

FRAUNHOFER INSTITUTE FOR CERAMIC TECHNOLOGIES AND SYSTEMS IKTS

TECHNOLOGIEVERBESSERUNG DER AUTOMATISCHEN SPRACHERKENNUNG FÜR DIE OBERSORBISCHE SPRACHE

Project Report

Ivan Kraljevski
Frank Duckhorn
Constanze Tschöpe
Matthias Wolff*

Fraunhofer Institut für Keramische Technologien und Systeme IKTS,
Maria-Reiche-Straße 2, 01109 Dresden
**Brandenburgische Technische Universität Cottbus–Senftenberg, Cottbus*

Kunde: Stiftung für das sorbische Volk, Postplatz 2, 02625 Bautzen

Table of Contents

1	Introduction	3
2	WP 1: Technological improvements of the classic recognizer	3
2.1	Description.....	3
2.2	Training Environment and Configuration	4
2.3	Extended Recognizer Application	4
2.3.1	General.....	4
2.3.2	Deliveries	5
2.3.2.1	Trained Models (HMM-Triphone and TDNN).....	5
2.3.2.2	Documentation	5
3	WP 2: Technology Improvement in the Creation of Language Models and Their Use in Combination with AI	6
3.1	Description.....	6
3.2	WP2.1: Re-implementation and optimization of Named Entity Recognition (NER) for detecting word classes in text data.	6
3.3	Named Entity Recognition (NER).....	6
3.3.1	The Textual Data	6
3.3.2	Rule-based NER Parser	7
3.3.3	SpaCy Pipeline	8
3.3.4	Deliveries	8
3.3.4.1	Documentation	9
3.4	WP2.2: Development of models for spelling correction of the results from other AI-based speech recognizers	9
3.4.1	Open AI Whisper.....	9
3.4.1.1	Transcription of the HSB-Corpus.....	9
3.4.2	Fine-tune a new OpenAI Whisper Model.....	10
3.4.3	Errors Analysis.....	10
3.4.4	Text Correction	11
3.4.4.1	Phonetisaurus FST	11
3.4.4.2	SymSpell.....	12
3.4.5	Deliveries	13
3.4.5.1	OpenAI_whisper.....	13
3.4.5.2	Phonetisaurus.....	14
3.4.5.3	SymSpell.....	14
3.4.5.4	Documentation	14
4	WP 3: Publication of Language Resources	15
4.1	Description.....	15
4.2	Speech Data.....	15
4.3	Deliveries	16
4.3.1	Releasing the Corpus	16
4.3.2	Output Files	17
4.3.3	Augmenting the Release	17
4.3.4	KALDI Datasets	17
4.3.5	Publishing the HSB-Corpus.....	17
4.3.6	Documentation	17
4.3.7	Publications.....	18

1 Introduction

This project aims to further develop the Upper Sorbian Speech-to-Text (STT) technology. We employed the available speech and language resources, and the development of an STT system based on a traditional approach, which includes acoustic, pronunciation, and language modeling.

Due to the scarcity of resources for Sorbian languages, our approach leverages sub-word and word-class modeling techniques. The word-class modeling is based on Finite-State Transducer definitions, which are applicable to both offline text parsing and integration into the decoding graph of the STT system. Word-class parsing is performed on the speech corpus and utilized for language modeling with complete words, sub-word units, or both. Additionally, the same definitions can be applied to Named Entity Recognition during the post-processing of recognized transcriptions.

This approach significantly reduces out-of-vocabulary words and enables greater customization of the recognizer for domain-specific applications. The system was implemented for the real-time transcription of church sermon broadcasts in Upper Sorbian. The domain-specific system achieved performance comparable to fine-tuned OpenAI Whisper models developed also by other initiatives while also providing a resource-efficient solution with semantically tagged recognition results.

