider 2020) as user interface library.

Vue.js is a declarative framework for the creation of UIs for web applications. In contrast to the more traditional Model-View-Controller (MVC) architecture, Vue.js implements the more modern concept of the Model-View-ViewModel (MVVM), which allows for the binding of individual parts of the UI directly to the data model. If the data model is changed, Vue.js identifies necessary updates on the level of the graphic UI (GUI) and applies them (see You 2020: Vue.js Introduction). This mechanism takes work-load off of the developers and reduces the risk of possible discrepancies between data model and the interface of the program. A drawback of this type of architecture is its comparatively high memory consumption in contrast to the MVC-architecture (see Gossman 2006).

Like other libraries and frameworks, Vue.js fosters the development of modular, component-oriented applications by employing single-file-components, i.e. small, self-contained program parts. These component parts usually have very limited, but specific functions – e.g. portrayal of a menu or a text box – and define fixed interfaces to communicate with other components. Combining these short, manageable components creates a component tree, which serves as GUI.

To further the development of Transcribe, an additional library of often-used standard components was employed as well. Vuetify (see Leider 2020: Vuetify) utilizes Material Design, a design system launched by Google (Google 2014: Material Design), and contains several so-called Widgets. While Transcribe is not completely bound to the rules and principles of Material Design, it orients itself on them. As a result, the inherent coherence of the rules and requirements of the Material Design framework, implemented in Vuetify, helps to facilitate the visual and functional consistency of Transcribe.

3. Audio Processing and Visualization in Transcribe

Initially, three important requirements for the program were identified:

(1) A speed and response-time comparable to that of a native desktop application
(2) The implementation as a web-based application
(3) The possibility of utilizing the application without a server back-end.

These requirements result in the need for an efficient, client-based method of decoding, visualizing and analyzing audio data. This, unfortunately, excludes traditional de-facto standards, as these are usually oriented towards server-side execution of these tasks, or in the context of native applications (see e.g. FFmpeg, FFmpeg 2016). The following describes issues which were faced during the fulfillment of these requirements, and their implemented or potential solutions.

3.1. Segmentation of Compressed Audio Data

The Web Audio API was specified in 2011 by the World Wide Web Consortium (W3C) (see W3C 201: WebAudio). It allows for the decoding of audio data in different formats. 'Decoding' in this context means the approximation (or, somewhat fuzzier: the 'tracing back') of compressed data to the originally digitized, time-discreet sequence of pulse-code modulated (PCM) signals. The individual signals depict the amplitude of a given sound wave in a specific point in time as samples, and are therefore suited for visualization and analysis of audio signals (in contrast to the compressed format). The decoding is obviously also a necessary step for playing the audio file, for which individual sampling points are converted into electrical voltage by means of a digital-analogue-converter, which powers the membrane of speakers, and thereby making it audible.

The current (Februar 2020) implementations of the decoder for ogg/vorbis in the popular browsers Chrome and Firefox do not allow to decode compressed audio streams continuously or in pre-defined chunks. After the handover of the audio-buffer to the decoder, the decoder will decode the audio in its entirety before handing it back to the client. The ratio of the duration of the audio material to the duration of its decoding is approximately 30:1, i.e. decoding about 30 minutes of audio material takes roughly one minute, becoming more balanced (in this case: slower) the longer the audio is.[1] During the decoding process, a considerable strain is put on the client, which limits other functions of the application. Other JavaScript-based decoders, such as Audiocogs (see Audiocogs 2015), have been considered, but ultimately did not bring a sufficient performance enhancement in comparison to the WebAudio API. In the context of Transcribe, and the SFB:DiOE, which deals with recordings of up to (sometimes over) two hours, it was vital to find a more efficient solution.

To solve this problem, an ogg-bitstream-parser and -chunker was formulated in Typescript, based on the specification of the RFC 3533 of Silva Pfeiffer (2003). These are able to read and chunk the binary format of ogg audio pages and containers (see table 1).

[1] These are reference values at best, established by internal tests. The tests were conducted on a standard MacBook Pro with an Intel Skylake i5 CPU@3,1GHz, without dedicated hardware for decoding, on both Firefox and Chrome.

Table 1 *Schematic representation of an .ogg page in binary*

Bit 0-7	Bit 8-15	Bit 16-32	Bit 24-31	Byte
Magic Number (Marker) for the beginning of „OggS“ in ASCII				0-3
Ogg-version	Header-type			4-7
Starting point as a 64-bit integer in miliseconds				8-11
		Serial number of		12-15
the bit stream		page		16-19
sequence		check		20-23
sum		number of segments		24-27
Vorbis-encoded audio segments				28–…

This allows to index and correctly chunk not fully loaded and still compressed ogg-audio files. These chunks are identified during the loading of the binary blob and can be handed to the decoder piece by piece. The decoded results can be handed piecewise as well to the functions responsible for visualization and analysis of the audio sample without delaying the users significantly during the process. Furthermore, this enables loading individual segments of an audio file in advance by HTTP-Range-Requests (see Mozilla Developer Network 2021: Range Requests), meaning that users can skip to a later part of the transcript without waiting for the complete audio file to be loaded and decoded.

