UNIVERSITY CEU - SAN PABLO

POLYTECHNIC SCHOOL

BIOMEDICAL ENGINEERING DEGREE

BACHELOR THESIS

A WHISPER-TIMESTAMPED BASED SYSTEM FOR QUERY-BY-EXAMPLE SPOKEN TERM DETECTION

Author: Sancha Barbara Kindler von Knobloch Luengo
Supervisor: Javier Tejedor Noguerales

July 2024

UNIVERSIDAD SAN PABLO-CEU
ESCUELA POLITÉCNICA SUPERIOR
División de Ingeniería

Datos del alumno

NOMBRE:

Datos del Trabajo

TÍTULO DEL PROYECTO:

Tribunal calificador

PRESIDENTE:	FDO.:
SECRETARIO:	FDO.:
VOCAL:	FDO.:

Reunido este tribunal el _______/_______/_______, acuerda otorgar al Trabajo Fin de Grado presentado por Don___________________________________la calificación de ___________________.

ACKNOWLEDGMENTS

Gracias a mi tutor Javier, por su ayuda y guía durante el desarrollo de este trabajo.

ABSTRACT

There is a wide range of informative materials stored in speech recordings that are not being accessed due to arduous retrieval processes. For this reason, there has been a growing interest in developing automatic systems to retrieve information from speech corpora. Current methods of finding recordings in which a certain word or short sentence is pronounced from an acoustic query, also known as Query by Example Spoken Term Detection (QbE STD), are particularly interesting for individuals with visual impairments, as a tool to access and manipulate digitally stored data.

The aim of this dissertation is to develop a QbE STD system based on a Whisper Automatic Speech Recognition (ASR) subsystem and a Spoken Term Detection subsystem to find putative occurrences within the speech corpora.

The system is tested on two different speech databases, MAVIR and SPARL22. The ASR subsystem obtained 91.20% accuracy in the transcription of the MAVIR development data and 83.39% accuracy in MAVIR test data. Furthermore, it obtained 93.47% accuracy of detection for the SPARL22 test database. In terms of the QbE STD, it obtained an Actual Term Weighted Value of 0.8695 for MAVIR development data, 0.6071 for MAVIR test data and 0.4547 for SPARL22 test data. Overall, this showcases the potential of the system to perform the extraction of valuable data stored in speech signals.

RESUMEN

Existe una amplia gama de documentos informativos almacenados en grabaciones de voz. Por tanto, hay un creciente interés en desarrollar sistemas automáticos para recuperar información de corpus de voz. Los métodos actuales para encontrar grabaciones en las que se pronuncia una palabra determinada o frase corta a partir de una consulta acústica, también conocidos como Query By Example Spoken Term Detection (QbE STD), son de especial interés para personas con discapacidad visual, como herramienta para acceder y manipular datos almacenados digitalmente.

El objetivo de este trabajo es desarrollar un sistema de QbE STD basado en la implementación de Whisper Timestamped como subsistema de reconocimiento automático de voz (RAV), seguido del desarrollo de un subsistema de detección de términos potencialmente presentes dentro de los corpus de voz.

El sistema fue probado en dos bases de datos de voz diferentes, MAVIR y SPARL22. La implementación del subsistema de reconocimiento automático de voz obtuvo un porcentaje de precisión en la transcripción de términos del 91.20% para los datos de MAVIR development, 83.39% para los datos de MAVIR test y 93.47% par los datos de SPARL22. Asimismo, el subsistema de QbE STD obtuvo un ATWV de 0.8695 para los datos de MAVIR development, 0.6071 para los datos de MAVIR test y de 0.4547 para los datos de SPARL22 test. En su totalidad, estos resultados señalan que este sistema de QbE STD puede ser una forma prometedora de realizar la extracción de datos almacenados en señales de voz.

INDEX

FIGURE INDEX

TABLE INDEX

INTRODUCTION

1.1 Importance of detecting information in speech signals

Speech is an intrinsic form of communication amongst humans. There has been an ever-increasing amount of heterogeneous speech data in the form of audio and audiovisual repositories. Encompassing everything from lectures to corporate meetings, today's digital age has enabled the rise of multimedia content and a consequent need for efficient and accessible search mechanisms.

Due to the large amounts of data available for processing, there has been a significant demand to perform this in an automated fashion. The first Automatic Speech Recognition (ASR) system dates to the 1950's with Bell Lab's Audrey and quickly evolved achieving multiple milestones in the field such as the development of IBM's shoebox shown in Figure 1, that demonstrated the feasibility of using computerized systems for speech recognition [1]. From this moment onwards, these has been a quasi-exponential transition into statistical models and deep learning algorithms to automatically detect information stored in speech signals. Modern ASR systems have become integral to a wide range of applications from virtual assistants to transcription services, reflecting decades of technological progress.

Figure 1: IBM's Shoebox in the 1960's [21]

Within the realm of speech processing, there are a variety of objectives in terms of the extraction of information. These range from the transcription into text of a given speech corpus, to the detection of specific terms within repositories. The latter is known as Spoken Term Detection. A variation of this method is Query-By-Example Spoken Term Detection (QbE STD), which involves searching for a term or phrase using an acoustic example of the term spoken. QbE STD can be implemented in devices without text-based capabilities and can also be applied to the development of language-independent STD systems [2]. Moreover, the development of Query-By-Example Spoken Term detection systems is particularly relevant in the context of individuals with visual impairments, as it eliminates the need for text-based input. Considering that there are well over a million individuals that are blind or suffering from limited vision and require assistive technology to interact with digital devices, building a QbE STD system is a must [3]. In doing so, they can engage with audio databases more effectively, using their voice to find relevant information and improving their overall experience interacting with digital systems.

1.2 Thesis objectives

The aim of the dissertation is to develop a QbE STD system. In the speech corpora that the system is tested with, queries are short text segments or words provided in the form of an acoustic input which are searched within the audio files, also known as utterances. During the query search process, if a query is identified within a utterance, it becomes a potential match or putative detection. The detection of speech signals within a larger speech signal is comprised of a two-step process, which in turn represents the subobjectives of the thesis. The first subobjective is to transcribe the queries and utterances from speech into text through the implementation of an ASR system. The second subobjective involves developing a subsystem to detect the text-based queries in the written utterances, including the confidence values assigned to the recognition of the queries within the utterances. These values are later analysed by a Detection Error Tradeoff system to evaluate the QbE STD system’s ability to identify potential matches as accurate instances of the desired term in utterances.

All in all, the objective of the thesis is the implementation of digital tools to develop a system that can be leveraged by individuals with visual impairments and implemented in tasks where current STD methods fall short due to the presence of text-based query inputs.

1.3 Thesis structure

The thesis is divided into State of the Art, Databases, Materials and Methods, Evaluation metrics, Results, Discussion and Conclusions sections. The second chapter (State of the Art) is a theoretical review, containing an explanation of the overall QbE STD methodology, a systematic review of the existing systems that are used to perform QbE STD and a specific description of Whisper and Whisper Timestamped as ASR methodologies. The third chapter (Databases) contains a review of the two Spanish speech repositories that are used to test the QbE STD system, including the length of the speech corpora and the quantity of queries and utterances present in each case. The fourth chapter (Materials and Methods) is dedicated to explaining the development of the system, and is divided into three sections: pre-processing, ASR implementation and QbE STD subsystem development and implementation. The fifth chapter (Evaluation metrics) includes the evaluation metrics, which are divided into two evaluation tools, at an ASR level and QbE STD evaluation analysis. The sixth chapter (Results) exposes the results of the previously described evaluation tools. In chapter seven (Discussion), the results are discussed. Finally, in chapter eight (Conclusions), the dissertation is concluded by summarizing the overall development and implementation of the system and proposing possible future work.

2 STATE OF THE ART

2.1 Spoken Term Detection

Spoken term detection methods are divided into two components, the ASR subsystem, and the STD subsystem. The first component makes use of an automatic speech recognition tool to convert a speech signal into text [4]. The second component is based on searching terms within the transcribed text, making putative detections that are deemed or ascertained as detections through confidence scoring by a decision maker. The overall process is displayed in Figure 2.

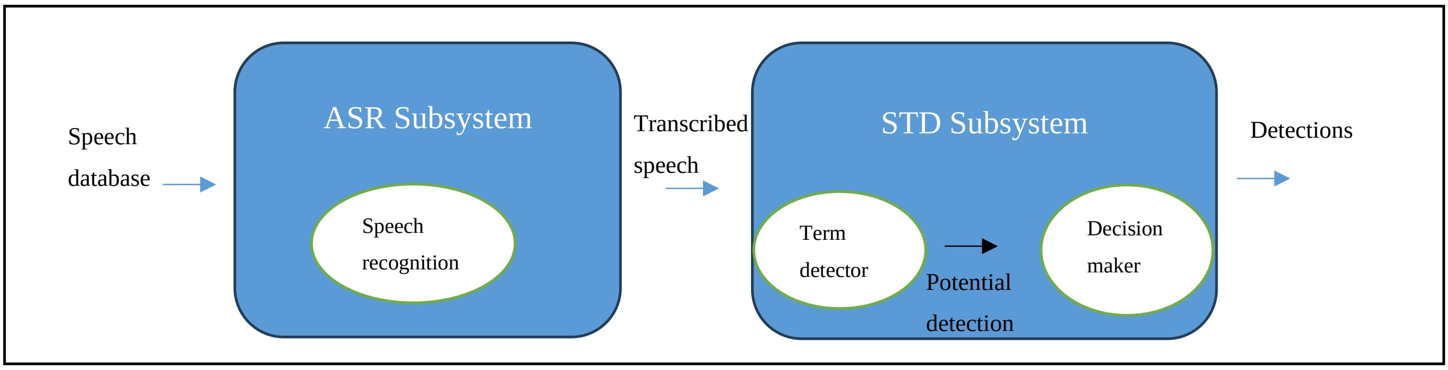


Figure 2: Illustration of Spoken Term Detection [2]

2.2 Overview of QbE STD Methods

Within STD, QbE STD refers to the specific case in which the user provides an example or a query in the form of a spoken term or phrase. This query is then used as a template to search for similar instances within the speech corpus. There are three methods for addressing QbE STD. The first method is based on textual transcription of the speech signal containing the query by applying an ASR system. Once transcription is obtained, the matching is performed with a STD technology. The second method is based on feature extraction and template matching of the query and utterance. The final method is the fusion of both previous methods [4].

With regards to the first method, phonetic transcription can be performed by a Large Vocabulary Speech Recognition System (LVCSR) and next Spoken Term Detection is performed through finite state transducers [5] or HMM-based systems [6]. Other methods include Lattice-based Search (PLS) and posterior matching based on dynamic programming [7]. At a word-level, other QbE STD systems implement Weighted Finite-State Transducers for matching [8].

The second method performs feature-based template matching. This methodology usually relies on Dynamic Time Warping (DTW) based template matching of mel frequency cepstral coefficient (MFCC), posteriors and bottleneck features extracted from Neural Networks (NN). Multiple variants of DTW are used, such as Segmental DTW [6] [7], Subspace-regularized [8] or Sub-sequence DTW [9]. Another approach considered substituting DTW for Convolutional Neural Networks with posterior feature vectors using pre-trained Feed Forward Neural Networks (FFN) [10].

The final approach is a hybrid implementation of the two previous methodologies. Query-by-Example Spoken Term Detection is expected to benefit from the fusion of heterogeneous systems. One approach is based on Majority Voting (MV) and score averaging [12]. Other methods include the combination of phone multigram representation and DTW [13] or Symbolic Search (SS) based approaches [14].

2.3 Whisper Automatic Speech Recognition System

Whisper is an ASR system developed by OpenAI that implements an encoder-decoder Transformer architecture, based on a methodology proposed by Varsani et al [17]. In their model, they rely on attention mechanisms to draw global dependencies between the input and output whilst reducing sequential computation. The attention mechanism allows models to focus on specific parts of the input function with varying levels of importance. Whisper makes use of both self-attention mechanisms, that focus on relationships within a single sequence and cross attention mechanisms, which focuses on relationships between two different sequences [17].

Unsupervised pre-training techniques to date used in speech recognition learn high quality representations of speech but require a supervised fine-tuning stage. This often limits their usefulness in ASR recognition techniques. Whisper employed large scale weak supervision and was trained on 680,000 hours of a diverse, multilingual speech dataset. In this way, it is a speech recognition system that works well in unseen scenarios, obtaining significantly accurate and robust results, approximating its performance to professional human transcribers [15].