2 WP 1: Technological Improvements of the Classic Recognizer

2.1 Description

The aim of the work package is to improve the speech recognition performance for recognizing the Upper Sorbian language from the previous project. To improve this, the following points from chapter 6.2.4 of the strategy document, which was developed in the last project, are to be implemented:

- a) Facilitation of acoustic training by implementing a training procedure that does not rely on the complete phonetic labelling of the training data.
- b) Support of acoustic models based on neural networks (TDNN models). For this purpose, the training procedure is extended, and the recognition software is adapted so that the new acoustic models can be employed.
- c) If sufficient new speech data resources are available, these will also be included in the training.

Deliverables:

- Training environment and configuration.
- Extended recognizer application and auxiliary programs with instructions.
- Trained models (HMM-Triphone and TDNN).
- Documentation.

2.2 Training Environment and Configuration

The configurations, tools and the models are in the folder “**training_env**” of the WP1 delivery package.

Pre-requisites (related to WP3 deliveries):

1. Installed KALDI environment.
2. Latest release of the HSB-Corpus.
3. Generated KALDI configuration files.
4. Generated corpus (hsb.corp) and the vocabulary (hsb.vocab).

Main workflow, detailed description is given in the documentation:

1. KALDI, train/test/dev data
2. Lexicon generation
3. Update absolute paths
4. Run training
5. Locate and convert models
6. Use models in reclKTS configuration

Step-by-step guidelines are given in the README.md file as a part of the delivery.

2.3 Extended Recognizer Application

2.3.1 General

Fraunhofer IKTS has developed proprietary software for speech and signal recognition (reclKTS), designed to offer a broader range of features compared to freely available alternatives, with the added benefit of potential commercial applications. Written in C and C++, the software is available as both a stand-alone application and a library. It is compatible with various architectures and operating environments, including Win-32/64, Linux-i386/amd64/arm64, and is optimized for signal processors.

This software handles the entire pipeline, from audio input to feature extraction, l-vector calculation, acoustic model computation, and decoding. A key focus of the implementation is resource efficiency. While speech recognition is a primary use case, the software is versatile enough to recognize technical and biological signals as well.

For feature extraction, the software supports classic MFCC features, including LDA feature transformation, which are compatible with KALDI's MFCC features. Alternatively, it offers customizable Fourier, wavelet, or cepstrum transformations for technical signals, which can be paired with a configurable filter bank and principal component analysis.

l-vector calculations for TDNN models are also supported and can be performed in real-time during signal input.

The acoustic models used in the software are trained externally and imported into the IKTS recognizer. The software supports TDNN and HMM models from KALDI for mono-, bi-, and tri-phones, as well as UASR HMM mono-phone models.

Unlike KALDI, the language model in this software offers greater flexibility. It supports statistical language models in ARPA format, predefined CFG grammars, or pre-

compiled automata in OpenFST format. The language model can also incorporate word classes, which can be defined as word lists, grammars, or OpenFST automata. Sub-word modelling is also supported, allowing the language model to include sub-word units or morphemes that are reassembled during post-processing.

The core component of the speech recognition engine is its highly optimized decoder, which has been engineered for maximum resource efficiency. It is built on an efficient FST implementation and features a custom variant of the token-passing algorithm. This decoder supports live detection for recordings of any length, with recognition results provided in real-time. The software uses an iterative backtracking algorithm to refine these results.

In addition to recognition capabilities, the software includes speech pause detection. For this purpose, it integrates the GMM-based VAD from the open-source WebRTC project. A dedicated state machine manages trigger offsets and minimum activation and deactivation times.

2.3.2 Deliveries

The latest version with the configuration examples is in the folder “**recikts_1.1.7**” of the WP1 delivery package.

2.3.2.1 Trained Models (HMM-Triphone and TDNN)

Both models are trained on the train set of the most current version of the speech **HSB-Corpus (v1.0)** with 76212 recordings (including augmented ones) from 163 speakers and total duration of 97:51:28.

