Cost effective, Multilingual, Privacy-driven voice-enabled Services

www.compriseh2020.eu

Call: H2020-ICT-2018-2020
Topic: ICT-29-2018
Type of action: RIA
Grant agreement Nº: 825081

WP Nº3: Multilingual personalised voice interaction

Deliverable Nº3.3: Final multilingual interaction library

Lead partner: TILDE

Version Nº: 1.0

Date: 30/01/2021

Document information	
Deliverable Nº and title:	D3.3 – Final multilingual interaction library
Version Nº:	1.0
Lead beneficiary:	TILDE
Author(s):	Askars Salimbajevs (TILDE), Jurgita Kapočiūtė-Dzikienė (TILDE)
Reviewers:	Dietrich Klakow (USAAR), Irina Illina (INRIA)
Submission date:	30/01/2021
Due date:	31/01/2021
Type[1]:	OTHER
Dissemination level[2]:	PU

Document history			
Date	Version	Author(s)	Comments
18/12/2020	0.1	Askars Salimbajevs	Draft of TOC
15/01/2021	0.2	Askars Salimbajevs & Jurgita Kapočiūtė-Dzikienė	Initial version
27/01/2021	0.3	Askars Salimbajevs & Jurgita Kapočiūtė-Dzikienė	Revision based on the reviewers' comments
30/01/2021	1.0	Akira Campbell & Emmanuel Vincent	Final version reviewed by the Project Manager and the Coordinator

[1] **R**: Report, **DEC:** Websites, patent filling, videos; **DEM:** Demonstrator, pilot, prototype; **ORDP:** Open Research Data Pilot; **ETHICS:** Ethics requirement. **OTHER:** Software Tools

[2] **PU:** Public**; CO:** Confidential, only for members of the consortium (including the Commission Services)

Document summary

This deliverable describes the “Final multilingual interaction library”, which is a collection of methods and software components developed within Tasks T3.1 and T3.2 of Work Package 3 to enable any user to interact with the dialogue system in any language using speech or text.

The first part of this document is devoted to research activities carried out in these tasks, concerning the integration of machine translation and speech-to-text on the one hand and the integration of machine translation and dialogue systems on the other hand. In this part we present:

- a “*synthetic* data pipeline” approach for training machine translation models that are robust to speech recognition errors;
- a disfluency detection model to filter out disfluent words from the speech-to-text output;
- monolingual and multilingual approaches for making dialogue system available in multiple languages.

The second part of the document is focused on the main software components of the multilingual interaction library which implement machine translation, dialogue management and tools for reproducing research results.

Because of the limitations of modern mobile devices, both machine translation and dialogue management are implemented as cloud-based services. Complete and up-to-date documentation of both components are provided online for developer convenience. Therefore, this deliverable only documents the base functions of the APIs and gives references to the full documentation.

Table of contents

1 Introduction

The goal of Work Package 3 is to enable any user to interact with dialogue systems in any language. This deliverable, entitled “D3.3 – Final multilingual interaction library”, focuses on the following sub-goals:

- combining Machine Translation (MT) with Speech-To-Text (STT) and Text-To-Speech (TTS) in order to provide a transparent interface between a user speaking a given language and a dialogue system (or possibly a human) in another language;
- combining MT with the components of a dialogue system, i.e., Spoken Language Understanding (SLU), dialogue management, and spoken language generation.

It should be noted that the final multilingual interaction library presented in this deliverable is a not a software library in the traditional sense (which can be statically or dynamically linked with some program), but a collection of APIs, software tools and methodologies which enable a developer to create multilingual voice-enabled dialogue systems or voice assistants. The STT and TTS components are not a part of the library and are not described here (see Work Package 4). While the library will be integrated into the COMPRISE SDK and the COMPRISE Cloud Platform, it can also be used separately.

Therefore, this deliverable is structured as follows. In Section 2 we describe the research conducted on robust integration of MT and STT, which involves a synthetic data pipeline, filtering methods to improve the quality of the synthetic data and a rule-based method for the generation of additional synthetic data. The proposed methods are evaluated in an MT scenario. We also report our results on disfluency detection in speech transcripts. Section 3 is devoted to research on the integration of MT and dialogue systems. We describe two effective solutions to make a dialogue system accessible in multiple languages: (1) MT of the training data and (2) using multilingual models. Section 4 focuses on software components of the library. We introduce the MT and dialogue management (Tilde.AI) APIs which will be used in the operating branch of COMPRISE. We also present tools for synthetic data filtering, disfluency detection and multilingual intent detection. We conclude in Section 5.

2 Integration of Machine Translation and Speech-to-Text

Integrating MT with STT allows a user speaking his/her own language to interact with a dialogue system in another language. This integration is not straightforward. First, we can distinguish between two modes of translation:

- *Simultaneous translation*: The translated text is provided at the same time as the speaker speaks.
- *Asynchronous translation*: The translated text is provided after the speaker finishes an utterance.

Simultaneous translation reduces the delay between speech and translation and can improve the user experience. This is especially important for long utterances and monologues. However, it requires both real-time STT and real-time MT and creates a number of additional technological and research challenges.

In COMPRISE, we are interested in voice assistant and dialogue scenarios where utterances are usually short. Under these circumstances translation can be performed at the end of a user input. This allows us to focus on **asynchronous translation**.

Furthermore, there are two main technical approaches for integrating STT and MT:

- *Sequential speech translation*: Perform speech recognition (STT) first, then translate the recognised text.
- *End-to-end translation*: Use a single component (usually a neural network) to translate to the target language directly from the audio input in the source language.

End-to-end translation is considered to be a more promising approach. Theoretically it allows the joint optimisation of STT and MT components for a particular task and should be more robust. However, current end-to-end systems are yet to reach state-of-the-art performance levels. Another downside to this approach is that it requires a rare kind of training data: translated speech corpora. State-of-the-art neural machines require millions of sentences for training, which would roughly correspond to thousands of hours of speech. Such data only exists for a few languages and in small quantities, compared to large existing corpora of parallel text and annotated speech. Therefore, in COMPRISE, we will focus on **sequential speech translation**.

The main problem with the sequential approach is the data quality mismatch between STT and MT:

- Due to potential recognition errors, the output of the STT system is a noisy textual representation of the *spoken* language. In addition, this representation does not have any punctuation and can contain disfluent and ungrammatical text.
- An MT system is trained on *written* language; therefore, the input is expected to have punctuation and be fluent and grammatically correct.

This mismatch results in poor translation quality if STT and MT components are naively combined. There are several possible approaches to tackle this issue:

- train MT on spoken language, e.g., on transcripts of STT training data;
- post-process STT output to adapt it for MT: insert punctuation, correct disfluencies;
- synthetically imitate spoken language features in the MT training data to train a robust MT model.

Each of these approaches have their own advantages and limitations. Theoretically, training MT on real spoken data should provide the best results. Unfortunately, STT datasets rarely come with translations and are often too small for training modern neural MT models. Still, if translation is available, STT data can be used for adaptation of a pre-trained MT model.

The post-processing approach can be a good option for high-resource languages, such as English, that have various spoken language processing tools and spoken language datasets available. This allows the development of both data-driven and rule-based solutions for correcting ungrammatical and disfluent utterances (Hassan *et al.*, 2014). However, for less-resourced languages (e.g., Latvian, Lithuanian, or Portuguese) which might not have required datasets or processing tools, following this approach is problematic.

The idea of the third approach is **to simulate STT errors** in the training data of MT. There are multiple ways to do this, for example, randomly substituting correct words by potentially confusing words using acoustic and linguistic embedding similarity (Simonnet *et al.*, 2018). This approach makes MT robust to STT errors, yet additional measures are needed to deal with ungrammatical and disfluent input.

In our work, we explored the latter approach, but decided to generate data with synthetic STT errors by using a pipeline of TTS and STT technologies, as will be detailed in Section 2.1. The advantage of this approach is that it generates not only substitution errors, but also insertions and deletions. In addition, it does not require additional data and only uses already available MT training data. However, this method is limited by the availability of STT and TTS for a given language. In the case of COMPRISE, such technologies will be available for the six languages (English, French, German, Latvian, Lithuanian, Portuguese) and language pairs needed in the project.

2.1 Synthetic data pipeline

The basic idea of this approach is to convert standard MT training data to a form that resembles the raw output from the STT system. This will introduce errors that are typically found in speech recognition output into the training data, and thus make the MT system more robust. To create the data, we propose the following training data pre-processing pipeline (see Figure 1):

1. Synthesise the source language sentences from the MT training dataset using TTS (the grey boxes in Figure 1).
2. Use STT to transcribe the synthesised sentences (the orange boxes in Figure 1).
3. Use STT transcriptions together with the original target sentences as the synthetic MT training dataset.

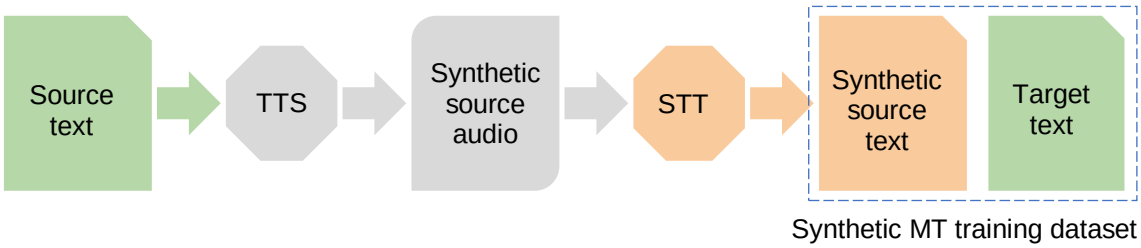

Figure 1: Synthetic data pipeline.

Before performing TTS and STT, the text needs to be pre-processed. For example, tokens that are pronounced incorrectly should be replaced to yield a correct pronunciation, and tokens that are not pronounced in spoken language (e.g., special characters) should be filtered out. Theoretically, the pre-processing step could also inject ungrammatical and disfluent segments into the sentences, but this was not performed with this data. This additional option will be explored in future research.

In the following experiments, we focus on Latvian-English speech-to-text MT. For speech synthesis, we use three of Tilde's Latvian TTS voices. Each sentence is synthesised with one of the three voices chosen at random. The source text is filtered from characters that should not be pronounced by the TTS (e.g., parentheses, quote signs, some punctuation signs).

For the STT, we then use Tilde's Latvian STT system which is based on a hybrid Hidden Markov Model and Time-Delay Neural Network acoustic model. The language model is implemented on a sub-word level using Byte-Pair Encoding (BPE) and consists of 4-grams, 6-grams, and a Recurrent Neural Network language model (RNNLM). However, in order to inject more errors into the resulting synthetic training data, only the 4-gram language model is used here.

As raw speech recognition output contains numbers written with words instead of digits, the MT system will have to learn how to translate words into digits. Therefore, to simplify the training of the MT system, we apply Tilde's in-house number normalisation tool for Latvian which rewrites numbers as digits. In the case when number normalisation tools are not available for a given language, raw transcripts from the STT system could also be used as the source text for MT training.

The pipeline for other languages can be constructed similarly: for STT we can use models available on the COMPRISE Cloud Platform and for TTS the open-source eSpeak synthesiser. The training dataset can be created from the WMT 2017 training dataset (Bojar *et al.*, 2017) or any other parallel sentence text corpus.

2.2 Data

The Latvian-English WMT 2017 training dataset (Bojar *et al.*, 2017) is used for training the MT system. The dataset consists of three smaller datasets:

- Europarl: proceedings of the European Parliament (637,599 sentence pairs).
- RAPID: press-releases taken from the Press Release Database of the European Commission (306,588 sentence pairs).
- DCEP: press-releases, session and legislative documents related to the European Parliament's activities and bodies (3,542,280 sentence pairs).

First, the data is processed by the Synthetic Data pipeline described in the previous section. The Word Error Rate (WER) of the synthesised data was approximately 20.7%. Then, we follow our baseline MT training recipe and pre-process the data using standard Moses (Koehn *et al.*, 2007) scripts for normalisation. First, punctuation was normalised (normalize-punctuation.perl). Then, the corpus was cleaned (clean-corpus-n.perl) by removing segments that are too long and segments where the ratio of source and target lengths exceeds 3. Such cleaning is necessary, because WMT data contain some erroneous translations.

After cleaning, 4,407,375 sentences remained in the training data. The data were further processed using the Moses tokeniser (tokenizer.perl) and truecaser (train-truecaser.perl and truecase.perl). Finally, words were split into sub-word units using byte-pair encoding[3] (Sennrich *et al.*, 2016) with 24.5k merge operations.

For tuning, we use the NewsDev2017 dataset from WMT 2017 (2003 sentence pairs) which is also processed using the same methods and tools.

The evaluation is performed on three datasets:

- NewsTest2017 from WMT 2017 (2001 sentence pairs).

[3] Subword Neural Machine Translation https://github.com/rsennrich/subword-nmt

- Synthetic NewsTest2017 which is NewsTest2017 processed by the same TTS-STT pipeline (2001 sentence pairs).
- Real-world test set “Tilde Balss” (1159 sentence pairs).

The real-world test set is based on a subset of data collected by the Tilde real-time Latvian STT. This subset contains 8,820 utterances and 37,782 words in 39 hours of audio (including silence). Utterances come from various domains: queries, short messages, addresses, interaction with a voice-enabled educational app, etc. In addition, it contains a lot of “noise”: laughing, English and Russian speech, untranslatable or ambiguous utterances, etc. Therefore, several rounds of semi-automatic filtering were performed to select meaningful utterances from this dataset. This resulted in a dataset of 1,159 Latvian utterances which were manually translated to English.

2.3 Filtering of the synthetic data

Although the speech processing workflow introduces real noise, it is also evident that it introduces errors due to the limitations of the workflow itself. For instance, the speech synthesiser is unable to pronounce foreign named entities and complex identifiers correctly. This results in misrecognition by the STT system (i.e., in such cases it either deletes words or recognises the mispronounced names as some other common phrases, which are not even necessarily phonetically similar). The synthesiser also drops most Unicode characters that it cannot pronounce, which also introduces noise in the parallel corpus.

Examples of errors introduced by the workflow itself are given in Table 1.

Table 1. Examples of noise introduced by the limitations of the speech processing tools.

Source / Target in the parallel corpus	**Synthetic segment / Translation (English)**
Pierre Schapira, Attīstības komiteja **Pierre Schapira**, Committee on Development	**ieviests papīra** attīstības komiteja **introduced paper** committee on development
Mieczysław Edmund Janowski Mieczysław Edmund Janowski	edmunda jānoski edmund jnoski
Skatīt arī **MEMO /** 14 **/** 597 See also **MEMO /** 14 **/** 597	skatīt arī **melo** 14 597 see also **lie** 14 597

To address these issues, we filter the generated synthetic data by discarding sentences:

- that contain website addresses or Roman or Arabic digits,
- that contain one character,
- for which the similarity based on the Levenshtein distance (Levenshtein, 1966) between the original and the synthetic sentence is lower than 0.9.

We applied the Levenshtein distance-based similarity threshold to identify segments that have been corrupted too much by the synthetic data generation workflow. For instance, this allows us to address issues introduced by the misrecognition of foreign named entities and complex identifiers.

2.4 Rule-based synthetic noise generation

After filtering, we performed an error analysis to investigate what types of errors common to STT systems were present in the filtered data. The results showed that 35.6% of all errors were suffix-related. This was to be expected as Latvian is a morphologically rich language with fusional morphology and incorrect inflections are common mistakes for STT systems (Salimbajevs & Strigins, 2015). Other errors were deletions (32.9%), insertions (25.2%), and the remaining 6.3% were related to other types of lexical misrecognitions.