The workflow that produces a transcription of a speech signal begins with an initial resampling of the audio to 16,000Hz, the application of an 80-channel log-magnitude Mel spectrogram computed on 25 millisecond windows with a stride of 10 milliseconds [18]. The spectrogram is normalized and passed through two convolutional layers to extract initial features. Both convolutional layers have a width of 3, with the first convolutional layer including a GELU Activation Function, and the second containing a stride of 2 which reduces temporal dimension. Afterwards, sinusoidal position embeddings are added to provide temporal context.

Once these steps are completed, the encoder transformer blocks are applied. The encoder section makes use of self-attention blocks that take generated Query (Q), Key (K) and V (Value) vectors derived from linear transformations of the input sequence and compute a compatibility function. The compatibility function, also known as attention function, can be described as mapping a query and a set of key-value pairs to an output. In fact, the computation is performed for multiple queries at once, forming a query matrix Q, key matrix K and value matrix V [17]. The output is computed by performing the dot product of queries of dimension d_k and values of dimension d_v, scaling the attention scores by the square root of the dimension of the keys and applying the softmax function as shown in Equation (1) to obtain the attention weights.

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

Instead of performing a single attention function for the input matrices of keys, values and queries, it is performed in a parallel manner in multiple, different linear projections, named as a Multi-Head Attention (MHA). In addition to attention sublayers, each block within the encoder has a feed-forward network (FFN) consisting of two linear transformations with a ReLU activation in between and a Multi-Layer Perceptron (MLP) placed after the MHA. This allows further processing information from the attention mechanism. In the encoder, the keys values and queries come from the output of the previous layer and is therefore known as self-attention layers. Ultimately, a normalization is applied to the encoder output before moving onto the decoder [18].

With respect to the decoder, it contains both multi-head self-attention and cross-attention layers whereby the query comes from the previous layer of the decoder, but the key and value are the values of the output of the encoder. This scheme enables every position in the decoder to attend over all positions in the input sequence [17].

The generated output is a sequence of tokens that represent transcription. The sequence is encoded using a byte-level BPE tokenizer [18]. A visual representation of sequence-to-sequence learning of Whisper is displayed in Figure 3.

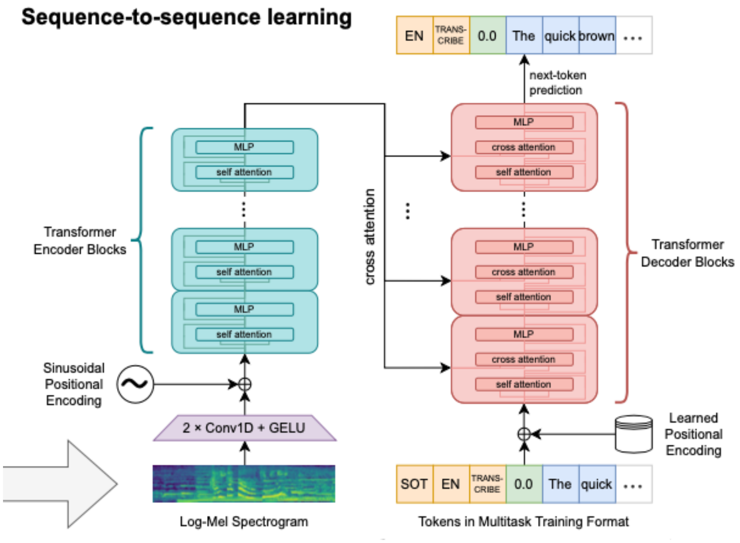
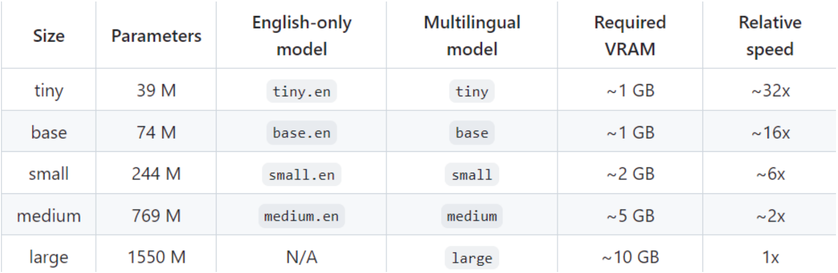


Figure 3: Whisper System Overview [18]

Whisper's performance varies depending on the choice of model and language. Depending on the objectives of the system's implementation, a smaller model can be an appropriate choice if less memory usage is preferable, or a larger model if there are no speed or VRAM constraints. The specific memory requirements and speed of the models are shown in Figure 4 [22].



Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~32x
base	74 M	`base.en`	`base`	~1 GB	~16x
small	244 M	`small.en`	`small`	~2 GB	~6x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x

Figure 4: Whisper models [22]

2.4 Whisper Timestamped Automatic Speech Recognition System

Whisper models can provide timestamps on speech segments, but do not predict word-level timestamps. A consequent implementation, Whisper Timestamped, makes use of Dynamic Time Warping applied to cross-attention weights to provide a more accurate start and end time estimation and confidence scores to the predicted word timestamps [20], creating a word alignment as displayed in Figure 5.

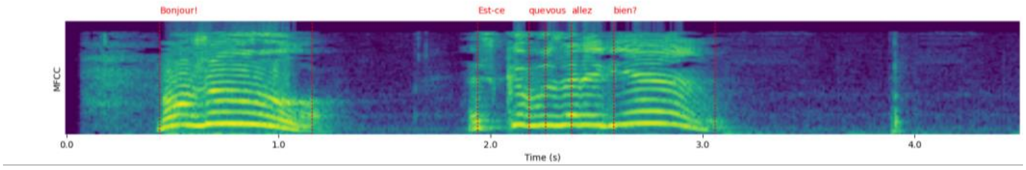


Figure 5: Plot of word alignment by Whisper Timestamped [20]

There is no previous implementation of Whisper nor Whisper Timestamped within QbE STD. This dissertation proposes the use of the latter ASR system, employing the transcription, start and end timestamps as well as the confidence scores for inputting into the STD system. Considering its state-of-the-art results, it can turn out to have a promising application within QbE STD systems.

3 DATABASES

3.1 Database overview

Two different speech corpora that include different acoustic environments and content have been employed for the evaluation of the QbE STD system. The first database, MAVIR, is comprised of Spanish conference talks and the audio files are divided into two sets: development and test [15]. Within each of these, there are audio files that represent utterances and queries. In the case of SPARL22, the dataset represents recordings from Spanish parliament sessions and uniquely contains test data, which are correspondingly divided into utterances and queries. The overall database schemea is shown in Figure 6.

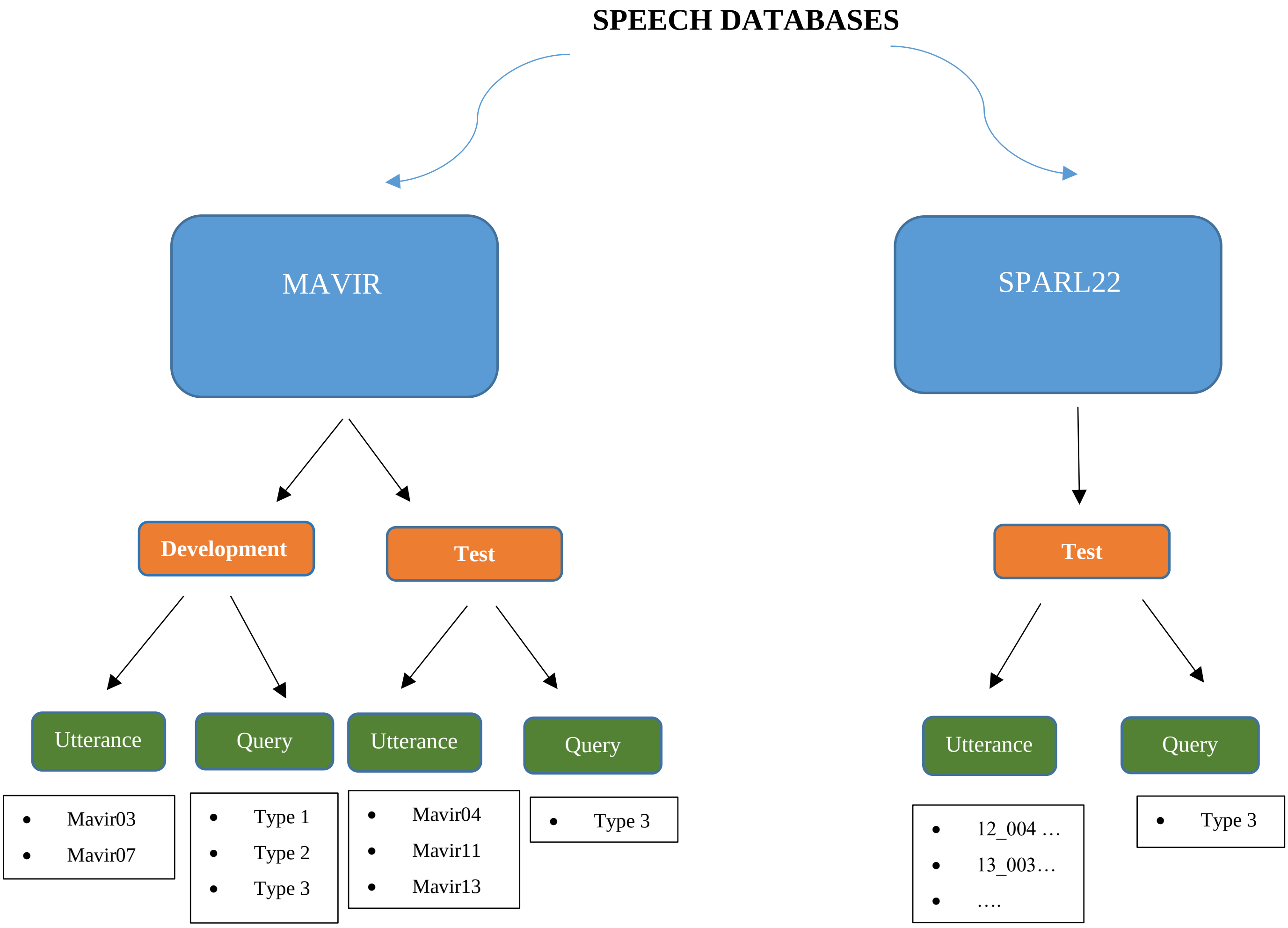


Figure 6: Databases used in the QbE STD System

The queries are divided into Type 1, Type 2 and Type 3. Each type is a folder containing a multitude of queries, where the differences between each type lie in the speakers that recorded them. In type 1, the queries are extracted from the same utterances where they will be searched, meaning the speaker is the same in both the queries and utterances. Type 2 query folders are recorded by speakers external to the database, as type 3 (another speaker adjacent to the utterance databases and type 2 queries).

3.2 MAVIR

The MAVIR database contains a collection of Spanish conference recordings from MAVIR workshops held in 2006, 2007 and 2008. This dataset features spontaneous recordings from various speakers from Spain and Latin America, with approximately 3 hours of speech material in total [16].

Originally, the speech data were captured in various audio formats, such as pulse-code modulation (PCM), a method used to digitally represent analogic signals, in one audio channel (mono) and two separate audio channels (stereo), MP3, at 22.05 kHz and 48kHz. These recordings were standardized to a uniform 16 kHz, single channel format to ensure audio quality and compatibility across different systems using a Sound eXchange (SoX) tool [16]. Except for one recording, all were made with a Digital TASCAM DAT model DA-p1, using different microphones including tabletop, floor standing and one lavalier microphone, ensuring equal recording conditions.

Furthermore, there was a variation in the distance from the speakers to the microphone, approximating the 50cm. Moreover, the conferences took place in spacious areas with significant capacity. This presents a realistic scenario and additional challenges including background noise and possible reverberations in the audio files.

Overall, the MAVIR development database within the dissertation contains two utterance files of around half an hour of duration each, one type 1, one type 2 and one type 3 query folders, with 102 queries each. Each query is three seconds long. The MAVIR test database contains three utterance files and one type 3 query folder.

3.3 SPARL22

The SPARL22 database contains speech recordings from the Spanish Parliament from 2016 sessions. Overall, the speech recordings make up a total of around two hours, all pertaining to the test dataset, as it will be used to evaluate the system performance in an unseen domain. There are 14 different utterances and 106 type 3 queries in the SPARL22 speech repository used to test the QbE STD proposed in this dissertation [15]. The utterances are around ten minutes long and the queries are three seconds long.