This parsing algorithm has a run-time complexity of *O(n)* and is optimized in such a way that for the use cases described, it can deliver results in the range of single-digit milliseconds. As of Feburary 2020, it is currently the only stand-alone implementation of such an algorithm in JavaScript, and is gonna be published as a library in the GitHub repository of the SFB DiOE.

3.2. Waveform Visualization

The term ‘waveform’ or ‘oscillogram’ means the visual representation of the envelope of acoustic waves in a diagram and is one of the most common graphical representations of audio data. In the specific application of transcribing spoken conversations, the waveform can only marginally give information

about acoustic phenomena such as pitch or articulation, however, it does aid with identifying pauses and differentiating between multiple speakers (see figure 1).[2]

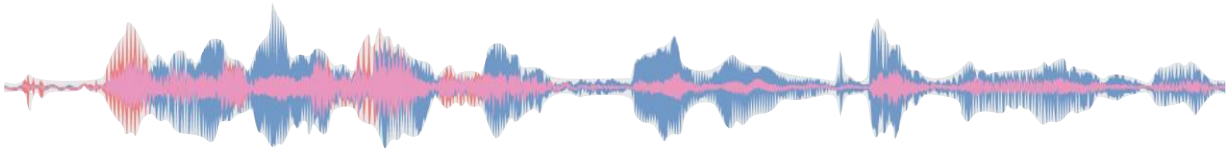


Figure 1 *Representation of a two-channel waveform in Transcribe*

As mentioned above, Transcribe operates within specific constraints that often exclude traditional server-sided solutions and require a great deal of efficiency of the algorithms used. After the evaluation of existing, client-based program libraries[3], a new solution was developed, with properties suitable for the task at hand.

For the output format of the waveforms, scalable vector graphics (SVG) were choosen, as these can be scaled without a loss of quality, in contrast to raster graphics. This furthermore allows for zooming in and out of the visualization without requiring a new computation of it. Additionally, both high- and low resolution screens can utilize the same visualization without any loss in either efficiency or quality. This obviously applies to all vector graphics.

The present solutions generates the complete SVG waveform once as an XML-string, and – in contrast to other solutions – not in iterations as a tree-like structure of Document-Object-Model elements (DOM-objects). The only access of the DOM during the execution is the embedding of the graphic in each segment of the document, which corresponds to the parsing of the structure as DOM-element. In this way, the present solution requires much less memory, resulting in a ten to 15 times shorter run-time than comparable DOM-based solutions (cf. Justice 2014). Because this solution avoids accessing the DOM-interface, it allows utilizing the algorithm in contexts which usually do not have access to the rendering engine of a browser – e.g. by using Node.js on servers, or aside of the main threads in Web Workers (which is the case for Transcribe).

Another novelty of this algorithm is that it allows for the simultaneous processing of two channels of a recording at once. Because two audio streams of one recording are necessarily of equal length, the same loop can be used to generate several independent wave forms. While this optimization seems trivial, it has not been employed by any of the libraries surveyed, and it saves about 35% of run-time.

The implementation presented here is, as of February 2020 and to our best knowledge, the fastest JavaScript-based library for the creation of SVG waveforms. Its optimizations result in a run time of about 15-20 miliseconds for a two-channel audio stream of 33 seconds of length, with a sample rate of 150 points per second. This corresponds to a ratio of length of audio stream to run time of about 1:2200. The implementation can be found as a library on GitHub in the repositories of Deutsch in Österreich.

[2] This is the case for stereo recordings, where each channel can be assigned to a speaker. This method of recording has proven itself to aid understanding of the transcriptors in the case of the SFB.

[3] Specifically DrawWave (https://github.com/meandavejustice/draw-wave), WaveSurfer (https://github.com/katspaugh/wavesurfer.js), and Audio-Waveform-SVG-Path (https://www.npmjs.com/package/audio-waveform-svg-path)

3.3. Spectrogram Visualization

As has been mentioned before, the waveform visualization of audio may serve as a signpost assisting transcriptors during longer stretches of conversation. It is, however, not suited for linguistic analysis proper, which may place its focus on acoustic and phonetic characteristics. The overlay of multiple signals or waves of different frequencies (caused by e.g. the overtones of a vowel sound) creates sometimes barely parseable visual representations. A spectrogram is an alternative visual representation of an audio signal, which not only visualizes a signals amplitude, but the whole spectrum of frequencies on a time axis. The spectrogram is therefore based on the transformation of the domain of time of a signal to the domain of its frequencies. Spectrograms are usually used for phonetic research.