The new material is included in the corpus for training. The “**test**” and “**dev**” set are the same, except the augmented data is also included, therefore it might look like the performance is degraded in comparison with the previous deliveries.

The corpus is currently only privately released.

In the **models/conversion_scripts** folder there are: the **am_kaldi2uasr.py**, **am_kaldi2recikts.py**, **ivec_kaldi2recikts.py** and the models with example YAML-configurations:

- **tdnn, merged_47_nnet.yaml**
- **tri1, merged_47_mfcc.yaml**

The existing lexicons and language models can be used with the new acoustic models.

2.3.2.2 Documentation

This section of the project report and the corresponding README.md in the delivery package.

3 WP 2: Technology Improvement in the Creation of Language Models and Their Use in Combination with AI

3.1 Description

The aim of this work package is to improve the language resources for training language models for the Upper Sorbian language and to enhance the accuracy of speech recognition outputs. The following points from chapter 6.2.4 of the strategy document, developed during the previous project, are to be implemented.

3.2 WP2.1: Re-implementation and optimization of Named Entity Recognition (NER) for detecting word classes in text data

The following sub-steps are to be carried out:

- a) Automatic annotation of text data with the NER parser of the FhG IKTS and a language model using defined lists of entities (persons, places, dates, times, organizations, etc.)
- b) Quality control of the annotations.
- c) Creation of a SpaCy NLP pipeline for named entity recognition for the Upper Sorbian language using the generated data.
- d) Use of this pipeline to tag word classes for N-Gram language models of specific domains or to post-process automatically generated transcriptions.

Deliverables:

- Pipeline for detecting word classes.
- Documentation and report with results.

Requirements (cooperation obligations of the client):

- Provision of text and language data for the development and evaluation of the models.
- Support in the identification of errors and quality control of annotations.

3.3 Named Entity Recognition (NER)

The python script for NER was re-factored and optimized. The word-classes definitions which were more suitable for the STT are simplified and expanded for the data-driven examples found in the "misa" textual data.

3.3.1 The Textual Data

The data originates from different public sources and contains religious texts, such as :

- "Church services" - transcriptions of recorded speech
- "Wozjewjenja" - written announcements in PDF format

Developed tools:

- `misa2corp.sh` - it processes PDF and subtitle files, converting them into text in uppercase. It then prepares two sets of corpora: one from PDF files and another from subtitle files, splitting both into training (`train.corp`) and testing (`test.corp`) datasets with an 80/20 split.
- `srt2txt.py` - processes SRT (subtitle) files by merging consecutive captions if the time gap between them is within a specified threshold. It uses the `pysrt` library to read subtitle files and merges captions that are close in time, creating a continuous text. The script can be used via the command line, where users specify the SRT file and the time threshold (in milliseconds) for merging.
- `pdf2txt.py` - extracts text from a PDF file and prints it. It uses the `PyPDF2` library to read the PDF and extract the text from each page. The script can be executed via the command line, where users specify the PDF file to process.

The `misa2corp.sh` will create the training (`train.corp`) and test (`test.corp`) data, combined from both speech transcriptions and the written announcements.

3.3.2 Rule-based NER Parser

The main task in the WP2.1 was to optimize and modify the NER parser (`nlp/ner.py`) for speed and accuracy.

Notable changes:

- preloading the word-class definitions,
- looping over word-classes and then input file lines,
- sorting by length and removing overlapping entities,
- support for time periods, percent and currency,
- support for proper names, surnames and locations,
- handling flexmes by matching the word stem and
- output parsed results into text and JSON-format suitable for Python SpaCy pipeline training.