Next, we analysed what types of suffix (or inflection) errors are present in the data. For this, we used the Tilde's Latvian Part-Of-Speech (POS) tagger[4] and extracted a list of the most common inflection changes. The analysis revealed that the 26 most common inflection changes amount to 90% of all suffix errors.

Since we found that 26 most common inflection changes amounted to 90% of all suffix errors, we devised a method that iterates through the original dataset in a random fashion and randomly generates in each sentence one of the error types for a random number of words for which that particular error type can be introduced.

As an additional step, we validated the generated errors using a vocabulary to make sure that the generation produced real Latvian words. After generation, we only selected sentences that had at least one error.

2.5 Speech translation evaluation

Since the paradigm-shifting success of Neural Machine Translation (NMT) systems at the 2016 Conference on Machine Translation (WMT) (Bojar *et al.*, 2016), and the invention of the self-attentional Transformer architecture (Vaswani *et al.*, 2017), Transformer-based NMT models have become the baseline choice for MT experiments.

Therefore, for the evaluation of the synthetic data pipeline we use the Marian NMT toolkit (Junczys-Dowmunt *et al.*, 2018) for the training of all NMT systems and the Marian base model configuration for the model hyper-parameters.

In order to generate the synthetic data, we use a selection of three Latvian TTS voices. For each sentence, we choose one of the voices at random. Characters that are not pronounced by TTS are filtered from the source text (e.g., parentheses, quotation marks, various other punctuation marks).

We use an STT system based on a hybrid Hidden Markov Model and Time-Delay Neural Network acoustic model and a sub-word language model. The language model is implemented using BPE and consists of 3 models: a 4-gram, a 6-gram and an RNNLM. In order to inject more errors into the resulting synthetic training data, only the 4-gram language model is used to produce synthetic data. As the raw speech recognition output contains numbers written with words not digits, the MT model would have to learn how to translate words into digits. Therefore, to simplify the training of the MT system, we apply a number normalisation tool for Latvian that re-writes numbers as digits. If number normalisation tools are not available for a given language, raw transcripts from the STT system could also be used as the source text for MT training.

[4] The source code can be found online at: https://github.com/pdonald/latvian

The speech processing workflow produced 4,407,364 synthetic sentences. After filtering, 1,921,043 sentences remained in the synthetic dataset. The rule-based synthetic noise generation workflow produced 753,693 sentences with vocabulary validation and 961,227 sentences without vocabulary validation. The vocabulary was built using the original training data.

For validation, we use the NewsDev2017 dataset from WMT2017. The dataset was also processed using the speech processing workflow. For validation during training, we use a combination of the clean and noisy validation sets.

The results in Table 2 show that, for the STT output, supplementing the original training data with synthetic noise increases the MT quality by up to 1.66 BLEU points. A similar tendency is evident when translating human-created transcripts that also may contain orthographic speech noise (e.g., truncated words, incorrect syntax, wrong punctuation, etc.). Only when translating clean publishable transcripts, the systems that are trained on noisy data show lower results than the baseline system. Nevertheless, the synthetic noise generation strategies have been successful in handling STT output better and achieving higher translation quality than the baseline system.

The results also show that the **best results were achieved when combining the filtered synthetic data and the data that is generated using rules with vocabulary validation**. The combined data allow increasing the translation quality by 1.87 BLEU points over the baseline system.

Table 2. Evaluation results (bold: highest score; †: statistically significant improvement over the baseline with $p<0.01$).

Training data	STT output		Ref. transcript		Ref. + punct.	
	BLEU	**ChrF2**	**BLEU**	**ChrF2**	**BLEU**	**ChrF2**
a. Original parallel data (baseline)	12.73	0.4395	14.67	0.4622	**20.90**	**0.5123**
b. Noisy synthetic data	12.61	0.4160	14.08	0.4306	13.94	0.4251
c. a + b	†14.33	0.4374	†16.08	0.4577	20.45	0.5068
d. Filtered synthetic data + a	†14.39	**0.4602**	†16.79	0.4854	19.37	0.4995
e. Rule-based data (no voc.) + a	12.08	0.4243	13.12	0.4377	18.57	0.4830
f. Rule-based data (with voc.) + a	11.47	0.4231	12.75	0.4367	18.94	0.4847
g. Rule-based data (no voc.) + d	13.72	0.4484	†16.00	0.4815	18.69	0.4973
h. Rule-based data (with voc.) + d	**†14.60**	0.4547	**†17.29**	**0.4907**	19.46	0.5028

2.6 Disfluency detection

Another important issue concerns ungrammatical spontaneous spoken language: people speak continuously without pauses and sentence breaks, often make repetitions, correct themselves, or stop in the middle of a sentence. This will create problems both for stock MT which expects clean input and for MT trained on synthetic data, because the synthetic data pipeline does not produce disfluencies. This means disfluencies should be detected and corrected before the resulting text gets translated by the MT model.

Most work on disfluency detection heavily relies on human-annotated data, which is scarce and expensive to obtain in practice, especially for a less-resourced language like

Latvian. Therefore, in the scope of WP3 **the self-supervised multi-task approach (Wang *et al.*, 2020) was investigated** to address the disfluency detection in speech recognition output.

The goal is to detect the added disfluent words by associating a label for each word, where the labels D and O mean that the word is an added word or a fluent word, respectively (see Figure 2).

I	want	a	flight	to	Boston	um	to	Denver
O	O	O	O	D	D	D	O	O

Figure 2. Example of disfluency tagged sentence.

The idea of the self-supervised approach by Wang *et al.* (2020) is to construct large-scale pseudo training data by randomly adding or deleting words from unlabelled data, and use it to train a multi-task model for joint:

- word tagging, i.e., detecting the added disfluent words,
- and sentence classification, i.e., distinguishing original sentences from grammatically incorrect sentences.

After self-supervision the model can be fine-tuned on a small real data set which is manually labelled (see Figure 3).

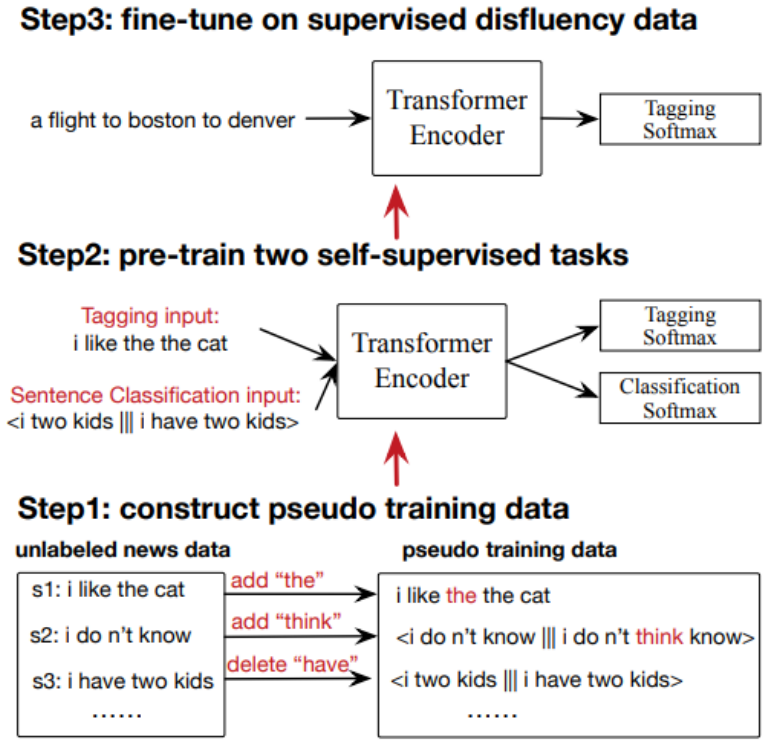

Figure 3. Self-supervised training of a disfluency detection model (Wang el al, 2020).

The model is an encoder network with an input embedding layer and two softmax output layers, one for the tagging task and one for the classification task (see Figure 4). We use a Transformer encoder with 512 hidden units, 8 heads, 6 hidden layers, GELU activations (Hendrycks and Gimpel 2016), and dropout of 0.1. We train our models with the Adam optimiser. For the joint tagging and sentence classification objectives, we use streams of 192 tokens and mini-batches of size 256, where approximately 30% of the

data are single sentences used for the tagging task, and another 70% are sentence pairs for the sentence classification task. We use a learning rate of 1e-5 and train for 50 epochs.

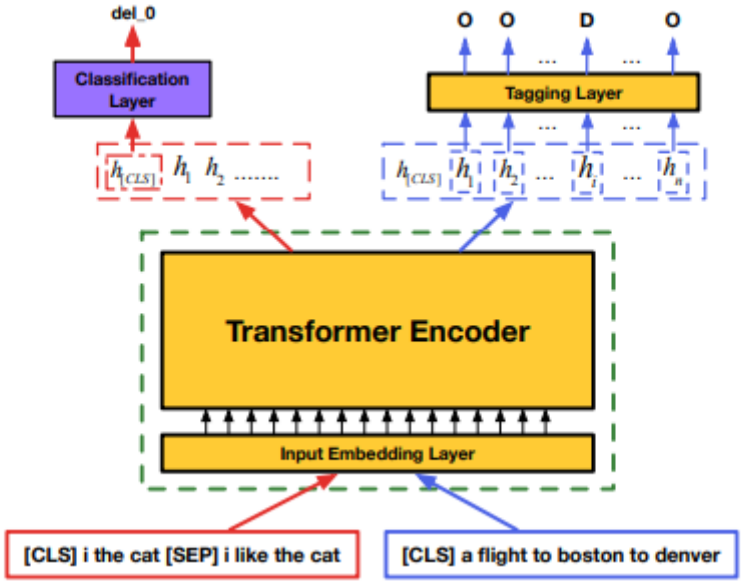

Figure 4. Disfluency detection model structure (Wang et al, 2020).

When fine-tuning, we use a batch size of 32, a learning rate of 1e-6, and 20 epochs. The parameters of the input embedding layer, the encoder layer, and the tagging layer are shared among the pre-training and fine-tuning phases. During the inference phase, only the output tagging layer is used.

The model is trained with self-supervision on large-scale pseudo training data created from unlabelled data (about 26M sentences from Latvian web news portals). Then, the model is fine-tuned on human-labelled data. A disfluency-tagged dataset (about 2,000 sentences) was created from phone call recordings at a tech support centre. 1,000 sentences are used for fine-tuning and another 1,000 for evaluation.

During the evaluation we compare the self-supervised model with a traditional model trained only on human annotated data (Table 3). **The self-supervised approach outperforms the baseline;** however, accuracy is too low to be used in practice.

Table 3. Evaluation of the self-supervised disfluency detection approach.

Model	Precision, %	Recall, %	F1-score, %
Baseline	52%	58%	54%
Self-supervision	**65%**	**71%**	**68%**

Such low results can be explained by the difficulty of the test data: the annotator self-agreement free-marginal kappa is 0.71 only. Therefore, a simpler disfluency tagged dataset was created which contains less ambiguities. 3,000 sentences were collected from real-world usage data of Tilde's real-time Latvian STT. Sentences were manually tagged and split into two subsets (1,000 for evaluation, 2,000 for fine-tuning).

Evaluation on the new data is presented in Table 4. As with the previous results, the self-supervised approach outperforms the baseline in terms of recall and F1 but achieves lower precision. Since the second dataset is bigger, we also performed some additional

experiments using 1,000 sentences for training and fine-tuning. This allows us to compare results with the previous experiment and observe that both the baseline model and the self-supervised model show much better results when tuned and tested on less ambiguous data of the same size. The improvement is especially significant in the case of baseline model training without any self-supervision.

Also, it can be seen that: (1) **adding more data improves the performance of both models**, (2) **the self-supervised approach allows us to achieve the same performance as the baseline using half the data**.

Table 4. Evaluation of self-supervised disfluency detection on a simpler data set.

Model	Precision, %	Recall, %	F1-score, %
Baseline (1,000 sentences)	75%	65%	70%
Baseline (all data)	**78%**	67%	72%
Self-supervision (1,000 sentences)	70%	75%	72%
Self-supervision (all data)	71%	**78%**	**74%**

2.7 Summary of results

The synthetic data pipeline approach was proposed and investigated to address the mismatch between the STT output and the MT training data. Several MT systems were trained and evaluated. It was noticed that using the synthetic data in MT training resulted in a clear benefit when translating STT output. The approach was further improved by special filtering of the synthetic data and the generation of additional data by rule-based methods. This increases the translation quality by 1.87 BLEU points over the baseline stock MT system.

The self-supervised approach was studied to address the disfluency detection in speech recognition output. Two disfluency-tagged datasets (2,000 and 3,000 sentences) were created for model fine-tuning and evaluation. On both datasets the self-supervised approach outperformed the baseline trained without self-supervision: 68% F1 versus 54% F1 on the first dataset and 74% versus 72% on second dataset. The second dataset is less ambiguous therefore both models show much better results. Also, it was observed that: (1) adding more data improves the performance of both models, (2) the self-supervised approach allows us to achieve the same performance as the baseline using half as much data.

3 Integration of Machine Translation and dialogue

English is a resource-rich language. In the machine learning (and especially deep learning) era, this fact explains why so much research has been done for English and so many accurate tools have been developed. The lack of resources or the proprietary nature often becomes an obstacle that hinders progress for some other languages, especially complex ones with a small number of speakers.

The goal of our research in Task T3.2 is to cross the barrier of monolingualism by offering effective multilingual dialogue solutions based on (1) a monolingual model trained on machine translated data for every language of interest and/or (2) a single multilingual

model trained on one or a few languages and used on other languages. The first approach completely relies on the accuracy of MT tools. The second one is a novel point-of-view to the problem, which has not been experimentally investigated much so far.

3.1 Intent detection data

In this research we aim to train a Natural Language Understanding (NLU) module, which is responsible for the comprehension of the user's intent. The NLU module is a key component in conversational (chatbot) systems, as well as in spoken dialogue interfaces where it is called SLU instead.

Formally, intent detection is a typical example of supervised text classification, which requires labeled instances for training. For this reason, we have used a manually prepared English dataset containing texts (i.e., user queries) with the assigned classes (i.e., corresponding chatbot answers). The dataset domain covers topics related to the application *Tildės Biuras*[5], i.e., prices, licenses, supported languages, used technologies. Instances in the dataset were shuffled and split into training and test subsets, keeping the proportion for training / test instances in each class equal to 80% / 20%, respectively (Table 5).

Table 5. Statistics about the used English benchmark dataset.

	Training	Testing
Numb. of intents	41	41
Numb. of instances	365	144
Instances per intent	8.9	3.5

In addition to English (EN), we have considered one more Germanic language (i.e., German – DE), two Romance languages (French – FR and Portuguese – PT) and two Baltic languages (Lithuanian – LT and Latvian – LV), differing from each other by such characteristics as morphology, derivational systems, sentence structure, etc. The diversity of languages will allow us to reveal whether there are regularities that cross the boundaries of each language.

The EN training dataset has been machine translated into DE, FR, LT, LV, and PT via Google, in order to simulate the real situation when no annotated training data is available for these languages and machine translated data is used to train the intent detection model instead. The test datasets for DE, FR, LT, LV, and PT were manually translated from EN instead. The size of the datasets in each language is summarised in Table 6.