4 MATERIALS AND METHODS

4.1 *Overview of the QbE STD System*

The QbE STD system is subdivided into three steps, preprocessing, ASR subsystem and QbE STD subsystem. The overall schema is shown in Figure 7. Each of these steps are explained in the sections below in greater detail.

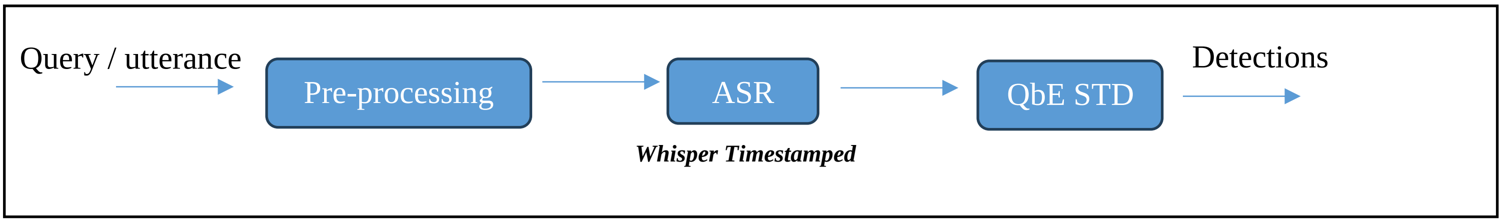


Figure 7: Architecture of the proposed Query by Example Spoken Term Detection system

4.2 *Data pre-processing*

4.2.1 Conversion of video to audio files

The utterances from repositories that are stored as .mpg type files require an appropriate conversion into a .wav type file for the ASR subsystem. This was carried out for MAVIR development utterances with the FFmpeg's open-source multi-media framework as follows:

```
ffmpeg -i input.mpg output.wav
```

4.3 *ASR Subsystem*

4.3.1 ASR Subsystem implementation

The next step involves the transcription of audio files into text by using the Whisper Timestamped tool presented previously. The ASR is run with all the combinations of parameters shown in Table 1 for the MAVIR development data, in order to obtain the model combination with the best recognition performance and use it for the recognition of MAVIR and SPARL22 test databases (for both transcription of queries and utterances).

Each model is tested with the accurate parameter, which is a shortcut to using the same default option as Whisper in terms of beam search and temperature increment on fallback parameters and without. Moreover, each of these combinations is coupled with the language that is set to Spanish to avoid recognition and transcription into other languages.

ASR System Input Combinations for MAVIR Development data
`Model --Tiny`
`Model --Tiny --accurate`
`Model --Small`
`Model --Small --accurate`
`Model --Medium`
`Model --Medium –-accurate`
`Model --Base`
`Model –-Base –-accurate`
`Model –-Large`
`Model –-Large –-accurate`

Table 1: ASR System input combinations for MAVIR development data

An example of the command line that runs Whisper Timestamped to perform the transcription is shown next:

```
Whisper_timestamped C:\Users\Sancha\development\mavir03.wav –-model --
small --output_dir C:\Users\Sancha\development\mavir03 --language
Spanish
```

For each utterance/query recognition process, various files were created. The one that will be of interest for addressing QbE STD is the .json file whose excerpt is shown in Figure 8.

```
{

  "text": " Muchas gracias Julio, buenos días. Adolfo Corujo, León, Dilbert, para qué le diría que improvisara? Podría haber sido más sencillo si le hubiera dado las líneas. Quiero agradecer a UNED que nos invita a participar en esta jornada Mavir, que además tiene un título, Tecnologías de la Lengua, en www.dospuntos-retos-mercados-potenciales. Para un chico de marketing, lo importante es eso de los mercados potenciales. Eso me ha atrapado la atención en las últimas semanas cuando estaba preparando esta charla. Ayer recordaba una cosa en cuanto al tema del mercado potencial que ahora hablando Isabel Aguilera me ha ayudado a cambiar.

(...)

"segments": [
  {
    "id": 0,
    "seek": 0,
    "start": 0.3,
    "end": 7.08,
    "text": " Muchas gracias Julio, buenos días. Adolfo Corujo, León, Dilbert, para qué le diría que improvisara?",
    "tokens": [
      50364,
      35669,
      16611,
      7174,
      1004,
      11,
(...)

],
    "temperature": 0.0,
    "avg_logprob": -0.20266801074035184,
    "compression_ratio": 1.479020979020979,
    "no_speech_prob": 0.09402094781398773,
    "confidence": 0.827,
    "words": [
      {
        "text": "Muchas",
        "start": 0.3,
        "end": 0.64,
        "confidence": 0.761
      },
```

Figure 8: Excerpt of the output JSON file from Whisper Timestamped

Whisper Timestamped creates a transcribed file divided into three main segments. Firstly, the whole transcribed text by the ASR system is shown, followed by the segments, where the tokenization is displayed. Finally, the words within the segment are displayed with their start and end timestamps and the confidence values, which are all necessary parameters for QbE STD.

4.3.2 ASR Subsystem evaluation

4.3.2.1 Formatting of ASR output file

To evaluate the accuracy of speech recognition of utterance and query files by Whisper Timestamped, a tool is implemented. It requires formatting the output json files and inputting three text files to calculate the accuracy metric. The first, is an MLF file format that is obtained through the conversion of the output json files of the ASR subsystem. For this, a python program that removes punctuation marks, transcribes numbers into text, and puts each transcribed word in a unique row of a single column is implemented. An MLF header title at the start and a full stop at the end of the file are added to create the MLF format. Making use of Python to do the formatting enables the use of existing modules that provided the required functionality whilst simplifying the code by importing json, re and num2words packages. The function requires the input json file (json_file_path) and outputs the formatted text file (text_file_path) as follows:

```
Write_words_to_text(json_file_path, text_file_path)
```

Overall, the MLF file follows the format shown in Figure 9.

```
#!MLF!#
"*/mavir03.rec"
muchas
gracias
julio
(…)
gracias
.
```

Figure 9: MLF formatted JSON file

This process is repeated for each file that is run through Whisper Timestamped: queries and utterances within development and test speech corpora. The queries, are joined in a single file for evaluating the ASR accuracy by implementing another python program, providing an output for all the queries as shown in Figure 10.

```
#!MLF!#
“*/DEV-0002.rec”
académico
.
“*/DEV-0003.rec”
noword
.
“*/DEV-0007.rec”
mis_asuntos
.
“*/DEV-0009.rec”
hilera
.
```

Figure 10: Recognition of queries in a single MLF file of MAVIR development queries

In scenarios where no word is recognized by the ASR system, the python program automatically writes “noword” in the file for that specific query as shown in Figure 15 for DEV-0003. This query combination process is also carried out for the utterances in MAVIR development and MAVIR and SPARL22 test data, to obtain a unique accuracy result for the utterances and queries within each speech corpus.

The second file required for ASR evaluation is the unique words present in the json file, which are obtained by implementing another python function that takes the previously formatted MLF file and outputs another text file with non-repeated words. The function is called as follows:

```
Uniquewords(‘C:\\Users\\Adriana\\Desktop\\MavirTest\\mavir13\\mavir
13.jsontotxt.txt’,
‘C:\\Users\\Adriana\\Desktop\\MavirTest\\mavir13\\mavir13.jsontotxt
nonrepeated.txt’)
```

The process of finding the unique words of the file is carried out for the output of the Whisper Timestamped tool for both utterance and query combinations. The output file in this case only contains non-repeated words and lacks any title or format.

The last file required for is the ground-truth to which the ASR recognition will be compared to in order to obtain the accuracy value.

4.3.2.2 Implementation of ASR evaluation tool

With the three file types mentioned in the previous section, the tool is run as shown right below. mapping_MLF.lab represents the ground truth, “combinacionquery1norepetidas.txt” represents the file with non-repeated words and “combinacionquery1.txt” is the MLF formatted file.

```
C:\Users\Adriana\Downloads\Hresults.exe -f -I
C:\Users\Adriana\Desktop\queries\mapping_MLF.lab
C:\Users\Adriana\Desktop\queries\query1\combinacionquery1no
repetidas.txt
C:\Users\Adriana\Desktop\queries\query1\combinacionquery1.t
xt
```

This command generates the output shown in Figure 11, which is comprised of the percentage accuracy and correct values, as well as number of insertions (I), substitutions (S), deletions (D) and labels. The meaning of these values is explained in detail in Section 5.2 ASR Evaluation.

```
-------------------------- File Results ---------------------
DEV-0002.rec: 100.00(100.00) [H= 1, D= 0, S= 0, I= 0, N= 1]
DEV-0003.rec: 0.00( 0.00) [H= 0, D= 0, S= 1, I= 0, N= 1]
DEV-0007.rec: 0.00( 0.00) [H= 0, D= 0, S= 1, I= 0, N= 1]
DEV-0009.rec: 0.00( 0.00) [H= 0, D= 0, S= 1, I= 0, N= 1]
DEV-0019.rec: 100.00(100.00) [H= 1, D= 0, S= 0, I= 0, N= 1
------------------------ Overall Results --------------------
SENT: %Correct=50.98 [H=52, S=50, N=102]
```

```
WORD: %Corr=50.98, Acc=50.98 [H=52, D=0, S=50, I=0, N=102]
==================================
```

Figure 11: Excerpt of ASR evaluation tool for MAVIR development Query 3

Once the MAVIR development data are processed, Whisper Timestamped is run for MAVIR and SPARL22 test data with the model that obtained the highest accuracy and the type of query that obtained the highest accuracy on MAVIR development query data.

4.4 QbE STD Subsystem

4.4.1 Overview of QbE STD Subsystem

The QbE STD subsystem development is divided into two main sections, transforming the JSON output file into a column-type formatted text file containing the word timestamps and confidence values, and the matching algorithm. As shown in Figure 12, once the utterance files are formatted, their content is compared with the recognised queries for matches.

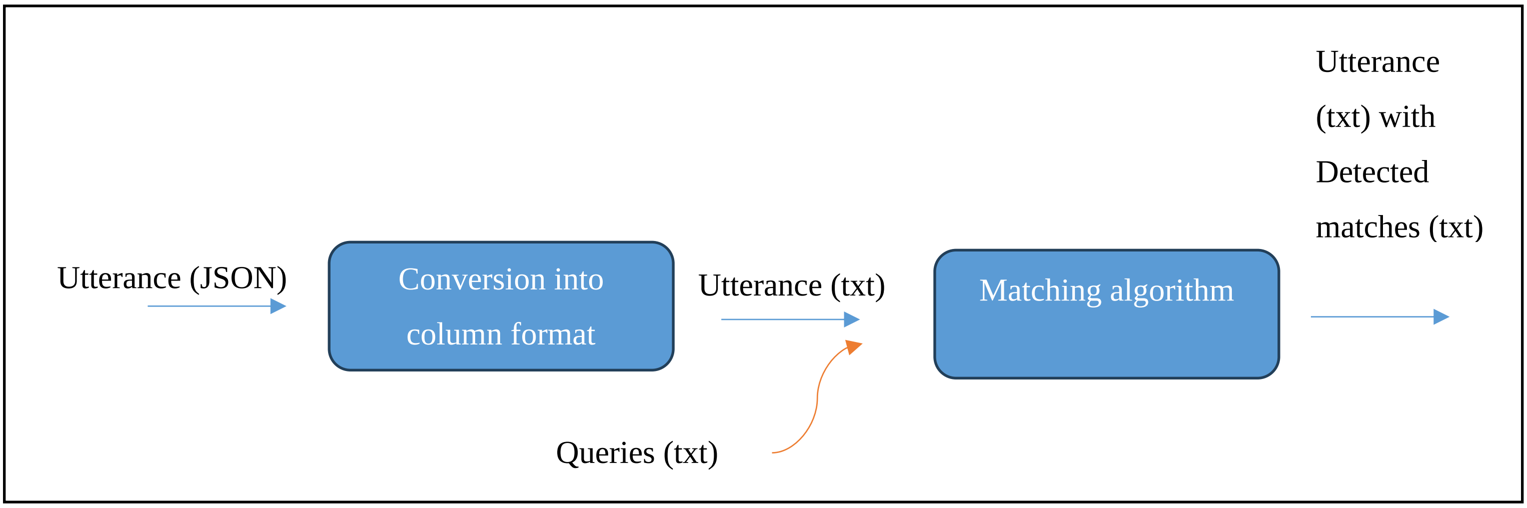

Utterance (JSON)
Conversion into column format
Utterance (txt)
Queries (txt)
Matching algorithm
Utterance (txt) with Detected matches (txt)

Figure 12: Architecture of the proposed Query by Example Spoken Term Detection subsystem.