The script uses a list of word-classes (`"wordclasses_parser.txt"`) with pre-compiled FSTs:

{CURRENCY}	<code>fst/currency.fst</code>	<code>*NEW*</code>
{PERCENT}	<code>fst/percent.fst</code>	<code>*NEW*</code>
{DATE}	<code>fst/DateUni.fst</code>	<code>*Updated: date ranges*</code>
{WEEKDAY}	<code>fst/weekdays.fst</code>	
{TIME}	<code>fst/ClockUni.fst</code>	<code>*Updated: simplified*</code>
{ORDINAL}	<code>fst/ORDUni1-31.fst</code>	
{CARDINAL}	<code>fst/NUM1-10^6.fst</code>	<code>*Updated: from NUM0-999*</code>
{NAME}	<code>fst/names.fst</code>	<code>*Updated: names*</code>
{SNAME}	<code>fst/surnames.fst</code>	<code>*Updated: surnames*</code>
{GPE}	<code>fst/places.fst</code>	<code>*Updated: locations*</code>

Step-by-step guidelines are given in the `README.md` file as a part of the delivery package.

The annotated corpus can be used as any other to create N-gram language models with existing tools, regardless the word/sub-word tokenization. The words in the word-classes will be not tokenized.

The debugging and the quality check of the annotations were done on a selected sentences to target specific FST grammars.

The quality was also confirmed in the SpaCy pipeline experiments, where the annotated text was used for successful training and evaluation.

3.3.3 SpaCy Pipeline

To train and test SpaCy NER pipeline, the script uses the `train_spacy.json` and `test_spacy.json`, either generated by the rule-based NER parser or manually annotated, as input for training and evaluation.

The test evaluation outputs the performance metrics: accuracy, precision, recall and f-score per tokens, entities and per entity types.

For example:

```
"token_acc": 1.0,  
"token_p": 1.0,  
"token_r": 1.0,  
"token_f": 1.0,  
"ents_p": 0.8885288399222294,  
"ents_r": 0.8288996372430472,  
"ents_f": 0.8576790741319987,  
"ents_per_type": {  
  "{WEEKDAY}": {  
    "p": 0.9882903981264637,  
    "r": 0.9929411764705882,  
    "f": 0.9906103286384976  
  },  
  "{DATE}": {  
    "p": 0.8444444444444444,  
    "r": 0.7037037037037037,  
    "f": 0.7676767676767676  
  },  
}
```

In the "parsing" mode the script loads the model and outputs annotated text with the entities as in the example:

```
"<ORDINAL:14.> WOSADNA <SNAME:RADA> PŘEPROŠUJE WŠITKICH ZAJIMCOW  
WUTROBNJE NA PASTORALNY PŘEDNOŠK <WEEKDAY:ŠTÝWÓRTK>, 21.3.,  
<TIME:19:00> HODŽINA KNJEZ <NAME:FLORIAN>"
```

3.3.4 Deliveries

In the WP2.1 the following software and resources are delivered:

- `misa_corp/tooling/misa2corp.sh`, extracting text corpus from SRT and PDF files.
- `misa_corp/tooling/srt2txt.py`, SRT files to text.
- `misa_corp/tooling/pdf2txt.py`, PDF to text.
- `nlp/ner.py`, improved and optimized FST based NER python script.
- `nlp/spacy_ner.py`, python script for NER training, testing and parsing using SpaCy pipeline.
- `word_classes/make_fst.sh`, creates the "fst" folder from FST definition files in the "dev_ikts" branch of the repo "speech_recognition_language_modeling/cfg_word_classes/inputs/word_classes".

- `word_classes/make_symlinks.sh`, creates symbolic links from the files in the input folder to a given output folder.
- `wordmap.txt`, mappings of abbreviations.
- `wordclasses_parser.txt`, ordered list of the word-classes tags and definition files used exclusively for parsing texts.
- `train_spacy.json`, training data generated from train.corp (misa domain).
- `test_spacy.json`, test data generated from test.corp (misa domain).
- `test_spacy_cnum.txt`, example sentences to check various FST grammars.
- `test_spacy_cnum_ner.(txt,json)`, parsed example file.

3.3.4.1 Documentation

This section of the project report and the corresponding README.md in the delivery package.