The aforementioned conversion to the domain of frequency is achieved through the so-called Fourier-transformation. The algorithm on which the most common implementation of this method is based was originally described by Carl Friedrich Gauß[4], which was rediscovered by Cooley and Tukey in 1965 (see Heideman et al. 1985: 265p), and is now known as the Fast-Fourier-Transformation (FFT).

Because of Transcribe's plug-in architecture, the spectrogram can – just like the wave forms – be selected as standard mode of visualization, and is completely scroll-able. In order to make this achievable without excessive loading times, the FFT implementation has to be as efficient as possible. The implementation currently used by Transcribe is an adaption of the method included in the signal processing library DSP.js (see Brooks 2017). It has been chosen due to its array of window functions, which makes it suitable for a variety of recording situations and levels of audio quality (see National Instruments 2019).

The current method is able to transform about 30 seconds of audio material in roughly 500-700 milliseconds, and visualize it as a spectrogram. While this is fast enough to be suitable for analysis, it may necessitate short waiting times while navigating in a longer transcript. The optimization of this process, and therefore, of the user experience (see Nielsen 2012) of Transcribe, is still a subject of development. The following approaches have been tried or considered;

(1) Expanding the FFT-package by Fødor Indutny with multiple window functions, which can, according to the benchmarks, achieve better perfomance under certain conditions (see Indutny 2017 and Audioplastic 2017).
(2) The implementation of the same algorithm in the WebAssembly run-time environment. In an initial trial run, the method mentioned above was ported into AssemblyScript and compiled in WebAssembly. This, however, did not result in a significant increase in performance. The trial run and tests are archieved on the GitHub repositories of Deutsch in Österreich for purposes of reproduction (cf. DiOE 2020: Transcribe).
(3) Parallelizing the FFT, or more specifically, the execution of the FFT via the WebGL-API specified by the Khronos Group (2020). This utilizes the possibility of expressing the Fourier transformation as multiplication of matrices. The viability of this approach could be proven outside of the Browser environment (see Rosenberg 2018), and Google's Machine-Learning Library TensorFlow offers a basic implementation for JavaScript environments (see Google 2019). The advantages of this method are especially pronounced with large amounts of data (see Demorest 2007 and Sasiki 2020).

[4] The original manuscript was unpublished. Heideman et. al. (1985: 266) date the writing of this manuscript around 1805.

The method suggested in (3) seems, at this point, the most viable route to pursue. At the same time, it also represents the largest deviation from the typical path of generating spectrograms.

4. Transcription with Transcribe

After having considered some of the more technical aspects, the following chapter turns towards the linguistic application of Transcribe. This includes a discussion of transcriptions and transcription conventions, especially in the context of a large-scale research project, and a possible application of Transcribe for machine translation, based on an N-gram approach as done by Tahel Singer. As mentioned above, Transcribe was developed in the context of the SFB DiOE, a large-scale variationist research program researching the variation, contact and perception of different varieties of German in Austria (for an overview of the research program, see Lenz 2018 or DiOE 2018). As the research program is situated in five different institutions, with nine different project parts, the research foci of the individual project parts are diverse – as are the requirements for the transcription of spoken language data.

Transcribe tackles this issue on two different fronts: on the one hand, issues stemming from multiple researchers and assistants working on the same transcripts are avoided by Transcribe constantly updating the changes made to the transcript, and providing a history of changes made. Transcribe utilizes the internal data base of the research project as back-end, and changes made by one user on a specific transcript are immediately updated and displayed for other users, avoiding issues of versioning local transcript data (see DiOE 2020: Transcribe). This means that researchers across the different institutions always work with the same, up-to-date transcript. On the other hand, Transcribe allows different transcription conventions, and parses them accordingly. Therefore, discourse-oriented project parts can transcribe their data according to the GAT2 standard, while other project parts can utilize eye-dialect or orthographic transcriptions.

While it may seem counterintuitive to employ several transcription conventions within one research program, the research matter at hand necessitates this. Obviously, there is no singular ‘correct’ transcription standard, and the transcription convention chosen ultimately needs to be suited for the concrete research at hand (cf. Nagy and Sharma 2013: 242). This is due to a variety of reasons, among them being the desired level of detail of the transcription, issues of searchability, or the feasibility of transcribing larger amounts of data. For this reason, several different transcription systems are employed throughout the SFB DiOE, among them close phonetic transcription with IPA symbols, eye-dialect and standardized orthographic transcripts, and transcripts modelled after the GAT2 standard. The choice of transcription convention used reflects the research interest of the individual project parts. But, as Kendall (2008: 337) cautions, the act of transcribing spoken data is far from theory neutral, and influences possible further analyses of the data. For this reason, Transcribe offers multiple tiers for transcription, which enables researchers to transcribe the same token phonetically, orthographically, or with an eye-dialect system. In such a way, transcribing audio data on different tiers can create parallel, time-aligned corpuses.

