# Automatic Audio Anonymization

by

Guillaume BARIL

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN INFORMATION TECHNOLOGY ENGINEERING
M.A.Sc.

MONTREAL, DECEMBER 17, 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

**BOARD OF EXAMINERS**

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Patrick Cardinal, Thesis Supervisor
Department of Software Engineering and Information Technology, École de technologie supérieure

Mr. Alessandro Lameiras Koerich, Thesis Co-supervisor
Department of Software Engineering and Information Technology, École de technologie supérieure

Mr. Pierre Dumouchel, President of the Board of Examiners
Department of Software Engineering and Information Technology, École de technologie supérieure

Mrs. Sylvie Ratté, Member of the Jury
Department of Software Engineering and Information Technology, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON DECEMBER 14, 2021

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

## ACKNOWLEDGEMENTS

First and foremost, I would like to extend my deepest gratitude to my supervisor Professor Patrick Cardinal for his guidance, support and knowledge. I would never have started this project without his advice. I also wish to thank my co-supervisor Professor Alessandro Lameiras Koerich for his suggestions and experience.

Special thanks to Serge Oligny and Guillaume Androz for their belief in my work and practical suggestions. I also very much appreciate the helpful advice from my colleagues and teachers.

I would like to acknowledge the financial support from Mitacs as part of the Mitacs Accelerate program.

Many thanks to my family, especially my father Benoit Baril, my mother Nathalie Péloquin, my grandmother Micheline Brin Baril and my significant other Myriam Jacquemin for all their love and support throughout this project.

This thesis is dedicated to my grandfather Donald Baril.

**Anonymisation automatique de l'audio**

Guillaume BARIL

## RÉSUMÉ

L'anonymisation de données est souvent une tâche réalisée par des humains. L'automatiser permettrait de réduire le coût et le temps requis pour réaliser cette tâche. Dans ce travail, on montre que l'anonymisation de données audio en français peut être automatisée. On propose un pipeline qui prend en entrée des fichiers audio avec leurs transcriptions et qui brouille les entités nommées présent dans l'audio.

Notre pipeline est formé de deux composantes. La première composante est l'aligneur qui va aligner les mots de la transcription avec l'audio et la deuxième composante est le modèle qui effectue la reconnaissance d'entités nommées. Ensuite, on remplace l'audio correspondant aux entités nommées par un silence. On a comparé plusieurs aligneurs et plusieurs modèles afin de trouver les meilleurs pour notre scénario.

On a testé notre pipeline sur une petite base de données annotée à la main et on a obtenu un score F1 de 76.9%. Ce résultat prouve qu'automatiser cette tâche est réalisable. Par contre, en ayant un jeu de données plus grand, il serait possible d'obtenir de meilleurs résultats, d'entraîner le modèle à reconnaître de nouvelles entités nommées et d'entraîner un modèle bout-en-bout qui obtiendrait probablement de meilleurs performances qu'un pipeline avec des composantes entraînées séparément.

**Mots-clés:** désidentification de l'audio, traitement du langage naturel, reconnaissance d'entités nommées, alignement forcé, apprentissage profond

**Automatic Audio Anonymization**

Guillaume BARIL

## ABSTRACT

Data anonymization is often a task carried out by humans. Automating it would reduce the cost and time required to complete this task. This work shows that the anonymization of audio data in French can be automated. We propose a pipeline, which takes audio files with their transcriptions and removes the named entities present in the audio.

Our pipeline is made up of two components. The first component is the aligner which will align the words in the transcript with the audio and the second component is the template which performs named entity recognition. Then, we replace the audio corresponding to the named entities with a silence. We compared several aligners and several models to find the best ones for our scenario.

We evaluated our pipeline on a small hand-annotated dataset, and it achieved a F1 score of 76.9%. This result proves that automating this task is feasible. However, by having a more extensive dataset it would be possible to get better results, train the model to recognize new named entities, and train an end-to-end model that would likely perform better than a pipeline with components trained separately.

**Keywords:** audio de-identification, natural language processing, named entity recognition, forced alignment, deep learning

## TABLE OF CONTENTS

Page

## LIST OF TABLES

Page

## LIST OF FIGURES

Page

## LIST OF ALGORITHMS

Page

## LIST OF ABBREVIATIONS

| | |
|---|---|
| HMM | Hidden Markov Model |
| EM | Expectation-Maximization |
| GMM | Gaussian Mixture Model |
| PDF | Probability Density Function |
| MFCC | Mel-Frequency Cepstral Coefficient |
| DFT | Discrete Fourier Transform |
| DCT | Discrete Cosine Transform |
| DFS | Depth-First Search |
| WFST | Weighted Finite-State Transducer |
| ASR | Automatic Speech Recognition |
| AM | Acoustic Model |
| LM | Language Model |
| CMLLR | Constrained Maximum Likelihood Linear Regression |
| FA | Forced Alignment |
| NER | Named Entity Recognition |
| NCCFr | Nijmegen Corpus of Casual French |
| MFA | Montreal Forced Aligner |
| SPPAS | SPeech Phonetization Alignment and Syllabification |
| CMVN | Cepstral Mean and Variance Normalization |

| | |
|---|---|
| RNN | Recurrent Neural Network |
| LSTM | Long Short-Term Memory |
| BiLSTM | Bidirectional Long Short-Term Memory |
| CNN | Convolutional Neural Network |
| CRF | Conditional Random Field |
| PE | Positional Encoding |
| IE | Input Embedding |
| FFN | Feed Forward Neural network |
| BERT | Bidirectional Encoder Representations from Transformers |
| MLM | Masked LM |
| NSP | Next Sentence Prediction |
| RoBERTA | Robustly Optimized BERT pretraining Approach |
| BPE | Byte-Pair Encoding |
| SLU | Spoken Language Understanding |
| NLU | Natural Language Understanding |
| STFT | Short-Time Fourier Transform |
| SGD | Stochastic Gradient Descend |
| NTE | No Type Error |

# INTRODUCTION

The more time passes, the more we live in a world where everything is digital. We can do everything online like shopping, managing our bank account, or watching a movie.

Companies understood this a long time ago. Nowadays, they collect data about their clients for multiple reasons: to improve their recommendations, understand customer needs, drive decision-making and much more. An example of data collected by businesses is the recordings from their call center.

In the last few years, there have been a huge amount of data breaches. "Black hat" hackers steal customers' personal information for many reasons like selling it or committing fraud. There are three main ways for companies to protect themselves from data breaches: limit access to the data, improve employees' security awareness, and patch system vulnerabilities.

## Problem Statement

Another way for companies to protect themselves from data breaches is to delete personal information when it is not needed. In most cases, call recordings are used for training purposes or to keep track of the agents' behavior. In these cases, the customer personal information is not important, so that it could be redacted.

Also, removing sensitive information from data would allow companies to make the datasets available to the public. Since gathering data is the most expensive phase of a new project, it could help the research community in their work.

In other words, we want to remove personal information from audio recordings.

**Scope and objectives**

Automatic de-identification of data is not a new task. There is a lot of research in this area. One example is de-identifying medical notes (Neamatullah *et al.*, 2008; Dernoncourt *et al.*, 2017; Liu *et al.*, 2017) to protect the confidentiality of patients. Another example is speaker anonymization (Bahmaninezhad *et al.*, 2018; Fang *et al.*, 2019; Patino *et al.*, 2021) to hide the speaker's identity. However, there is not a lot of research in audio de-identification. To the best of our knowledge, Cohn *et al.* (2019) are the only researchers who explored this topic. That being said, Cohn *et al.* (2019) used a pipeline with two components : Automatic Speech Recognition (ASR) and Named Entity Recognition (NER).

In other words, audio de-identification consists of finding named entities in speech and removing them afterwards. There is not a lot of end-to-end NER from speech models (Ghannay *et al.*, 2018; Yadav *et al.*, 2020), but NER on text is a popular research area (Huang *et al.*, 2015; Lample *et al.*, 2016; Devlin *et al.*, 2019; Liu *et al.*, 2019). It is important to note that these methods are language-dependent. For example, retraining the model from Devlin *et al.* (2019) on French data is another research of its own (Martin *et al.*, 2020; Le *et al.*, 2020).

Our main objective is to determine if anonymizing French audio automatically by removing named entities is a feasible task. This proof of concept will be used to determine if the project has enough potential to build real-life applications.

Due to limited resources, this work does not cover ASR. Instead, we assume that the audio has already been passed through an ASR model. Then, we apply Forced Alignment (FA) to align each word with the audio recording. Finally, we find the named entities in the transcription, and we remove them from the audio.

To achieve our main objective, we have established two intermediate goals. The first goal is to determine the best French FA algorithm to align each word with the audio recording, and

the second goal is to train a French NER model. This research focuses on anonymizing French audio recordings only. In addition, because our partner organization is a bank, we did our best to use financial data when possible.

### Contributions

In this work, we create a pipeline to anonymize French audio recordings automatically. Moreover, we show that it does not require a lot of data to achieve respectable performance. All things considered, our contributions are as follow :

- Create a speech corpus with word level boundaries in French
- Evaluate which FA algorithm is best for French
- Train a NER model on a French corpus of financial news
- Develop a pipeline to anonymize French audio automatically[1]

### Thesis outline

This thesis is organized as follows:

- Chapter 1 introduces the theory on which this research is based.
- Chapter 2 presents related works and state-of-the-art FA algorithms and NER architectures.
- Chapter 3 describes our methodology, our corpora, and FA and NER algorithms we decided to evaluate.
- Chapter 4 describes the experiments carried out on the FA and NER models and presents the models constituting the pipeline.
- Chapter 5 presents the pipeline created to anonymize audio recordings.

Finally, we conclude the thesis and present our recommendations for future work.

[1] Available at https://github.com/gbaril/audio_anonymization

# CHAPTER 1

# THEORETICAL BACKGROUND

In this section, we present the algorithms used by the pipeline components. The first component is the Forced Alignment (FA) and the second component is the Named Entity Recognition (NER), as we can see in Figure 1.1. For more details on the pipeline architecture, see Sections 3.1 and 5.1.

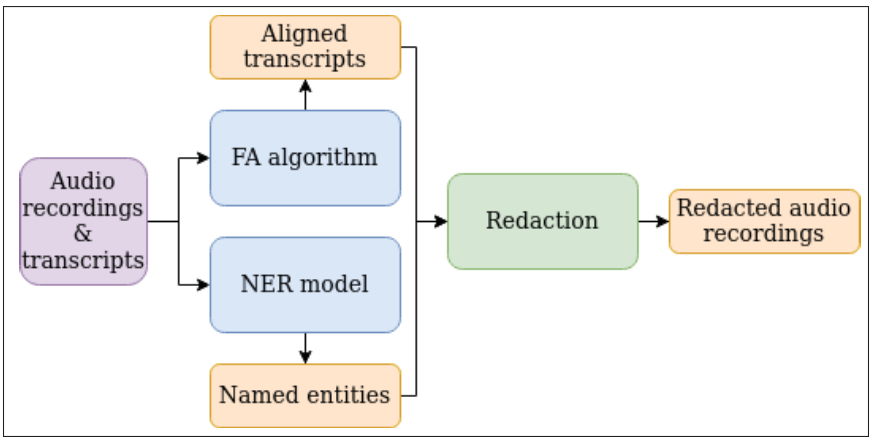


Figure 1.1 Pipeline architecture overview

Firstly, there are four algorithms for FA. The first algorithm is the Mel-frequency Cepstral Coefficients, which are the features of the sound used by the Hidden Markov Model. The second algorithm is the Hidden Markov Model which models the probability of seeing all phonemes based on sound features. The third algorithm is the Gaussian Mixture Model, which models the probability of each sound feature independently. Also, we introduce Weighted Finite-State Transducers, which are the core of several Automatic Speech Recognition (ASR) systems. Finally, the last algorithm is the Gaussian Mixture Model, which independently models the probability of each sound feature.