4.4.2 QbE STD Subsystem implementation

4.4.2.1 Conversion of json file into formatted column file

The first step to develop the QbE STD system involves the transformation of the JSON output from Whisper Timestamped into a column-based format. Through the implementation of a python script, a text output file is generated and structured into five columns. The first column denotes the query identifier, the second and third columns indicate the start and end times respectively, the fourth column represents the accuracy score of the transcribed word from Whisper Timestamped, and the fifth column details the duration of each matched query. Notably, the duration parameter is computed by the script, by subtracting the start time from the end time values present in the original JSON output of Whisper Timestamped. The function is run as follows:

```
json_file_path =
'C:\\Users\\Adriana\\Desktop\\SPARL22\\13_000400_005_0_19422_642923\\1
3_000400_005_0_19422_642923.wav.words.json'
output_file_path =
'C:\\Users\\Adriana\\Desktop\\SPARL22\\13_000400_005_0_19422_642923\\1
3_000400_005_0_19422_642923matriz.txt'
matrix_formatting(json_file_path, output_file_path)
```

An example output file with the column format is shown in Figure 13.

```
gracias 3.18 3.46 0.463 0.28 13_000400_005_0_19422_642923
señora 3.64 3.76 0.961 0.12 13_000400_005_0_19422_642923
presidenta 3.76 4.58 0.916 0.82 13_000400_005_0_19422_642923
es 4.86 5.08 0.697 0.22 13_000400_005_0_19422_642923
mi 5.08 5.22 0.986 0.14 13_000400_005_0_19422_642923
primera 5.22 5.56 0.997 0.34 13_000400_005_0_19422_642923
intervención 5.56 6.22 0.996 0.66 13_000400_005_0_19422_642923
desde 6.22 6.46 0.974 0.24 13_000400_005_0_19422_642923
esta 6.46 6.76 0.992 0.3 13_000400_005_0_19422_642923
tribuna 6.76 7.14 0.964 0.38 13_000400_005_0_19422_642923
y 7.14 7.32 0.664 0.18 13_000400_005_0_19422_642923
no 7.32 7.48 0.987 0.16 13_000400_005_0_19422_642923
```

Figure 13: SPARL22 file in column format

This step is necessary for the utterances in MAVIR development and MAVIR and SPARL22 test data.

4.4.2.2 Matching algorithm

After generating the output file in the specified column format, an additional program is employed to replace the words in column file, with the query filenames of the queries recognized by ASR that match the words in the output file.

To do so, the program first extracts all the queries of the repository and appends them to a list of words for comparison. Subsequently, each row in the column file, are iteratively compared against every word in the query list. In the case that a match is identified, the entry in the first column (representing the matched word) is replaced with the identifier of the corresponding query. For instance, in Figure 14, a match is found in the fourth row, where a word is substituted with "TEST-0062," corresponding to the identifier of the matched query.

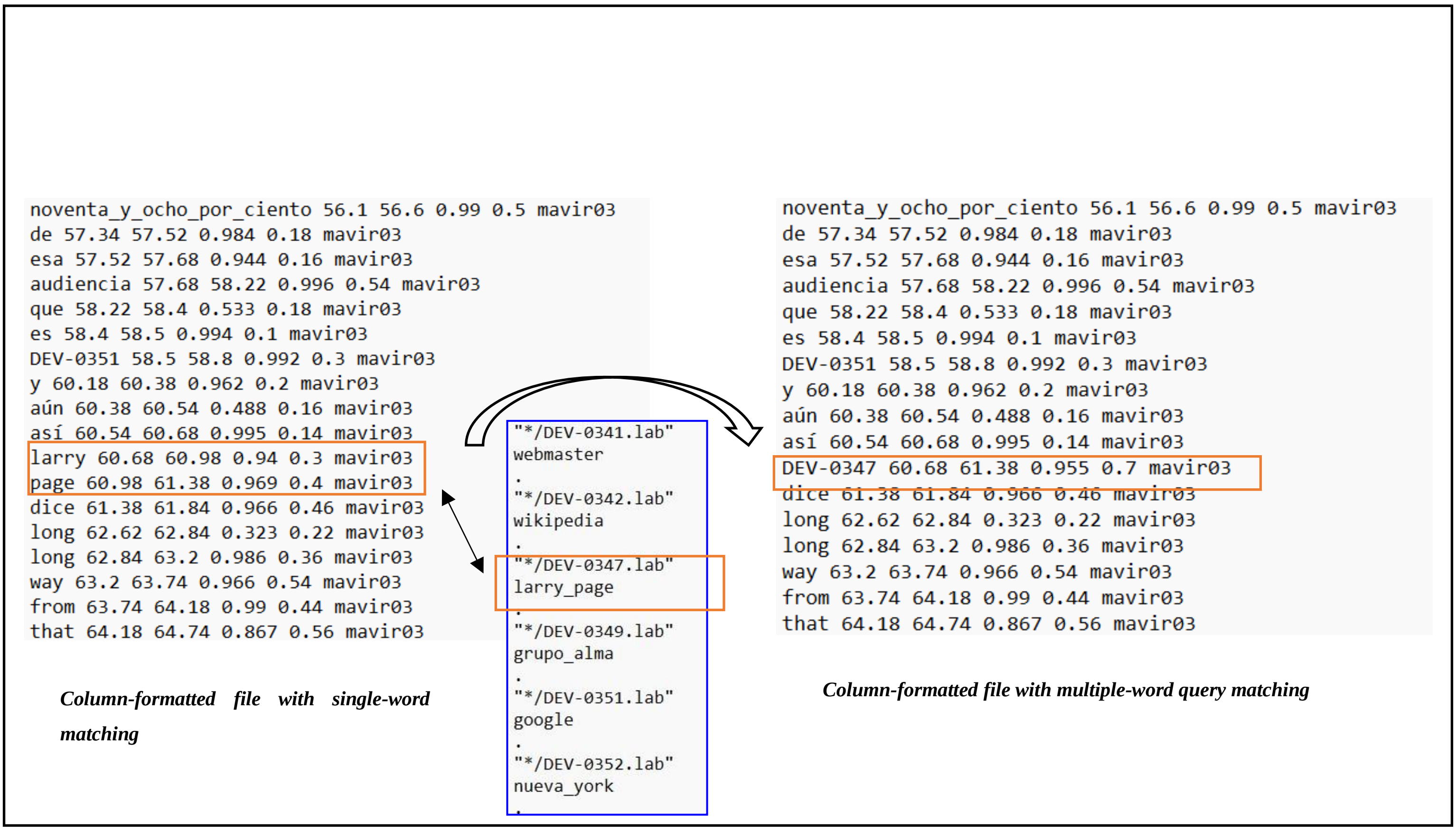
```
a 11.0 11.1 0.997 0.1 13_000400_005_0_19422_642923

millones 11.1 11.46 0.999 0.36 13_000400_005_0_19422_642923

de 11.46 11.66 0.997 0.2 13_000400_005_0_19422_642923

TEST-0062 11.66 12.16 0.996 0.5 13_000400_005_0_19422_642923

españoles 12.16 12.57 0.927 0.41 13_000400_005_0_19422_642923

en 12.57 12.76 0.997 0.19 13_000400_005_0_19422_642923

el 12.76 12.92 0.997 0.16 13_000400_005_0_19422_642923
```

Figure 14: SPARL22 file with single word swapping

An additional function is implemented to consider search of queries that consist of multiple words, which have separate start, end, and accuracy values in the Whisper Timestamped JSON file. This function extends the methodology of the previous implementation by incorporating specific logic to handle these scenarios.

The function sequentially compares each word of the recognized query with subsequent words in the json file. If there is a match between a word in the column-formatted json file and the multiple-word query, it continues to compare the next word of the query with the next row of the file iteratively, until all words match, or a discrepancy arises. In the case that all the words of the recognized query are aligned, the function takes the start time of the first word as the detection start time and the end time of the last word as the detection end time. The confidence score is computed as the average of all the words' confidence values and the duration is obtained by subtracting the start to the end time value.

Figure 15 showcases how *Larry page*, which is a query recognized by the ASR from the query 3 package with identifier DEV-0347 is detected in MAVIR development utterances.



```
noventa_y_ocho_por_ciento 56.1 56.6 0.99 0.5 mavir03
de 57.34 57.52 0.984 0.18 mavir03
esa 57.52 57.68 0.944 0.16 mavir03
audiencia 57.68 58.22 0.996 0.54 mavir03
que 58.22 58.4 0.533 0.18 mavir03
es 58.4 58.5 0.994 0.1 mavir03
DEV-0351 58.5 58.8 0.992 0.3 mavir03
y 60.18 60.38 0.962 0.2 mavir03
aún 60.38 60.54 0.488 0.16 mavir03
así 60.54 60.68 0.995 0.14 mavir03
larry 60.68 60.98 0.94 0.3 mavir03
page 60.98 61.38 0.969 0.4 mavir03
dice 61.38 61.84 0.966 0.46 mavir03
long 62.62 62.84 0.323 0.22 mavir03
long 62.84 63.2 0.986 0.36 mavir03
way 63.2 63.74 0.966 0.54 mavir03
from 63.74 64.18 0.99 0.44 mavir03
that 64.18 64.74 0.867 0.56 mavir03
```

Column-formatted file with single-word matching

```
"*/DEV-0341.lab"
webmaster
.
"*/DEV-0342.lab"
wikipedia
.
"*/DEV-0347.lab"
larry_page
.
"*/DEV-0349.lab"
grupo_alma
.
"*/DEV-0351.lab"
google
.
"*/DEV-0352.lab"
nueva_york
.
```

```
noventa_y_ocho_por_ciento 56.1 56.6 0.99 0.5 mavir03
de 57.34 57.52 0.984 0.18 mavir03
esa 57.52 57.68 0.944 0.16 mavir03
audiencia 57.68 58.22 0.996 0.54 mavir03
que 58.22 58.4 0.533 0.18 mavir03
es 58.4 58.5 0.994 0.1 mavir03
DEV-0351 58.5 58.8 0.992 0.3 mavir03
y 60.18 60.38 0.962 0.2 mavir03
aún 60.38 60.54 0.488 0.16 mavir03
así 60.54 60.68 0.995 0.14 mavir03
DEV-0347 60.68 61.38 0.955 0.7 mavir03
dice 61.38 61.84 0.966 0.46 mavir03
long 62.62 62.84 0.323 0.22 mavir03
long 62.84 63.2 0.986 0.36 mavir03
way 63.2 63.74 0.966 0.54 mavir03
from 63.74 64.18 0.99 0.44 mavir03
that 64.18 64.74 0.867 0.56 mavir03
```

Column-formatted file with multiple-word query matching

Figure 15: Multiple word swapping example from MAVIR Development Query 3

4.4.3 QbE STD Subsystem evaluation

4.4.3.1 Formatting matching file into XML format

The second evaluation tool, which evaluates the system at a QbE STD level, requires an input which contains the information from the previously created column format but in an XML file type.

To do this, the first step is to concatenate the column utterance files of each database (MAVIR development, MAVIR test and SPARL22 test independently). Secondly, another program is created to output a file with a list of all the queries that were detected from the combined utterance, calling the function as follows:

```
input_file =
'C:\\Users\\Adriana\\Desktop\\TFG\\resultados\\joinmatrizswapmultiple.
txt'
output_file =
'C:\\Users\\Adriana\\Desktop\\TFG\\resultados\\onlydevmatrizswapmultip
le.txt'
onlydev_lines(input_file, output_file)
```

Where the “joinmatrizswapmultiple.txt” file above is the input column file for MAVIR development utterances, and the output file is another columnar text file only containing recognized words as shown in Figure 16.

```
DEV-0253 1.7 2.2 0.995 0.5 mavir07
DEV-0356 11.13 12.22 0.821 1.09 mavir07
DEV-0217 18.26 18.6 0.982 0.34 mavir07
DEV-0168 29.48 29.9 0.986 0.42 mavir07
DEV-0168 30.96 31.34 0.992 0.38 mavir07
DEV-0351 32.98 33.32 0.866 0.34 mavir07
DEV-0168 44.7 45.08 0.99 0.38 mavir07
DEV-0116 48.52 48.92 0.953 0.4 mavir07
DEV-0298 54.02 54.38 0.999 0.36 mavir07
DEV-0116 54.8 55.18 0.988 0.38 mavir07
DEV-0203 55.54 55.88 0.999 0.34 mavir07
DEV-0116 59.12 59.52 0.997 0.4 mavir07 (…)
```

Figure 16: File with only matched terms for MAVIR development

From this newly created file, the non-repeated query terms are extracted by calling the following function:

```
unique_words =
find_unique_words('C:\\Users\\Adriana\\Desktop\\TFG\\resultados\\onlyd
evmatrizswapmultiple.txt')
```

With both files, the non-repeated queries and the column file with every instance of detected queries, a loop iterates through the list of unique detections and within each unique detection, another loop iterates through each of the rows of the column file. In this way, each time a unique detection is encountered in the outer loop, it is added to the XML output file, in a unique format shown in Figure 17 (equalling a detected termlist_id). Afterwards, the column file is iterated and when there is a match between the outer query and a row within the column file, the row is added to the output file. An Example of this is shown in Figure 17, where the unique term DEV-0253, has been found in the column file in three different rows corresponding to instances in mavir07 (at tbeg 1.7) and mavir03 files (at tbeg 419.28 and 1111.44).