3.4 WP2.2: Development of Models for Spelling Correction of the Results from other AI-based Speech Recognizers

The following sub-steps are to be realized:

- a) Perform speech recognition for all data in the HSB-Corpus using OpenAI Whisper.
- b) Compare the results with the reference transcriptions.
- c) Identify incorrect words and create a dataset with word pairs (correct and incorrect).
- d) Train an AI model (FST, RNN, etc.) to convert incorrect words into correct ones.
- e) Evaluate an alternative approach using a language model for transfer learning.

Deliverables:

- A successful model for spelling correction.
- Documentation and report with results.

Requirements (cooperation obligations of the client):

- Provision of text and language data for the development and evaluation of the models.
- Support in the identification of errors and quality control of annotations.

3.4.1 Open AI Whisper

3.4.1.1 Transcription of the HSB-Corpus

Transcription using fine-tuned OpenAI whisper-small model in GGML format:

```
python transcribe.py file_list base_folder
```

where the "file_list" contains relative paths to the recordings under the "base_folder".

Step-by-step guidelines are given in the README.md file as a part of the delivery package.

The results are written simultaneously into a CSV file in the following format, for example:

```
file,wer,transcription,correct_sentence
2024_03_17-clips/0012-2024_03_17,50.0,a tuž je by to snano naležnosť tež,a tuž je
bytoštna naležnosť tež
2024_03_17-clips/0022-2024_03_17,33.33,amen modleny so,amen modlmy so
2024_03_17-clips/0083-2024_03_17,36.36,turisko tam steja hdyž so dwanaće ja po
štoljo pokazuja kóždy z nich,turisca tam steja hdyž so dwanaće japoštoljo pokazuja
kóždy z nich
2024_03_17-clips/0044-2024_03_17,0.0,kiž ma na swět přińć,kiž ma na swět přińć
```

3.4.2 Fine-tune a new OpenAI Whisper Model

Fine-tuning on the current HSB-Corpus using “HuggingFace” modules.

- ft_whisper_prep.py data pre-processing and storing into HF datasets,
- ft_whisper_train.py use HF datasets for fine-tuning,
- ft_whisper_test.py test on a HF dataset part (train,test,dev).

3.4.3 Errors Analysis

For each sentence where there are misspellings, the pair of erroneous and correct sequence of words are presented.

The total WER is calculated and printed at the end.

```
python analyse_text.py
```

The transcriptions yielded the following WER and CER performance on the current HSB-Corpus (v1.0) and the additional independent “misa150” set:

dataset	WER (%)	CER (%)
train_set_v3	1.73	0.30
test_set_v3	7.04	1.17
dev_set_v3	12.49	2.10
misa150_v3	23.27	4.40

Compared with the old (version 1) OpenAI Whisper HSB (some of the test and dev set were used in the training):

dataset	WER (%)	CER (%)
test_set_v1	12.49	2.80
dev_set_v1	10.41	1.87
misa150_v1	33.17	6.14

3.4.4 Text Correction

The transcribed sentences of the whole HSB corpus are paired with the correct ones into a CSV file.

Augmentation for spelling errors from the transcriptions (dataset_v3.csv) and split to train/test data. The augmented data have the following WER/CER:

Dataset	WER (%)	CER (%)
dataset_v3.csv	2.43	0.42
train_aug.csv	54.69	10.86
test_aug.csv	54.79	10.84

The data splits from the dataset_v3.csv have the following WER/CER:

Dataset	WER (%)	CER (%)
train_70K.csv	2.44	0.43
test_7K.csv	2.33	0.41

The resulting combined datasets (train_70K + train_aug, test_7K + test_aug) have the following WER:

Dataset	WER (%)	CER (%)
train.csv	36.39	7.21
test.csv	43.56	8.61

3.4.4.1 Phonetisaurus FST

The folder `Text_Correction/phonetisaurus` contains the scripts for training and testing of the Phonetisaurus G2P model for text corrections. The training of the model is done with:

```
python phonetisaurus/psaurus_tc_train.py results/train.csv  
train_ps.txt
```

where `train_ps.txt` is the data that contains the misspelled/correct word pairs, used to train the model (1.5 million pairs).