Table 6. Statistics about datasets in different languages: number of words.

Language	Training	Testing
EN	2,826	1,090
DE	2,369	877
FR	2,743	1,222
LT	1,929	751
LV	1,991	855

[5] Tildės Biuras. https://www.tilde.lt/tildes-biuras

PT	2,812	1,133

3.2 Methodology

Our intent detection models consist of two parts: (1) a text vectorisation model trained on a large unlabeled text corpus and (2) a classification model trained on intent detection data in a supervised manner.

For the vectorisation, BERT (Bidirectional Encoder Representations from Transformers; Devlin et al., 2019) was selected because it is a robust solution for disambiguation problems since homonyms (words with the same spelling but different meanings) are represented by different word vectors depending on their context. We have investigated two categories of pre-trained BERT models for vectorisation:

- **Word embedding (BERT-w)**. We have tested 4 monolingual word embedding models (for EN only): bert-base-cased, bert-base-uncased, bert-large-cased, and bert-large-uncased (where cased and uncased stand for models trained on cased or lower-cased texts, respectively), and 2 multilingual word embedding models (covering all of our target languages, i.e., EN, DE, FR, LT, LV, and PT): bert-base-multilingual-cased, and bert-base-multilingual-uncased.[6]
- **Sentence embedding (BERT-s)**. We have tested 4 monolingual sentence embedding models (for EN only): roberta-base-nli-stsb-mean-tokens, roberta-large-nli-stsb-mean-tokens, bert-large-nli-stsb-mean-tokens, and distilbert-base-nli-stsb-mean-tokens, and 4 multilingual sentence embedding models: distiluse-base-multilingual-cased-v2, xlm-r-distilroberta-base-paraphrase-v1, xlm-r-bert-base-nli-stsb-mean-tokens, and distilbert-multilingual-nli-stsb-quora-ranking.[7]

For the classification, we have investigated the following approaches:

- **BERT-w + CNN**: fixed BERT word embeddings with a Convolutional Neural Network (CNN) classifier. CNNs were introduced by LeCun et al. (1998), but in our experiments we have used the 1D CNN architecture adjusted for text (Kim, 2014). The CNN architecture and hyper-parameters have been refined by Tilde for various language processing tasks. The CNN gets a vectorised query (i.e., a sequence of word embeddings) as input and learns to detect relevant patterns (consisting of 2, 3 or more adjacent tokens, so-called n-grams, regardless of their position in the text) that have a major impact on the estimation of the intent.
- **BERT-w + BERT**: BERT word embedding model fine-tuned on our data for the downstream intent detection task.[8]

[6] The detailed description of these BERT word embedding models can be found at https://huggingface.co/transformers/pretrained_models.html.

[7] The detailed description of these BERT sentence embedding models can be found at https://www.sbert.net/docs/pretrained_models.html.

[8] Fine-tuning is performed by adding a fully connected classification layer on top of the embedding of the special [CLS] token and backpropagating the gradient through the entire architecture. Before fine-tuning, the [CLS] token embeddings output by the pre-trained BERT-w models do not provide a good representation of the sentence. By contrast, the [CLS] token embeddings output by the pre-trained BERT-s models provides a reliable generic sentence representation.

- **BERT-s + FFNN**: fixed BERT sentence embeddings (as introduced and tested for similarity tasks by Reimers & Gurevych (2019)) with a Feed Forward Neural Network classifier.
- **BERT-s + COS:** fixed BERT sentence embeddings with cosine similarity measure. During the test phase the cosine similarity was calculated between each test instance and all training instances. The test instance was assigned with the label of the training instance with which the similarity was the highest.

These four approaches were implemented in *Python* with the *Tensorflow*[9], *Keras*[10] and *PyTorch*[11] libraries. The word and sentence transformer models were taken from the *huggingface*[12] repository.

3.3 Experiments and results

The training datasets described in Section 3.1 were used to create models, i.e., to train and tune on the 80% / 20% split from the shuffled training data. Then the created models were tested on the test datasets. The performance of each model was evaluated with the *accuracy*, *precision*, *recall*, and *f-score* metrics. The evaluation of *accuracy*, *precision*, *recall* and *f-score* metrics was performed using *sklearn.metrics*[13].

The *accuracy*, *precision*, *recall* and *f-score* values were averaged in 5 runs and the confidence intervals were calculated for all methods except BERT-s + COS. BERT-s + COS calculates a fixed similarity metric between two pre-trained vectors, therefore any run gives absolutely the same results (and confidence intervals cannot be calculated).

A model is considered reasonable if the calculated accuracy is above *random* and *majority* baselines:

$$\text{random\_baseline} = \sum_j P^2(c_j) \tag{3.1}$$

$$\text{majority\_baseline} = \max_j P(c_j) \tag{3.2}$$

with $P(c_j)$ the probability of class c_j. In our experiments *random* and *majority* baselines are equal to ~0.04 and ~0.09, respectively. The low random baseline value demonstrates the difficulty of the task for which random guess cannot be a solution. The low majority baseline values show that the dataset is not biased towards any class.

When comparing evaluation results, it is important to determine whether differences are statistically significant. For this purpose, McNemar's test (McNemar, 1947) with 95% confidence ($\alpha = 0.05$) has been used. Differences are considered statistically significant if the calculated *p* value is below $\alpha = 0.05$. The evaluation of statistical significance was performed using the *statsmodels.stats.contingency_tables* module in *Python*.

We investigated the following training and test conditions:

[9] TensorFlow, a free and open-source software library for machine learning. https://www.tensorflow.org/
[10] Keras: the Python deep learning API. https://keras.io
[11] PyTorch is an open source machine learning library. https://pytorch.org/
[12] Hugging face AI community. https://huggingface.co
[13] Scikit-learn, a free software machine learning library. https://scikit-learn.org

- **Monolingual** (see Section 3.3.1), where training and testing are done on one and the same target language. These experiments will demonstrate the performance of monolingual models trained on machine translated data.
- **Multilingual (train EN)** (see Section 3.3.2), where training is done on the EN training dataset, but testing is performed on the test dataset of some other target language (e.g., DE, FR, LT, LV, PT). These experiments will reveal if multilingual models trained in one language (EN in our case) can be useful for another (target) language.
- **Combined (train EN + target)** (see Section 3.3.3). These experiments combine the previous two approaches and are similar to multilingual (train EN). The only difference is that instead of EN alone, we use two training datasets of two languages (i.e., EN plus MT to the target language), whereas testing is done on the target language only. These experiments will show if the model benefits from training on the original dataset of a language (different from the target) complemented with the machine translated samples in the target language.
- **Combined (train all)** (see Section 3.3.4)**,** where training is done on all training datasets of all languages (except the target language) and testing is done on the test dataset of the target language. This is similar to the above method, but represents the scenario in which machine translated data for the target language cannot be obtained or are of very poor quality.

3.3.1 Monolingual experiments

For the monolingual set of experiments, we have tested all four approaches described in Section 3.2 on separate languages. Monolingual model evaluation results with BERT-w + CNN, BERT-w + BERT, BERT-s + FFNN and BERT-s + COS are presented in Tables 8, 9, 10, and 11 in Appendix A, respectively.

To see clearly which monolingual model is the best for each language, we have summarised these accuracies in Figure 5 by selecting the best word or sentence embedding for each language and each of the 4 classification methods. Methods based on sentence embeddings outperform methods based on word embeddings. The best results are from BERT-s + FFNN followed by BERT-s + COS. In most cases, the differences between their accuracies are not statistically significant.

Word embedding-based methods use sequences of concatenated word vectors to represent texts/sentences, whereas sentence embeddings aggregate the meaning of the text/sequence as a whole. Hence, it seems that sentence embeddings are the more natural way to represent texts for any language for the intent detection task.

The best overall accuracy (~0.842) is achieved on English data. This is not surprising as this result represents the topline: it uses the manually prepared training dataset and the sentence embedding model *roberta-base-nli-stsb-mean-tokens* that has been trained on English data only. This explains why the result is the best for English and worse for the other languages with multilingual sentence embeddings (i.e., *xlm-r-distilroberta-base-paraphrase-v1* and *xlm-r-bert-base-nli-stsb-mean-tokens*).

More importantly, these results clearly show that the approach of machine translating the training data is successful. The performance of models trained on machine translated data only is very close to the topline and is similar across all 5 languages, despite the fact that some languages are more closely related than the others.

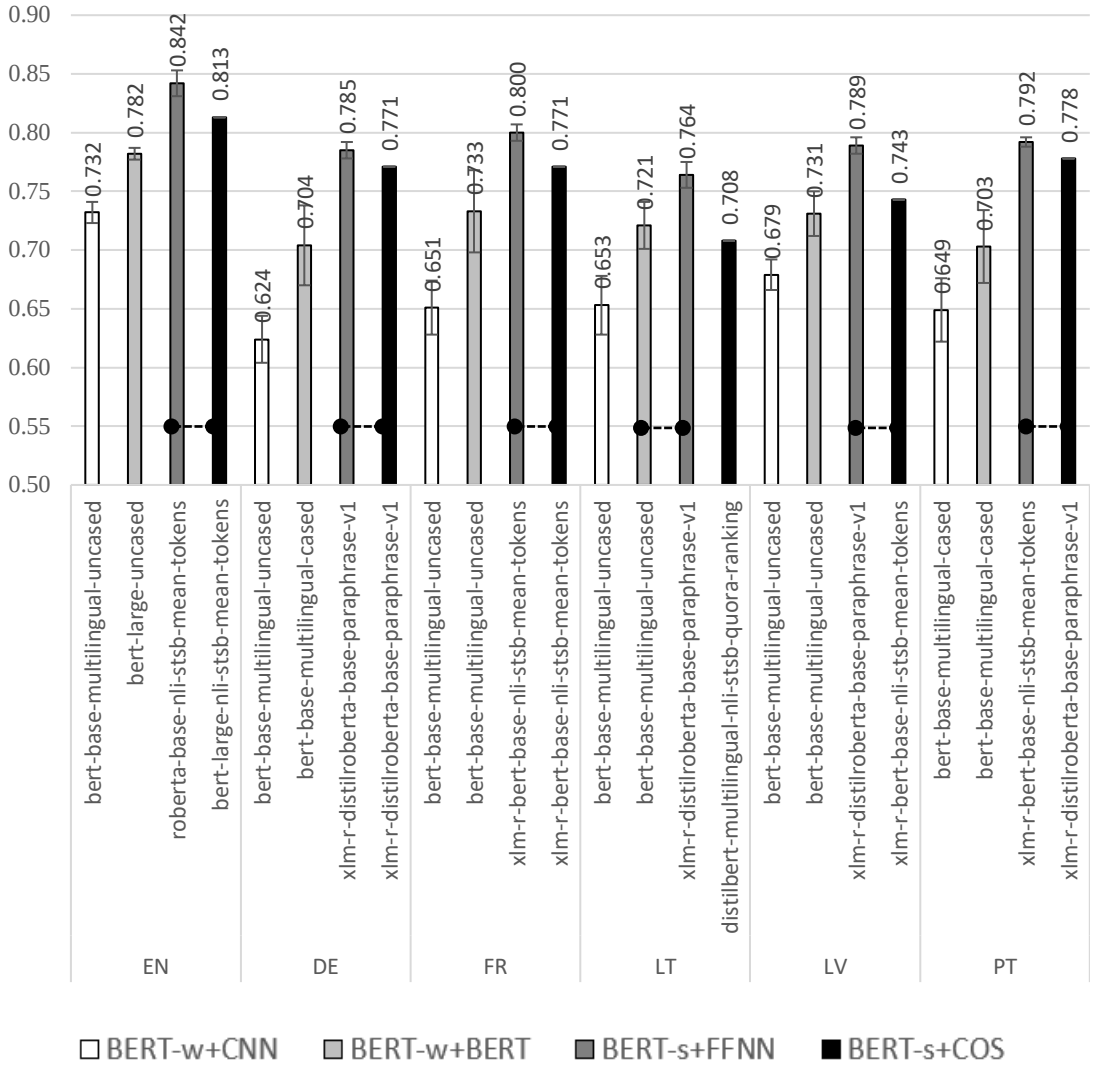

Figure 5. Intent detection accuracies and confidence intervals with BERT-w + CNN, BERT-w + BERT, BERT-s + FFNN, BERT-s + COS monolingual models. Dashed lines connect the best accuracy for a given language with those accuracies for which the difference is not statistically significant.

3.3.2 Training multilingual models on English only

The following experiments were performed under the "train EN" condition, i.e., we train-multilingual models on the EN training dataset alone and test on some other target language. The results with BERT-w + CNN, BERT-w + BERT, BERT-s + FFNN and BERT-s + COS are summarised in Tables 12, 13, 14, and 15 in Appendix A, respectively. The highest accuracies for each language are given in Figure 6.

Accuracies with sentence embeddings exceed 70%. This is surprisingly good as models haven't seen any training data in the target language. For GE and FR, BERT-s + FFNN is a better option whereas, for LT, LV and PT, BERT-s + COS achieves higher accuracy. This difference is not statistically significant for DE, LT and PT. Out of the 4 multilingual sentence embeddings, *xlm-r-bert-base-nli-stsb-mean-tokens* seems to be slightly better for LT and PT and *xlm-r-distilroberta-base-paraphrase-v1* for DE, FR, and LV.

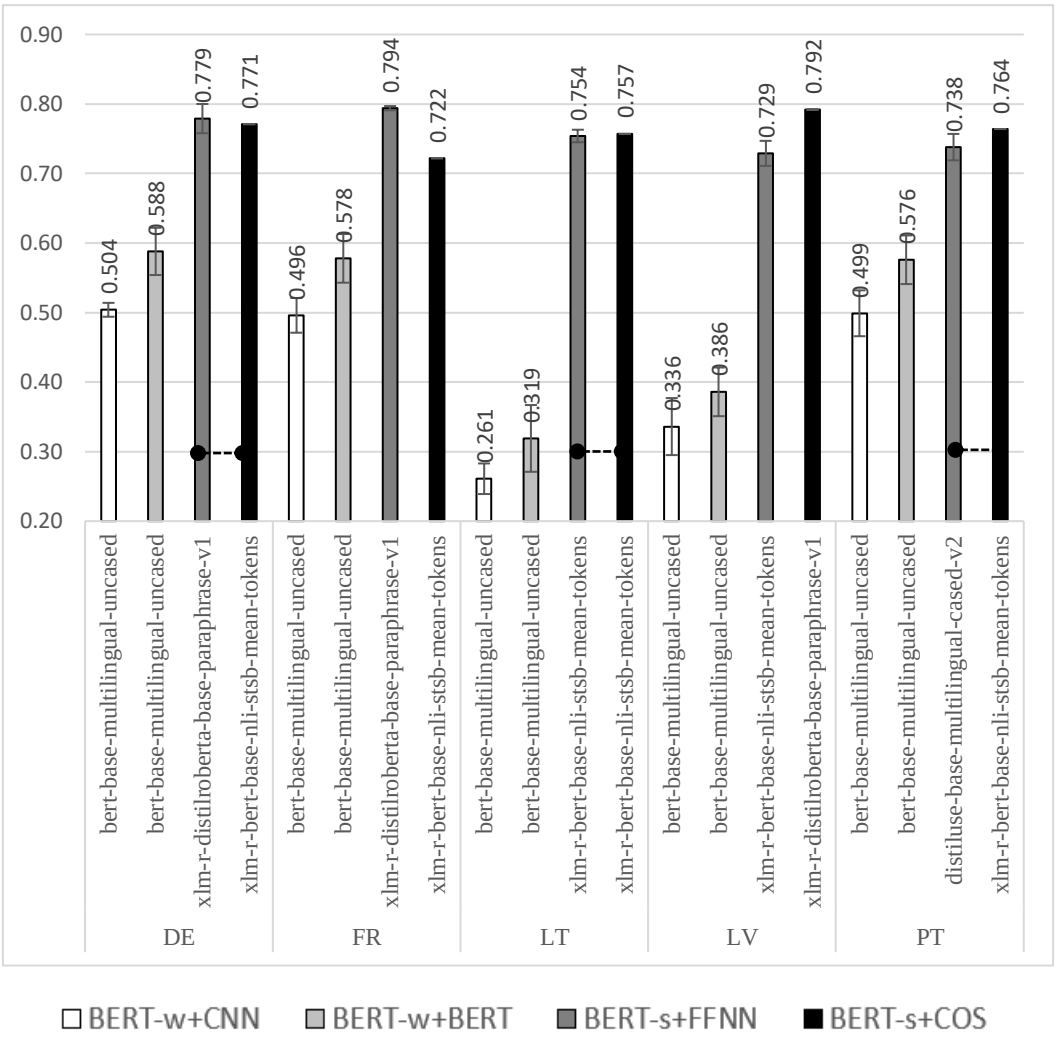

Figure 6. Intent detection accuracies and confidence intervals with BERT-s + FFNN, BERT-s + COS multilingual (train EN) models. For the notation see Figure 5.