Also, there are three algorithms for NER. The first algorithm is the Long Short-Term Memory, the second is the Convolutional Neural Network, and the third is the Transformer. All of them

are deep neural network architectures. Finally, there is another algorithm called Conditional Random Fields, which are very similar to Hidden Markov Models.

## 1.1 Mel-frequency Cepstral Coefficients

The Mel-frequency Cepstral Coefficients (MFCCs) are commonly used features in ASR systems. The whole pipeline is summarized in Figure 1.2.

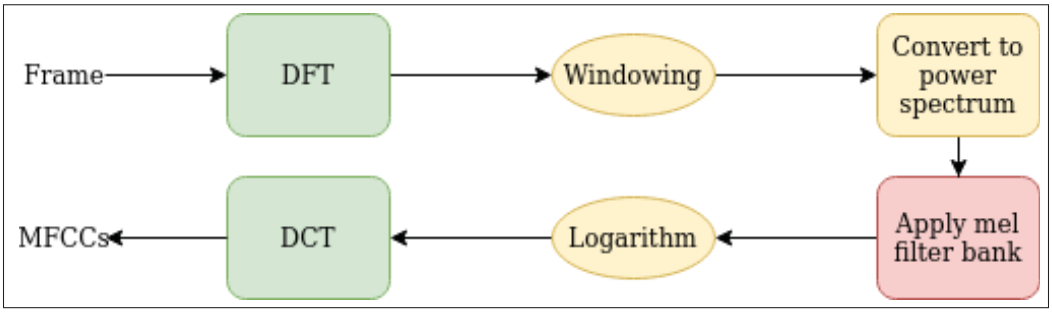


Figure 1.2 Mel-frequency Cepstral Coefficients computation pipeline

Let's define an audio signal $x$ with a frequency $f_x$ where $x(i)$ is the $i^{\text{th}}$ sample. Each frame $f$ contains $I$ samples starting at $f * (o - 1)$ where $o$ is the step size. The total number of frame $F$ is equal to $\frac{|x|-I}{o} + 1$. We can then declare $\hat{x} \in \mathbb{R}^{F \times I}$ where $\hat{x}_{fi}$ is the $i^{\text{th}}$ sample of the $f^{\text{th}}$ frame as described in Kopparapu & Laxminarayana (2010).

$$\hat{x}_{fi} = x(f * (o - 1) + i) \tag{1.1}$$

The first step to obtain the MFCCs is to calculate the $N$-points Discrete Fourier Transform (DFT) of each frame $f$ with eq. (1.2).

$$X_{fn} = \sum_{k=0}^{N-1} \hat{x}_{fk} w(k) e^{-2\pi i k n / N} \tag{1.2}$$

where $X \in \mathbb{R}^{F \times N}$ and $w(k)$ is a window function. Note that $N \geq I$ is usually a power of 2 and that $\hat{x}_{fk}$ is equal to 0 if $k > I$. The window function is important because when frequencies are

not multiples of $f_x/N$, the DFT coefficients will not equal zero for frequencies not in the original signal. This is called spectral leakage and it happens when the processed signal is not a full period. The window function prevents this by smoothing each ends of a frame and generating a periodic signal. One popular function is the hamming window defined as

$$w(n) = 0.54 - 0.46\cos(\frac{2\pi n}{N}) \tag{1.3}$$

Now, we can calculate the power spectrum with eq. (1.4). It is a matrix of the same size as the DFT.

$$P_{fn} = \frac{|X_{fn}|^2}{N} \tag{1.4}$$

Thirdly, we calculate the log energy of this power spectrum on the mel scale using a Mel filter bank $H \in \mathbb{R}^{N\times M}$. We calculate $Y \in \mathbb{R}^{F\times M}$ where $Y_{fm}$ is the $m^{\text{th}}$ log energy coefficient of the $f^{\text{th}}$ frame with eq. (1.5). The number of filters $M$ usually ranges between 20 and 40.

$$Y_{fm} = \ln \sum_{k=0}^{N-1} P_{fk} H_{km} \tag{1.5}$$

Finally, we obtain the first $U$ MFCCs by calculating the Discrete Cosine Transform (DCT) of $Y$. The $u^{\text{th}}$ MFCC of the $f^{\text{th}}$ frame is given by

$$c_{fu} = \sum_{m=0}^{M-1} Y_{fm} \cos(\frac{u\pi(2m-1)}{2M}) \tag{1.6}$$

where $c \in \mathbb{R}^{F\times U}$. We usually keep the first 12 or 13 coefficients.

### 1.1.1 Computing Mel filter bank

To compute the Mel filter bank, we first need to define functions to convert Hertz to Mel and vice versa.

$$\begin{aligned} \hat{m}(h) &= 2595\log(1 + \frac{h}{700}) \\ \hat{h}(m) &= 700(10^{m/2595} - 1) \end{aligned} \tag{1.7}$$

Then, we need four parameters for this algorithm: the minimum frequency $h_{\min}$, the maximum frequency $h_{\max}$, the number of filters $M$ and the sampling frequency $f_x$. Firstly, we calculate the Mel frequency resolution $\delta\phi$ as described in Sigurðsson *et al.* (2006).

$$\delta\phi = \frac{\hat{m}(h_{\max}) - \hat{m}(h_{\min})}{M+1} \tag{1.8}$$

Secondly, we calculate $\phi(m)$ which represents the $m^{\text{th}}$ triangular Mel filter bank. It contains $M+2$ elements.

$$\phi(m) = \lfloor \frac{(N+1)\hat{h}(m\delta\phi + m_{\min})}{f_x} \rfloor \tag{1.9}$$

Finally, $\phi(m)$ is used to calculate the Mel filter bank $H \in \mathbb{R}^{M\times N}$ given by

$$H_{km} = \begin{cases} 0 & \text{if } k < \phi(m-1) \\ \frac{k-\phi(m-1)}{\phi(m)-\phi(m-1)} & \text{if } \phi(m-1) \le k < \phi(m) \\ 1 & \text{if } k = \phi(m) \\ \frac{\phi(m+1)-k}{\phi(m+1)-\phi(m)} & \text{if } \phi(m) < k \le \phi(m+1) \\ 0 & \text{if } k > \phi(m+1) \end{cases} \tag{1.10}$$

The complete procedure to compute the Mel filter bank is summarized in algorithm 1.1.

Algorithm 1.1 Mel filter bank computation

**Input:** minimum frequency $h_{\min}$, maximum frequency $h_{\max}$, number of filters $M$ and sampling frequency $f_x$
**Output:** Mel filter bank $H$

```
δφ ← calculate with eq. (1.8);
for m ← 0 to M + 1 do
    φ(m) ← calculate with eq. (1.9);
for k ← 1 to K do
    for m ← 1 to M do
        H_km ← calculate with eq. (1.10);
```

### 1.1.2 Differential and acceleration coefficients

Differential and acceleration coefficients, also known as delta and delta-delta coefficients, can be seen as the derivative and the second derivative of MFCCs. It is defined as

$$d(\phi_m) = \frac{\sum_{i=1}^{D} i(\phi_{m+i} - \phi_{m-i})}{2\sum_{i=1}^{D} i^2} \tag{1.11}$$

where $D$ usually equals 2 or 3. To obtain delta-delta coefficients, we simply compute $d(d(\phi_m))$. In total, we have $3M$ coefficients usable by HMMs.

## 1.2 Gaussian Mixture Models

The Gaussian distribution is the most common Probability Density Function (PDF) and it has the following form

$$\mathcal{N}(x; \mu, \Sigma) = p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu)^{\mathsf{T}}\Sigma^{-1}(x-\mu)) \tag{1.12}$$

where $\mu \in \mathbb{R}^n$ is the mean vector and $\Sigma \in \mathbb{R}^{n\times n}$ is the covariance matrix. Figure 1.3 shows an example of a 2-dimensional Gaussian distribution.

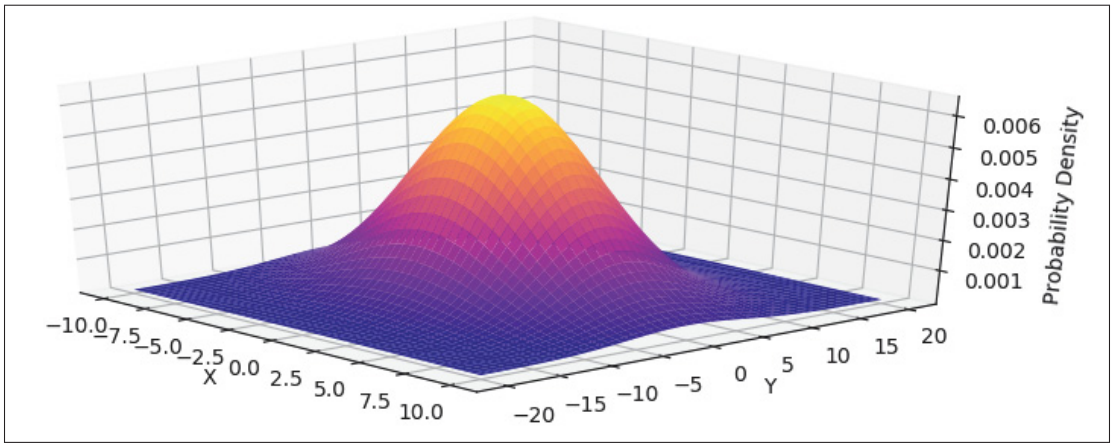


Figure 1.3 Illustration of a multivariate Gaussian distribution

The Gaussian Mixture Model (GMM) is a mixture model using $M$ Gaussian distributions $C_1, ..., C_M$. This distribution is defined as

$$p(x) = \sum_{m=1}^{M} c_m \mathcal{N}(x; \mu_m, \Sigma_m) \tag{1.13}$$

where $c_m$ is the weight associated to the $m$-th mixture. It is important to note that the sum of all weights $c_m$ must equal to one. Otherwise, the PDF will not be normalized. Figure 1.4 shows an example of a 2-dimensional GMM with two mixtures[2].