The model with a default name `text2text.fst` is used for evaluation on the `test.csv` set.

```
python phonetisaurus/psaurus_tc_test.py results/test.csv
```

The model achieved the following results on the test.csv only on misspelled and correctly transcribe sentences separately:

Dataset	WER (%)		CER (%)	
	Before	After	Before	After
Test.csv				
Misspelled	29.35	18.97	11.08	4.77
Correct	0.00	0.14	0.00	0.03

The phonetisaurus is trained only on pairs of words and cannot correct errors where the words are either concatenated or split.

Although most of the cases the words were properly corrected, the model is sensitive to unseen words. Rarely the model provides wrong result for a correctly spelled word. Additionally, the computational time is an issue to be employed for real time corrections.

This approach is only partially usable for correction of transcription errors.

3.4.4.2 SymSpell

The official repo is:

<https://github.com/wolfgarbe/SymSpell>

and the python wrapper:

<https://github.com/mammothb/sympellpy.git>

The folder `Text_Correction/sympell` contains the scripts for text correction using SymSpell module:

- `bigrams.py`, estimate unigrams and bigrams from a text corpus,
- `sympell_test.py`, use unigram and bigram,
- `sympell_kenlm_test.py`, use KenLM ARPA model, and
- `gold.corp`, large (> 3 mil unique words) corpus in Upper Sorbian.

KenLM ARPA language model is used for optional re-scoring and must be installed with the path to the binaries set.

Dataset	WER (%)		CER (%)	
	before	after	before	after
Test.csv				
Aggregated	42.39	22.76	9.20	6.10
Misspelled	54.34	29.18	11.79	7.81
Correct	0.00	0.00	0.00	0.00

Using additional LM (gold.arpa) for re-scoring of the word hypotheses, the results on the whole test set (results/test.csv):

Dataset	WER (%)		CER (%)	
	before	after	before	after
Test.csv				
Aggregated	42.39	32.60	9.20	9.83
Misspelled	54.34	37.18	11.79	11.06
Correct	0.00	0.00	16.36	5.45

It is obvious that the additional LM for re-scoring provided worse results, replacing the correct words with wrong ones. It is important to have enough textual data matching a domain.

Using only unigrams and bigrams resulted better results where the WER decreased from 42.39% to 22.76%.

As in the case of phonetisaurus, misspelled words with more than 2 different characters are rather difficult to correct due to the ambiguous word candidates.

The difference in the WER and CER with the above for the test.csv are due to the averaging the per sentence and not calumniating then on the whole transcriptions.

This method is computationally efficient even for vocabulary with more than million-word forms, however in the case of more than few incorrect characters per word it is not performing well.

3.4.5 Deliveries

In this part of the work-package the following software and resources are delivered:

3.4.5.1 OpenAI_whisper

In the OpenAI_whisper/ folder the following tools and files are located:

- ft_whisper_prep.py - data preparation and conversion into HF format
 - ft_whisper_train.py - fine-tuning of whisper-small model (lang: czech)
 - ft_whisper_test.py - test the model on given file list of recordings
 - transcribe.py - transcribe corpus using binary whisper model
-
- ggml-model_v1.bin - old, fine-tuned whisper-small model
 - ggml-model.q8_0_v1.bin - quantized binary model (old)
 - ggml-model_v3.bin - new fine-tuned whisper-small model (lang: czech)
 - ggml-model.q8_0_v3.bin - quantized binary model (new)
-
- results/analyse_text.py - transcription analysis for WER/CER
 - results/train_v3.csv - transcriptions from train.flst (hsbcorpus_v1.0)
 - results/dev_v3.csv - transcriptions from dev.flst (hsbcorpus_v1.0)
 - results/dev_v3.log - log of the detailed error analysis
 - results/test_v1.csv - transcriptions test.flst (hsbcorpus_v1.0), old model
 - results/test_v1.log - log of the detailed error analysis