To produce an output file where the detections go in increasing temporary order of appearance, as shown in Figure 17, an ultimate function is implemented, which calls upon a built-in python function to arrange the beginning time in ascending order (sorted). In a similar manner to the detections of unique queries within the column files, the tbeg values are extracted for every “detected_termlist” matches, iterated and ordered by matching tbeg values within the XML file, such that smaller tbegs are matched and written first into the ordered output XML file and go in increasing size. The overall format of the ordered XML file is shown in Figure 17.

```
<stdlist
termlist_filename="C:\Users\Adriana\Desktop\TFG\resultados\onlydevmatrizswapmultiple.xml
" indexing_time="1.000" language="spanish" index_size="1" system_id="fake">
<detected_termlist termid="DEV-0253" term_search_time="24.3" oov_term_count="1">
<term file="mavir07" channel="1" tbeg="1.7" dur="0.5" score="0.995" decision="YES"/>
<term file="mavir03" channel="1" tbeg="419.28" dur="0.44" score="0.994" decision="YES"/>
<term file="mavir03" channel="1" tbeg="1111.44" dur="0.38" score="0.992" decision="YES"/>
</detected_termlist>
<detected_termlist termid="DEV-0356" term_search_time="24.3" oov_term_count="1">
<term file="mavir07" channel="1" tbeg="11.13" dur="1.09" score="0.821" decision="YES"/>
<term file="mavir07" channel="1" tbeg="201.64" dur="0.86" score="0.881" decision="YES"/>
</detected_termlist>
```

```
<detected_termlist termid="DEV-0217" term_search_time="24.3" oov_term_count="1">
<term file="mavir07" channel="1" tbeg="18.26" dur="0.34" score="0.982" decision="YES"/>
<term file="mavir07" channel="1" tbeg="152.52" dur="0.48" score="0.976" decision="YES"/>
<term file="mavir03" channel="1" tbeg="179.22" dur="0.26" score="0.996" decision="YES"/>
<term file="mavir03" channel="1" tbeg="187.58" dur="0.5" score="0.982" decision="YES"/>
<term file="mavir03" channel="1" tbeg="913.66" dur="0.44" score="0.986" decision="YES"/>
<term file="mavir03" channel="1" tbeg="932.28" dur="0.32" score="0.994" decision="YES"/>
<term file="mavir03" channel="1" tbeg="1020.16" dur="0.76" score="0.999" decision="YES"/>
<term file="mavir03" channel="1" tbeg="1038.2" dur="0.36" score="0.994" decision="YES"/>
<term file="mavir03" channel="1" tbeg="2121.38" dur="0.36" score="0.985" decision="YES"/>
</detected_termlist>
(…)
<detected_termlist termid="DEV-0132" term_search_time="24.3" oov_term_count="1">
<term file="mavir03" channel="1" tbeg="2112.14" dur="0.14" score="0.633" decision="YES"/>
</detected_termlist>
</stdlist>
```

Figure 17: Final XML formatted file for MAVIR Development

As we can see from above, the query with title DEV-0253, which corresponds to the word “presentación” has appeared three times, twice in mavir03 and once in mavir07. In the case of DEV-0217, which represents the word “negocio” appears a total of nine times in MAVIR development files. A stdlist title and ending “</stdlist>” is also included to create the XML format required for inputting into the QbE STD evaluation program to find the final statistical values.

4.4.3.2 QbE STD Subsystem evaluation implementation

With the ordered XML file, a program is run to compute the QbE STD evaluation metric by firstly downloading MINGW64, an open-source development environment for Git. The environment variables that are required to run the program are:

- P: Path to the file directory
- I: Path to the directory containing the “STDEval.pl” (NIST) script
- O: Path to the directory for output results
- F: Path to the XML file with the detections

These paths are stored within a .sh script file that are called from Git with the following command line:

```
bash ./score_qbestd_2022_MAVIR_development.sh
```

The output of the program is seen in Figure 19. The ATWV are computed for MAVIR development and test data as well as SPARL22 data.

```
+--------------------------------------------------------------------------------------------+
| Indexing Time:               1.0000                                                        |
| Language:                   spanish                                                        |
| Index size (bytes):               1                                                        |
| System ID:                     fake                                                        |
| Coefficient C:               0.1000                                                        |
| Coefficient V:               1.0000                                                        |
| Trials Per Second:           1.0000                                                        |
| Probability of a Term:       0.0001                                                        |
|   Decision   Score                         OK   (Max   NO:   -9999.0000,   Min   YES:   0.3800)
|
+----------------------------------------------+---------------------------------------------+
|                                       Search |                    ALL                      |
|                                              |                          Occ.               |
|    TermID                      Text     Time |   Ref  Corr   FA  Miss  Value P(FA)  P(Mis)|
+----------------------------------------------+---------------------------------------------+
| DEV-0002                  DEV-0002      0.00 |     0     0     0     0    N/A    N/A   N/A |
| DEV-0003                  DEV-0003     24.30 |     2     2     0     0 1.000 0.00000 0.000 |
| DEV-0007                  DEV-0007     24.30 |     9     9     0     0 1.000 0.00000 0.000 |
| DEV-0009                  DEV-0009     24.30 |     3     3     0     0 1.000 0.00000 0.000 |
(…)
| DEV-0363                  DEV-0363     24.30 |     2     1     1     1 0.450 0.00028 0.500 |
+----------------------------------------------+---------------------------------------------+
| Totals, Actual Occ. Weighted Value   1530.90 |   425   392     6    33 0.921 0.00189 0.078 |
| Means (N/A excl.)                      15.01 |     6     5     0     0 0.893 0.00003 0.105 |
+----------------------------------------------+---------------------------------------------+
|                                              |  Number of Trials                3592       |
|                                              |  Total Speech Time (sec.)        3591.9     |
|                                              |  A. T-Weighted Value (N/A excl.)     0.870  |
+----------------------------------------------+---------------------------------------------+
```

```
+---------------------------------------------------------------------+
|                DET Curve Analysis Summary                  |
+-------------+----------------------+------------------------------+
|          |     Weighted    |                Decision   |
| Description | Max Value | A. Value |   P(Fa) P(Miss)      Score   |
+-------------+-----------+----------+------------------------------+
+-------------+-----------+----------+------------------------------+
|        ALL |   0.8695   |  0.8695   | 0.00003   0.105   3.800000e-01 |
+-------------+-----------+----------+------------------------------+
```

Figure 18: Excerpt of the output result for the QbE STD system for MAVIR development data

4.5 Complete Code of QbE STD system

The complete code to run the Materials and Methods outlined above is available in the public Github repository in the following link:

https://github.com/SanchaKvK/A-Whisper-Timestamped-Based-System-for-Query-By-Example-Spoken-Term-Detection

5 EVALUATION METRICS

5.1 Overview of Evaluation Metrics

The evaluation of the QbE STD detection system is done at two stages, at the ASR subsystem level and at the QbE STD subsystem, as showcased in figure 19.

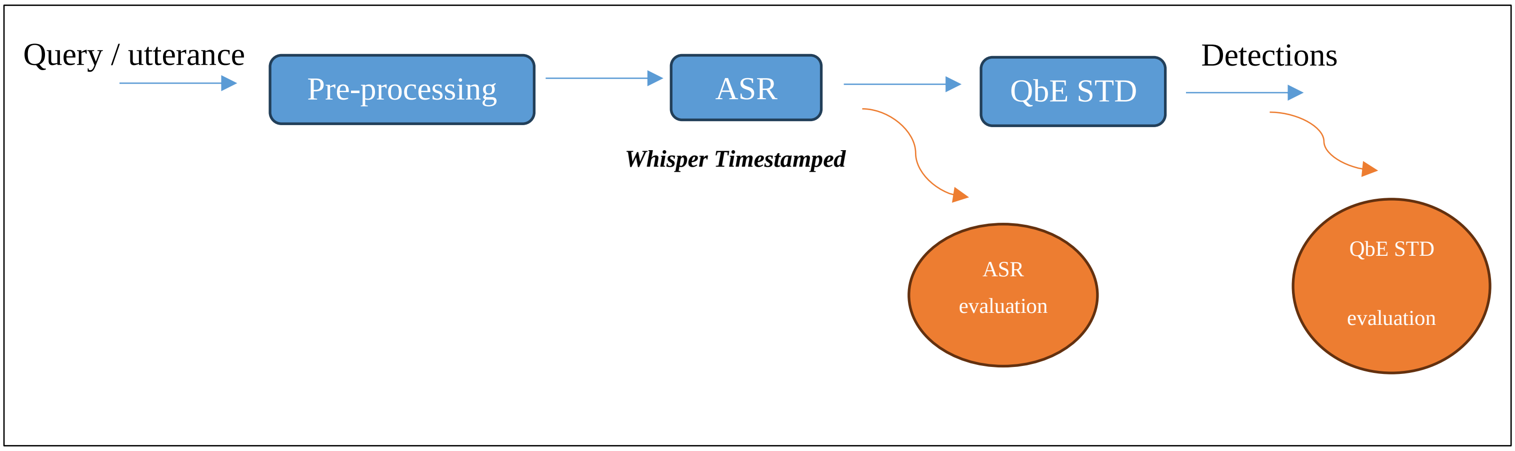


Figure 19: Architecture of proposed Query by Example Spoken Term Detection and integration with System Evaluation methods

5.2 ASR Evaluation

5.2.1 ASR SubSystem evaluation with HResults

HResults is a tool within the Hidden Markov Model Toolkit (HTK) which is used to evaluate the performance of ASR systems. This compares the output of an ASR system, with the original reference transcription of the repository files and outputs a percentage accuracy value. This metric is computed from the number of insertion (I), deletion (D) and substitution (S) errors carried out by the ASR subsystem considering the total number of words in the reference transcription (N), according to equation 2.

$$Percent\ Accuracy = \frac{N-D-S-I}{N} \times 100\% \tag{2}$$

Once the MAVIR development data are processed, Whisper Timestamped is run for MAVIR and SPARL22 test data with the model that obtains the highest accuracy and the type of query that obtains the highest accuracy on MAVIR development query data.

5.3 QbE STD Evaluation

5.3.1 Overview of QbE STD Evaluation Metric

The second evaluation metric aims to evaluate the final QbE STD performance by using a *Perl* tool provided by NIST [18] . This involves detecting the presence or absence of a specific query in utterances. The ATWV is a metric proposed by the National Institute of Standards and Technology (NIST). It helps understand how well the system performs in terms of detecting queries accurately (hit) while balancing false positives (false alarm) and false negatives (miss) in a single main metric [16]. For each unique query, the computation of ATWV considers hits and false alarms and averages them over all queries as shown in Equation (3).

$$ATWV = \frac{1}{|\Delta|}\sum_{T \in \Delta}\left(\frac{N_{hit}^{T}}{N_{true}^{T}} - \beta\,\frac{N_{FA}^{T}}{L-N_{true}^{T}}\right) \tag{3}$$

where Δ denotes the set of queries and $|\Delta|$ is the number of queries. N_{true}^{T} represents the number of actual present instances of query T in the speech signal. N_{hit}^{T} and N_{FA}^{T} denote the numbers of hits and false alarms of the query T respectively. L is the length of the audio recording in seconds. β is a weight factor set to 999.9, which places a greater emphasis on maximizing the recall value by prioritizing capturing as many true instances of the query even at the cost of increased false alarms. This is useful in scenarios where missing a query is more detrimental than having extra false detections, with a 10:1 ratio in terms of recall with respect to precision.

A correct detection is found when it appears within a $\pm$ 15-s interval from the ground-truth timestamp. The ATWV is computed with the actual decision threshold of the system which is adjusted based on development data. If this results in the establishment of a sub-optimal threshold, it can reduce the overall performance of the system for which another metric has been calculated to evaluate upper-bound performance independent of the system's set decision threshold. It is known as the

Maximum Term Weighted Value (MTWV), and it is computed using the optimal threshold and confidence scores provided by the system.