Once again, we believe that the explanation of the sentence embeddings' success lies in the nature of vectorisation. A sentence is not a sequence of words (as with word embeddings), but their cumulative semantical meaning. Different languages have distinct word orders, expressions or functional words (articles, modal verbs, etc.), therefore the word embedding sequences representing a given sentence are very different. Despite this, the meaning of a given sentence remains the same across all languages, which explains the success of sentence embedding-based methods.

3.3.3 Training multilingual models on both English and target language

The experiments in this section are very similar to those in Section 3.3.2. The only difference is that the training is performed on two datasets of two languages (i.e., EN + the machine translated data for target language), whereas testing is done only on the target language. This represents the scenario in which both approaches (MT of training data and multilingual models) are combined. In these experiments we have tested only sentence embedding-based models, because they demonstrated much better performance in all previous comparative experiments.

The accuracies for BERT-s + FFNN and BERT-s + COS are summarised in Tables 16 and 17 in Appendix A, respectively. The best accuracies for each target language are presented in Figure 7.

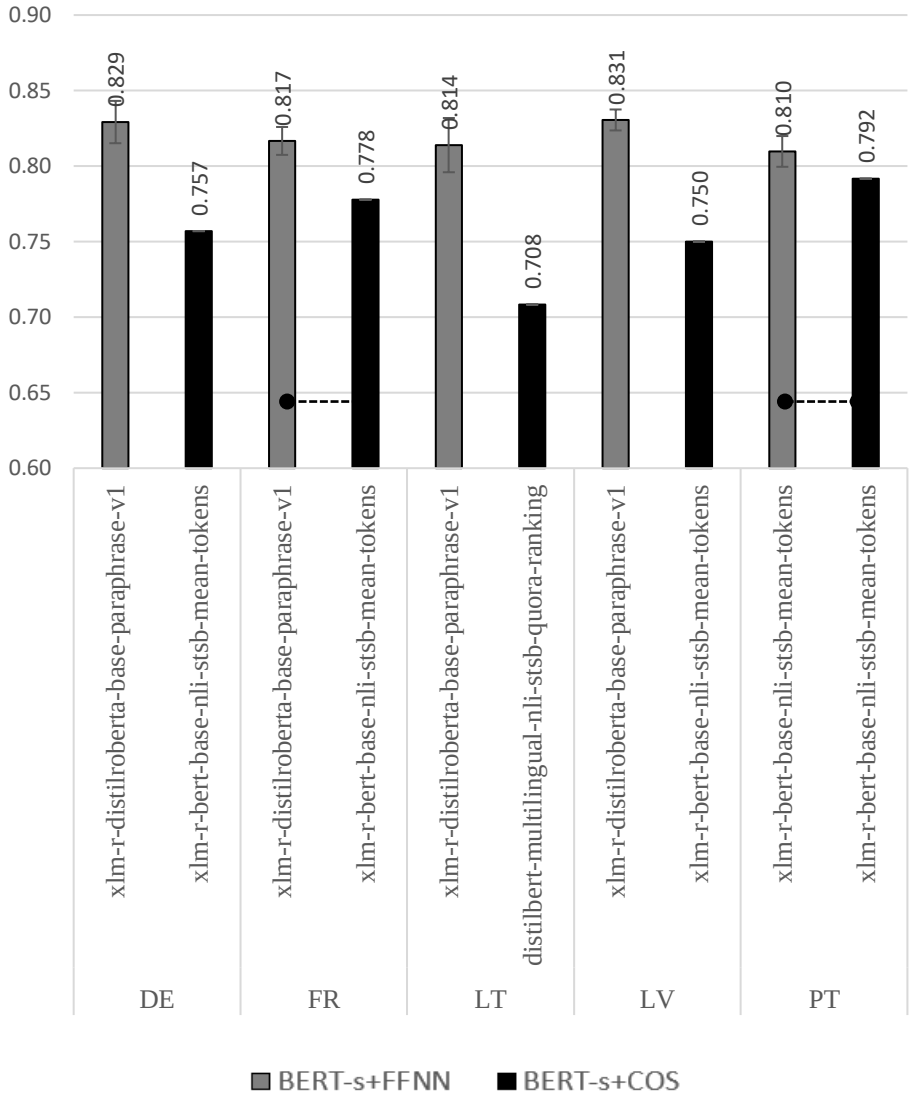

Figure 7. Intent detection accuracies and confidence intervals with BERT-s + FFNN and BERT-s + COS multilingual (train EN + target) models. For the notation see Figure 5.

The best accuracies exceeding 80% are achieved with the BERT-s + FFNN method. For FR and PT languages, the differences between BERT-s + FFNN and BERT-s + COS are insignificant. The obvious sentence embedding model winner with BERT-s + FFNN is *xlm-r-distilroberta-base-paraphrase-v1*, except for PT. However, its difference from *xlm-r-bert-base-nli-stsb-mean-tokens* is less than 0.3% and insignificant.

3.3.4 Training multilingual models on all languages

In this subsection we describe the final set of experiments, in which multilingual model training was performed on all training datasets for all languages (excluding the target one, i.e., “train all” condition) and testing was performed on the test dataset of the target language. This represents the scenario when MT for the target language is not available, hence we try to improve the model by using machine translated data for other languages.

The accuracies for BERT-s + FFNN and BERT-s + COS are summarised in Tables 18 and 19 in Appendix A, respectively. The best accuracies for each target language are presented in Figure 8.

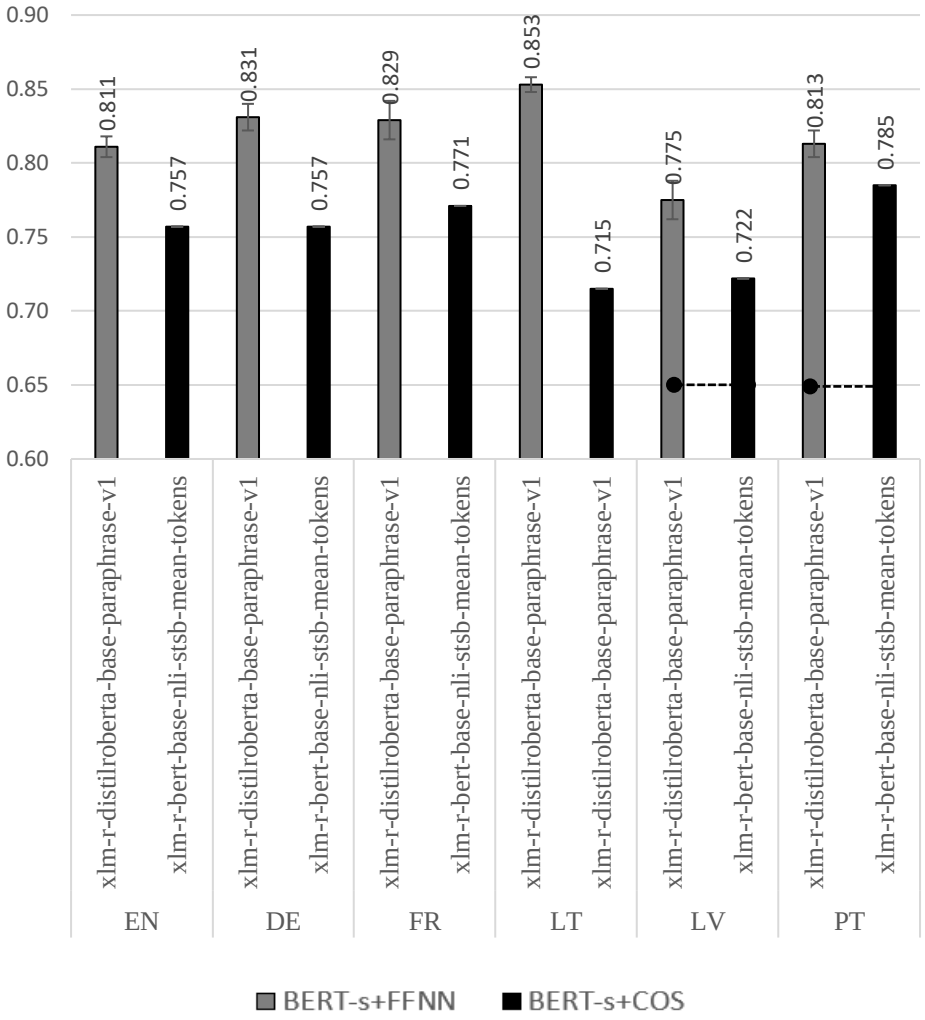

Figure 8. Intent detection accuracies and confidence intervals with BERT-s + FFNN and BERT-s + COS multilingual (train all) models. For the notation see Figure 5.

Zooming into Tables 18 and 19 and Figure 8 allows us to claim that the obvious winner is the BERT-s + FFNN method. For LV and PT, the difference in accuracy with BERT-s + COS is not significant. The best achieved accuracies exceed 80% (except for LV), which means that they exceed the results in Section 3.3.2 (train EN) and are competitive with the results in Section 3.3.3 (train EN + target). The good performance of these models can be easily explained: more training data (training data of different languages) and more diverse data (examples of different sentence structures for different languages with the same meaning) lead to more robust models.

3.3.5 Summary of results

To easily compare the best achieved accuracies in the monolingual and the three multilingual experiments, we have summarised them in Figure 9. The best results are obtained by the “train all” approach (train on all available languages, except the target) with the BERT-s + FFNN method, followed by multilingual training on the original EN and machine translated target language datasets (train EN + target), with the exception of LV

for which the opposite holds. There is no clear explanation why the LV language falls out of the picture, but it is obvious that it benefits most from having training samples of its own language.

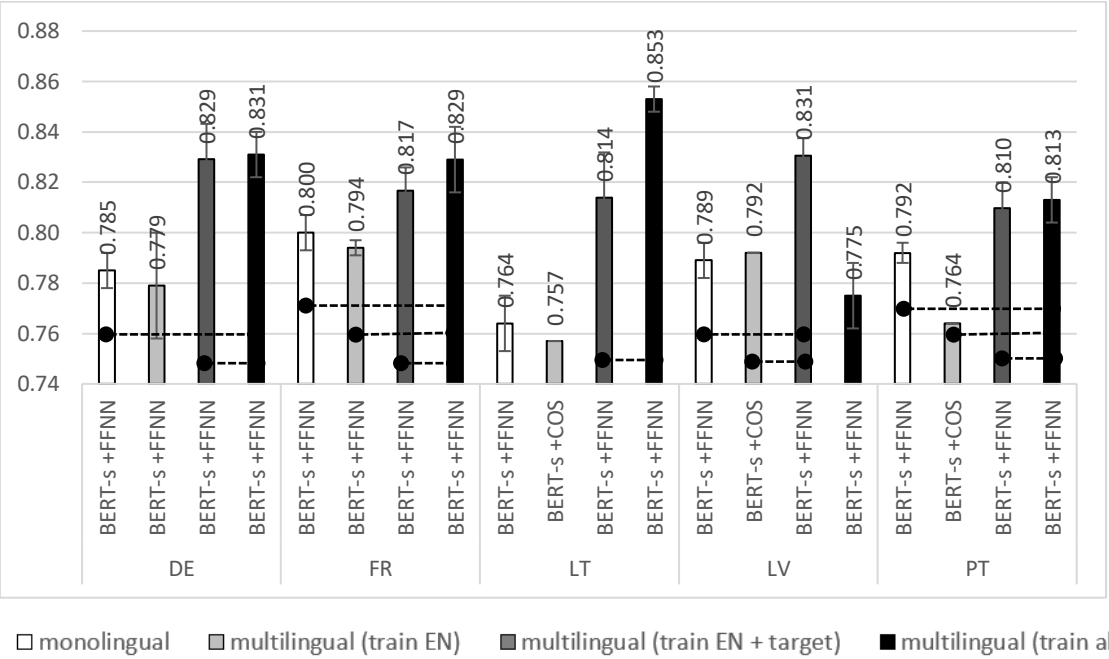

Figure 9. Intent detection accuracies and confidence intervals for different languages with monolingual and multilingual approaches. For the notation see Figure 5.

3.4 Conclusions and future work

The research presented in Section 3 attempts to solve a two-fold problem: 1) the intent-detection problem for several languages (English, German, French, Lithuanian, Latvian and Portuguese); 2) the annotated data scarcity problem, because such data for some languages does not exist. For this reason, the English benchmark training dataset (involving 41 intents) is machine translated via Google, whereas the test dataset is manually translated into all 5 languages. This is done intentionally to trying and simulate the situation when annotated data is available in other languages, but not the target language. The trained model is evaluated with manually prepared data, because it will be used by people asking questions in their own language.

The intent detection problem was solved by using two types of vectorisation (BERT word and sentence embeddings) and four classification (CNN, BERT fine-tuning, FFNN, and Cosine similarity) approaches. The annotated data scarcity problem was tackled with testing the previously described approaches under the following conditions: monolingual (when training is done on the machine translated data), multilingual (when a multilingual model is trained on English alone), and combined (a multilingual model trained on English complemented with machine translated samples in the target language or in several languages excluding the target one). The experiments revealed that both monolingual and multilingual approaches are equally effective. Furthermore, these approaches are complementary and can be combined, which leads to superior results for all target languages. It must also be noted that sentence embeddings are strongly recommended for the intent detection problem in all languages.

The best accuracy on English data is ~0.842, but similar accuracy levels can be reached for the other languages, namely ~0.831, ~0.829, ~0.853, ~0.831, and ~0.813 for German, French, Lithuanian, Latvian, and Portuguese, respectively. This allows us to conclude that the intent detection problem can effectively be solved in a language-independent fashion without having necessary training resources in the target language.

As future research, we plan to continue working in the multilingual direction by exploring different datasets and other classification problems. The most desirable result would be to finally make sure that the annotated data scarcity problem for some languages is not a problem anymore.

4 Software components

This section describes the components of the multilingual interaction library. Several of these components (Sections 4.1.2, 4.3, 4.4, 4.5) are new, while the others were presented in Deliverable "D3.1 – Initial multilingual interaction library" (Submitted to the European Commission on February 28, 2020 − Public). For the sake of completeness, all of them are presented below.