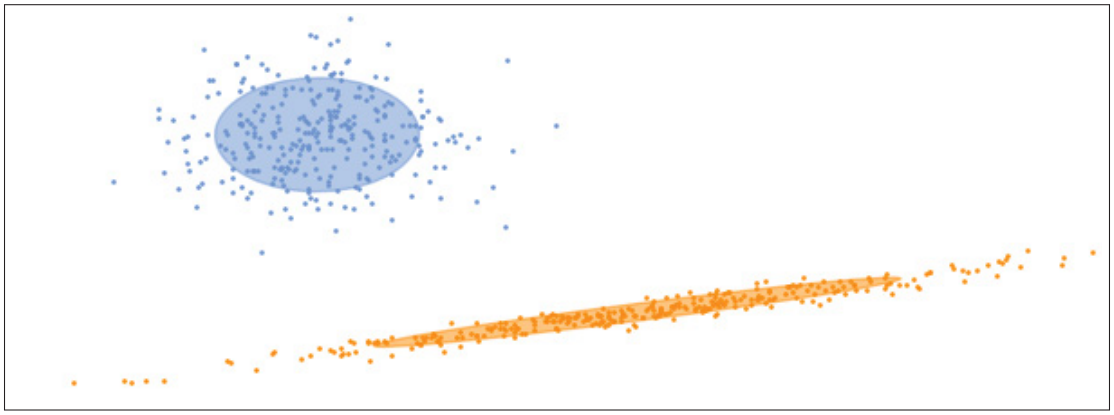

Figure 1.4 Illustration of a Gaussian Mixture Model with two mixtures

Also, let's define $z_t$ as the actual mixture component for $x_t$. We can calculate the probability that $x_t$ is modeled by the component $m$. We multiply the probability that $x_t$ belongs to the component $m$ by the probability that $x_t$ was generated by that component and we normalize it. More formally, it is defined as

$$w_m^t = P(z_t = C_m | x_t) = \frac{P(z_t = C_m) P(x_t | z_t = C_m)}{P(x_t)} = \frac{c_m \mathcal{N}(x_t; \mu_m, \Sigma_m)}{\sum_{k=1}^{M} c_k \mathcal{N}(x_t; \mu_k, \Sigma_k)} \tag{1.14}$$

GMMs use an Expectation-Maximization algorithm which is an iterative process to learn the best parameters $c_m$, $\mu_m$ and $\Sigma_m$ for $1 \leq m \leq M$. The first step is the *expectation step*. In this

[2] Taken from https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm.html

step, we have to compute the probability that each data point $x_t$ was generated by the $m$-th mixture, which is $w_m^t$. The second step is the *maximization step*. In this step, we update all the parameters $c_m$, $\mu_m$ and $\Sigma_m$ for $1 \leq m \leq M$ as follow

$$\begin{aligned}
\bar{c}_m &= \frac{1}{T}\sum_{t=1}^{T} w_m^t \\
\bar{\mu}_m &= \frac{\sum_{t=1}^{T} w_m^t x_t}{\sum_{t=1}^{T} w_m^t} \\
\bar{\Sigma}_m &= \frac{\sum_{t=1}^{T} w_m^t (x_t - \mu_m)(x_t - \mu_m)^\top}{\sum_{t=1}^{T} w_m^t}
\end{aligned} \tag{1.15}$$

These formulas are obtained by derivating the likelihood function with respect to $c_i$, $\mu_i$ and $\Sigma_i$. For more details, see Bishop (2006). We repeat the *expectation* and the *maximization* steps until convergence.

## 1.3 Hidden Markov Models

As defined in Rabiner (1989), the first order Markov property indicates that the state $t$ depends only upon the previous state $t-1$. Given $n$ states $S_1, ..., S_n$ and defining $q_t$ as the actual state at time $t$, the probability of $q_t$ being equal to state $S_i$ is defined by

$$P(q_t = S_i | q_1, ..., q_{t-1}) = P(q_t = S_i | q_{t-1} = S_j) \tag{1.16}$$

The most basic model using this property is the Markov Chain. Two parameters define this model. The first is a transition probability distribution $A \in \mathbb{R}^{n \times n}$ where the probability of passing from state $i$ to state $j$ is $A_{ij}$ as seen in eq. (1.17). The second is an initial state $\pi \in \mathbb{R}^n$ where the

probability of beginning in state $i$ is $\pi_i$ as seen in eq. (1.18).

$$A_{ij} = P(q_t = S_i | q_{t-1} = S_j) \tag{1.17}$$

$$\pi_i = P(q_1 = S_i) \tag{1.18}$$

When the state sequence that maximizes the probability is not directly observable but can be determined through observations, the Hidden Markov Model (HMM) is used. These $m$ observations are defined as $O_1, ..., O_m$ and $x_t$ is the actual observation at time $t$. This model needs a third parameter which is the emission probability distribution $B = \{b_j(x_t)\}$ where the probability of observing $x_t$ at time $t$ given that we are in state $j$ is $b_j(x_t)$ as seen in eq. (1.19).

$$b_j(x_t) = P(x_t | q_t = S_j) \tag{1.19}$$

For convenience, the notation $\lambda = (n, m, A, B, \pi)$ will be use to indicate the set of parameters of the HMM. Also, let's define $X_t = x_1, ..., x_t$ and $Q_t = q_1, ..., q_t$. The following sections present the main algorithms used to solve problems using an HMM.

### 1.3.1 Forward-Backward algorithm

The Forward-Backward algorithm is useful to calculate variables used by other algorithms. Let's introduce the forward variable $\alpha_t(j)$ defined as

$$\alpha_t(j) = P(X_t, q_t = S_j | \lambda) \tag{1.20}$$

It is the probability of observing $X_t$ and being in state $j$ at time $t$. It can be calculated inductively with algorithm 1.2, which is illustrated in Figure 1.5.

In the same way, the backward variable $\beta_t(i)$ is defined as

$$\beta_t(i) = P(x_{t+1}, ..., x_T, q_t = S_i | \lambda) \tag{1.21}$$

It is the probability of observing $x_{t+1}, ..., x_T$ and being in state $i$ at time $t$. It can be calculated inductively with algorithm 1.3, which is illustrated in Figure 1.5.

**Problem 1** We can now solve the first problem which is to determine the probability of a sequence of observations given a model. More formally, we want to compute $P(X_T|\lambda)$. With algorithm 1.2, we can calculate it efficiently with eq. (1.22) for any given $t$ where $1 \leq t < T$.

$$P(X_T|\lambda) = \sum_{i=1}^{n} \alpha_t(i)\beta_t(i) = \sum_{i=1}^{n} \alpha_T(i) \tag{1.22}$$

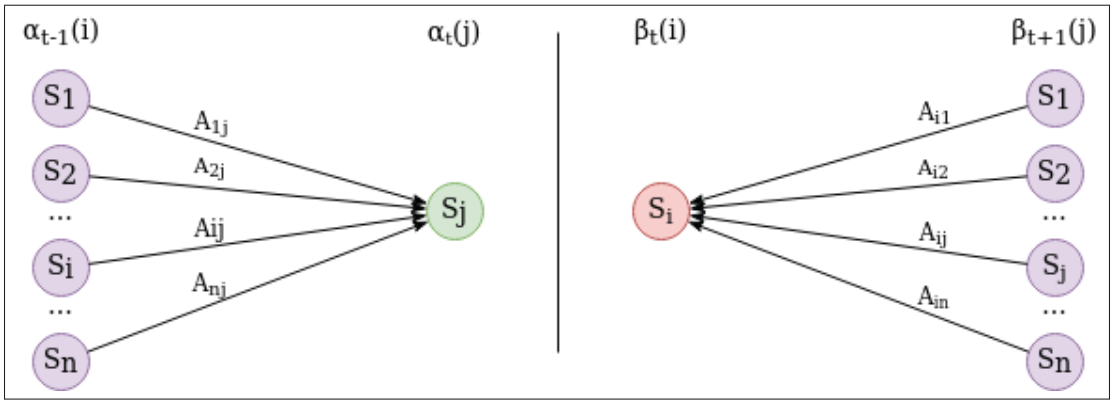


Figure 1.5 Illustration of the computation of (left) the forward variable $\alpha_t(j)$ and (right) the backward variable $\beta_t(i)$
Adapted from Rabiner (1989)

Algorithm 1.2 Forward algorithm

**Input:** Model parameters $\lambda$ and the list of observations $X_T$
**Output:** Forward variable $\alpha$

**for** $j \leftarrow 1$ *to* $n$ **do**
  $\alpha_1(j) \leftarrow \pi_j b_j(x_1)$; /* Initialization */
**for** $t \leftarrow 2$ *to* $T$ **do**
  **for** $j \leftarrow 1$ *to* $n$ **do**
    $\alpha_t(j) \leftarrow b_j(x_t) \sum_{i=1}^{n} \alpha_{t-1}(i)A_{ij}$; /* Induction */

Algorithm 1.3 Backward algorithm

**Input:** Model parameters $\lambda$ and the list of observations $X_T$
**Output:** Backward variable $\beta$

**for** $i \leftarrow 1$ *to* $n$ **do**
  $\beta_T(i) \leftarrow 1$; /* Initialization */
**for** $t \leftarrow T-1$ *to* $1$ **do**
  **for** $i \leftarrow 1$ *to* $n$ **do**
    $\beta_t(i) \leftarrow \sum_{j=1}^{n} \beta_{t+1}(j) A_{ij} b_j(x_{t+1})$; /* Induction */

### 1.3.2 Viterbi algorithm

The Viterbi algorithm is used to determine the most likely sequence of hidden states given a sequence of observation. It uses the variable $\delta_t(j)$ which is defined as

$$\delta_t(j) = \max_{Q_{t-1}} P(X_t, Q_{t-1}, q_t = S_j|\lambda) \tag{1.23}$$

It is the probability associated to the sequence of states $Q_t$ maximizing the probability of ending in state $j$ at time $t$ that accounts for the observations $X_t$ given the model $\lambda$. To keep track of the previous state which maximized eq. (1.23), we define the variable $\psi_t(j)$ corresponding to the state $j$ maximizing $\delta_t(j)$.

Algorithm 1.4 Viterbi algorithm

**Input:** Model parameters $\lambda$ and the list of observations $X_T$
**Output:** Highest probabilities $\delta$ and the path sequence $\psi$

**for** $j \leftarrow 1$ *to* $n$ **do**
  $\delta_1(j) \leftarrow \pi_j b_j(x_1)$; /* Initialization */
  $\psi(j) \leftarrow 0$;
**for** $t \leftarrow 2$ *to* $T$ **do**
  **for** $j \leftarrow 1$ *to* $n$ **do**
    $\delta_t(j) \leftarrow b_j(x_t) \max_i[\delta_{t-1}(i) A_{ij}]$; /* Induction */
    $\psi_t(j) \leftarrow \text{argmax}_i[\delta_{t-1}(i) A_{ij}]$;

**Problem 2** The second problem is exactly what algorithm 1.4 solves. In other words, it is to determine the sequence of states $Q_T$ which maximize $P(Q_T, X_T|\lambda)$ knowing $X_T$. Let's define $q_t^*$ corresponding to the index of the state maximizing $P(Q_T, X_T|\lambda)$ at time $t$. Then, we can calculate $q_t^*$ for all $t$ with a backtracking algorithm using eq. (1.24). The state at time $t$ correspond to $S_{q_t^*}$.

$$q_t^* = \begin{cases} \text{argmax}_j \delta_T(j) & \text{if } t = T \\ \psi_{t+1}(q_{t+1}^*) & \text{if } 1 \leq t < T \end{cases} \tag{1.24}$$

Finally, we can calculate the maximum probability $P^*$ of $P(Q_T, X_T|\lambda)$ with eq. (1.25).

$$P^* = \max_j \delta_T(j) \tag{1.25}$$

Viterbi algorithm works, but it does not scale well because the complexity is $O(Tn^2)$. But, there is two solutions for this problem. The first one is to use a Beam Search with a beam size of $B \ll n$. When calculating $\delta_t(j)$, we will use only the $B$ highest $\delta_{t-1}(i)$. With this method, the algorithm now has a complexity of $O(TnB)$, but it might not find the best answer.