There are two additional metrics within the system evaluation toolkit which are the probabilities of miss and the probability of false alarm of the system, as shown in Equations 4 and 5.

$$p(Miss) = 1 - \frac{N_{hit}}{N_{true}} \tag{4}$$

$$p(FA) = \frac{N_{FA}}{L - N_{true}} \tag{5}$$

N_{true} is the actual instances of the query in the utterance.

6 RESULTS

6.1 ASR Results

6.1.1 MAVIR Development

HResults is implemented first to calculate the accuracy of ASR for MAVIR development data. The accuracy obtained for every combination of models with and without the accurate parameters are displayed in Table 2.

Model Type	**With accurate (%)**	**Without Accurate (%)**
Tiny	76.94	35.72
Tiny accurate	76.94	80.46
Small	89.92	89.78
Base	84.94	76.89
Medium	90.80	**91.20**
Large	90.20	56.02

Table 2: ASR for MAVIR Development Utterances

There is an overall positive increase in the accuracy values moving from tiny to large parameters. The model that provided the best accuracy was the medium without the accurate option and it was used for transcribing MAVIR and SPARL22 test, whose results are presented next. This trend is in line with the speed and accuracy tradeoffs of the

Whisper Timestamped models, where smaller models require less memory and are faster and the opposite happens with larger models [19].

The values for the ASR subsystem for the three query types (Query 1, Query2 and Query 3) for MAVIR development are shown in Table 3. As the Type 3 query folder obtained the highest accuracy, this query type was used in MAVIR test and SPARL22 test.

Query 1	Query 2	Query 3
50.98	88.24	93.14

Table 3: Table 4: ASR for MAVIR Development Queries

6.1.2 MAVIR Test

The ASR subsystem for MAVIR test obtained a combined accuracy value of 83.39%. In terms of queries, it obtained an accuracy of 86.79%.

6.1.3 SPARL22 Test

The ASR subsystem achieved an overall accuracy of 93.47% across the SPARL22 test dataset, encompassing the fourteen utterances present. Furthermore, it attained 71.30% accuracy for the queries.

6.2 QbE STD System Evaluation Results

Table 3 shows the values obtained for the Actual Term Weighted Value, the Maximum Term Weighted Value, the P (Miss) and P (Fa) for MAVIR development and test, and SPARL22 test databases. As shown in Table 4, MAVIR development obtained the highest ATWV and MTWV values. On the other hand, SPARL22 obtains the lowest figures. These results will be discussed next.

Data	ATWV	MTWV	P(Fa)	P(Miss)
MAVIR Development	0.8695	0.8695	0.00003	0.105
MAVIR Test	0.6071	0.6174	0.00008	0.307
SPARL22 Test	0.4547	0.4573	0.00006	0.485

Table 5: QbE STD results for MAVIR (development and test) and SPARL22 databases

7 DISCUSSION

In terms of ASR results, the medium model has been the one that presented the best performance. There are cases, in which the ASR outputs an excerpt such as:

(…) escritor que se llama Nick Kong, que es un escritor que se llama Nick Kong, que es un escritor que se llama Nick Kong, que es un escritor que se llama Nick Kong, que es un escritor que se llama Nick Kong, que es un escritor que se llama Nick (…)

This speech recognition phenomenon is known as "hallucination" when using the transformer technology and resulted in a worse ASR accuracy percentage value [19] for certain models such as the large model in MAVIR test utterances.

Overall, the Whisper Timestamped ASR system has been able to accurately transcribe speech recordings with an accuracy of 91.20% for MAVIR development, 83.39% for MAVIR test and 93.47% for SPARL22 test databases. The highest values for the SPARL22 database is due to SPARL22 presenting speech recordings from parliament sessions, which are more intelligible and present less noise than MAVIR data, which is more spontaneous.

Nevertheless, the QbE STD system has presented better results on MAVIR test data than on SPARL22 data, 0.6071 vs. 0.4547. Although the SPARL22 data obtained better ASR performance that MAVIR test data, the queries selected for SPARL22 data present a worse ASR performance, with an accuracy of ASR of 71.30% which is significantly lower than that obtained for MAVIR test data (86.79. This means that the chosen for MAVIR data are less likely to be misrecognized or confused with other words. Moreover, SPARL test query data contains certain names that are not in Spanish, such as "Michael" in TEST-0168 that can contribute to worse recognition. This may also mean that the lower accuracy in the ASR for MAVIR development data comes from other common words or stop words, not the target query terms.

It is significant to note that the values for MTWV and ATWV are very similar and equal in the case of MAVIR development with 0.8695, indicating that the detection threshold has been well calibrated by keeping all occurrences provided by the ASR subsystem as actual query STD detections in the QbE STD system.

8 CONCLUSION

QbE STD is a field of ever-increasing interest. Having vast amounts of data stored in speech signals, there is an existing need to access data stored through specific query searches. This is particularly relevant in patients with reduced or impaired vision, where there is an even greater unmet need. Through this automated system, these individuals can have an improved experience manipulating and accessing digitally stored databases for diverse purposes. The aim of this dissertation has been to develop a QbE STD system from a state-of-the-art approach based on Whisper-Timestamped.

The system is comprised of two main blocks or subsystems, the ASR subsystem, performed by Whisper Timestamped, and the QbE STD subsystem developed in the dissertation. In order to test the system's capabilities at detecting spoken terms in speech repositories, two different metrics were calculated at the levels of the subsystems, the ASR recognition accuracy and the ATWV value for the QbE STD.

The development of the dissertation has provided insights on the potential of applying Whisper Timestamped as an Automatic Speech Recognition system for detecting queries within utterances in all speech-based signals.

To further understand the scope and limitations of the system, it would be interesting to apply the system to a diverse range of databases and see how results differ in multiple scenarios. Furthermore, to deal with hallucinations by the ASR system, a Voice Activity Detector (VAD) could be run before applying Whisper Timestamped to avoid hallucinations due to errors in training data [20].

In conclusion, this work serves as a basis for future research into the use of Whisper Timestamped as a tool to help patients suffering from impaired vision in their digital tasks and for STD systems that do not have a text input. By addressing the additional steps mentioned above, this methodology can be implemented to create a significant impact in the lives of individuals by developing an accessible, cost effective and accurate QbE STD system.

9 REFERENCES

[1] Deepgram, “Deepgram,” 02 06 2024. [Online]. Available: https://deepgram.com/learn/the-history-of-automatic-speech-recognition.

[2] G. Deekshitha and L. Mary, “Multilingual spoken term detection: a review,” *International Journal of Speech Technology,* p. 15, 2020.

[3] “Blindness Statistics,” National Federation of the Blind, 01 2019. [Online]. Available: https://nfb.org/resources/blindness-statistics. [Accessed 05 06 2024].

[4] J. Tejedor, D. T. Toledano, X. Anguera, A. Varona, L. F. Hurtado, A. Miguel and J. Colas, “Query-by-Example Spoken Term Detection ALBAYZIN 2012 Evaluation: overview, systems, results, and discussion,” *EURASIP Journal on Audio, Speech, and Music Processing,* p. 17, 2013.

[5] C. Parada, A. Sethy and B. Ramabhadran, “Query-by-Example Spoken Term Detection For OOV terms,” *IEEE,* vol. 31, no. 11th , 2005.

[6] Y. Zhang and J. R. Glass, “Unsupervised spoken keyword spotting via segmental dtw on gaussian posteriorgrams,” *IEEE Workshop on Automatic Speech Recognition & Understanding (ASRU),* pp. 398-403, 2009.

[7] A. S. Park and J. R. Glass, “Unsupervised pattern discovery in speech,” *IEEE Transactions on Audio, Speech, and Language Processing,* vol. 16, pp. 186-197, 2008.

[8] D. Ram and A. Asaei, “Sparse subspace modeling for query by example spoken term detection,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* vol. 26, pp. 1130-1143, 2018.

[9] M. Muller, “Information retrieval for music and motion,” *Springer,* vol. 2, 2007.

[10] D. Ram, L. Miculicich and H. Bourlard, “CNN Based Query By Example Spoken Term Detection,” *Interspeech 2018,* pp. 92-96, 2018.

[11] I. Szoke, L. Burget, F. Grezl, J. Honza, C. and L. Ondel, “Calibration and fusion of query-by-example systems - BUT SWS 2013,” *IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP),* 2014.

[12] A. Abad, M. Penagarikano, L. J. Rodriguez-Fuentes and A. Varona, “On the Calibration and Fusion of Heterogeneous Spoken Term Detection Systems,” *Interspeech,* pp. 1-6, 2013.

[13] P. Lopez-Otero, J. Parapar and A. Barreiro, “Efficient query-by-example spoken document retrieval combining phone multigram representation and dynamic time warping,” *ELSEVIER Information Processing and Management,* vol. 56, pp. 43-60, 2019.

[14] P. Yang, H. Xu, X. Xiao, L. Xie, C.-C. Leung, H. Chen, J. Yu, H. Lv, L. Wang, S. Jun Leow, B. Ma, E. S. Chng and H. Li, “The NNI Query-by-Example System for MediaEval 2014,” *Institute for Infocomm Research,* pp. 1-2, 2014.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, “Attention Is All You Need,” *Conference on Neural Information Processing Systems,* no. 31, p. 15, 2017.

[16] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey and I. Sutskever, “Robust Speech Recognition via Large-Scale Weak Supervision,” *arXiv,* 2020.

[17] J. Tejedor and D. T. Toledano, “Whisper-based spoken term detection systems for search on speech ALBAYZIN evaluation challenge,” *EURASIP Journal on Audio, Speech, and Music Processing*, p. 20, 2024.

[18] J. Tejedor, D. T. Toledano, J. M. Ramirez, A. R. Montalvo and J. I. Alvarez-Trejos, “The multi-domain international search on speech 2020 ALBAYZIN,” *MDPI*, vol. 11, no. 18, 2021.

[19] D. Jurafsky and J. H. Martin, “Chapter 8: Hidden Markov Models,” in *Speech and Language Processing*, 2024, p. 17.

[20] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev and P. Woodland, HTK Book Alpha Version 3.4, Cambridge University Engineering Department, 2006.

[21] “A Brief History of ASR: Automatic Speech Recognition,” Medium, 31 04 2019. [Online]. Available: https://towardsdatascience.com/a-brief-history-of-asr-automatic-speech-recognition-95de6c014187. [Accessed 03 06 2024].

[22] GitHub, “Multilingual Automatic Speech Recognition with word-level timestamps and confidence,” GitHub, 2022. [Online]. Available: https://github.com/linto-ai/whisper-timestamped. [Accessed 01 02 2024].