4.1 Machine translation

Ideally, MT would be part of the COMPRISE Client Library and run locally on a mobile device. However, MT requires a large amount of computational, run-time memory, and storage resources, which are typically unavailable on mobile devices. Since the development of on-device MT would require a separate project on its own and the development of a new MT service is out of the scope of COMPRISE, we provide two alternative solutions: cloud-based translation via Tilde MT, or containerised MT systems.

4.1.1 Machine Translation API

This section provides the documentation of the basic Tilde MT API functions which are called by the COMPRISE Client Library (see Work Package 4). Full documentation can be found online.[14] The Tilde MT API implements a RESTful calling style over HTTPS, i.e., all calls are inherently encrypted.

4.1.1.1 Authentication

All Tilde MT API requests must contain an authentication token in the HTTP header that identifies the user of Tilde MT.

Developers of applications using COMPRISE libraries will need to contact Tilde and get their client-id *authorisation_token*. Below is the header example of an HTTP request with an authorisation token:

```
GET https://www.letsmt.eu/ws/service.svc/json/GetSystemList?appID=appid HTTP/1.1
client-id: authorisation_token
```

4.1.1.2 List of available systems

Tilde MT API provides a method to list available translation systems for an authenticated user:

[14] Tilde Machine Translation API. https://www.tilde.com/developers/machine-translation-api

```
GET https://www.letsmt.eu/ws/service.svc/json/GetSystemList?appID=appid HTTP/1.1
client-id: authorisation_token
```

where *appid* is a string application identifier chosen by the developer.

The response will contain a JSON document containing the description of the systems available to the user or the application. The table below presents a short description of the main metadata entries of the MT system:

Name	Type	Description
ID	string	MT system ID
SrcLanguage	complex	Source language (IETF RFC 5646)
TrgLanguage	complex	Target language (IETF RFC 5646)
Domain	string	Translation domain
Title	string	Title of the MT system
Description	complex	Description of the MT system
Metadata	complex	More metadata about the system

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 916
{
"System":[
{
"Description":{  … },
"Domain":"law",
"ID":"smt-ea364952-816a-4491-9fbc-35b41056730a",
"Metadata":[  … ]
}, … ]}
```

4.1.1.3 Translation

Knowing the MT system ID, developers can translate plain text or text with inline tags using the Tilde MT API:

```
GET https://www.letsmt.eu/ws/service.svc/json/Translate?appID=appid&sys-
temID=sys&text=text client-id: authorisation_token
```

where:

1. *appid* is a string application identifier.
2. *sys* is a string translation system ID, obtained using the method in Section 3.1.2.
3. *text* is the text to be translated (with or without inline tags).

For translating longer text, it is recommended to use the HTTP POST method:

```
POST https://www.letsmt.eu/ws/service.svc/json/Translate HTTP/1.1
client-id: authorisation_token
Content-Type: application/json; charset=utf-8
Content-Length: XXX
{
"appID": "appid",
"systemID": "sys",
"text": "saule spīd spoži",
"options": ""
}
```

If a request is successful, the response will contain the translated text:

```
HTTP/1.1 200 OK
Content-Length: 23
Content-Type: application/json; charset=utf-8
"the sun shines bright"
```

In the case of an error the response will contain an error description:

```
code: 22
exception: systemNotFound
description: Translation system not found
```

4.1.2 Containerised Machine Translation systems

Containerised MT systems were also developed within WP3. The Docker containers are meant to be run inside an instance of the COMPRISE Personal Server, so that the input and output texts are not exposed to any cloud service provider.

Currently, the translation is performed on the CPU only. This is adequate for the translation of short queries that are typical with voice assistant systems. This also reduces hardware requirements and therefore increases the range of servers on which these containerised MT systems may be installed. A version with GPU support can be provided by Tilde upon request.

To load the Docker image into your system run the following command:

```
sudo docker load -i image name.tar.gz
```

After that you can run the Docker container and map the two exposed ports: port 10000 for the paragraph translation API, and port 5000 for the sentence translation API. For example, for the LV->EN translation Docker, the following command can be used:

```
sudo docker run --rm -p 10000:10000 -p 5000:5000 tilde-en-lv-translator
```

The Docker containers include the NMT models which are made accessible for translation via two different web services (each exposing a different API):

1. a web service for translating paragraphs, which automatically performs sentence splitting;
2. a lower-level web service for translating single sentences or arrays of sentences, which does not perform sentence splitting.

The paragraph translation API can be called as follows:

```
$ curl http://localhost:10000/translate -X POST -H "Content-Type: application/json" -d '{"texts": ["Hello world! The weather is nice.", "This is another paragraph."]}'
```

The result will be a JSON object containing both original and translated paragraphs.

```
{"translations": [{"text": "Hello world! The weather is nice.", "translation": "Sveiks, pasaule! Laiks ir jauks."}, {"text": "This is another paragraph.", "translation": "\u0160\u012b ir v\u0113l viena rindkopa."}]}
```

Single sentences can be translated using the following request to the sentence translation API:

```
$ curl http://localhost:5000/translate -X POST -H "Content-Type: application/json" -d '{"text": "Single sentence example."}'
```

The result will be a plaintext translation of the source sentence:

```
Viena teikuma piemērs.
```

The sentence API can also be used to translate an array of sentences:

```
$ curl http://localhost:5000/translate/batch -X POST -H "Content-Type: application/json" -d '{"texts": ["Hello world!", "The weather is nice.", "No paragraphs here, just sentences."]}'
```

The result will be a JSON array of translations:

```
["Sveiks, pasaule!","Laiks ir jauks.","Šeit nav rindkopu, tikai teikumi."]
```

All of the relevant services in the container are managed with `supervisord`, which keeps the log files for each of the services in the /tmp directory. To access the log files, run the following commands:

```
sudo docker ps

# identify the relevant container id

sudo docker exec -ti <id_from_above> bash

cd /tmp

less *.log
```

Table 7 lists the provided MT system Docker images and the URLs for download.

Table 7. Dockerised MT systems.

Direction	URL
EN -> LV	https://mt-tilde-tmp.s3-eu-west-1.amazonaws.com/docker-image-tilde-en-lv-translator.tar.gz
EN -> DE	Provided by Tilde upon request.
EN -> FR	Provided by Tilde upon request.

Direction	URL
EN -> PT	Provided by Tilde upon request.
EN -> LT	Provided by Tilde upon request.
LV -> EN	https://devcomprise.blob.core.windows.net/mt-dockers/docker-image-tilde-lv-en-translator.tar.gz
DE -> EN	Provided by Tilde upon request.
FR -> EN	Provided by Tilde upon request.
PT -> EN	Provided by Tilde upon request.
LT -> EN	Provided by Tilde upon request.

4.1.3 Terms of Use

Tilde grants the rights to use Tilde's MT technologies (both MT API and MT containers for the COMPRISE Personal Server) to the project partners for non-commercial purposes in the scope of the COMPRISE project. The project partners may use these MT technologies for research, development, evaluation, and dissemination activities. The rights to use Tilde's MT technologies outside of the project scope or for commercial purposes can be agreed upon with Tilde.

4.2 Dialogue Management

The Dialogue Management functionality of the multilingual interaction library is provided by Tilde.AI's cloud-based dialogue system (see Figure 10). This system comprises SLU, Dialogue Management, and Spoken Language Generation in an integrated framework.

Normally, the developer of a mobile app will create a dialogue application with the Tilde.AI dashboard by defining a dialogue scenario, intents, actions, etc. Then, the created dialogue can be integrated into the mobile app using the Bot Service API.

In some cases, a developer might want to implement dialogue management and text generation externally and only use Tilde.AI for SLU functions. For these cases, a separate NLU API is provided by the Tilde.AI dialogue system.

Interaction with the Bot Service API and Tilde.AI NLU API are integrated into the COMPRISE Client Library, therefore developers using COMPRISE SDK do not need to access these APIs directly.

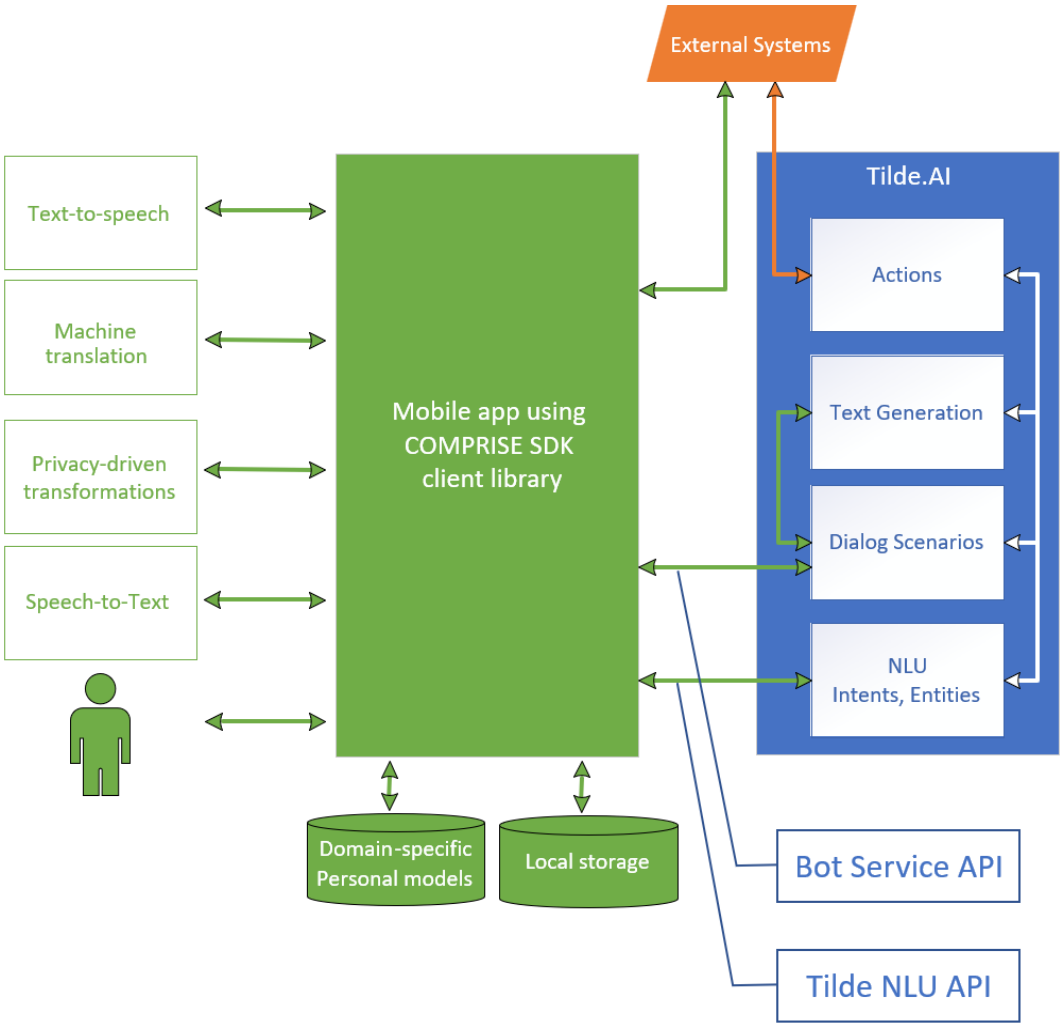

Figure 10: Interaction between the Tilde.AI dialogue system and the COMPRISE Client Library.

4.2.1 Dialogue management API

The Tilde.AI dialogue system is hosted on Azure Bot Service and uses Microsoft Bot Framework for user communication. The preferred method of interacting with the Bot Framework is the SDK provided by Microsoft for the following programming languages and platforms: Python, JavaScript, .NET, and Node.js.

If the SDK is not available for the chosen programming language or if it cannot be used for some other reason, then there are several Bot Service APIs that can be called directly:

- Bot Framework REST API.
- Direct Line API.

Full documentation of these APIs can be found on Microsoft's website[15,16]. Therefore, here we will only provide a short description of base Direct Line API 3.0 functions.

4.2.1.1 Authentication

A mobile app can authenticate requests to Direct Line API 3.0 either by using a `secret` obtained from the Tilde.AI dashboard or by using a `token` which is obtained at runtime. The `secret` or `token` should be specified in the authorisation header of each request using the following format:

```
Authorization: Bearer SECRET_OR_TOKEN
```

A Direct Line secret is a master key that can be used to access any conversation that belongs to the associated bot. A secret can also be used to obtain a token. Secrets do not expire.

A Direct Line token is a key that can be used to access a single conversation. A token expires but can be refreshed.

4.2.1.2 Start a conversation

Direct Line conversations are explicitly opened by the mobile app and may run as long as the user and dialogue system participate and have valid credentials. While the conversation is open, both the dialogue system and the user may send and receive messages.

```
POST https://directline.botframework.com/v3/directline/conversations
Authorization: Bearer SECRET_OR_TOKEN
```

If the request is successful, the response will be a JSON object containing an ID for the conversation, a token, a value that indicates the number of seconds until the token expires, and a stream URL that the client may use to receive activities via a WebSocket stream:

```
{
"conversationId": "abc123",
"token": "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qbOF5xnR2vtCX7CZj0LdjAPG-
fiCpg4Fv0y8qbOF5xPGfiCpg4Fv0y8qqbOF5x8qbOF5xn",
"expires_in": 1800,
"streamUrl": "https://directline.botframework.com/v3/directline/conversa-
tions/abc123/stream?t=RCurR_XV9ZA.cwA..."
}
```

4.2.1.3 Send an activity to the dialogue system

Using the Direct Line 3.0 protocol, the user and the dialogue system may exchange different types of activities, including message activities, typing activities, etc.

---

[15] Bot Framework REST API reference
https://docs.microsoft.com/en-us/azure/bot-service/rest-api/bot-framework-rest-connector-api-reference?view=azure-bot-service-4.0

[16] Bot Framework Direct Line API 3.0 reference
https://docs.microsoft.com/en-us/azure/bot-service/rest-api/bot-framework-rest-direct-line-3-0-api-reference?view=azure-bot-service-4.0

The following snippet provides an example of how to send a text message to a dialogue system:

```
POST https://directline.botframework.com/v3/directline/conversations/convid/activities
Authorization: Bearer SECRET_OR_TOKEN
Content-Type: application/json
[other headers]
```

where *convid* is a string identifier of the conversation obtained after starting a conversation.

The request body shall contain a JSON object describing the text message:

```
{
    "type": "message",
    "from": {        "id": "user1"    },
    "text": "hello"
}
```

If the POST request is successful, the response contains a JSON payload that specifies the ID of the activity that was sent to the bot.