### 1.3.3 Baum-Welch algorithm

Before exploring the Baum-Welch algorithm, let's define the variable $\gamma_t(i)$ which represents the probability of being in state $i$ at time $t$ given the observations $X_T$ and the model $\lambda$. More formally, it is defined as

$$\gamma_t(i) = P(q_t = S_i|X_T, \lambda) \tag{1.26}$$

It can be expressed using only the variables calculated with the Forward-Backward algorithm using eq. (1.22). It gives the following equation

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(X_T|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{n} \alpha_t(j)\beta_t(j)} \tag{1.27}$$

Then, $q_t^*$ for $1 \leq t \leq T$ can be calculated using $\gamma_t(i)$ with eq. (1.28).

$$q_t^* = \text{argmax}_i \gamma_t(i) \tag{1.28}$$

Since $\gamma_t(i)$ is a computationally expensive algorithm, we can use Viterbi's algorithm to approximate it.

Baum-Welch algorithm is an Expectation-Maximization (EM) algorithm. It uses $\gamma_t(i)$ calculated previously and another variable named $\xi_t(i, j)$ which is the probability of transitioning from state $i$ to state $j$ at time $t$ given the observations $X_T$ and the model $\lambda$. More formally, it is defined as

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j | X_T, \lambda) \tag{1.29}$$

With the forward and the backward variables, it can be calculated with eq. (1.30), which is illustrated in Figure 1.6.

$$\xi_t(i, j) = \frac{\alpha_t(i) A_{ij} b_j(X_{t+1}) \beta_{t+1}(j)}{P(X_T|\lambda)} = \frac{\alpha_t(i) A_{ij} b_j(X_{t+1}) \beta_{t+1}(j)}{\sum_{k=1}^{n} \sum_{l=1}^{n} \alpha_t(k) A_{kl} b_l(X_{t+1}) \beta_{t+1}(l)} \tag{1.30}$$

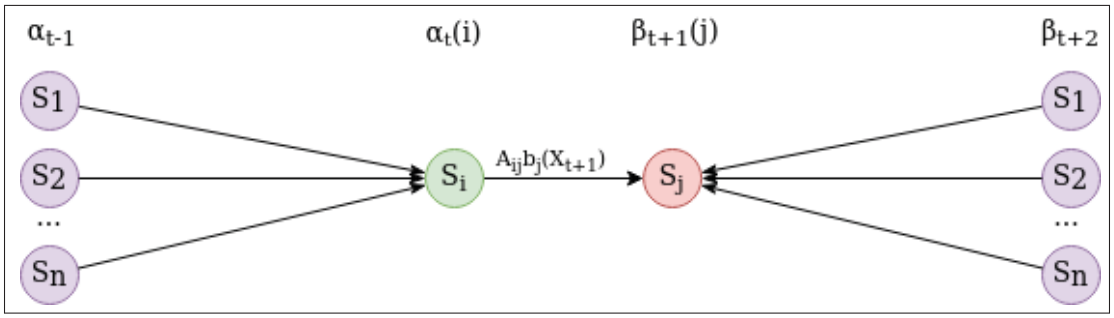


Figure 1.6 Illustration of the computation of $\xi_t(i, j)$
Adapted from Rabiner (1989)

**Problem 3** The third and last problem is exactly what algorithm 1.5 solves. In other words, it is to adjust the parameters of a model $\lambda$ to maximize $P(X_T|\lambda)$.

Algorithm 1.5 Baum-Welch algorithm

**Input:** Model parameters $\lambda$ and the list of observations $X_T$
**Output:** Updated model parameters $\lambda$

**while** $\lambda$ *not converged* **do**
  **for** $t \leftarrow 1$ *to* $T$ **do**
    **for** $i \leftarrow 1$ *to* $n$ **do**
      $\gamma_t(i) \leftarrow$ calculate with eq. (1.27);
      **for** $j \leftarrow 1$ *to* $n$ **do**
        $\xi_t(i,j) \leftarrow$ calculate with eq. (1.30);
  **for** $i \leftarrow 1$ *to* $n$ **do**
    $\bar{\pi}_i \leftarrow \gamma_1(i)$; /* Update $\pi_i$ */
    **for** $j \leftarrow 1$ *to* $n$ **do**
      $\bar{A}_{ij} \leftarrow \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$; /* Update $A_{ij}$ */
    **for** $t \leftarrow 1$ *to* $T$ **do**
      $\bar{b}_i(x_t) \leftarrow \frac{\sum_{u=1}^{T} \gamma_u(i) \text{ if } x_u = x_t}{\sum_{u=1}^{T} \gamma_u(i)}$; /* Update $b_i(x_t)$ */
  $\lambda \leftarrow \bar{\lambda}$

### 1.3.4 Continuous emission densities

Sound is a continuous signal. Therefore, the emission probability in HMMs should be a PDF instead of a discrete probability function. One popular approach is to redefine $b_j(x_t)$ as a GMM.

$$b_j(x_t) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(x_t; \mu_{jm}, \Sigma_{jm}) \tag{1.31}$$

With this new definition, we need to modify algorithm 1.5 to update the parameters of the GMMs. Let's define $\gamma_t(i,m)$ the probability of being in state $i$ at time $t$ while occupying the $m$-th mixture component. With equations (1.26) and (1.14), we can define $\gamma_t(i,m)$ more formally as

$$\gamma_t(i,m) = P(q_t = S_i | X_T, \lambda) P(z_t = C_m | x_t) \tag{1.32}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{n} \alpha_t(j)\beta_t(j)} \frac{c_{im}\mathcal{N}(x_t; \mu_{im}, \Sigma_{im})}{\sum_{k=1}^{M} c_{ik}\mathcal{N}(x_t; \mu_{ik}, \Sigma_{ik})} \tag{1.33}$$

With $\gamma_t(i, m)$, we can change line 16 of the Baum-Welch algorithm to update all three parameters of the GMMs using eq. (1.34). These equations are very similar to eq. (1.15) but for more details, see Xuan *et al.* (2001).

$$\begin{aligned}
\bar{c}_{jm} &= \frac{\sum_{t=1}^{T} \gamma_t(j, m)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \gamma_t(j, k)} \\
\bar{\mu}_{jm} &= \frac{\sum_{t=1}^{T} \gamma_t(j, m) x_t}{\sum_{t=1}^{T} \gamma_t(j, m)} \\
\bar{\Sigma}_{jm} &= \frac{\sum_{t=1}^{T} \gamma_t(j, m)(x_t - \mu_{jm})(x_t - \mu_{jm})^{\mathsf{T}}}{\sum_{t=1}^{T} \gamma_t(j, m)}
\end{aligned} \tag{1.34}$$

### 1.3.5 Weighted Finite-State Transducers

As defined in Mohri & Pereira (2002), Weighted Finite-State Transducers (WFSTs) are used to represent multiple elements composing an ASR model. It is an automata with an initial and a final state (Mohri *et al.*, 2008). Also, on each of its transitions, it has an input label, an output label and a weight. Figure 1.7 shows an example of a WFST mapping a phone sequence to words in its lexicon composed of the words *data* and *dew*.

Let's define the alphabet $\Sigma$ representing all the input symbols and the alphabet $\Omega$ representing all the output symbols with their associated weights. The transducer is a function $T : \Sigma \rightarrow \Omega$ that maps an input symbol to an output symbol. In our example, $\Sigma = \{d, ae, ey, uw, dx, t, ax\}$ and $\Omega = \{data, dew\}$. Everything else not part of $\Sigma$ is rejected.

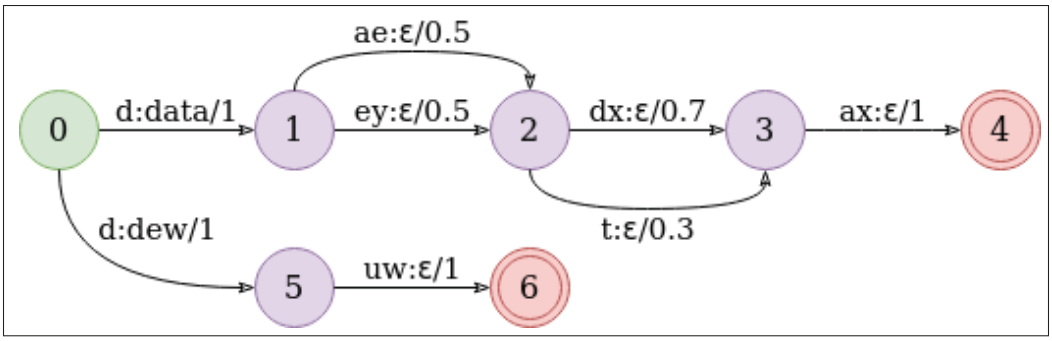


Figure 1.7 Example of a Weighted Finite-State Transducer
Taken from Mohri & Pereira (2002)

There is multiple ways to combine weights depending on what they represent. We denote this operation with $\otimes$. In our example, we added the weights of each transition because they are logarithms of probabilities. If the transducer represents an HMM, the weights could be probabilities that we multiply together. Also, in WFST, a transition may not consume an input or produce an output label. We use the symbol $\epsilon$ to represent this situation. In our example, we can see one transition from state 2 to state 4 that consumes the input *a* without producing any output.

#### 1.3.5.1 Operations on transducers

There are three important operations to understand how WFSTs are applied in speech recognition: composition, determinization and minimization. Given a transducer on alphabets $\Sigma_1$ and $\Omega_1$ and another transducer on alphabets $\Sigma_2$ and $\Omega_2$, the composition intersects both transducers to create a new one on alphabets $\Sigma_1$ and $\Omega_2$ (see Figure 1.8). This operation allows us to construct ASR models more easily.

The determinization converts a non-deterministic transducer into a deterministic one. In speech recognition, a deterministic transducer contains only one sequence of state per input, which reduce the time and space needed to process inputs. It uses the operation $\oplus$ to combine paths with the same input label (see Figure 1.9).