A further danger identified by Nagy and Sharma (2013: 242), which especially endangers the reusability of transcripts, concerns the usage of punctuation in transcription and the ambiguity this can create. While there are some (competing) standardized protocols in place which formalize the meaning of punctuation, this issue is greatly elevated by supplying the audio recordings to the transcript via time-alignment. Additionally, Transcribe allows for customizable type-token parsing, meaning that

certain punctuation conventions can be coded into the transcript, and become meaningful to the software, e.g. indicating contractions with underscores, which changes the token type of the respective tokens. This type-token parser can be customized using regular expressions.

As a result of these features of Transcribe, especially the possibility of transcribing the same token on different tiers, and the modular programming of Transcribe, a further use-case is introduced in the following – the application of Transcribe in machine translation. This translation task between a transcription tier featuring an eye-dialect transcript of Viennese to a tier of a standard-orthographic Standard German following an N-Gram approach was developed as part of T. Singer's bachelor's thesis at the Technical University of Vienna. No deep learning features are involved in the main translation task, following the assumption that supervised machine learning methods, i.e., a single layer of local memory network, would be sufficient for this translation task and provide satisfactory results. An additional focus is an experimental approach toward handling unknown words, also known as out-of-vocabulary (OOV) words, involving a heuristic method and a prediction task of an external pretrained model provided by BERT ('Bidirectional Encoder Representations from Transformers'). The linguistic value of such a task is obvious: by virtue of such a model, it would become possible to automatically add a tier of orthographic standard transcription to an existing eye-dialect transcript. This does not only greatly enhance the searchability of a given transcript, it also allows for the application of other resources, such as automatic Part-of-Speech-tagger, which usually achieve the best results when confronted with the orthographic standard of a given language.

4.1. A Possible Application of Transcribe – Machine Translation Viennese to Standard German

The following therefore addresses the need for machine translation capable of handling language varieties that are not standardized and mostly suffer from lack of natural language processing (NLP) resources. The translation of Viennese from its dialectal form to its orthographic standard, which tends to be identical to Standard German, is the intended goal, while preserving language phenomena that can be found in slightly different grammar rules, idioms and ways of expression.

The data at the core of this work does not stem from the SFB DiOE proper, but rather L. M. Breuer's dissertation project (Breuer 2021), in cooperation with the project part 11 of the SFB DiOE at the Centre for Translation Studies at the University of Vienna. The corpus documents the variety of the German language and coexisting varieties of speech in Vienna. The data is constructed in such a way that the source language does not have any capitalized words and the target language includes capitalized words whenever it is necessary (e.g. proper nouns).

4.2 Approaches and Methods

The ambiguity of individual words characterizing the source language poses a great challenge for this particular translation task and influences the complexity of the translation task (see Trost 2016). The work combines different methods to achieve an optimal translation output; the main translation task is accomplished by observing the context of the word, following the N-gram approach. This approach is in accordance with the principle that single words may not be the best atomic units for translation, especially in this specific case, where the phenomenon of ambiguity is common and can be resolved only by considering a wider context (see Koehn 2009: 127–154).

Modeling with N-grams includes the probability distribution of tuples of the size N that construct word sequences. A unigram represents the sequence of single words, bigrams the sequences of pairs and trigrams the sequences of three consecutive words and they resemble different degrees of translation units. Each gram is assigned a probability using the chain rule probability of the following scheme:

$$P(w_1 \dots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \dots P(w_n|w_{1:n-1}) = \prod_{k=1}^{n} P(w_k|w_{1:k-1})$$

where:

$P(w_1 \dots w_n)$: the probability of a sentence word$_1$, word$_2$, … word$_n$.

It is given by the multiplication of the conditional probabilities;

$P(w_1)$: the probability of the first word;

$P(w_2|w_1)$: the probability of w_2 to appear based on the knowledge that w_1 appeared beforehand; and

$P(w_3|w_{1:2})$: the probability of w_3 to appear based on the knowledge that the sequence w_1 w_2 came before (cf. Koehn 2009: 181–216).

For handling the out-of-vocabulary words, a special mechanism based on known language patterns of sound shifting in Viennese was developed, applying a deeper character-based analysis.

The machine translation consists of the following statistical components; a language model based on trigrams of the target language, a phrase table consisting of uni-, bi- and trigrams of the translation units and their corresponding statistics and a stack decoder. The phrase table enables a deeper resolution of the single words or pair of words for seeking better and more precise translation alternatives. The stack decoder maintains the decoding procedure efficiently; once the input sentence is segmented into phrases, the output sentence is built sequentially from left to right, creating multiple hypotheses that can be referred to as the translation options. The stack manages the hypothesis expansion, i.e., the build-up of the translated sentence, and each hypothesis with the same number of words translated is placed in the same stack (see Koehn 2009: 155–180).