```
{
    "id": "0001"
}
```

4.2.1.4 Receive activities from the dialogue system

By using the Direct Line 3.0 protocol, clients can receive activities via a WebSocket stream or retrieve activities by issuing HTTP GET requests.

To retrieve messages for a specific conversation using HTTP GET, the client should issue the following request:

```
GET https://directline.botframework.com/v3/directline/conversations/convid/activities?watermark=w
Authorization: Bearer SECRET_OR_TOKEN
```

where:

- *convid* is a string identifier of the conversation obtained after starting a conversation.
- w is an optional watermark parameter to indicate the most recent message seen by the client.

The response will be a JSON object containing new activities (messages) from the user and the dialogue system:

```
{
    "activities": [
        {
            "type": "message",
            "channelId": "directline",
            "conversation": {
                "id": "abc123"
```

```
                },
                "id": "abc123|0001",
                "from": {
                      "id": "bot1"
                },
                "text": "Nice to see you, user1!"
          }
     ],
     "watermark": "0001a-95"
}
```

Clients should page through the available activities by advancing the watermark value until no activities are returned.

4.2.2 Spoken language understanding

A developer might want to implement their own dialogue management and text generation systems, but does not want or does not have the capability to develop SLU. In this case, a developer can use the NLU API provided by Tilde.AI.

The mobile app might be interested in two particular functions of NLU API: intent detection and entity recognition. Full documentation of these and other NLU API functions can be found on the Tilde.AI developer portal.[17] However, in this document we will only provide a short description of intent detection and entity recognition functions.

4.2.2.1 Authentication

All Tilde.AI NLU API requests must contain a subscription key in the HTTP header that identifies a user. This key can be obtained after signing up on the Tilde.AI developer portal. Below is the header example of an HTTP request with an authorisation token:

```
GET https://dev-nlu1-am.azure-api.net/api/Train/languages
Ocp-Apim-Subscription-Key: subscription_key
```

4.2.2.2 Intent detection

The intent of the given text can be detected using the following POST request:

```
POST https://dev-nlu1-am.azure-api.net/api/Guess/AppId/LangId
Ocp-Apim-Subscription-Key: subscription_key
```

where:

- *AppId* is the app to be used to detect the intent. It is obtained via other API methods (see full documentation) or by using the COMPRISE SDK.
- *LangId* is the language of the text, e.g., “en-en”, “lv-lv”, etc.

The request body shall contain a text wrapped in double quotes.

```
"Hello. Can you detect intent on this text."
```

[17] Tilde.AI developer portal https://dev-nlu1-am.portal.azure-api.net/docs/services/natural-language-understanding-nlu/operations/PostAddEntities

The response will be a JSON object containing the detected intents sorted by descending confidence.

```
[
        { "intentid": "greeting", "confidence": 0.607079566 },
        { "intentid": "some_intent", "confidence": 0.261552066 },
        { "intentid": "some_intent_2", "confidence": 0.131368339 }
]
```

4.2.2.3 Entity recognition

The Tilde NLU API can be used to extract entities from a given text. Entities should be defined beforehand using specialised API functions (see full documentation) or by using the COMPRISE SDK. Entity recognition is available by performing an HTTP POST request:

```
POST https://dev-nlu1-am.azure-api.net/api/Entity/AppId/LangId
Ocp-Apim-Subscription-Key: subscription_key
```

where

- *AppId* is the app to be used for entity recognition. It is obtained via other API methods (see full documentation) or by using the COMPRISE SDK.
- *LangId* is the language of the text, e.g. “en-en”, “lv-lv”, etc.

The request body shall contain a text for which entity recognition should be performed:

```
"I would like to book a flight to London”
```

If the POST request is successful, the response will be a JSON object containing the detected entities and their position in the input.

```
[
        {
        "dim": "@is_alive_NoNoKnowldgeAPI",
        "body": "is_alive_NoNoKnowldgeApi",
        "value": { "value": "is_alive_NoNoKnowldgeAPI", "type": "value" },
        "start": 0,
        "end": 39

        }, {
        "dim": "@destination",
        "body": "destination",
        "value": { "value": "London", "type": "value" },
        "start": 33,
        "end": 39
                }
]
```

“is_alive_NoNoKnowldgeAPI” is an example entity which is hardcoded for every app for debugging.

4.2.3 Terms of Use

Tilde grants the rights to use the Tilde.AI API and user interface to the project partners for non-commercial purposes in the scope of the COMPRISE project. The project partners may use these technologies for research, development, evaluation, and dissemination activities. The rights to use Tilde.AI technologies outside of the project scope or for commercial purposes can be agreed upon with Tilde.

4.3 Synthetic data pipeline tools

Relevant code for reproducing the synthetic data filtering and rule-based error generation results described in Sections 2.3 and 2.4 is published in the COMPRISE GitLab repository.[18]

Before running the provided scripts, one should install the conda[19] package management system and create an environment by running:

```
conda env create -f environment.yml.
```

The synthetic speech translation dataset can be filtered by running the following command:

```
python speech_translation.py filter --workdir= --source= --source_asr= --target= --source_pos= --result_source= --result_source_asr --result_target --result_source_pos
```

where:

- source – file containing sentences in the source language;
- source_asr – file containing source sentences processed by synthetic data pipeline;
- source_pos – POS tags for source sentences in Moses factored data format;[20]
- target – file containing sentences in the target language.

Sentences in input files are separated by the newline character, so that each line contains exactly one sentence. All input files should be aligned line-by-line.

In order to perform Rule-Based Synthetic Noise Generation first a suffix noise or suffix error model should be learned from the filtered synthetic dataset. This can be done by:

```
python speech_translation.py learn_noise_model --workdir=workdir --source= --source_pos= --source_asr= --result_model=
```

Then, the suffix noise model can be applied to any parallel dataset.

```
python speech_translation.py apply_noise_model --source= --source_pos= --target= --result_source= --result_target= --noise_model=
```

[18] https://gitlab.inria.fr/comprise/deliverables/deliverable-d33/-/tree/master/speech_translation

[19] Conda open-source package management system and environment management system. https://docs.conda.io/en/latest/

[20] Moses factored data format. http://www.statmt.org/moses/manual/manual.pdf (page 241)

STT-like word splitting and merging noise can be added to the parallel training data by running:

```
python speech_translation.py generate_split_merge --source= --target= --result_target= --result_source=
```

We also provide an example script for creating the Kaldi data directory from a text file by applying a TTS engine. The resulting data directory can then be processed by Kaldi to create the synthetic training data for the MT engine.

The script is provided only as an example since it relies on TILDE's TTS voices (which are not provided). However, relevant sections can be easily replaced with calls to other TTS engines and then the script can be run as follows:

```
python synth_dataset.py datadir < text_file
```

where:

- datadir – name of the output directory (must exist before calling);
- text_file – text file to synthesise (each sentence on a separate line).

4.4 Disfluency detection

The multi-task self-supervised disfluency detection model code implemented within Task T3.1 is published in the COMPRISE GitLab repository.[21]

4.4.1 Training

The disfluency detection model training script depends on the SRILM language modelling toolkit that must be acquired separately. After installation, the correct path for SRILM should be inserted into the Makefile.

Before training you need to obtain a monolingual text corpus. Such corpus can be obtained for Latvian and some other languages from the WMT2017 webpage.

Then, the corpus should then be tokenised, lowercased, filtered from punctuation and other non-word tokens and split into files *train.txt*, *dev.txt*, and *test.txt*.

You will also need some amount of labelled data that should be stored in the file *tune.txt* where disfluent words are tagged with ":D" and other words with ":O". See example below:

```
each:O utterance:O in:O separate:O line:O lowercased:O without:O punctuation:O with:O disfluent:D words:O tagged:O uhm:D with:O D:O
```

To parse *tune.txt* file and prepare data for fine-tuning the following command should be executed:

```
make tune.testtag
```

Finally, the training can be started:

[21] https://gitlab.inria.fr/comprise/deliverables/deliverable-d33/-/tree/master/disfluency-detector

```
make tune.mdl
```

This will train a multi-task disfluency detection model in a self-supervised way on unlabelled training data from train.txt and then perform fine-tuning on human labelled data.

4.4.2 Inference

After model training and fine-tuning is finished, the model can be used to tag any plain text file.

For example, in order to tag a .txt file one can use the following command:

```
python3 tag.py --iter tune.mdl . input.txt > output.tags
```

4.5 Multilingual intent detection

The scripts of the multilingual BERT-s + FFNN and BERT-s + COS methods are published in the COMPRISE GitLab repository.[22]

Before running these scripts, the following Python packages must be installed: *numpy*, *scikit-learn*, *tensorflow*, *keras*, *pytorch*, *transformers*, *sentence_transformers*.

The scripts BERT_s_FFNN.py and BERT_s_COS.py for BERT-s + FFNN and BERT-s + COS, respectively, are located in the main directory (in our case, *multilingual*). All the training and test data is located in *multilingual/data*. The *data* directory contains subdirectories for separate languages (in our case: *de*, *en*, *fr*, *lt*, *lv*, *pt*) and two text files *train_cls.txt* and *test_cls.txt* (see Figure 11).

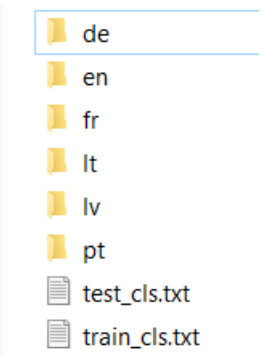

Figure 11. Distribution of sub-directories and files in directory data.

Each sub-directory contains two text files *train.txt* and *test.txt* with training and test instances (one text per line) used in the intent detection task (a snippet of the training file *train.txt* is in Figure 12). Since the order of instances in *train.txt* and *test.txt* files for different languages is the same, the corresponding classes are stored in *multilingual/data/train_cls.txt* and *multilingual/data/test_cls.txt* files (a snippet of the training classes file *train_cls.txt* is in Figure 13).

[22] https://gitlab.inria.fr/comprise/deliverables/deliverable-d33/-/tree/master/multilingual-intent-detection

```
Where can I download Tildės Biuras?
Where can I find Tildės Biuras?
Tildės Biuras address for downloading
Where can I find Tilde's installation file?
Do you have a demo version of Tildės Biuras?
I want to try a demo version of Tildės Biuras
Can I try Tildės Biuras for free?
What versions of Windows can be used for Tildės Biuras?
Windows for Tildės Biuras
```

Figure 12. Snippet of the *train.txt* file in the *multilingual/data/en* directory.

```
TB_download_q
TB_download_q
TB_download_q
TB_download_q
TB_about_demo_q
TB_about_demo_q
TB_about_demo_q
TB_about_os_q
TB_about_os_q
```

Figure 13. Snippet of the *train_cls.txt* file in the *multilingual/data* directory.

The script BERT_s_FFNN.py can be run with the following command (e.g., python BERT_s_FFNN en de):

```
python BERT_s_FFNN.py <train_language> <test_language>
```

where <*train_language*> and <*test_language*> parameters determine sub-directories in the *data* directory and correspond to separate languages used for training and testing the multilingual intent detection model. The trained model is used on the evaluation dataset and the calculated *accuracy*, *precision*, *recall* and *f-score* values are stored in the *test_result.txt* file, located in the main *multilingual* directory.

The script BERT_s_COS.py can be run with the following command (e.g., python BERT_s_COS.py en pt):

```
python BERT_s_COS.py <train_language> <test_language>
```

where <*train_language*> and <*test_language*> parameters, as in the previous case, determine separate training and test languages. The evaluation results are also stored in the *test_result.txt* file (see the snippet in Figure 14). The first column determines the used method (BERT-s + FFNN or BERT-s + COS), the second – languages used for training and test; the third – sentence embedding model; the fourth, fifth, sixth and seventh columns represent *accuracy*, *precision*, *recall* and f-score values, respectively.