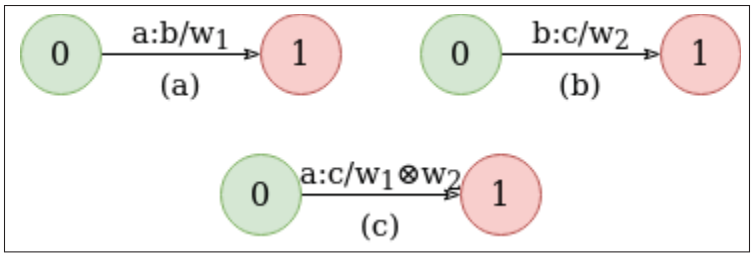


Figure 1.8 Composition of (a) ∘ (b)
into a new transducer (c)
Adapted from Mohri & Pereira (2002)

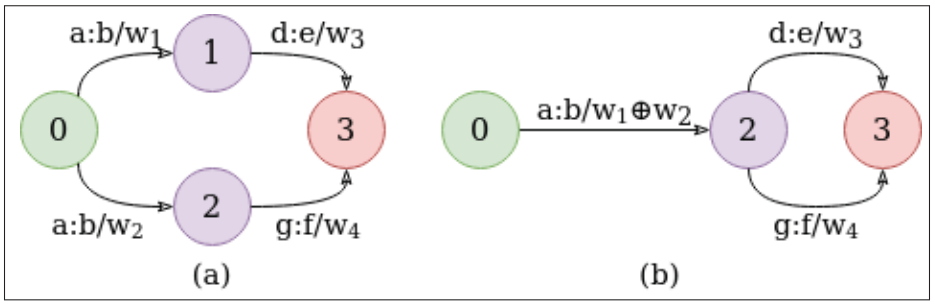


Figure 1.9 The determinization of (a)
gives the deterministic transducer (b)
Adapted from Mohri & Pereira (2002)

The minimization reduces the number of states in a transducer to its minimum while being equivalent (Mohri & Pereira, 2002). It is done in two phases. The first phase pushes weights among transitions towards the initial state as much as possible and the second phase minimizes the transducer (see Figure 1.10). Because the transducer contains a minimal amount of state, the computations will be more efficient.

## 1.4 Conditional Random Fields

A Conditional Random Field (CRF) is a special case of Markov Random Field (Wallach, 2004). Firstly, let's define the observation sequences $X = (x_1, \ldots, x_n)$ and the label sequences $Q = (q_1, \ldots, q_n)$. Then, let $G = (V, E)$ be a graph such that $Q = \{q_v | v \in V\}$. If $G$ is

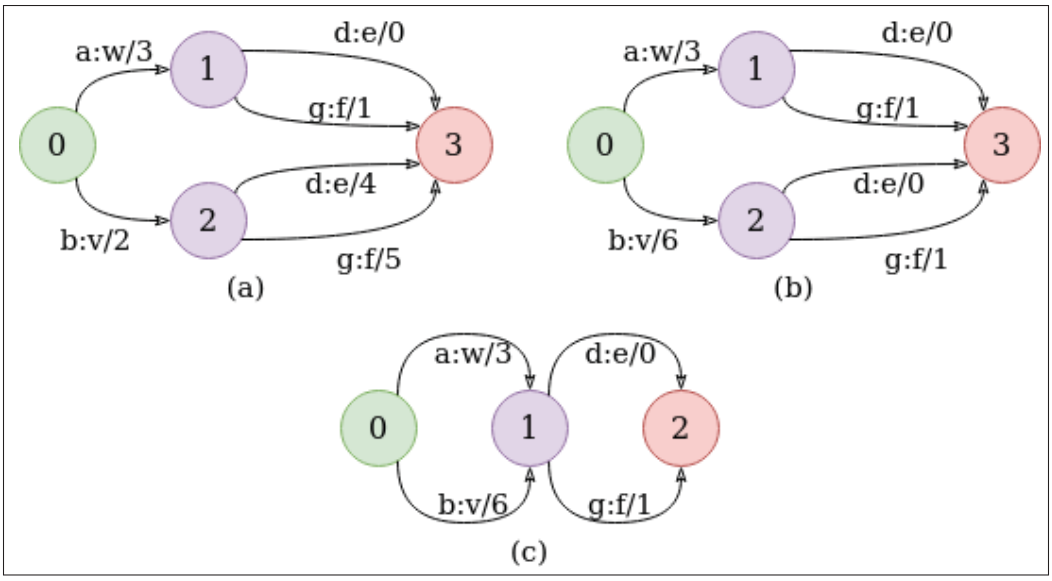


Figure 1.10 (a) Deterministic transducer (b) Equivalent transducer obtained by weight pushing (c) Equivalent minimal transducer Adapted from Mohri & Pereira (2002)

conditioned on $X$ and if each random variable $q_v$ obeys the Markov Property, then $G$ is a CRF (Lafferty *et al.*, 2001).

CRFs are similar to HMMs, but they have some differences. Firstly, HMMs are generative models whereas CRFs are discriminative models. In other words, HMMs learn the distribution of labels by estimating $P(X|Q)$ and $P(Q)$ whereas CRFs learn the boundary between labels by estimating $P(Q|X)$. As we can see in Figure 1.11, another difference is that HMMs only capture dependencies between a state and its corresponding observation whereas CRFs capture dependencies between a state and all the observations.

The conditional probability of a linear chain CRF is defined as

$$P(Q|X,\lambda) = \frac{1}{Z(X)} \exp \sum_{t=1}^{n} \sum_{i=1}^{f} \lambda_i F_i(q_{t-1}, q_t, X, t) \tag{1.35}$$

where $Z(X)$ is a normalization factor, $F() \in \mathbb{R}$ are the $f$ feature functions and $\lambda_i$ are the learned parameters of the model. Each feature function looks at a pair of adjacent states $q_{t-1}$ and $q_t$, the observation sequence $X$ and the position in the sequence $t$.

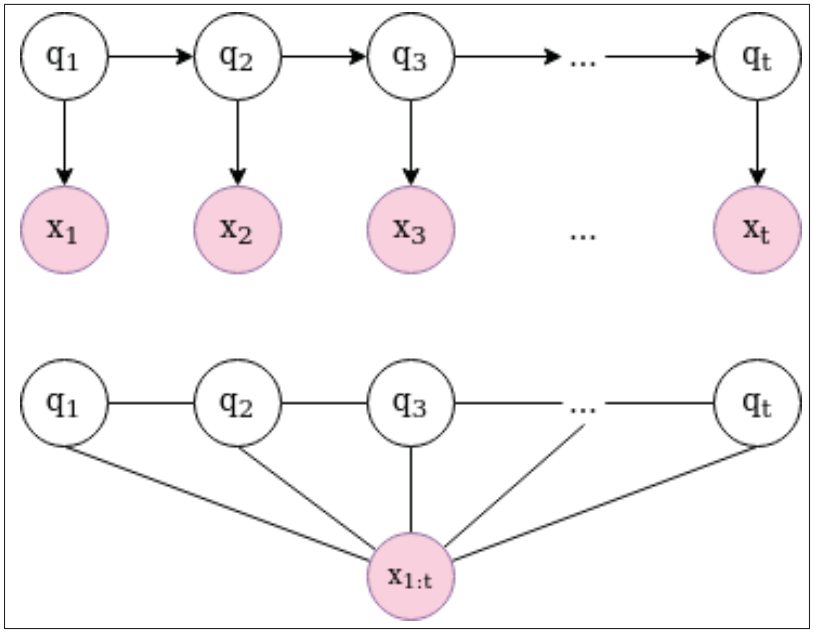


Figure 1.11 Comparaison between (top) a HMM and (bottom) a CRF

### 1.4.1 Objective function

CRFs are trained using an iterative scaling algorithm maximizing the log-likelihood objective function. Given the training sequences $\{(X_1, Q_1), \ldots, (X_m, Q_m)\}$, we can define the following function

$$\mathcal{F}_i(Q, X) = \sum_{t=1}^{n} F_i(q_{t-1}, q_t, X, t) \tag{1.36}$$

And the objective function $O(\lambda)$ is

$$O(\lambda) = \sum_{j=1}^{m} \log P(Q_j|X_j, \lambda) = \sum_{j=1}^{m} \left[ \sum_{i=1}^{f} \lambda_i \mathcal{F}_i(Q_j, X_j) - \log Z(X_j) \right] \tag{1.37}$$

The partial derivative of $O(\lambda)$ with respect to $\lambda_i$ is

$$\frac{\partial O(\lambda)}{\partial \lambda_i} = E_{\tilde{p}(Q_{1:m}, X_{1:m})}[F_i(Q_{1:m}, X_{1:m})] - \sum_{j=1}^{m} E_{p(Q_{1:m}|X_j,\lambda)}[F_i(Q_{1:m}, X_j)] \tag{1.38}$$

where $E_{\tilde{p}(Q_{1:m}, X_{1:m})}$ is the empirical distribution over the training sequences. Because there is too many possible label sequences to compute $E_{p(Q_{1:m}|X_j,\lambda)}$, we use an algorithm similar to the Baum-Welch algorithm (see Section 1.3.3).

### 1.4.2 Feature functions

In this section, we present basic examples of feature function. Let's define our first example $F_1$ as the probability of passing from state $i$ to state $j$.

$$F_1(q_{t-1}, q_t, X, t) = \begin{cases} 1 & \text{if } q_{t-1} = S_i \text{ and } q_t = S_j \\ 0 & \text{otherwise} \end{cases} \tag{1.39}$$

Multiplied with parameter $\lambda_1$, it is equivalent to the transition probability distribution $A_{ij}$ in HMMs. For our second example, we define $F_2$ as the probability of observing $O_i$ at time $t$ given that we are in state $j$.

$$F_2(q_{t-1}, q_t, X, t) = \begin{cases} 1 & \text{if } x_t = O_i \text{ and } q_t = S_j \\ 0 & \text{otherwise} \end{cases} \tag{1.40}$$

Similar to the previous example, if we multiply $f_2$ with $\lambda_2$, it is equivalent to the emission probability distribution $b_j(x_t)$ in HMMs. For our final example, we show that CRFs can learn more things than HMMs. Let's define $F_3$ as the probability of observing $O_i$ at time $t_1$ given that we are in state $j$ at time $t$.

$$F_3(q_{t-1}, q_t, X, t) = \begin{cases} 1 & \text{if } x_{t-1} = O_i \text{ and } q_t = S_j \\ 0 & \text{otherwise} \end{cases} \tag{1.41}$$

This feature function is impossible for HMMs to model because they only capture dependencies between a state and its corresponding observation. Also, it is important to note that feature functions can return any real value. Thus, there is an infinite amount of possible feature functions.

## 1.5 Long Short-Term Memory networks

A Recurrent Neural Network (RNN) is a type of neural network used to process sequential data. It takes as input a sequence of $n$ vectors $x_1, \ldots, x_n$ and outputs another sequence $h_1, \ldots, h_n$. We say that RNNs have a memory because $h_t$ depends on $x_t$ and on $h_{t-1}$. Let's define $\hat{x_t} = [x_t, h_{t-1}, 1]$. Now, we can formally define $h_t$ as followed

$$h_t = \tanh(W\hat{x_t}) \tag{1.42}$$

where $W$ and $b$ are respectively the weights and the bias learned during training. Figure 1.12 illustrates an RNN. One major problem with RNNs is that it is very hard to learn long-term dependencies due to a vanishing gradients problem (Bengio *et al.*, 1994).