9 APPENDIX

9.1 HResults

9.1.1 HResults for MAVIR Development Files

File	Combination	Accuracy
Mavir03	Tiny	38.67
Mavir03	Tiny accurate	78.00
Mavir03	Small	91.58
Mavir03	Small Accurate	90.60
Mavir03	Base	73.98
Mvir03	Base Accurate	85.90
Mavir03	Medium	93.20
Mavir03	Medium Accurate	92.88
Mavir03	Large	76.17
Mavir03	Large Accurate	92.40
Mavir07	Tiny	30.57
Mavir07	Tiny accurate	75.09
Mavir07	Small	86.65
Mavir07	Small Accurate	86.44
Mavir07	Base	81.97
Mavir07	Base Accurate	83.27
Mavir07	Medium	87.71

Mavir07	Medium Accurate	87.19
Mavir07	Large	20.91
Mavir07	Large Accurate	86.36

9.1.2 HResults for MAVIR Development Query 1 Folder Files

File (Folder 1)	Accuracy
DEV-0002	100
DEV-0003	0
DEV-0007	0
DEV-0009	0
DEV-0019	100
DEV-0021	0
DEV-0025	100
DEV-0027	0
DEV-0032	100
DEV-0033	0
DEV-0035	100
DEV-0036	100
DEV-0037	100
DEV-0039	100
DEV-0040	100
DEV-0042	100
DEV-0045	100
DEV-0053	0
DEV-0056	100
DEV-0059	0
DEV-0060	100
DEV-0066	0
DEV-0067	100
DEV-0071	100

DEV-0077	100
DEV-0081	0
DEV-0082	100
DEV-0088	100
DEV-0090	100
DEV-0098	0
DEV-0101	0
DEV-0106	0
DEV-0116	100
DEV-0120	0
DEV-0127	100
DEV-0130	100
DEV-0132	0
DEV-0134	0
DEV-0138	0
DEV-0145	100
DEV-0146	100
DEV-0149	100
DEV-0150	0
DEV-0151	0
DEV-0153	0
DEV-0165	100
DEV-0167	100
DEV-0168	0
DEV-0173	0
DEV-0181	100

DEV-0183	0
DEV-0188	100
DEV-0193	0
DEV-0195	100
DEV-0198	0
DEV-0203	0
DEV-0206	0
DEV-0208	0
DEV-0209	0
DEV-0214	100
DEV-0217	0
DEV-0218	0
DEV-0222	100
DEV-0233	0
DEV-0235	100
DEV-0237	100
DEV-0239	0
DEV-0243	100
DEV-0244	0
DEV-0248	100
DEV-0252	0
DEV-0253	100
DEV-0256	100
DEV-0258	100
DEV-0260	0
DEV-0266	100

DEV-0273	0
DEV-0278	0
DEV-0285	100
DEV-0291	100
DEV-0298	100
DEV-0301	0
DEV-0310	100
DEV-0313	0
DEV-0316	100
DEV-0324	100
DEV-0330	100
DEV-0331	0
DEV-0336	100
DEV-0337	0
DEV-0339	0
DEV-0340	0
DEV-0341	0
DEV-0342	100
DEV-0347	0
DEV-0349	0
DEV-0351	100
DEV-0352	0
DEV-0356	0
DEV-0360	100
DEV-0362	0
DEV-0363	100

Total	50.98

9.1.3 HResults for MAVIR Development Query 2 Folder Files

File (Folder 2)	Accuracy
DEV-0002	100
DEV-0003	100
DEV-0007	100
DEV-0009	100
DEV-0019	100
DEV-0021	100
DEV-0025	100
DEV-0027	0
DEV-0032	100
DEV-0033	100
DEV-0035	100
DEV-0036	100
DEV-0037	100
DEV-0039	100
DEV-0040	100
DEV-0042	100
DEV-0045	100
DEV-0053	100
DEV-0056	100
DEV-0059	100

DEV-0060	100
DEV-0066	100
DEV-0067	100
DEV-0071	100
DEV-0077	100
DEV-0081	100
DEV-0082	100
DEV-0088	100
DEV-0090	100
DEV-0098	100
DEV-0101	100
DEV-0106	100
DEV-0116	100
DEV-0120	100
DEV-0127	100
DEV-0130	100
DEV-0132	0
DEV-0134	0
DEV-0138	100
DEV-0145	100
DEV-0146	100
DEV-0149	100
DEV-0150	0
DEV-0151	0
DEV-0153	100
DEV-0165	100

DEV-0167	100
DEV-0168	100
DEV-0173	100
DEV-0181	100
DEV-0183	100
DEV-0188	100
DEV-0193	0
DEV-0195	100
DEV-0198	0
DEV-0203	100
DEV-0206	100
DEV-0208	100
DEV-0209	100
DEV-0214	100
DEV-0217	100
DEV-0218	0
DEV-0222	100
DEV-0233	100
DEV-0235	100
DEV-0237	100
DEV-0239	100
DEV-0243	100
DEV-0244	100
DEV-0248	100
DEV-0252	100
DEV-0253	100

DEV-0256	100
DEV-0258	100
DEV-0260	100
DEV-0266	100
DEV-0273	100
DEV-0278	100
DEV-0285	100
DEV-0291	100
DEV-0298	100
DEV-0301	100
DEV-0310	100
DEV-0313	100
DEV-0316	100
DEV-0324	100
DEV-0330	100
DEV-0331	0
DEV-0336	100
DEV-0337	100
DEV-0339	100
DEV-0340	100
DEV-0341	0
DEV-0342	100
DEV-0347	0
DEV-0349	100
DEV-0351	100
DEV-0352	100

DEV-0356	0
DEV-0360	100
DEV-0362	100
DEV-0363	100
Total	88.24

9.1.4 HResults for MAVIR Development Query 3 Folder Files

File (Folder 3)	**Accuracy**
DEV-0002	100
DEV-0003	100
DEV-0007	100
DEV-0009	100
DEV-0019	100
DEV-0021	100
DEV-0025	100
DEV-0027	0
DEV-0032	100
DEV-0033	100
DEV-0035	100
DEV-0036	100
DEV-0037	100
DEV-0039	100
DEV-0040	100

DEV-0042	100
DEV-0045	100
DEV-0053	100
DEV-0056	100
DEV-0059	100
DEV-0060	100
DEV-0066	100
DEV-0067	100
DEV-0071	100
DEV-0077	100
DEV-0081	100
DEV-0082	100
DEV-0088	100
DEV-0090	100
DEV-0098	100
DEV-0101	100
DEV-0106	100
DEV-0116	100
DEV-0120	100
DEV-0127	100
DEV-0130	100
DEV-0132	0
DEV-0134	100
DEV-0138	100
DEV-0145	100
DEV-0146	100

DEV-0149	100
DEV-0150	0
DEV-0151	100
DEV-0153	100
DEV-0165	100
DEV-0167	100
DEV-0168	100
DEV-0173	100
DEV-0181	100
DEV-0183	100
DEV-0188	100
DEV-0193	100
DEV-0195	100
DEV-0198	0
DEV-0203	100
DEV-0206	100
DEV-0208	100
DEV-0209	100
DEV-0214	100
DEV-0217	100
DEV-0218	0
DEV-0222	100
DEV-0233	100
DEV-0235	100
DEV-0237	100
DEV-0239	100

DEV-0243	100
DEV-0244	100
DEV-0248	100
DEV-0252	100
DEV-0253	100
DEV-0256	100
DEV-0258	100
DEV-0260	100
DEV-0266	100
DEV-0273	100
DEV-0278	100
DEV-0285	100
DEV-0291	0
DEV-0298	100
DEV-0301	100
DEV-0310	100
DEV-0313	100
DEV-0316	100
DEV-0324	100
DEV-0330	100
DEV-0331	0
DEV-0336	100
DEV-0337	100
DEV-0339	100
DEV-0340	100
DEV-0341	100

DEV-0342	100
DEV-0347	100
DEV-0349	100
DEV-0351	100
DEV-0352	100
DEV-0356	100
DEV-0360	100
DEV-0362	100
DEV-0363	100
Total	93.14

9.1.5 HResults for MAVIR Test Utterances

File	Accuracy
Mavir04	86.86
Mavir11	89.55
Mavir13	76.84

9.1.6 HResults for MAVIR Test Query 3 Folder Files

File (Folder 3)	Accuracy

TEST-0000	100
TEST-0001	100
TEST-0004	100
TEST-0005	100
TEST-0006	100
TEST-0009	100
TEST-0011	100
TEST-0013	100
TEST-0016	100
TEST-0019	100
TEST-0020	100
TEST-0021	0
TEST-0023	100
TEST-0025	100
TEST-0028	100
TEST-0029	100
TEST-0030	100
TEST-0036	0
TEST-0038	100
TEST-0040	0

TEST-0041	100
TEST-0043	100
TEST-0045	100
TEST-0047	100
TEST-0048	100
TEST-0050	100
TEST-0051	100
TEST-0053	100
TEST-0055	100
TEST-0060	100
TEST-0064	100
TEST-0065	100
TEST-0067	0
TEST-0069	100
TEST-0070	100
TEST-0071	0
TEST-0074	100
TEST-0075	100
TEST-0078	100
TEST-0079	100

TEST-0083	0
TEST-0085	0
TEST-0087	100
TEST-0088	100
TEST-0090	0
TEST-0092	100
TEST-0093	100
TEST-0094	100
TEST-0096	100
TEST-0097	100
TEST-0098	100
TEST-0101	100
TEST-0103	100
TEST-0104	100
TEST-0107	100
TEST-0109	100
TEST-0110	100
TEST-0112	100
TEST-0114	100
TEST-0115	100

TEST-0119	100
TEST-0120	100
TEST-0121	100
TEST-0123	100
TEST-0127	100
TEST-0128	100
TEST-0134	100
TEST-0135	100
TEST-0137	100
TEST-0140	100
TEST-0142	100
TEST-0144	100
TEST-0145	100
TEST-0149	100
TEST-0150	
TEST-0153	100
TEST-0156	100
TEST-0158	100
TEST-0159	100
TEST-0162	100

TEST-0167	100
TEST-0168	100
TEST-0169	100
TEST-0172	100
TEST-0174	100
TEST-0175	100
TEST-0178	100
TEST-0179	0
TEST-0180	100
TEST-0183	100
TEST-0186	100
TEST-0188	100
TEST-0190	100
TEST-0192	0
TEST-0193	100
TEST-0196	100
TEST-0197	100
TEST-0199	100
TEST-0201	0
TEST-0203	0

TEST-0204	100
TEST-0205	0
TEST-0210	100
TEST-0212	100
TEST-0216	100
TEST-0218	0
Total	86.79

9.1.7 HResults for SPARL22 Test Utterances

File	Accuracy
12_000400_003_0_16430_586456	97.43
12_000400_148_0_18727_632388	86.53
12_000400_153_0_18748_633006	97.06
13_000327_002_0_19437_643241	82.94
13_000400_002_1_19376_642366	94.05
13_000400_002_1_19376_642375	94.52
13_000400_003_0_19381_642398	93.49
13_000400_003_0_19381_642399	96.29
13_000400_004_0_19388_642448	89.34
13_000400_005_0_19422_642922	95.90
13_000400_005_0_19422_642923	95.83
13_000400_005_0_19422_642932	92.06
13_000400_007_0_19432_643097	90.75
13_000500_003_1_19421_642906	96.46

9.1.8 HResults for SPARL22 Test Query 3 Folder Files

File (Folder 3)	Accuracy
TEST-0000	0
TEST-0008	100
TEST-0011	100
TEST-0012	0
TEST-0013	0
TEST-0017	100
TEST-0018	100
TEST-0024	100
TEST-0025	100
TEST-0026	100
TEST-0028	100
TEST-0030	0
TEST-0033	100
TEST-0034	0
TEST-0039	100
TEST-0042	100
TEST-0043	100
TEST-0046	100
TEST-0048	0
TEST-0056	100
TEST-0058	100
TEST-0061	0
TEST-0062	100

TEST-0063	100
TEST-0072	100
TEST-0073	0
TEST-0076	100
TEST-0078	100
TEST-0084	100
TEST-0089	0
TEST-0094	100
TEST-0095	100
TEST-0096	100
TEST-0099	100
TEST-0100	100
TEST-0101	0
TEST-0103	100
TEST-0104	100
TEST-0105	100
TEST-0106	100
TEST-0110	100
TEST-0111	100
TEST-0112	100
TEST-0113	100
TEST-0114	0
TEST-0115	100
TEST-0119	0
TEST-0121	100
TEST-0122	100

TEST-0124	0
TEST-0125	100
TEST-0132	100
TEST-0138	0
TEST-0143	0
TEST-0145	0
TEST-0147	100
TEST-0152	100
TEST-0153	100
TEST-0154	100
TEST-0158	100
TEST-0159	100
TEST-0160	0
TEST-0161	0
TEST-0162	100
TEST-0163	100
TEST-0167	100
TEST-0168	0
TEST-0169	0
TEST-0180	100
TEST-0184	0
TEST-0187	0
TEST-0188	0
TEST-0189	100
TEST-0191	100
TEST-0196	100

TEST-0198	100
TEST-0201	0
TEST-0202	100
TEST-0205	100
TEST-0206	0
TEST-0207	100
TEST-0208	0
TEST-0209	0
TEST-0212	100
TEST-0215	100
TEST-0216	0
TEST-0217	100
TEST-0219	100
TEST-0220	0
TEST-0222	100
TEST-0225	100
TEST-0231	100
TEST-0235	100
TEST-0236	100
TEST-0242	0
TEST-0244	100
TEST-0245	100
TEST-0246	100
TEST-0248	100
TEST-0249	0
TEST-0252	100