```
BERT-s+FFNN     en->de  distiluse-base-multilingual-cased-v2    0.757   0.827   0.711   0.702
BERT-s+COS      en->pt  distiluse-base-multilingual-cased-v2    0.674   0.815   0.663   0.647
BERT-s+FFNN     en->en  distiluse-base-multilingual-cased-v2    0.771   0.866   0.751   0.740
```

Figure 14. Snippet of the *test_result.txt* file in the *multilingual* directory.

It is important to notice, that the same scripts can be used in the monolingual experiments. The only difference is that the *<train_language>* and *<test_language>* parameters have to be equal (e.g., python BERT_s_FFNN.py en en).

5 Conclusion

This document presented the work performed in Tasks T3.1 and T3.2 of Work Package 3. This work resulted in a collection of software components, APIs and methodologies, which is called the “Final Multilingual Interaction Library” and which enables any user to interact with dialogue systems in any language.

A synthetic data pipeline approach was proposed to overcome the mismatch between the output of the STT and the expected input of the MT. The idea is to imitate typical errors in the STT output by processing the MT training data with a TTS and STT pipeline. Several MT systems were trained and evaluated. It was noticed that using synthetic data in MT training resulted in a clear benefit when translating STT output. The result can be further improved if we apply special filtering to the synthetic data and also augment it with data generated by rule-based methods, this increases the translation quality by 1.87 BLEU points over the baseline system.

A self-supervised approach was studied to address disfluency detection in the speech recognition output. Two disfluency-tagged datasets (2,000 and 3,000 sentences) were created for model fine-tuning and evaluation. On both datasets the self-supervised approach outperformed the baseline trained without self-supervision: 68% F1 versus 54% F1 on the first dataset and 74% versus 72% on second dataset. The second dataset is less ambiguous therefore both models show much better results. Also, it was observed that: (1) adding more data improves the performance of both models, (2) the self-supervised approach allows us to achieve the same performance as the baseline using half as much data.

We proposed two solutions to make dialogue systems accessible in multiple languages when only English training data is available, which are based on monolingual (using machine translated training data) or multilingual (trained on EN data, but tested on another language) models. The analysis of results from different angles allows us to conclude that both solutions are equally good when sentence embeddings are used for vectorisation. Furthermore, these approaches are complementary and can be combined, which leads to superior results for all languages.

The second half of the deliverable described the main software components of the Final Multilingual Interaction Library. We provided documentation for the main functions of the Tilde MT API and the Tilde.AI dialogue system. Full, up-to-date documentation of these APIs are provided online (links can be found in the corresponding sections of the document). We also provided descriptions for various tools developed within the research activities, namely: synthetic data pipeline processing and filtering tools, multi-task self-supervised disfluency detection model implementation and multilingual intent detection model implementation.

6 Bibliography

Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., . . . & Negri, M. (2016). Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers* (pp. 131-198).

Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huang, S., Huck, M., Koehn, P., Liu, Q., Logacheva, V., Monz, C., Negri, M., Post, M., Rubino, R., Specia, L., & Turchi, M. (2017). Findings of the 2017 Conference on Machine Translation (WMT17). In *Proceedings of the Second Conference on Machine Translation*, *Volume 2: Shared Task Papers* (pp. 169–214).

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT 2019*, Minneapolis, MN, USA, 2–7 June 2019 (pp. 4171–4186).

Hassan, H., Schwartz, L., Hakkani-Tür, D., & Tür, G. (2014). Segmentation and disfluency removal for conversational speech translation. In *Proceedings of INTERSPEECH* (pp. 318–322).

Hendrycks, D., and Gimpel, K. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. arXiv preprint arXiv:1606.08415.

Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., . . . & Martins, A. F. (2018). Marian: Fast Neural Machine Translation in C++. In *Proceedings of the 56th Annual Meeting of the ACL, System Demonstrations* (pp. 116-121).

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 25–29 October 2014 (pp. 1746–1751).

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., . . . Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL, Interactive Poster and Demonstration Sessions* (pp. 177–180).

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition; *IEEE:* Pasadena, CA, USA, (pp. 2278–2324).

Levenshtein, V.I. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals, Soviet Physics Doklady 10(8) (1966), 707–710.

McNemar, Q.M. (1947). Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. *Psychometrika*, 12(2), 153–157.

Reimers, N & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using SiameseBERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* 2019, Hong Kong, China, November 2019 (pp. 3982–3992).

Salimbajevs, A., & Strigins, J. (2015). Error Analysis and Improving Speech Recognition for Latvian language. In Proceedings of 10th International Conference on Recent Advances in Natural Language Processing (RANLP 2015) (pp. 563–569).

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the ACL* (pp. 1715–1725).

Simonnet, E., Ghannay, S., Camelin, N., & Estève, Y. (2018). Simulating STT errors for training SLU systems. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation* (pp. 3157-3162).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998-6008).

Wang S, Che W, Liu Q, Qin P, Liu T, Wang WY. Multi-Task Self-Supervised Learning for Disfluency Detection. In: The Thirty-Fourth {AAAI} Conference on Artificial Intelligence, {AAAI} 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, {IAAI} 2020, The Tenth {AAAI} Symposium on Educational Advances in Artificial Intelligence, {EAAI} 2020, New York, NY, USA, February 7-12, 2020 [Internet]. {AAAI} Press; 2020. p. 9193–200.

A Detailed experimental results

Table 8. Monolingual experiments with the BERT-w + CNN method. The table contains averaged *accuracy*, *precision*, *recall* and *f-score* values followed by confidence intervals. The best results for each language are emphasised in bold.

Language	BERT model	Accuracy	Precision	Recall	F-score
EN	bert-base-cased	0.697±0.016	0.778±0.017	0.674±0.027	0.722±0.020
	bert-base-uncased	0.714±0.014	0.793±0.009	0.689±0.016	0.737±0.011
	bert-large-cased	0.714±0.019	0.794±0.024	0.685±0.016	0.735±0.019
	bert-large-uncased	0.653±0.022	0.739±0.023	0.628±0.022	0.679±0.021
	bert-base-multilingual-cased	0.703±0.019	0.782±0.017	0.672±0.019	0.723±0.018
	bert-base-multilingual-uncased	**0.732**±0.009	**0.801**±0.014	**0.696**±0.012	**0.745**±0.012
DE	bert-base-multilingual-cased	0.614±0.026	0.745±0.034	0.560±0.028	0.639±0.027
	bert-base-multilingual-uncased	**0.624**±0.020	**0.748**±0.019	**0.578**±0.025	**0.652**±0.022
FR	bert-base-multilingual-cased	0.640±0.014	0.743±0.020	0.601±0.014	0.665±0.009
	bert-base-multilingual-uncased	**0.651**±0.023	**0.781**±0.029	**0.628**±0.027	**0.696**±0.027
LT	bert-base-multilingual-cased	0.651±0.015	**0.776**±0.018	**0.572**±0.017	**0.659**±0.017
	bert-base-multilingual-uncased	**0.653**±0.025	0.774±0.025	0.569±0.025	0.656±0.024
LV	bert-base-multilingual-cased	0.651±0.018	0.783±0.027	0.612±0.021	0.687±0.020
	bert-base-multilingual-uncased	**0.679**±0.013	**0.783**±0.012	**0.643**±0.012	**0.706**±0.010
PT	bert-base-multilingual-cased	**0.649**±0.027	**0.762**±0.022	**0.600**±0.037	**0.670**±0.019
	bert-base-multilingual-uncased	0.632±0.015	0.725±0.022	0.597±0.026	0.655±0.024

Table 9. Monolingual experiments with the BERT-w + BERT method. For the notation see Table 8.

Language	BERT model	Accuracy	Precision	Recall	F-score
EN	bert-base-cased	0.749±0.019	0.827±0.021	0.739±0.023	0.780±0.020
	bert-base-uncased	0.765±0.033	**0.836**±0.016	0.751±0.027	0.791±0.020
	bert-large-cased	0.688±0.150	0.813±0.009	0.678±0.201	0.719±0.156
	bert-large-uncased	**0.782**±0.005	0.828±0.014	**0.776**±0.010	**0.801**±0.010
	bert-base-multilingual-cased	0.732±0.011	0.804±0.009	0.712±0.011	0.755±0.009
	bert-base-multilingual-uncased	0.768±0.027	0.829±0.022	0.768±0.027	0.798±0.022
DE	bert-base-multilingual-cased	**0.704**±0.034	**0.770**±0.035	**0.681**±0.036	**0.723**±0.035
	bert-base-multilingual-uncased	0.703±0.029	0.760±0.026	0.672±0.031	0.713±0.027
FR	bert-base-multilingual-cased	0.715±0.017	0.785±0.012	0.702±0.023	0.741±0.014
	bert-base-multilingual-uncased	**0.733**±0.035	**0.794**±0.022	**0.718**±0.031	**0.754**±0.024
LT	bert-base-multilingual-cased	0.692±0.022	0.741±0.023	0.669±0.029	0.703±0.021
	bert-base-multilingual-uncased	**0.721**±0.020	**0.767**±0.024	**0.688**±0.024	**0.725**±0.024
LV	bert-base-multilingual-cased	0.710±0.016	0.785±0.014	0.700±0.021	0.740±0.017
	bert-base-multilingual-uncased	**0.731**±0.019	**0.798**±0.023	**0.729**±0.028	**0.762**±0.025
PT	bert-base-multilingual-cased	**0.703**±0.031	**0.778**±0.028	**0.679**±0.031	**0.725**±0.027
	bert-base-multilingual-uncased	0.699±0.018	0.773±0.019	0.673±0.017	0.719±0.012

Table 10. Monolingual experiments with the BERT-s + FFNN method. For the notation see Table 8.

Lang	BERT model	Accuracy	Precision	Recall	F-score
EN	roberta-base-nli-stsb-mean-tokens	**0.842**±0.011	0.774±0.018	**0.806**±0.009	0.762±0.006
	roberta-large-nli-stsb-mean-tokens	0.808±0.014	0.871±0.014	0.795±0.017	0.831±0.015

Lang	BERT model	Accuracy	Precision	Recall	F-score
	bert-large-nli-stsb-mean-tokens	0.817±0.009	0.863±0.019	0.806±0.012	0.833±0.015
	distilbert-base-nli-stsb-mean-tokens	0.799±0.009	0.857±0.014	0.785±0.011	0.819±0.010
	distiluse-base-multilingual-cased-v2	0.760±0.020	0.843±0.019	0.728±0.024	0.781±0.021
	xlm-r-distilroberta-base-paraphrase-v1	0.806±0.011	**0.872**±0.011	0.793±0.015	**0.831**±0.011
	xlm-r-bert-base-nli-stsb-mean-tokens	0.806±0.006	0.857±0.013	0.789±0.011	0.821±0.008
	distilbert-multilingual-nli-stsb-quora-ranking	0.790±0.008	0.835±0.009	0.770±0.015	0.801±0.011
DE	distiluse-base-multilingual-cased-v2	0.735±0.010	0.833±0.011	0.703±0.014	0.762±0.006
	xlm-r-distilroberta-base-paraphrase-v1	**0.785**±0.007	0.836±0.012	**0.770**±0.008	0.802±0.009
	xlm-r-bert-base-nli-stsb-mean-tokens	0.774±0.005	**0.849**±0.016	0.769±0.010	**0.807**±0.011
	distilbert-multilingual-nli-stsb-quora-ranking	0.692±0.022	0.785±0.018	0.678±0.024	0.727±0.021
FR	distiluse-base-multilingual-cased-v2	0.731±0.012	0.824±0.011	0.696±0.018	0.754±0.015
	xlm-r-distilroberta-base-paraphrase-v1	0.782±0.011	0.840±0.014	0.760±0.016	0.798±0.013
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.800**±0.007	**0.843**±0.003	**0.791**±0.010	**0.816**±0.005
	distilbert-multilingual-nli-stsb-quora-ranking	0.754±0.016	0.784±0.017	0.724±0.020	0.753±0.017
LT	distiluse-base-multilingual-cased-v2	0.640±0.005	0.776±0.011	0.571±0.010	0.658±0.006
	xlm-r-distilroberta-base-paraphrase-v1	**0.764**±0.011	**0.843**±0.011	0.706±0.019	0.768±0.015
	xlm-r-bert-base-nli-stsb-mean-tokens	0.732±0.021	0.803±0.018	0.692±0.024	0.744±0.022
	distilbert-multilingual-nli-stsb-quora-ranking	0.751±0.010	0.825±0.021	**0.718**±0.008	0.768±0.013
LV	distiluse-base-multilingual-cased-v2	0.685±0.013	0.814±0.011	0.660±0.019	0.729±0.015
	xlm-r-distilroberta-base-paraphrase-v1	**0.789**±0.007	**0.869**±0.004	0.756±0.013	**0.809**±0.007
	xlm-r-bert-base-nli-stsb-mean-tokens	0.786±0.011	0.844±0.019	0.744±0.015	0.791±0.015
	distilbert-multilingual-nli-stsb-quora-ranking	0.756±0.007	0.818±0.011	**0.761**±0.009	0.789±0.010
PT	distiluse-base-multilingual-cased-v2	0.700±0.021	0.802±0.022	0.669±0.023	0.730±0.022
	xlm-r-distilroberta-base-paraphrase-v1	0.779±0.017	**0.885**±0.008	0.777±0.019	0.827±0.013
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.792**±0.004	0.856±0.011	**0.789**±0.007	**0.821**±0.007
	distilbert-multilingual-nli-stsb-quora-ranking	0.750±0.016	0.809±0.016	0.733±0.026	0.770±0.020

Table 11. Monolingual experiments with the BERT-s + COS model. For the notation see Table 8.

Lang	BERT model	Accuracy	Precision	Recall	F-score
	roberta-base-nli-stsb-mean-tokens	0.764	0.833	0.748	0.788
	roberta-large-nli-stsb-mean-tokens	0.757	0.837	0.752	0.793
	bert-large-nli-stsb-mean-tokens	**0.813**	0.862	**0.800**	**0.830**
	distilbert-base-nli-stsb-mean-tokens	0.757	0.841	0.744	0.790
	distiluse-base-multilingual-cased-v2	0.694	0.824	0.679	0.745
	xlm-r-distilroberta-base-paraphrase-v1	0.708	0.801	0.680	0.735
	xlm-r-bert-base-nli-stsb-mean-tokens	0.778	**0.864**	0.769	0.814
EN	distilbert-multilingual-nli-stsb-quora-ranking	0.715	0.822	0.732	0.775
	distiluse-base-multilingual-cased-v2	0.736	0.811	0.749	0.779
	xlm-r-distilroberta-base-paraphrase-v1	**0.771**	**0.857**	**0.768**	**0.810**
	xlm-r-bert-base-nli-stsb-mean-tokens	0.757	0.852	0.739	0.792
DE	distilbert-multilingual-nli-stsb-quora-ranking	0.674	0.797	0.693	0.741
	distiluse-base-multilingual-cased-v2	0.688	0.777	0.683	0.727
	xlm-r-distilroberta-base-paraphrase-v1	0.743	**0.864**	0.737	0.795
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.771**	0.863	**0.761**	**0.808**
FR	distilbert-multilingual-nli-stsb-quora-ranking	0.701	0.830	0.717	0.769
	distiluse-base-multilingual-cased-v2	0.667	0.791	0.648	0.713

Lang	BERT model	Accuracy	Precision	Recall	F-score
LT	xlm-r-distilroberta-base-paraphrase-v1	0.674	0.798	0.626	0.702
	xlm-r-bert-base-nli-stsb-mean-tokens	0.694	**0.808**	0.658	**0.725**
	distilbert-multilingual-nli-stsb-quora-ranking	**0.708**	0.753	**0.688**	0.719
LV	distiluse-base-multilingual-cased-v2	0.653	0.798	0.681	0.735
	xlm-r-distilroberta-base-paraphrase-v1	0.694	0.816	0.690	0.747
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.743**	**0.827**	**0.723**	**0.772**
	distilbert-multilingual-nli-stsb-quora-ranking	0.667	0.749	0.719	0.734
PT	distiluse-base-multilingual-cased-v2	0.639	0.749	0.651	0.697