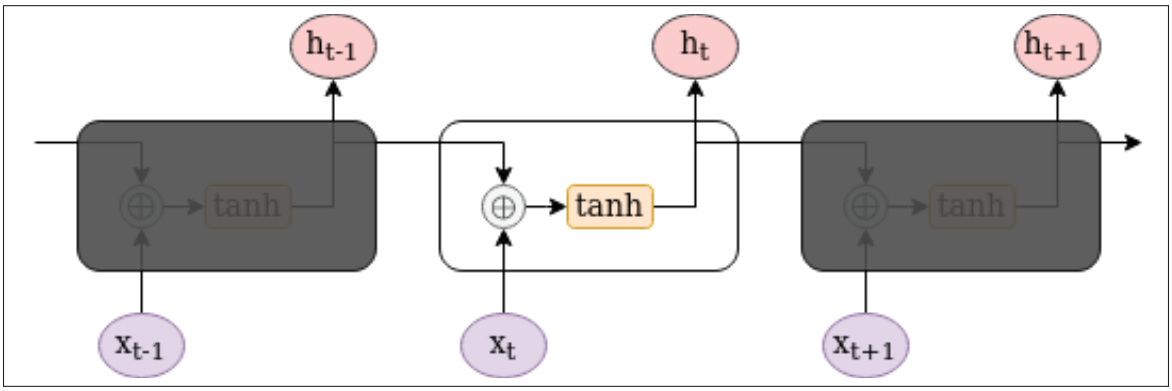


Figure 1.12 Architecture of a Recurrent Neural Network
Adapted from Olah (2015)

Long Short-Term Memory (LSTM) networks, developed by Hochreiter & Schmidhuber (1997), have been designed to solve this long-range dependency problem. As we can see in Figure 1.13, a LSTM cell is a little bit more complex than a RNN cell. It uses three gates to control which

information to keep and which information to forget. Each gate is a simple linear transformation followed by a sigmoid function. The result, which is between 0 and 1, acts as a valve either allowing the data to flow or blocking it.

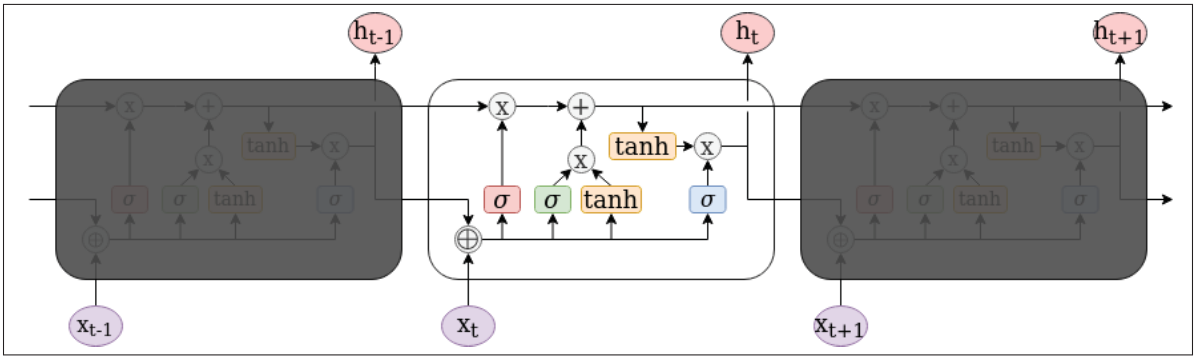


Figure 1.13 Architecture of a Long Short-Term Memory Network
Adapted from Olah (2015)

The first gate, named the Forget gate $f$, is represented by the red sigma symbol in Figure 1.13. Intuitively, it controls how much information from the previous cell state $c_{t-1}$ to keep. More formally, it is defined as

$$f_t = \sigma(W_f \hat{x}_t) \tag{1.43}$$

The second gate, named the Input gate $i$, is represented by the green sigma symbol in Figure 1.13. Near it, we can see a tanh function. It is the same function as eq. (1.42), but we refer to it as the internal cell state $\bar{c}_t$. Intuitively, this gate controls how much $\bar{c}_t$ will affect the current cell state $c_t$. More formally, we can define

$$i_t = \sigma(W_i \hat{x}_t) \tag{1.44}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \tag{1.45}$$

The last gate, named the Output gate $g$, is represented by the blue sigma symbol in Figure 1.13. Intuitively, it controls how much information from $c_t$ is needed to calculate the cell output $h_t$.

More formally, we can define these variables as

$$o_t = \sigma(W_o \hat{x}_t) \tag{1.46}$$

$$h_t = o_t \odot \tanh(c_t) \tag{1.47}$$

### 1.5.1 Bidirectional Long Short-Term Memory

With a LSTM, the $t^{\text{th}}$ cell only use its left context. But, when we do some tasks like named entity recognition, we also have access to the right context because the whole sequence is available. In this case, using one forward LSTM $\overrightarrow{h}$ (left context) and one backward LSTM $\overleftarrow{h}$ (right context) can improve performance. We can then concatenate both outputs to obtain the left-right context representation $r_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$. The resulting model is referred to as a Bidirectional LSTM (BiLSTM) as seen in Figure 1.14 (Graves & Schmidhuber, 2005).

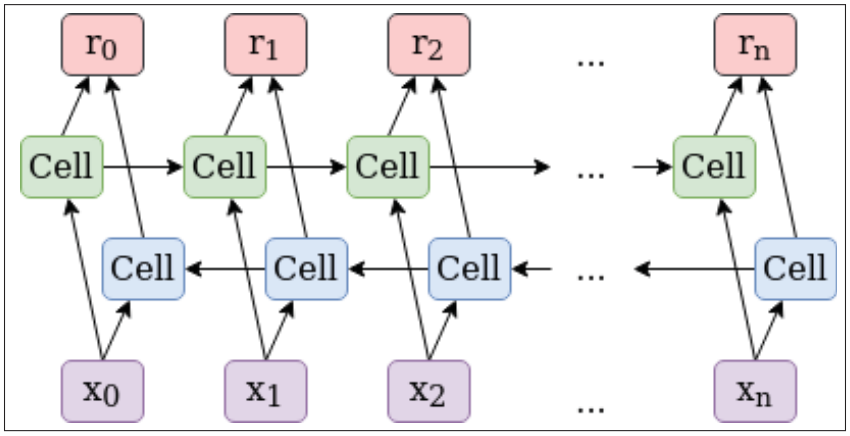


Figure 1.14 Bidirectional Long Short-Term Memory Network

## 1.6 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of deep neural network (LeCun *et al.*, 1990). Since CNNs are not used in the experiments, we do not do an in-depth review. For more details, we refer you to O'Shea & Nash (2015); Albawi *et al.* (2017). In fact, we do not explore them

since, following our preliminary experiments, we found that CNNs do not perform as well as the other models we explored.

That being said, a CNN is made of three types of layer: convolutional layers, pooling layers and fully connected layers. Figure 1.15 shows an example of a simple CNN[3] with a two dimensional input.

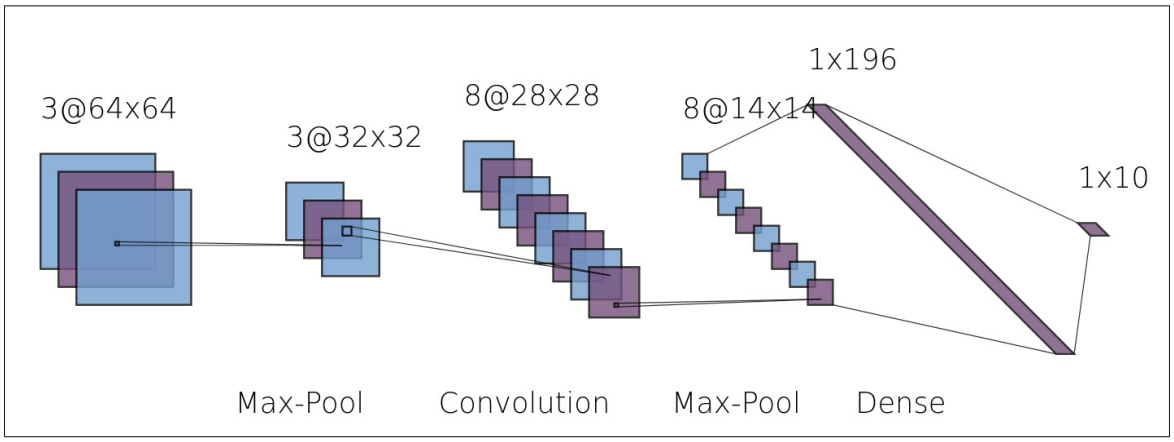


Figure 1.15 Convolutional Neural Network example

The convolutional layer uses a kernel. A kernel is a small matrix compared to the image with learnable parameters and it is used to calculate activation maps. If we have an input of size $W \times W \times D$, the activation map size $A$ is equal to

$$A = \frac{W - K + 2P}{S} + 1 \tag{1.48}$$

where $K$ is the kernel size, $P$ is the amount of padding and $S$ is the stride. Then, the pooling layer reduces the size of the activation maps. One of the most common type of pooling is max-pooling with a kernel size of $2 \times 2$ and a stride of 2. If we have an activation map of size $A \times A \times D$, the output size $O$ is equal to

$$O = \frac{A - K}{S} + 1 \tag{1.49}$$

[3] Created with http://alexlenail.me/NN-SVG/LeNet.html

Finally, the fully connected layer is a standard neural network where the input size is the same as the pooling layer size and the output size depends on the problem.

## 1.7 Transformers

The Transformer (Vaswani *et al.*, 2017) is a neural sequence transduction model. Unlike previous model architectures, it rely on attention rather than recurrence. There is two main motivations behind this new architecture. The first motivation is to reduce the path length to learn long-range dependencies and the second motivation is to improve computation by allowing parallelization.

Let's compare recurrent, convolutional and Transformer models. Firstly, recurrent models have a path length growing linearly because it processes tokens sequentially and it does not allow parallelization. Then, convolutional models improved these aspects. It has a path length growing logarithmically and it allows parallelization. Finally, Transformers enhance this by having a constant path length and by allowing parallelization during training.

Transformers have an encoder-decoder structure (see Figure 1.16). Both the encoder and decoder are composed of a stack of $N$ identical layers. Also, between each sub-layers, there is a residual connection (He *et al.*, 2016) and a layer normalization (Ba *et al.*, 2016). The encoder has two sub-layers: a multi-head self-attention mechanism and a fully connected Feed Forward Neural network (FFN). Furthermore, the decoder has three sub-layers: a masked multi-head self-attention mechanism, a multi-head cross-attention mechanism and a fully connected FFN. Finally, a linear and softmax layer is applied to transform the decoder output into prediction probabilities.

The encoder uses learned embedding to convert input tokens to vectors of dimension $D$ and the decoder uses its output of the previous step as input. The next sections will describe what is the positional encoding added to the input embedding and how multi-head attention works.