TEST-0253	100
TEST-0254	100
TEST-0255	100
TEST-0256	100
TEST-0257	100
TEST-0258	100
TEST-0259	100
Total	71.30

9.2 JSON Code

9.2.1 Formatting Code

9.2.1.1 Json file into text format for HResults implementation

```
import json
import unidecode
from num2words import num2words
import re
import os
import roman

def split_words(text):
    return text.split()

def is_valid_float(word):
    try:
        float_value = float(word)
        return True
    except ValueError:
        return False

def number_to_spanish_word(word):
    return num2words(word, lang='es')

def is_roman_numeral(word):
    roman_pattern = 
r'^M{0,4}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$'
    match = re.match(roman_pattern, word)
    return match is not None

def clean_and_translate(word):
    word = 
word.rstrip('.').rstrip(',').rstrip('?').lstrip('¿').rstrip('...').rst
rip('-').lstrip('¡').rstrip('!').rstrip('...').rstrip('…')
    if not word:
        print("Empty string encountered. Returning as is.")
        return word

      # (…) Not included specific file scenarios for brevity
       if '.' in word:
        word =  word.replace('.', '')
        if is_valid_float(word):
            if len(word) > 12:
                word = "dosmilceros"
                return word
            else:
                word = number_to_spanish_word(int(word))
                word = word.lower()
                return word
    if '%' in word:
        number_part,percent_part = word.split('%')
        if ',' in number_part:
            number_firstpart,number_secondpart = word.split(',')
            number_secondpart = number_secondpart.rstrip('%')
            number_translationfirstpart = 
number_to_spanish_word(int(number_firstpart))
            number_translationsecondpart = 
number_to_spanish_word(int(number_secondpart))
            word = f"{number_translationfirstpart} coma 
{number_translationsecondpart} por ciento"
        else:
            number_translation = 
number_to_spanish_word(int(number_part))
            word = f"{number_translation} por ciento"
        word = word.lower()
        return word

    if ',' in word:
        if word.count(',')==1:
            number_firstpart,number_secondpart = word.split(',')
            if len(number_secondpart)>1:
                combinenumber = number_firstpart + number_secondpart
                word = number_to_spanish_word(int(combinenumber))
                word = word.lower()
            else:
                number_translationfirstpart = 
number_to_spanish_word(int(number_firstpart))
                number_translationsecondpart = 
number_to_spanish_word(int(number_secondpart))
                word = f"{number_translationfirstpart} coma 
{number_translationsecondpart}"
                word = word.lower()
            return word
        else:
            word =  word.replace(',', '')
            word = number_to_spanish_word(word)
            word = word.lower()
        return word
    if is_valid_float(word):
        word = number_to_spanish_word(word)
    if is_roman_numeral(word):
        convertedromannumber = roman.fromRoman(word)
        word = number_to_spanish_word(int(convertedromannumber))
        word = word.lower()
        return word
    word = word.lower()
    return word
def write_words_to_text(json_file_path, text_file_path):
    try:
        with open(json_file_path, 'r', encoding='utf-8') as json_file:
            json_content = json.load(json_file)
        words = []
        for key, value in json_content.items():
            if isinstance(value, str):
                words.extend(split_words(value))
        words.pop()
        cleaned_words = []
        for word in words:
            cleaned_word = clean_and_translate(word)
            if cleaned_word:
                cleaned_words.extend(split_words(cleaned_word))
        with open(text_file_path, 'w', encoding='utf-8') as text_file:
            text_file.write('#!MLF!#\n')
            text_file.write(f'"*/{os.path.splitext(os.path.basename(js
on_file_path))[0]}"\n')
            print(' '.join(cleaned_words))
            for cleaned_word in cleaned_words:
                text_file.write(cleaned_word + '\n')
            text_file.write('.\n')
        print(f"Words written to {text_file_path}")

    except Exception as e:
        print(f"An error occurred un error de: {e}")
```

9.2.1.2 Creating a text file with unique words for HResults Implementation

```
def uniquewords(combinedfile, outputfile):
    try:
        with open(combinedfile, 'r') as file1:
            seen_words = set()
            words = []

            for line in file1:
                cleaned_line = line.strip()
                if not cleaned_line.startswith('"*/mavir') and not
cleaned_line == '.' and not cleaned_line.startswith('#!MLF!#'):
                    for word in cleaned_line.split():
                        if word not in seen_words:
                            seen_words.add(word)
                            words.append(word)

        unique_words = '\n'.join(words)
        with open(outputfile, 'w') as output_file:
            output_file.write(unique_words)

        print(f"Unique words file successfully created: {outputfile}")

    except Exception as e:
        print(f"An error occurred: {e}")
```

9.2.2 Creating the column / matrix format file and matching

9.2.2.1 Column formatting function

```
def calculate_duration(start_time, end_time):
    duration = end_time - start_time
    return duration


def matrix_formatting(json_file, output_file_path):
    with open(json_file, 'r', encoding='utf-8') as file:
        data = json.load(file)
        segments = data['segments']
        with open(output_file_path, 'w', encoding='utf-8') as
output_file:
            for segment in segments:
                segment_text = segment['text']
                words = segment['words']
                for word in words:
                    word_text = word['text']
                    cleaned_word = clean_and_translate(word_text)
                    if ' ' in cleaned_word:
                        cleaned_word =  cleaned_word.replace(' ', '_')
                    word_start = word['start']
                    word_end = word['end']
                    word_confidence = word['confidence']
                    word_duration = round((word_end - word_start),2)
                    output_file.write(cleaned_word+" ")
                    output_file.write(str(word_start)+" ")
                    output_file.write(str(word_end)+" ")
                    output_file.write(str(word_confidence)+" ")
                    output_file.write(str(word_duration)+" ")
                    output_file.write("12_000400_003_0_16430_586456"+"
\n")
```

9.2.2.2 Swapping / matching query – utterance function

```
from decimal import Decimal, ROUND_HALF_UP
import math
def round_half_up(n, decimals=0):
    if isinstance(n, float):
        decimals = max(0, len(str(n).split('.')[1]))
    multiplier = 10**decimals
    result = math.floor(n * multiplier + 0.5) / multiplier
    if len(str(result).replace(".", "").lstrip("0")) == 4:
        result = round(result, 3)
    truncated_number = int(result * 1000) / 1000.0
    return truncated_number
def multiplewordswapping(mapping_file, input_matrix, output_matrix):
    mapping_content = []
    mapping_with_underscore = []
    mapping_without_underscore = []
    mismatched_items = []
    last_mismatch_position = []
    with open(mapping_file, 'r') as f:
        next(f)
        lines = f.readlines()
        for i in range(0, len(lines), 3):
            dev_title = lines[i].strip().split('/')[-1].split('.')[0]
            word = lines[i+1].strip()
            full_stop = lines[i+2].strip()
            if '_' in word:
                mapping_with_underscore.append((dev_title, word,
full_stop))
            else:
                mapping_without_underscore.append((dev_title, word,
full_stop))

    mapping_content.extend(mapping_with_underscore)
    mapping_content.extend(mapping_without_underscore)

    with open(input_matrix, 'r') as input_file, open(output_matrix,
'w') as output_file:
        written_multiple_words = False
        mismatch_occured = False
        for line in input_file:
            parts = line.strip().split()
            first_word = parts[0] if parts else None
            found_mapping = False
            mismatch_occured = False
            for dev_title, word, full_stop in mapping_content:
                if first_word == word:
                    parts[0] = dev_title
                    output_file.write(' '.join(parts) + '\n')
                    found_mapping = True
                    break

                if '_' in word:
                    doublematch = []
                    mismatch = []
                    multiplewords = word.split('_')
                    lineword = first_word
                    confidencevalues = []
                    for i in range(len(multiplewords)):
                        for dev_title1, word1, full_stop1 in
mapping_content:
                            if lineword == word1:
                                parts[0] = dev_title1 #DEV...
                                doublematch.append(' '.join(parts) +
'\n')
                        if lineword == multiplewords[i]:
                            confidencevalues.append(parts[3])
                            if i==0:
                                startvalue = float(parts[1])
                                mismatch.append((lineword,parts[1],par
ts[2],parts[3],parts[4],parts[5]))
                            if i == len(multiplewords) - 1:
                                endvalue = float(parts[2])
                                duration = round((endvalue-
startvalue),2)
                                confidencevalues = [Decimal(value) for
value in confidencevalues]
                                previoustoround = (sum(float(value)
for value in confidencevalues) / len(confidencevalues))
                                confidence =
round_half_up((sum(float(value) for value in confidencevalues) /
len(confidencevalues)),4)
                                output_line = dev_title + ' ' +
str(startvalue) + ' ' + str(endvalue) + ' ' + str(confidence) + ' ' +
str(duration) + ' mavir07\n'
                                output_file.write(output_line)
                                for match in doublematch:
                                    output_file.write(''.join(map(str,
match)))
                                written_multiple_words = True
                            else:
                                line = input_file.readline()
                                parts = line.strip().split()
                                lineword = parts[0] if parts else None
                                mismatch.append((lineword,parts[1],par
ts[2],parts[3],parts[4],parts[5]))
                        else:
                            if len(mismatch)>0:
                                mismatch_occured = True
                                mismatched_items.extend(mismatch)
                                last_mismatch_position.append(len(mism
atch))
                                break
                            break

            if not found_mapping and not written_multiple_words:
                if mismatch_occured:
                    for mismatch_item in
mismatched_items[(len(mismatched_items)-last_mismatch_position[-1]):]:
                        output_file.write(' '.join(map(str,
mismatch_item))+'\n')
                else:
                    output_file.write(line)

            if written_multiple_words:
                written_multiple_words = False
            if mismatch_occured:
                mismatch_occured = False
```

9.2.3 Creating the XML for Detection Error Tradeoff Analysis

9.2.3.1 Development of file with rows containing only DEV swapped words

```
def onlydev_lines(input_file, output_file):
    with open(input_file, 'r') as f_in, open(output_file, 'w') as 
f_out:
        for line in f_in:
            if line.startswith('DEV'):
                f_out.write(line)
```

9.2.3.2 Development of file with unique matched queries present

```
def find_unique_words(input_file):
    unique_words = []
    with open(input_file, 'r') as f_in:
        for line in f_in:
            word = line.strip().split()[0]  # Extract the first word
            if word not in unique_words:
                unique_words.append(word)
    return unique_words

def filter_lines_with_unique_words(input_file, unique_words, 
output_file):
    with open(input_file, 'r') as f_in, open(output_file, 'w') as 
f_out:
        for line in f_in:
            word = line.strip().split()[0]  # Extract the first word
            if word in unique_words:
                f_out.write(line)
```

9.2.3.3 Text to XML file transformation code

```
def xml_file(input_file, unique_words, output_file):
    with open(output_file, 'w') as f_out:
        f_out.write('<stdlist 
termlist_filename="C:\\Users\\Adriana\\Desktop\\TFG\\resultados\\onlyd
evmatrizswapmultiple.xml" indexing_time="1.000" language="spanish" 
index_size="1" system_id="fake">\n')
        for word in unique_words:
            f_out.write(f'<detected_termlist termid="{word}" 
term_search_time="24.3" oov_term_count="1">\n')
            with open(input_file, 'r') as f_in:
                for line in f_in:
                    if line.startswith(word):
                        columns = line.strip().split()
                        term_file = columns[5]
                        tbeg = columns[1]
                        dur = columns[4]
                        score = columns[3]
                        f_out.write(f'<term file="{term_file}" 
channel="1" tbeg="{tbeg}" dur="{dur}" score="{score}" 
decision="YES"/>\n')
            f_out.write('</detected_termlist>\n')
        f_out.write('</stdlist>\n')
```

9.2.3.4 Ordering XML file in increasing tbeg of matching

```
def sort_lines_within_termlist(detected_termlist):
    terms =  []
    sorted_lines = []
    sorted_numbers =[]
    for line in detected_termlist:
        parts = line.strip().split()
        tbeg_part = parts[3]
        print(tbeg_part)
        tbeg_value = float(tbeg_part.split('=')[1].strip('"'))
        terms.append((tbeg_value))
    sorted_numbers = sorted(terms)
    for term in sorted_numbers:
    for number in sorted_numbers:
        for line in detected_termlist:
            partsofline = line.strip().split()
            tbeg_part = partsofline[3]
            tbeg_value = float(tbeg_part.split('=')[1].strip('"'))
            if number == tbeg_value:
                sorted_lines.append(line)
    return sorted_lines

def order_detected_termlists(input_file, output_file):
    with open(input_file, 'r') as f_in, open(output_file, 'w') as
f_out:
        first_line = f_in.readline()
        f_out.write(first_line)

        detected_termlist = []

        for line in f_in:
            if line.startswith('<detected_termlist'):
                f_out.write(line)
            elif line.startswith('</detected_termlist>'):
                sorted_lines =
sort_lines_within_termlist(detected_termlist)
                for sorted_line in sorted_lines:
                    f_out.write(sorted_line)
                detected_termlist = []
                f_out.write(line)
            elif line.startswith('<term'):
                detected_termlist.append(line)
            elif line.startswith('</stdlist>'):
                f_out.write(line)
```