4.3 Handling Out-Of-Vocabulary Words

The default stack decoder does not support a partial translation in case of unknown words, also known as out-of-vocabulary (OOV) words and returns an empty value. With unknown words, we refer to tokens that are not contained in the corpus. The bigger the corpus is, the fewer words are found to be unknown to the system during the lookup process. For this reason, a pre-sentence analysis (also known as the preprocessing step) for the detection of such words is needed and can be easily done by a lookup function in the unigram-based dictionary. The unknown words are divided into two categories: Words that appear neither in the source language nor in the target language and words that do not appear in the source language, but do belong to the target language. The second category exists due to the intelligibility of Viennese and standard German and the switching phenomenon.

4.3.1 Prediction Process

Detected in-target words are added as their translation to the lookup dictionary, i.e., the function is described by the mapping f(w) = w, and the phrase table with the assigned probability value 1.0. This prevents the phrase table's error key from leading to an empty return value and enables the dynamic

enlargement of the corpus giving it "learning" qualities that spares future unnecessary extra processing for the same word occurrence.

As for the rest of the unknown words, a heuristic was developed applying the known language patterns regarding sound shifting in Viennese, which are reflected as changes in vowels observing the data as textual data type (see Breuer 2021). The heuristic aims at regaining unknown words using the character level approach as part of the preprocessing step. The information is saved as a dictionary data type supported by Python, where the key stands for the vowel in the source language and the value(s) for the possible occurrences in the target language. The method alternates the vowel and checks with the help of the lookup mechanism, if the word appears in the corpus, either in the source or in the target language. In case of a match, the word is added accordingly, and the workflow proceeds to the main translation task via the N-grams.

Special handling is done for unknown words that hold the letter <g>, based on the grammatical structure of verbs in the past participle tense. In Viennese, the form of a verb in this tense tends to be shortened by omitting the following letter <e> (e.g. *gestürzt*, 'fallen', being shortened to *gstürzt* in Viennese). There are two forms where <g> appears; either at the beginning of the sentence, in this case, the whole word holds only one part, or it appears in the middle of the word, belonging to the so-called separable verbs (e.g. *abgestürzt*, 'crashed'). These verbs are created by two parts, where – in certain grammatical contexts – each can be used independently. Each of the parts can occur both in dialect and in its standard German version. After identifying which form the OOV word has, a lookup is conducted for each part of the word. The mechanism is then similar to the previous one and the omitted <e> letter is appended after the <g>, and the newly created word that is strongly believed to belong to the target language is added to the corpus.

This heuristic might not cover all unknown words in an input sentence, however, it greatly contributes to the translator. It establishes a further step towards the computational understanding of a not-standardized language, while this kind of inferring and vowel alternation is done in most cases naturally by German speakers in the Bavarian language space.

4.3.2 Experimenting with German BERT using masks for predictions

For the task of finding possible candidates for the unknown words, an experiment with BERT's capability of predictions was conducted. BERT, which stands for Bidirectional Encoder Representations from Transformers, was developed and published in 2018 using unsupervised deep learning techniques and enables a wide variety of NLP tasks. Deep learning is applied in BERT via artificial neural networks that contain multi-layer transformers. A BERT model is constructed with two steps; pre-training and fine-tuning.

One of BERT's unique training approaches is the Masked Language Modeling (MLM), which can be described as a fill-in-the-blank task. In this case, a model bases its prediction regarding the next suitable word by observing the context words surrounding the mask token (see Devlin et al 2018).

We experimented with BERT's language task supported by the pre-trained models with the 'fill-mask' pipeline.[5] The 'dbmbz/bert-base-german-cased' model was used as the model and as the tokenizer for the pipeline. It provides a further look into the integration of the existing Standard German

[5] https://huggingface.co/dbmdz/bert-base-german-cased [last access 02. 09. 2021]

applications with the particular data of Viennese. The idea behind using it for predicting unknown words is to benefit from the similarity of both languages, especially when the source language lacks resources. The results can lead to a broader overview of the success of such integration and reflect some language phenomena of Viennese in comparison to Standard German.

Only the sentences that contain one or more unknown words are considered for this extra feature. The filled-in mask requires a sentence that contains a mask token [MASK]. The current fill-mask for the pre-trained models supports a mask prediction with a limitation of one masked word per input sentence. Therefore, the language task for such sentences is executed sequentially; for each unknown word, one mask is placed and the rest of the positions are filled with the original unknown words. At the end, the final sentence is constructed by inserting at each position the chosen candidate per mask.