- results/test_v3.csv - transcriptions test.flst (hsbcorpus_v1.0), new model
- results/test_v3.log - log of the detailed error analysis

3.4.5.2 Phonetisaurus

In the `phonetisaurus` folder the following tools and files are located:

- psaurus_tc_train.py - training script
- psaurus_tc_test.py - text correction and evaluation
- txt2txt.fst - model trained on results/train.csv
- psaurus_err.log - results of wrong transcriptions from results/test.csv
- psaurus_corr.log - results of correct transcriptions from results/test.csv

3.4.5.3 SymSpell

In the `symspell` folder the following tools and files are located:

- bigrams.py - unigrams and bigrams from a text corpus
- gold.corp - large (> 3 mil unique words) corpus
- symspell_kenlm_test.py - use KenLM ARPA model
- symspell_test.py - use unigram and bigram
- gold_bigram_counts.txt - bigrams
- gold_unigram_counts.txt - unigrams
- model.arpa - ARPA model
- sym_spell_kenlm_test.log - logfile
- sym_spell_test.log - logfile

3.4.5.4 Documentation

This section of the project report and the corresponding README.md in the delivery package.

4 WP 3: Publication of Language Resources

4.1 Description

The publication of language data enables other groups to work on Upper Sorbian speech recognition.

Additionally, it is a prerequisite for the creation of scientific publications on this work. The following steps are implemented:

- a) Identification of speech recordings that can be published.
- b) Creation of metadata, licensing information, and documentation (analogous to LibriSpeech).
- c) Preparation of a technical paper describing the corpus and submission to ESSV25.
- d) Publication of the data on GitHub or as an archive with a download link.

Deliverables:

- Processed speech and text data for publication.
- Joint paper on the publication.

Requirements (client's obligations):

- Support with legal matters and obtaining consent from speakers.

4.2 Speech Data

The corpus (HSB-Corpus) is organized into several repositories, representing specific domains: the pilot project corpus (HSB), ten movies (SCF), ten studio recordings (SCM), male (SCK) and female (SCW) speakers for TTS systems, and the validated set of the Common Voice HSB dataset v5.1 (CV).

The repositories are updated with each release, and the recordings are converted to 16kHz, 16-bit PCM WAV format when necessary. Transcriptions are stored in a separate folder with corresponding files, formatted as one word per line, in uppercase, UTF-8 encoded, and free of special characters.

Each audio-transcription pair is assigned a universally unique identifier (UUID) during the release compilation. Consequently, each new version of the corpus features updated file names for recordings and transcriptions. This process also helps randomize the file order, which is particularly beneficial for ensuring speaker anonymity in movies and documentaries.

Table 1. presents key statistical properties of the corpus, including vocabulary richness measured through the Type-Token Ratio (TTR), which compares the number of unique words (types) to the total number of words (tokens) in a text.

The corpus was augmented with inclusion of Gaussian and real background noise and applying time-stretching, time-shifting, and simulation of impulse responses of large inner spaces, effectively doubling the amount of data.

The resulting speech corpus including the augmented recordings comprises over 70 thousand sentences from 163 speakers, totaling more than 97 hours of audio. The corpus compilation procedure is configurable and speakers or different sub-corpora can be excluded from the release.

	Combined	HSB	SCF	SCM	CV	SCK	SCW
#Speakers	163	31	106	23	16	1	1
#Recordings	38106	6313	14439	12473	1350	2442	1089
Duration	48:56:23	11:34:40	08:06:25	21:46:20	02:27:58	02:48:27	02:12:32
Avg. Duration (s)	4.62	6.6	2.02	6.28	6.58	4.14	7.3
Avg. Words	7.74	7.29	4.6	11.07	9.57	7.97	11.18
Type-Token Ratio	0.13	0.08	0.11	0.21	0.43	0.28	0.47

4.3 Deliveries

Upper Sorbian Speech corpus is composed of the various speech corpora of different domains: HSB from the feasibility study - (Smart Lamp), Movies (SCF) and recordings (SCM) and the validated set of the Common Voice hsb dataset v5.1.