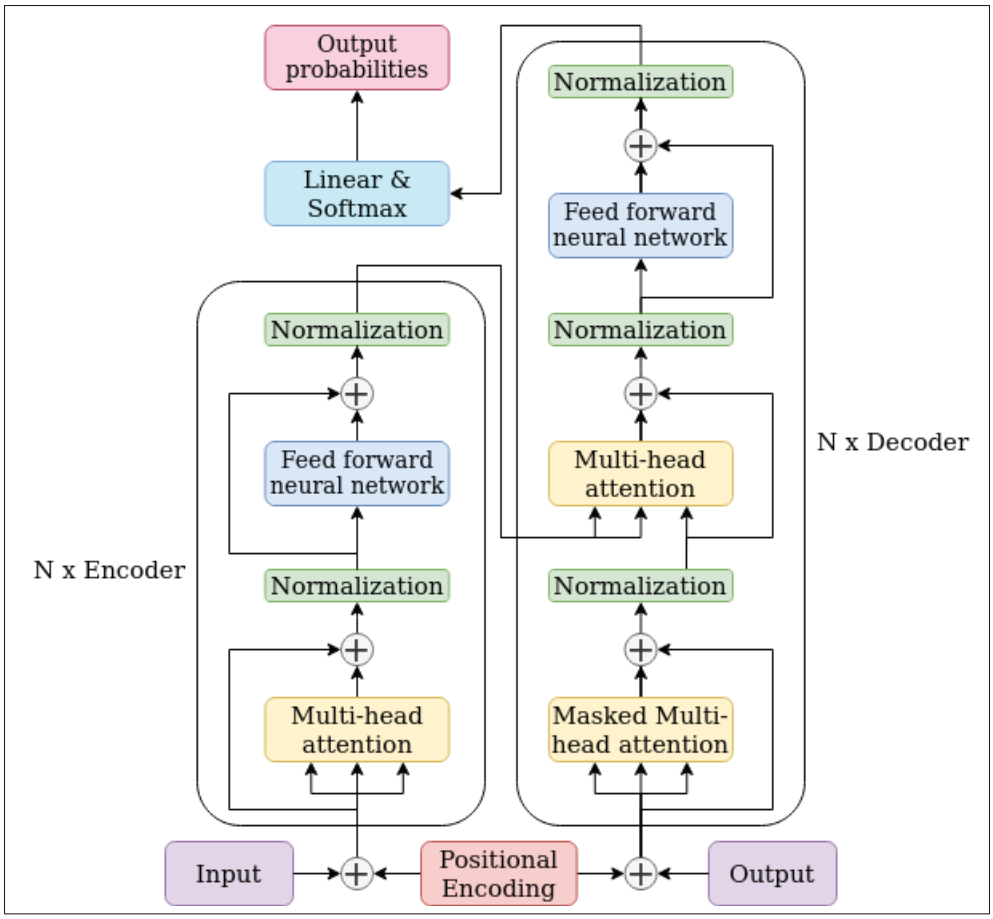


Figure 1.16 Illustration of the architecture of a Transformer
Taken from Vaswani *et al.* (2017)

### 1.7.1 Positional encoding

Since transformers contain no recurrence and no convolution, the model does not have any sense of position for each token. The authors of Vaswani *et al.* (2017) propose to add positional encoding (PE) to the input embedding (IE).

The positional encoding vector of the $p^{\text{th}}$ token $PE_p \in \mathbb{R}^D$ is equal to

$$PE_{pi} = \begin{cases} \sin(p/10^{4i/D}) & \text{if } i \text{ is even} \\ \cos(p/10^{4(i-1)/D}) & \text{if } i \text{ is odd} \end{cases} \tag{1.50}$$

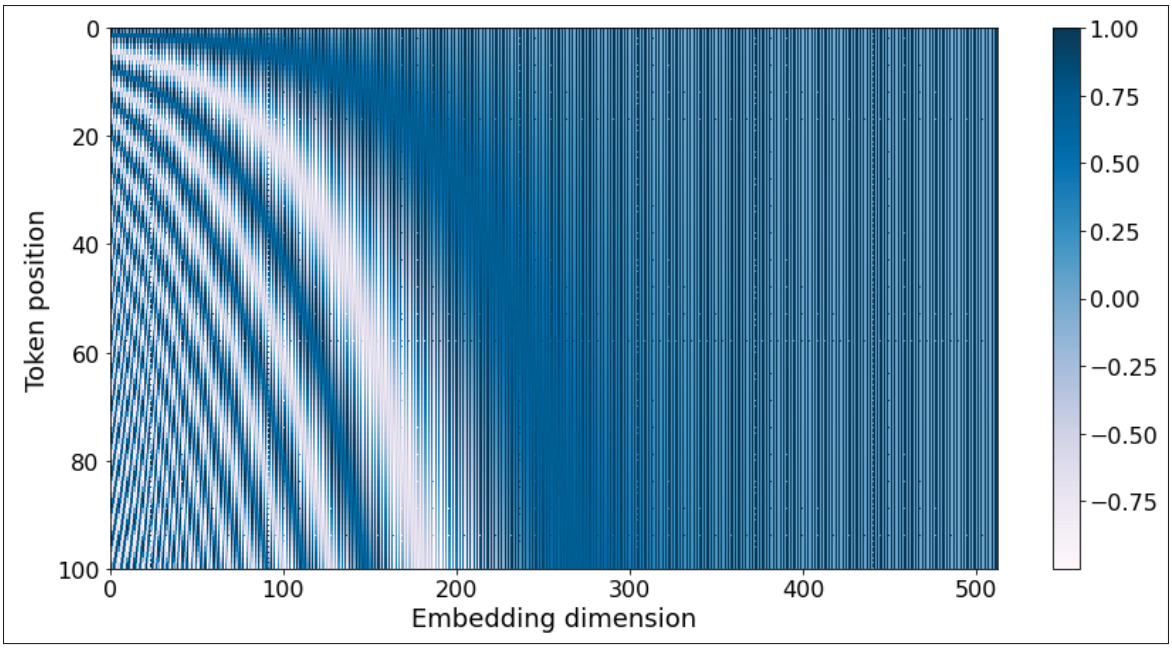


Figure 1.17 Visual representation of the positional encoding of 100 tokens with D=512

where $1 \leq i \leq D$. Figure 1.17 shows the result of this function on 100 tokens with a dimension of 512. This function allows the model to learn to attend by relative position instead of absolute position. In other words, the model does not have to learn relationship between two words at every position. They hypothesized this is because for any offset $k$, $PE_{p+k}$ can be represented as a linear function of $PE_p$ (Yan *et al.*, 2019). More formally, there is a transformation matrix $M$ where $M$ does not depend on $p$ for which eq. (1.51) holds.

$$PE_{p+k} = MPE_p \tag{1.51}$$

Finally, the positional encoding has the same dimensionality as the input embedding. Thus, we can sum them together to add the positional information to the input. More formally, it is defined as

$$x_p = PE_p + IE_p \tag{1.52}$$

### 1.7.2 Attention mechanism

The intuition behind the attention mechanism is the following. A set of queries $Q$ are compared to a set of keys $K$ to determine their compatibility. Each values $V$ is paired with a key. The greater the compatibility between a query and a key, the greater influence the corresponding value will have on the output of the attention mechanism. Each $P$ queries and each $P$ keys are vector of dimension $d_k$ and each values are vector of dimension $d_v$. To determine the compatibility $c \in \mathbb{R}^{P \times P}$ between $Q$ and $K$, we compute the dot products between them.

$$c(Q, K) = QK^\top \tag{1.53}$$

Then, we normalize $c(Q, K)$ to obtain non-negative weights summing to one by applying a softmax function. Also, the authors of Vaswani *et al.* (2017) suspect that when $d_k$ grows, the dot products grow large in magnitude causing vanishing gradient when passed to the softmax function. By assuming that $Q$ and $K$ are normalized independent random variables, their dot product has mean 0 and standard deviation $\sqrt{d_k}$. Thus, we can standardize the dot products by dividing by $\sqrt{d_k}$, giving us the scaled dot product $s(Q, K) \in \mathbb{R}^{P \times P}$.

$$s(Q, K) = \text{softmax}(\frac{c(Q, K)}{\sqrt{d_k}}) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}}) \tag{1.54}$$

Finally, we use the weights $s(Q, K)$ to compute a linear combination of $V$ to obtain the output of the attention mechanism $a(Q, K, V) \in \mathbb{R}^{P \times d_v}$ as seen in eq. (1.55). In Transformers, there is three types of attention mechanism. We will explain them intuitively using a French-English neural machine translation problem as an example.

$$a(Q, K, V) = s(Q, K)V = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V \tag{1.55}$$

**Self-attention** is the attention mechanism used in the encoder. The question here is "How relevant is one word in the French sentence to the other words in the same sentence?". All variables Q, K and V are calculated using the input embedding.

**Masked self-attention** is the first attention mechanism used in the decoder. The question here is "How relevant is one word in the English sentence to the previous words in the same sentence?". All variables Q, K and V are calculated using the output embedding. Since the decoder is autoregressive, the prediction at position $t$ should only depends on prior predictions. To ensure this, we use a mask $M \in \mathbb{R}^{P\times P}$ which is an upper triangular matrix with $-\infty$ above the main diagonal. The reason for this mask is because $\text{softmax}(-\infty) = 0$, leaving zero compatibility scores for future predictions. Masked self-attention $ma(Q, K, V)$ is defined as

$$ma(Q, K, V) = \text{softmax}(M + \frac{QK^\top}{\sqrt{d_k}})V \tag{1.56}$$

**Cross-attention** is the second attention mechanism used in the decoder. The question here is "How relevant is one word from the French sentence to the other words in the English sentence?". The variables $K$ and $V$ are the output of the encoder and $Q$ is the output of the previous layer of the decoder.

### 1.7.3 Multi-head attention

The authors find it beneficial to use multiple attention heads to learn multiple representations simultaneously. A multi-head attention uses $h$ heads and each head has three learned projection matrices as seen in eq. (1.57).

$$head_i(Q, K, V) = ma(QW_i^Q, KW_i^K, VW_i^V) \tag{1.57}$$

where $W_i^Q \in \mathbb{R}^{D\times d_k}$, $W_i^K \in \mathbb{R}^{D\times d_k}$ and $W_i^V \in \mathbb{R}^{D\times d_v}$. The multi-head attention ($mh$) concatenate all $h$ heads and linearly project the learned representation back to the original embedding dimensionality.

$$mh(Q, K, V) = concat(head_1, \ldots, head_h)W^O \tag{1.58}$$

where $W^O \in \mathbb{R}^{hd_v\times D}$. Also, to simplify calculations, the authors chose use $d_k = d_v = D/h$, thus making $W^O$ a square matrix.

## 1.8 Summary

In summary, we presented different algorithms used by the pipeline components. Recall that the first component is the FA and the second component is the NER. In the next section, we will explore the literature to determine the best models for our components. These models are based on the algorithms we have seen in this section.

# CHAPTER 2

# LITERATURE REVIEW

This section first goes through the literature to find the existing algorithms that constitute the pipeline. Firstly, we analyze different ASR toolkits which are used by FA algorithms. Then, we analyze existing French FA algorithms. Finally, we examine different NER model architectures.

In a second step, we explore another way of achieving our objective using end-to-end models. Unlike the pipeline with multiple components, these models optimize the ASR and NER models jointly.