The return value from the pipeline includes a few candidates (normally 3–5) that are model-dependent and sorted by their probabilities. Hence, choosing the best candidate based on its similarity to the original OOV word makes more sense and can improve results, as the decision based on the Viennese model creates a stronger correlation rather than the external Standard German one that lacks the sensitivity for this particular data. The Levenshtein Distance is ideal for implementing such a selection mechanism that is edit distance-based.

4.4 Results of the Translation

4.4.1 Methods for Evaluation

One- dimensional quantitative analysis is not sufficient to assess the properties of the machine translation, due to the particularity of the data and the heuristic method. Therefore, different testing methods were used in order to facilitate a wider understanding of the quality of this machine translation. This includes quantitative as well as qualitative methods, that are especially essential for determining the quality of the heuristic for the out-of-vocabulary.

For general statistics, we refer to absolute translation correctness as a hit/miss rate, i.e., the output sentence is identical to the expected translation, and correct partial translation, which can be measured with the WER score (word error rate) and with BLEU (bilingual evaluation understudy). For this work, the main method for evaluating the quality of translation is chosen to be WER score. The quantitative results refer to case-sensitive as well as case-insensitive. The case-sensitive check aims to measure how well the machine observes the orthographic grammar rule regarding capitalization. This property can be treated as another quality check of the machine. It is, however, not a vital criterion because an external grammar checker is able to perform this task.

The formula for WER calculation goes as follows: $WER = (S + I + D) / N$

where:[6]

S… stands for substitutions (replacing a word)

I… stands for insertions (inserting a word)

D… stands for deletions (omitting a word)

N… the total number of words appearing in the sentence

[6] See https://deepgram.com/blog/what-is-word-error-rate/ [last access 03. 09. 2021]

4.4.2 Test Set

The training set for creating the corpus consists of 100,000 parallel input sentences.

The test set consists of 24,000 parallel sentences, creating a relation of about 80%–20% training/test data (see table 2).

Table 2 *BLEU Score*

WER	BLEU	Case-sensitive	Case-insensitive
4.4731%	≈1	5,096/23,207 sentences incorrect 78.041% correct	3,423/23,207 sentences incorrect 85.25% correct

Such a high average BLEU score indicates that there is a high level of uni-/bi-/tri- and 4-grams correlation and that the data for training and testing might be overfitting. No external model for references is used, but can be included in future work.

4.4.3. Evaluation of the OOV Heuristic

Reviewing the results manually (see table 3), the words that the machine managed to regain via the heuristic illustrate the property of flexibility of this machine translation. It enables the understanding and deeper processing of Viennese, based on the language's patterns and nuances. The majority of the unknown words belong to the Standard German domain, and they are not recognized by the system because they have not appeared in the training set.

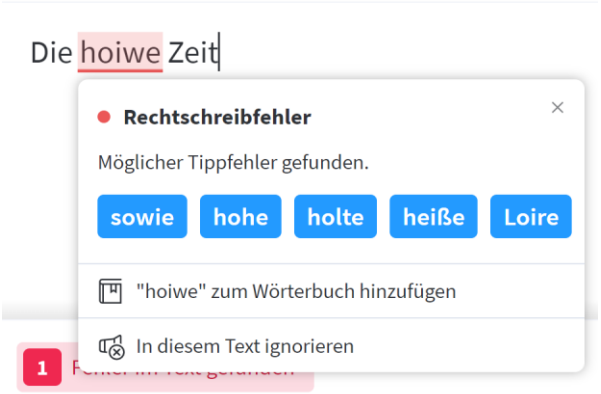
Table 3 *OOV heuristics*

Overall unknown words	Gained back words	Regain rate
4,106	221	5.38%

Some relevant examples: *aamol* (Viennese 'once') was changed to *amol*. This word is an example of adapting to the source language model, from this point it can be further processed by the translation task which will produce *einmal* (Standard German 'once'). It shows the flexibility of the system to handle anomalous inputs. Another similar example shows the capability of the system to cope with different varieties: *hoiwe* (Viennese 'half') was changed to *hoibe* that will be eventually translated to *halbe* (Standard German 'half').

The grammar correction task for *hoiwe* was tested with LanguageTool (2021), as shown in figure 2:

Figure 2 *LanguageTool translation for hoiwe*



This example leads us to the conclusion that external German language models might not be sensitive and adequate enough for Viennese.

4.4.4. Evaluation of BERT's Mask Prediction

The average WER value for the sentences rebuilt with BERT's masks stands at ≈20.63%, whereas for the same sentences without BERT's intervention (meaning the output translation that used only the OOV heuristic) the WER is 14.87%.

Overall only 21 words out of 3,237 processed sentences with unknown words were found to belong to the Standard German domain. This result is very poor, as not even 1% was recognized well for this task.

Overall 211 out of 3,237 sentences scored better than the output translation, i.e., a lower word error rate, which is about 6.5%.