	xlm-r-distilroberta-base-paraphrase-v1	**0.778**	**0.870**	**0.776**	**0.820**
	xlm-r-bert-base-nli-stsb-mean-tokens	0.771	0.868	0.742	0.800
	distilbert-multilingual-nli-stsb-quora-ranking	0.708	0.796	0.701	0.746

Table 12. Multilingual experiments (train EN) with the BERT-w + CNN model. For the notation see Table 8.

Language	BERT model	Accuracy	Precision	Recall	F-score
DE	bert-base-multilingual-cased	0.369±0.008	0.710±0.018	0.318±0.015	0.439±0.015
	bert-base-multilingual-uncased	**0.504**±0.010	**0.738**±0.034	**0.436**±0.019	**0.547**±0.020
FR	bert-base-multilingual-cased	0.435±0.031	0.711±0.029	0.359±0.030	0.476±0.022
	bert-base-multilingual-uncased	**0.496**±0.025	**0.748**±0.030	**0.445**±0.031	**0.557**±0.026
LT	bert-base-multilingual-cased	0.219±0.016	0.702±0.020	0.197±0.025	0.307±0.031
	bert-base-multilingual-uncased	**0.261**±0.022	**0.669**±0.049	**0.246**±0.033	**0.359**±0.039
LV	bert-base-multilingual-cased	0.222±0.026	0.686±0.048	0.191±0.023	0.298±0.027
	bert-base-multilingual-uncased	**0.336**±0.041	**0.687**±0.029	**0.271**±0.035	**0.387**±0.035
PT	bert-base-multilingual-cased	0.410±0.071	0.757±0.054	0.324±0.061	0.449±0.063
	bert-base-multilingual-uncased	**0.499**±0.033	**0.769**±0.016	**0.399**±0.053	**0.524**±0.047

Table 13. Multilingual experiments (train EN) with the BERT-w + BERT model. For the notation see Table 8.

Language	BERT model	Accuracy	Precision	Recall	F-score
DE	bert-base-multilingual-cased	0.525±0.034	0.694±0.039	0.512±0.023	0.589±0.024
	bert-base-multilingual-uncased	**0.588**±0.034	**0.748**±0.023	**0.573**±0.034	**0.648**±0.029
FR	bert-base-multilingual-cased	0.568±0.037	0.724±0.067	0.544±0.060	0.621±0.060
	bert-base-multilingual-uncased	**0.578**±0.035	**0.746**±0.072	**0.570**±0.054	**0.646**±0.060
LT	bert-base-multilingual-cased	0.215±0.026	**0.697**±0.050	0.259±0.014	0.377±0.009
	bert-base-multilingual-uncased	**0.319**±0.048	0.644±0.024	**0.342**±0.041	**0.446**±0.040
LV	bert-base-multilingual-cased	0.303±0.054	**0.697**±0.064	0.326±0.027	0.444±0.037
	bert-base-multilingual-uncased	**0.386**±0.035	0.694±0.029	**0.383**±0.032	**0.493**±0.029
PT	bert-base-multilingual-cased	0.536±0.035	0.676±0.020	0.498±0.044	0.572±0.034
	bert-base-multilingual-uncased	**0.576**±0.035	**0.724**±0.017	**0.551**±0.040	**0.625**±0.030

Table 14. Multilingual experiments (train EN) with the BERT-s + FFNN model. For the notation see Table 8.

Lan	BERT model	Accuracy	Precision	Recall	F-score
DE	distiluse-base-multilingual-cased-v2	0.760±0.012	0.832±0.012	0.716±0.013	0.769±0.010
	xlm-r-distilroberta-base-paraphrase-v1	**0.779**±0.021	**0.870**±0.014	**0.750**±0.025	**0.806**±0.020
	xlm-r-bert-base-nli-stsb-mean-tokens	0.749±0.010	0.838±0.013	0.730±0.007	0.780±0.009
	distilbert-multilingual-nli-stsb-quora-ranking	0.700±0.010	0.818±0.025	0.677±0.011	0.741±0.013

Lan	BERT model	Accuracy	Precision	Recall	F-score
FR	distiluse-base-multilingual-cased-v2	0.718±0.016	0.823±0.024	0.690±0.020	0.751±0.022
	xlm-r-distilroberta-base-paraphrase-v1	**0.794**±0.003	**0.878**±0.012	**0.774**±0.004	**0.823**±0.004
	xlm-r-bert-base-nli-stsb-mean-tokens	0.767±0.014	0.851±0.022	0.741±0.014	0.792±0.017
	distilbert-multilingual-nli-stsb-quora-ranking	0.707±0.009	0.821±0.018	0.676±0.018	0.741±0.010
LT	distiluse-base-multilingual-cased-v2	0.647±0.024	0.754±0.034	0.614±0.034	0.677±0.034
	xlm-r-distilroberta-base-paraphrase-v1	0.657±0.023	0.811±0.016	0.610±0.014	0.696±0.010
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.754**±0.009	**0.841**±0.013	**0.717**±0.012	**0.77**4±0.012
	distilbert-multilingual-nli-stsb-quora-ranking	0.625±0.018	0.759±0.025	0.585±0.012	0.660±0.016
LV	distiluse-base-multilingual-cased-v2	0.613±0.015	0.768±0.029	0.575±0.018	0.657±0.020
	xlm-r-distilroberta-base-paraphrase-v1	0.726±0.016	**0.852**±0.015	**0.695**±0.025	**0.765**±0.015
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.729**±0.018	0.825±0.014	0.682±0.025	0.747±0.020
	distilbert-multilingual-nli-stsb-quora-ranking	0.618±0.010	0.778±0.021	0.580±0.017	0.664±0.017
PT	distiluse-base-multilingual-cased-v2	**0.738**±0.019	0.853±0.014	0.714±0.026	0.777±0.020
	xlm-r-distilroberta-base-paraphrase-v1	0.771±0.011	0.864±0.013	**0.743**±0.016	0.799±0.013
	xlm-r-bert-base-nli-stsb-mean-tokens	0.771±0.011	**0.875**±0.006	0.742±0.009	**0.803**±0.007
	distilbert-multilingual-nli-stsb-quora-ranking	0.699±0.007	0.800±0.017	0.675±0.012	0.732±0.012

Table 15. Multilingual experiments (train EN) with the BERT-s + COS model. For the notation see Table 8.

Lang	BERT model	Accuracy	Precision	Recall	F-score
DE	distiluse-base-multilingual-cased-v2	0.715	0.809	0.687	0.743
	xlm-r-distilroberta-base-paraphrase-v1	0.764	0.841	0.736	0.785
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.771**	**0.859**	**0.767**	**0.811**
	distilbert-multilingual-nli-stsb-quora-ranking	0.667	0.724	0.657	0.689
FR	distiluse-base-multilingual-cased-v2	0.653	0.792	0.669	0.725
	xlm-r-distilroberta-base-paraphrase-v1	0.701	**0.851**	0.691	**0.763**
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.722**	0.826	**0.692**	0.753
	distilbert-multilingual-nli-stsb-quora-ranking	0.660	0.740	0.646	0.690
LT	distiluse-base-multilingual-cased-v2	0.618	0.749	0.617	0.677
	xlm-r-distilroberta-base-paraphrase-v1	0.736	**0.897**	0.702	0.788
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.757**	0.830	**0.761**	**0.794**
	distilbert-multilingual-nli-stsb-quora-ranking	0.611	0.721	0.614	0.663
LV	distiluse-base-multilingual-cased-v2	0.576	0.727	0.599	0.657
	xlm-r-distilroberta-base-paraphrase-v1	**0.792**	**0.882**	**0.756**	**0.814**
	xlm-r-bert-base-nli-stsb-mean-tokens	0.778	0.842	0.748	0.792
	distilbert-multilingual-nli-stsb-quora-ranking	0.597	0.760	0.623	0.685
PT	distiluse-base-multilingual-cased-v2	0.674	0.815	0.663	0.731
	xlm-r-distilroberta-base-paraphrase-v1	0.743	0.859	0.723	0.785
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.764**	**0.860**	**0.739**	**0.795**
	distilbert-multilingual-nli-stsb-quora-ranking	0.681	0.779	0.692	0.733

Table 16. Combined approach (train EN + target) with the BERT-s + FFNN model: results for the target languages (column *Lang*). For the other notation see Table 8.

Lang	BERT model	Accuracy	Precision	Recall	F-score
DE	distiluse-base-multilingual-cased-v2	0.781±0.009	0.843±0.01	0.762±0.018	0.800±0.014
	xlm-r-distilroberta-base-paraphrase-v1	**0.829±0.014**	**0.880±0.00**	**0.818±0.015**	**0.848±0.012**

Lang	BERT model	Accuracy	Precision	Recall	F-score
	xlm-r-bert-base-nli-stsb-mean-tokens	0.779±0.010	0.863±0.00	0.766±0.015	0.811±0.010
	distilbert-multilingual-nli-stsb-quora-ranking	0.733±0.007	0.811±0.01	0.716±0.010	0.761±0.006
FR	distiluse-base-multilingual-cased-v2	0.787±0.013	0.846±0.00	0.765±0.020	0.803±0.014
	xlm-r-distilroberta-base-paraphrase-v1	**0.817±0.009**	**0.870±0.00**	**0.807±0.012**	**0.837±0.011**
	xlm-r-bert-base-nli-stsb-mean-tokens	0.800±0.015	0.844±0.00	0.791±0.014	0.817±0.011
	distilbert-multilingual-nli-stsb-quora-ranking	0.785±0.009	0.820±0.01	0.781±0.010	0.800±0.009
LT	distiluse-base-multilingual-cased-v2	0.729±0.004	0.800±0.01	0.708±0.011	0.751±0.007
	xlm-r-distilroberta-base-paraphrase-v1	**0.814±0.018**	**0.875±0.01**	**0.779±0.026**	**0.824±0.019**
	xlm-r-bert-base-nli-stsb-mean-tokens	0.767±0.018	0.826±0.01	0.731±0.018	0.776±0.014
	distilbert-multilingual-nli-stsb-quora-ranking	0.765±0.008	0.823±0.01	0.752±0.011	0.786±0.008
LV	distiluse-base-multilingual-cased-v2	0.739±0.020	0.824±0.03	0.755±0.020	0.787±0.022
	xlm-r-distilroberta-base-paraphrase-v1	**0.831±0.007**	**0.892±0.01**	**0.804±0.007**	**0.846±0.006**
	xlm-r-bert-base-nli-stsb-mean-tokens	0.800±0.019	0.864±0.01	0.756±0.022	0.807±0.019
	distilbert-multilingual-nli-stsb-quora-ranking	0.740±0.011	0.827±0.01	0.738±0.013	0.780±0.012
PT	distiluse-base-multilingual-cased-v2	0.761±0.018	0.845±0.01	0.749±0.018	0.794±0.013
	xlm-r-distilroberta-base-paraphrase-v1	0.807±0.005	**0.874±0.00**	0.793±0.008	**0.831±0.007**
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.810±0.010**	0.867±0.00	**0.796±0.013**	0.830±0.010
	distilbert-multilingual-nli-stsb-quora-ranking	0.781±0.011	0.856±0.00	0.765±0.019	0.808±0.012

Table 17. Combined approach (train EN + target) with the BERT-s + COS model: results for the target languages (column *Lang*). For the other notation see Table 8.

Lang	BERT model	Accuracy	Precision	Recall	F-score
DE	distiluse-base-multilingual-cased-v2	0.722	0.802	0.715	0.756
	xlm-r-distilroberta-base-paraphrase-v1	**0.757**	0.832	0.738	0.782
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.757**	*0.861*	**0.739**	**0.796**
	distilbert-multilingual-nli-stsb-quora-ranking	0.681	0.802	0.699	0.747
FR	distiluse-base-multilingual-cased-v2	0.688	0.764	0.681	0.720
	xlm-r-distilroberta-base-paraphrase-v1	0.743	0.859	0.730	0.789
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.778**	**0.871**	**0.773**	**0.819**
	distilbert-multilingual-nli-stsb-quora-ranking	0.715	0.836	0.725	0.776
LT	distiluse-base-multilingual-cased-v2	0.674	0.811	0.681	**0.741**
	xlm-r-distilroberta-base-paraphrase-v1	0.674	0.798	0.626	0.702
	xlm-r-bert-base-nli-stsb-mean-tokens	0.694	**0.816**	0.658	0.728
	distilbert-multilingual-nli-stsb-quora-ranking	**0.708**	0.753	**0.688**	0.719
LV	distiluse-base-multilingual-cased-v2	0.653	0.757	0.681	0.717
	xlm-r-distilroberta-base-paraphrase-v1	0.694	0.816	0.690	0.747
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.750**	**0.831**	**0.735**	**0.780**
	distilbert-multilingual-nli-stsb-quora-ranking	0.667	0.749	0.719	0.734
PT	distiluse-base-multilingual-cased-v2	0.646	0.753	0.655	0.701
	xlm-r-distilroberta-base-paraphrase-v1	0.771	0.854	**0.767**	0.808
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.792**	**0.866**	**0.767**	**0.814**
	distilbert-multilingual-nli-stsb-quora-ranking	0.701	0.795	0.699	0.744

Table 18. Combined approach (train all) with the BERT-s + FFNN model: results for the target languages (column *Lang*). For the other notation see Table 8.

Lang	BERT model	Accuracy	Precision	Recall	F-score
	distiluse-base-multilingual-cased-v2	0.790±0.008	0.836±0.015	0.786±0.012	0.810±0.013

Lang	BERT model	Accuracy	Precision	Recall	F-score
EN	xlm-r-distilroberta-base-paraphrase-v1	**0.811**±0.007	**0.854**±0.007	**0.799**±0.011	**0.826**±0.007
	xlm-r-bert-base-nli-stsb-mean-tokens	0.811±0.023	0.851±0.024	0.796±0.024	0.823±0.022
	distilbert-multilingual-nli-stsb-quora-ranking	0.781±0.009	0.848±0.009	0.750±0.011	0.796±0.008
DE	distiluse-base-multilingual-cased-v2	0.785±0.004	0.835±0.006	0.772±0.010	0.802±0.007
	xlm-r-distilroberta-base-paraphrase-v1	**0.831**±0.009	**0.872**±0.013	**0.803**±0.010	**0.836**±0.011
	xlm-r-bert-base-nli-stsb-mean-tokens	0.761±0.010	0.825±0.004	0.746±0.015	0.784±0.008
	distilbert-multilingual-nli-stsb-quora-ranking	0.742±0.012	0.800±0.018	0.718±0.017	0.756±0.010
FR	distiluse-base-multilingual-cased-v2	0.771±0.007	0.844±0.009	0.762±0.010	0.801±0.008
	xlm-r-distilroberta-base-paraphrase-v1	**0.829**±0.013	**0.880**±0.013	0.820±0.019	0.849±0.016
	xlm-r-bert-base-nli-stsb-mean-tokens	0.818±0.007	0.876±0.004	0.803±0.012	0.838±0.005
	distilbert-multilingual-nli-stsb-quora-ranking	0.728±0.005	0.807±0.009	0.727±0.010	0.765±0.009
LT	distiluse-base-multilingual-cased-v2	0.733±0.013	0.818±0.008	0.744±0.014	0.780±0.011
	xlm-r-distilroberta-base-paraphrase-v1	**0.853**±0.005	**0.891**±0.005	**0.846**±0.008	**0.868**±0.005
	xlm-r-bert-base-nli-stsb-mean-tokens	0.793±0.016	0.841±0.016	0.757±0.024	0.797±0.020
	distilbert-multilingual-nli-stsb-quora-ranking	0.754±0.005	0.836±0.008	0.727±0.015	0.778±0.008
LV	distiluse-base-multilingual-cased-v2	0.729±0.004	0.825±0.008	0.739±0.004	0.780±0.004
	xlm-r-distilroberta-base-paraphrase-v1	**0.775**±0.013	**0.858**±0.009	**0.746**±0.010	**0.798**±0.010
	xlm-r-bert-base-nli-stsb-mean-tokens	0.775±0.009	0.835±0.021	0.746±0.014	0.788±0.015
	distilbert-multilingual-nli-stsb-quora-ranking	0.656±0.013	0.745±0.021	0.658±0.016	0.699±0.016
PT	distiluse-base-multilingual-cased-v2	0.758±0.008	0.839±0.009	0.756±0.013	0.795±0.011
	xlm-r-distilroberta-base-paraphrase-v1	**0.813**±0.009	**0.874**±0.005	**0.791**±0.015	**0.830**±0.006
	xlm-r-bert-base-nli-stsb-mean-tokens	0.792±0.004	0.851±0.019	0.776±0.014	0.812±0.010
	distilbert-multilingual-nli-stsb-quora-ranking	0.700±0.015	0.819±0.021	0.678±0.018	0.742±0.015

Table 19. Combined approach (train all) with the BERT-s + COS model: results for the target languages (column *Lang*). For the other notation see Table 8.

Lang	BERT model	Accuracy	Precision	Recall	F-score
EN	distiluse-base-multilingual-cased-v2	0.694	0.802	0.714	0.755
	xlm-r-distilroberta-base-paraphrase-v1	0.708	0.837	0.674	0.747
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.757**	**0.838**	0.726	**0.778**
	distilbert-multilingual-nli-stsb-quora-ranking	**0.757**	0.808	**0.749**	**0.778**
DE	distiluse-base-multilingual-cased-v2	0.694	0.813	0.692	0.748
	xlm-r-distilroberta-base-paraphrase-v1	0.750	0.817	**0.730**	0.771
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.757**	**0.825**	**0.730**	**0.774**
	distilbert-multilingual-nli-stsb-quora-ranking	0.701	0.770	0.680	0.722
FR	distiluse-base-multilingual-cased-v2	0.681	0.790	0.686	0.734
	xlm-r-distilroberta-base-paraphrase-v1	0.722	0.820	0.717	0.765
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.771**	**0.862**	**0.755**	**0.805**
	distilbert-multilingual-nli-stsb-quora-ranking	0.688	0.767	0.683	0.722
LT	distiluse-base-multilingual-cased-v2	0.604	0.740	0.607	0.667
	xlm-r-distilroberta-base-paraphrase-v1	**0.715**	**0.821**	0.672	**0.739**
	xlm-r-bert-base-nli-stsb-mean-tokens	0.708	0.785	**0.683**	0.731
	distilbert-multilingual-nli-stsb-quora-ranking	0.646	0.749	0.650	0.696
LV	distiluse-base-multilingual-cased-v2	0.604	0.759	0.636	0.692
	xlm-r-distilroberta-base-paraphrase-v1	0.688	0.807	0.648	0.718
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.722**	**0.855**	**0.707**	**0.774**
	distilbert-multilingual-nli-stsb-quora-ranking	0.646	0.780	0.671	0.721

Lang	BERT model	Accuracy	Precision	Recall	F-score
PT	distiluse-base-multilingual-cased-v2	0.688	0.808	0.682	0.740
	xlm-r-distilroberta-base-paraphrase-v1	0.750	**0.853**	0.745	0.796
	xlm-r-bert-base-nli-stsb-mean-tokens	**0.785**	0.852	**0.781**	**0.815**
	distilbert-multilingual-nli-stsb-quora-ranking	0.674	0.786	0.684	0.731