They are included as submodules in the repository and invoked each time when making a release.

<https://github.com/kogmatd/hsbcorpus.git>

The version of the release is set in the file VERSION.

The license of the release is given in the file LICENSE.

Step-by-step guidelines are given in the README.md file as a part of the delivery.

4.3.1 Releasing the Corpus

Automatic script pulls the repo and the submodules, reverse the changes if any and invokes the corpus-specific preprocessing scripts to unify the audio and transliterations format and the naming convention.

In the creation process the audio and transliterations were checked for missing ones and given an URI name.

This means that the files in the release cannot be backtracked to the original files.

The subfolders in the “sig” and “tr1” folders are derived from the repo names ensuring their unique abbreviations.

The “metadata” folder contains the configurations for the corpus:

- dev_speaker_id.txt - speaker IDs that will be part of the “dev” dataset.
- test_speaker_id.txt - speaker IDs that will be part of the “test” dataset.
- exclude_speaker_id.txt - speaker IDs that will be excluded from the release.

- name_mapping_2020.txt - speaker IDs mapping for the speech_corpus_2020_study_lamp
- name_mapping_cv.txt - speaker IDs mapping for the speech_corpus_common_voice

4.3.2 Output Files

The audio files that are missing transliterations are listed in the “**mis_sig.log**” and the transliteration files without matching audio are listed in the “**mis_tr1.log**”.

- sig/ - folder with audio files in respective subfolders
- tr1/ - folder with transliteration files in respective subfolders
- all.flst - file list of all files in the release
- dev.flst - file list of the “development” dataset
- test.flst - file list of the “test” dataset
- train.flst - file list of the “train” dataset
- metadata.csv - speaker IDs, total duration and file counts
- hsb.corp - textual corpus of the transliterations in upper case
- hsv.vocab - word list - vocabulary of the hsb.corp

4.3.3 Augmenting the Release

Optionally the release can be expanded by augmenting the audio files by adding gaussian noise, time stretching, shifting, adding background noise and room simulation reverbation.

The folders with a suffix “**_AUG**” are added to the release effectively doubling the amount of the data. In this case, the file lists and the metadata must be recreated.

4.3.4 KALDI Datasets

Create KALDI compatible data on a local workstation with absolute paths, ready to be used with a corresponding KALDI recipe:

Folder “**kaldi**” will be created with the “**dev**”, “**test**” and “**train**” subfolders, each containing:

- corp - corpus of the dataset,
- text - utterance ids to transliteration mappings,
- utt2spk - utterance ids to speaker mappings,
- wav.scp - utterance to wav file mappings (absolute paths).

The KALDI data contains absolute paths and should not be part of a public release.

4.3.5 Publishing the HSB-Corpus

The content of folder “**release_\$VERSION**” where the version can be manually set in the **VERSION** file is manually archived in zip (tar) files, excluding the “**.log**” files.

The zipped archive will be provided as an official release withing the public repository.

4.3.6 Documentation

This section of the project report and the corresponding **README.md** in the delivery package.

4.3.7 Publications

Kraljevski I., Duckhorn F., Sobe D., Tschöpe C., Wolff M., "Preserving Language Heritage Through Speech Technology: The Case of Upper Sorbian", International Conference on Speech and Computer, 2024, Springer Nature Switzerland Cham., http://dx.doi.org/10.1007/978-3-031-77961-9_1

Kraljevski I., Duckhorn F., Sobe D., Tschöpe C., Wolff M., "Speech-To-Text in Upper Sorbian: Current State", ESSV 2025, Halle/Salle, 3-5 March 2025.