4.5 Discussion of Further Translation Work

The high rate of the quantitative methods confirms the first assumption that an N-gram model with no neural network involved would also return relatively good results. Additionally, the machine managed to identify the need for capitalization for the formal second person singular pronoun *Sie* and *Ihnen* (which are formally identical with the second person plural pronouns, except for capitalization) in most cases.

The quality of the results of the correct partial translation has reached a point from which external models and libraries can be integrated to enhance the final output translation, e.g. autocomplete correction, grammar correction, etc. This also leads to a possible approach for future work, where the output translation of this machine is treated as a pivot language between the dialect and Standard German.

The OOV heuristic has shown the flexibility of the system to deal with deviations from the existing corpus by alternating specific vowels based on known patterns and nuances of Viennese. Also if the rate of regaining unknown words is pretty low (as described about 5%), it provides the system with the ability to handle language varieties in a way that is still not fully supported with the existing models and transformers.

From the results, it can be concluded that almost none of the candidates that were offered by BERT's prediction mechanism was correct, based on the comparison of the WER values.

This feature could be still used as fine-tuning for the words that do belong to the Standard German domain that suffer from misspellings, as the examples above showed.

However, it is clear from the poor results that Viennese poses a language that might be very similar to Standard German and have multiple intersection sets with it, yet consists of very particular figures of speech, idioms and ways of expressions. These do not necessarily intersect with Standard German. Most of the examples where no match is found between the best candidate offered by BERT and the intended word demonstrate this particularity. Thus, this implies that the current NLP tools and models for Standard German might not be fully adequate for handling dialects in general and Viennese in particular. While the equation of the Viennese dialect with Standard German might be problematic, the grammatic, syntactic and morphologic similarities between these two varieties can and should be utilized. This leads to a possible future need for more specialized or extended models that include trained data based on dialects.

5. Conclusion

This paper aimed to describe some of the core functionalities and innovations of Transcribe. Starting with its methods for audio processing and the visualization of audio data, continuing with Transcribe's utility for linguistic transcription, especially in the context of larger, collaborative research projects with different foci, and concluding with a possible application for machine translation. We are confident that further possible applications can and will be combined with Transcribe. Additionally, we want to expand Transcribe by several other functions, among them the possibility to easily connect it with a cloud-based back-end. We hope that this will lower the threshold of using it in the context of smaller-scale, decentralized research groups, who might not be able to afford hosting a server, but still want to work collaboratively on transcription. Additional features are still being conceptualized, developed and tested, but suggestions and comments from the linguistic and IT-community are welcome. The aim of Transcribe is not just to be a tool to enable researchers and any others working with language data to transcribe their data as convenient and possible and as detailed as necessary, but also to highlight some of the methodological implications of the act of transcription.

Transcribe is at this point still a work in progress, with a first stand-alone version to be released soon. The software is open-source and free, and can the code can be found in the GitHub Repository of the SFB DiOE (see DIOE 2020). Additionally, the public release of Transcribe, which is planned for 2021, will be announced on the website of the SFB DIOE.

References

Audiocogs 2015: *Ogg.js. GitHub Repository*. https://github.com/audiocogs/ogg.js [last access 04. 02. 2020].

Audioplastic 2017: *FFT Benchmarks. GitHub Repository*. https://github.com/audioplastic/fft-js-benchmark [last access 04. 02. 2020].

Breuer, Ludwig Maximilian 2021: *„Wienerisch" vertikal. Theorie und Methoden zur stadt-sprachlichen syntaktischen Variation am Beispiel einer empirischen Untersuchung in Wien*. Dissertation at the University of Vienna

Brook, Corban 2017: *DSP.js. Digital Signal Processing for Javascript*. GitHub-Repository. https://github.com/corbanbrook/dsp.js/ [last access 14. 02. 2020].

Chammas, Adriana, Quaresma, Manuela and Mont'Alvão, Cláudia 2015: A Closer Look on the User Centred Design. *Procedia Manufacturing* 2015/3: 5397–5404.

Debasmita, Saha, Ardhendu, Mandal, and Pal, S. 2015: User Interface Design Issues for Easy and Efficient Human Computer Interaction: An Explanatory Approach. *International Journal of Computer Sciences and Engineering* 2015/3: 127–135.

Demorest, Paul 2007: GPU Benchmarking. https://www.cv.nrao.edu [last access 07. 02. 2020].

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton and Toutanova, Kristina 2018: Bert: *Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805.

DiOE 2018: Overview. https://dioe.at/en/article-details/ [last access: 03.09.2021]

DiOE 2020: Github Repositories of the SFB: Deutsch in Österreich. https://github.com/german-in-austria [last access 03. 09. 2021].

European Commission 2016: *Open innovation, Open Science, open to the world*. A vision for Europe. Brussels: European Commission, Directorate-General for Research and Innovation. https://dx.doi.org/10.2777/061652 [last access 17. 5. 2018].

FFmpeg 2016: ffmpeg tool (Version be1d324) [Software]. https:// ffmpeg.org/ [last access 12. 02. 2020].

Google 2014: Material Design. https://material.io [last access 01. 02. 2020].

Google 2019: Tensorflow.JS. URL: https://js.tensorflow.org/api/0.13.3/#spectral.fft [last access 07. 02. 2020].

Gossman, John 2006: Advantages and Disadvantages of M-V-VM. https://docs.microsoft.com/en-gb/archive/blogs/johngossman/advantages-and-disadvantages-of-m-vvm [last access 01. 02. 2020].

Heideman, Michael, Johnson, Don and Burrus, Charles 1985: Gauss and the history of the fast Fourier transform. *Archive for History of Exact Sciences* 34: 265-277.

Indutny, Fødor 2017: *FFT.js*. GitHub Repository. https://github.com/indutny/fft.js [last access 03. 02. 2020].

Justice, David 2014: *DrawWave*. GitHub Repository. https://github.com/meandavejustice/draw-wave [last access 20. 02. 2020].

Kendall, Tyler 2008: On the History and Future of Sociolinguistic Data. *Language and Linguistics Compass* 2/2: 332–351.

Khronos Group 2020: WebGL 2.0. *Specification*. https://www.khronos.org/registry/ webgl/specs/latest/2.0/ [last access 22. 02. 2020].

Koehn, Philipp 2009: *Statistical Machine Translation*. Cambridge: Cambridge University Press

Krasner, Glenn E. and Pope, Stephen T. 1988: A Cookbook for Using the Model-View-Controller User Interface Paradigm. *Journal of Object-Oriented Programming* 88/4: 26–49.

LanguageTool 2021: https://languagetool.org/de/ [last access 03. 09. 2021].

Leider, John 2020: *Vuetify*. Material Design Component Framework. https://vuetifyjs.com [last access 01. 02. 2020].

Lenz, Alexandra N. 2018: The Special Research Programme: German in Austria: Variation – Contact – Perception. *Sociolinguistica* 32/1: 269–277.

Linklater, Greg, Marais, Craig, and Herbert, Alan 2018: Offline-First Design for Fault Tolerant Applications. SATNAC 2018, South Africa, 260–265.

Mozilla Developer Network 2021: Range Requests. https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests [last access 03. 09. 2021].

Nagy, Naomi and Sharma, Devyani 2013: Transcription. In: Podesva, Robert J. and Sharma, Devyani (eds.): *Research Methods in Linguistics*. Cambridge: Cambridge University Press, 235–256.

National Instruments 2019: Schnelle Fourier Transformation. https://www.ni.com/de-at/innovations/white-papers/06/understanding-fftsand-windowing.html#section--241931811 [last access: 02. 02. 2020].

Nielsen, Jakob and Molich R. 1990: Heuristic evaluation of user interfaces, Proc. ACM CHI'90 Conf. Seattle, 249–256.

Nielsen, Jakob 1999: Do Interface Standards Stifle Design Creativity? Jakob Nielsen's Alertbox, August 22, 1999. http://www.useit.com/alertbox/990822.htm [last acess 12.2.2020].

Nielsen, Jakob 2012: User Satisfaction vs. Performance Metrics. https://www.nngroup.com/articles/satisfaction-vs-performance-metrics/ [last access 17. 02. 2020].

Pfeiffer, Silvia 2003: *The Ogg Encapsulation Format*. Request For Comments 3533. *https://tools.ietf.org/html/rfc3533* [last access 01. 02. 2020].

Raskin, Jeff 1994: *Intuitive equals Familiar*. Communications of the ACM 37/9, 17 https://www.asktog.com/papers/raskinintuit.html [last access 18. 2. 2020].

Rosenberg, Duane 2018: GPU parallelization of a hybrid pseudospectral fluid turbulence framework using CUDA. Atmosphere Journal of Physics 11/2, 178-200. https:// arxiv.org/pdf/1808.01309.pdf [last access: 01 02. 2020].

Sasiki, Kay 2020: Fast Fourier Transform in TensorFlow.js WebGL backend. https:// www.lewuathe.com/webgl-implementation-of-fast-fourier-transform.html [last access 01. 02. 2020]

Trost, Harald 2016: *A Hybrid Approach to Statistical Machine Translation Between Standard and Dialectal Varieties. Human Language Technology*. Challenges for Computer Science and Linguistics: 6th Language and Technology Conference, LTC 2013, Poznań, Poland, December 7-9, 2013. Revised Selected Papers. Vol. 9561.

World Wide Web Consortium 2011: Web Audio. https://www.w3.org/TR/2011/WD-webaudio-20111215/ [last access 03. 02. 2020].

You, Evan 2020: Vue.Js. The Progressive JavaScript Framework. https://vuejs.org [last access 01. 02. 2020].