## 2.1 Speech recognition toolkits

ASR toolkits are used to transform speech into text. FA models are based on these toolkits to align a text with its corresponding audio. Therefore, it was mandatory that the toolkits we explored support French.

**Hidden Markov Model Toolkit (HTK)** It is a tool to manipulate HMMs (Young *et al.*, 2015). It can be used to train acoustic models (AM) and language models (LM). To use these models, we need a decoder. HTK has its own decoder named HVite, but there is also a more popular one called Julius (Lee & Kawahara, 2009). The decoding algorithm is a two-pass forward-backward search (Lee *et al.*, 1998). The first pass consists of a Viterbi beam search generating a word trellis index. But, for faster decoding, the LM uses reduced constraint. For example, instead of using N-gram probabilities, the LM will use only 2-gram probabilities. The second pass performs a stack decoding search using the output of the first pass as an heuristic. Also, the second pass will use the LM without any constraint. Figure 2.1 shows the algorithm pipeline.

**Kaldi** Is another popular toolkit developed by Povey *et al.* (2011). It uses four WFSTs to represent every part of the ASR model (Mohri *et al.*, 2008). The resulting graph, called HCLG,

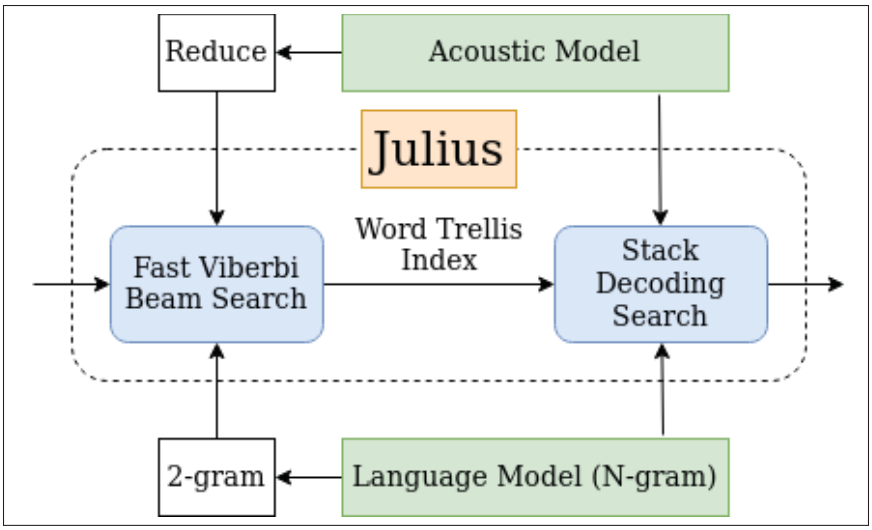


Figure 2.1 Julius decoding algorithm using an AM and a LM

is equal to

$$\text{HCLG} = H \circ C \circ L \circ G \tag{2.1}$$

where $\circ$ is the composition operation. The first letter $G$ is the grammar transducer (or the language model). It represents all possible combination of words and output the probability of a word sequence.

The second letter $L$ is the pronunciation lexicon transducer. It can be seen as a dictionary mapping a sequence of phones to words. Thus, we have the transducer

$$L \circ G$$

mapping phones to words restricted to the grammar $G$. The next letter $C$ is the context-dependency transducer. It maps triphones, which are context-dependent, to context-independent phones. Also, it guarantees that the phoneme sequence is a valid triphone sequence. So, we now have the transducer

$$C \circ L \circ G$$

mapping triphones to words restricted to the grammar $G$. The last letter $H$ is the HMM. It maps a sequence of observations to triphones. Each triphone is modeled with a 3-state HMM and the transducer $H$ is the union of all these HMMs (Cardinal, 2013). Finally, we have the transducer

$$H \circ C \circ L \circ G$$

mapping a sequence of observations to a word sequence restricted by the grammar $G$. Let *min* be the minimization operation and *det* be the determinization operation, we can optimize HCLG with eq. (2.2).

$$\text{HCLG} = min(det(H \circ det(C \circ det(L \circ G)))) \tag{2.2}$$

To conclude, we need to be careful in which context we use Julius, because it mainly uses HTK models and they are under a restrictive license. Other than that, both models seem promising regarding their performance due to their two pass decoding algorithm. Also, both architectures are very modular. It means that it will be easy to integrate our own AM or LM and it will be easy to add elements in a pretrained dictionary.

## 2.2 Forced alignment algorithms

There are multiple FA algorithms. A lot of them are based on HTK or Kaldi and some of them, like FAVE (Rosenfelder *et al.*, 2014), only work on English. There is also web API like MAUS (Strunk *et al.*, 2014), but because our final goal is to use recordings with sensible information, we did not explore this path. In this section, we explore the most recent algorithms supporting French.

**Montreal Forced Aligner (MFA)** It is an algorithm developed by McAuliffe *et al.* (2017) and based on Kaldi. It uses a HMM/GMM architecture with three training phases for the acousting model. The first phase consists of training monophone GMMs to generate basic alignment. Then, the second phase uses these models to train triphone GMMs which take into account the context of a phoneme. Also, it uses a decision tree to cluster triphones because there

is too many of them and it will probably not have enough data to see all of them. Finally, based on the triphone models, it uses Constrained Maximum Likelihood Linear Regression (CMLLR) to calculate speaker-adapted GMMs (Gales, 1998; Povey & Saon, 2006).

Table 2.1 shows the accuracy at different tolerances between gold standard annotations and the predictions made by the model on the Buckeye corpus (Pitt *et al.*, 2007). Note that this corpus is in English.

Table 2.1 Accuracy on the Buckeye corpus at different tolerances (ms) for absolute differences between gold standard annotations and predictions

| | ≤10ms | ≤25ms | ≤50ms | ≤100ms |
|---|---|---|---|---|
| Word boundaries | 0.33 | 0.68 | 0.88 | 0.97 |
| Phone boundaries | 0.41 | 0.77 | 0.93 | 0.97 |

**SPeech Phonetization Alignment and Syllabification (SPPAS)** It is the second algorithm developed by Bigi (2015) and based on Julius. Training is done in three steps. Firstly, it phonetizes the text with a language-independent algorithm (Bigi, 2016). This phonetic representation will be use by the following steps. Then, it creates a flat start monophone model. In other words, it will initialize the model by calculating the average of all the features in the corpus and then train the model with the Viterbi algorithm. Finally, it creates a tied-state triphone model which is a decision tree of triphone HMMs. It uses a decision tree for the same reason as MFA: to avoid sparsity.

The authors also worked on a Quebec French model (Lancien *et al.*, 2020). Since a future objective is to apply the developed pipeline on audio recording with Quebecker clients, we will not need to train our own model. In another paper (Bigi & Meunier, 2018), the authors evaluated SPPAS on various spontaneous speech corpus in French. Their final results are shown in Table 2.2.

To conclude, one of the advantage of MFA is that it can take audio file with a sampling as low as 8 kHz. SPPAS cannot do that by default because it has been trained on audio with a sampling of

Table 2.2 Accuracy on the spontaneous French at different tolerances (ms) for absolute differences between gold standard annotations and predictions

| | ≤20 | ≤30 | ≤40 | ≤50 | ≤80 |
|---|---|---|---|---|---|
| Word boundaries | 0.861 | 0.939 | 0.965 | 0.976 | 0.992 |

16 kHz. Because 8 kHz is the minimum acceptable in the industry, it means MFA can align every audio files.

## 2.3 Named entity recognition models

In this section, we explore multiple NER model architectures. They have been split in two categories : LSTM-based and Transformer-based. We start with LSTM-based models and then, we explore the newer Transformer-based models.

### 2.3.1 LSTM-based models

Before Transformers, LSTMs were one of the best model for NLP tasks such as NER because this type of neural network could learn long-term dependencies. There is a lot of different architectures based on LSTMs.

Huang *et al.* (2015) did some experiments using simple architectures. For example, they used a forward LSTM, a BiLSTM and a CRF. In their paper, they also created a more complex network by combining a BiLSTM with a CRF layer on top. The BiLSTM allow the model to use past and future input features and the CRF layer allow the model to use sentence level tag information. As we can see in Figure 2.2, the CRF uses the BiLSTM left-right context representations as input.

Another architecture introduced by Lample *et al.* (2016) is a BiLSTM-CRF network combining both word and character-level context (see Figure 2.3). Unlike previous architectures which only use word-level context, this new architecture have proved to have good performance with little domain specific knowledge (Yadav & Bethard, 2019). Because there are not many resources

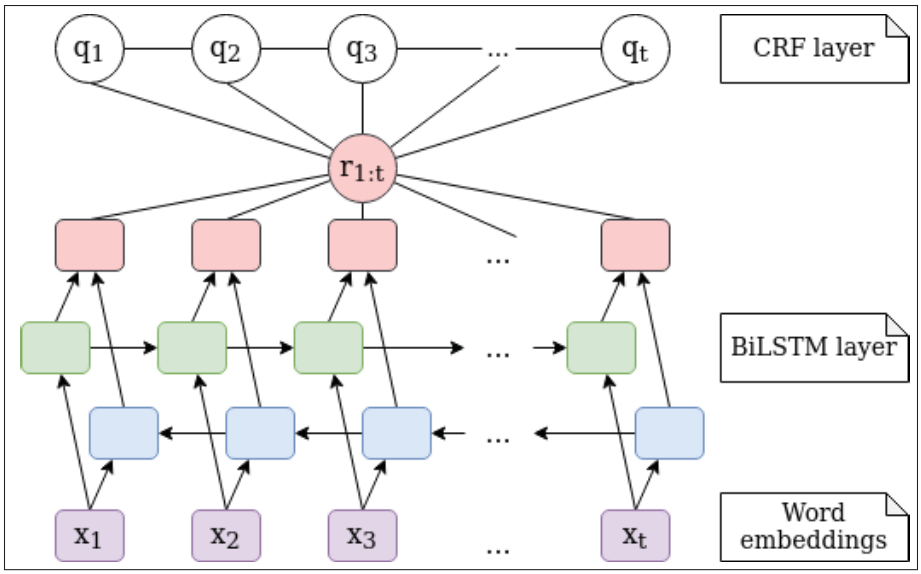


Figure 2.2 BiLSTM-CRF with word embeddings model architecture
Taken from Huang *et al.* (2015)

in French, this aspect is very important because it allows us to use a model trained on a large amount of data in another context which may not be exactly the same domain.

Also, Lample *et al.* (2016) introduce a new model inspired by transition-based dependency parsing with LSTMs (Dyer *et al.*, 2015), also called Stack-LSTM. The idea is to augment the LSTM with a stack pointer. The current location of the pointer determines which cell in the LSTM provides $c_{t-1}$ and $h_{t-1}$ to the current cell. Each model uses one buffer and two stacks. The buffer contains all unprocessed tokens, the first stack represents the scratch space and the second stack, called output stack, contains the completed segments. There is three transition operations on these models :

- **Shift** transition moves a token from the buffer to the stack.
- **Reduce** transition pops all elements from the stack and combine them to form a segment. This segment is labeled and copied to the output stack.
- **Out** transition moves a token from the buffer directly into the output stack.

Figure 2.4 shows an example of a transition sequence with a Stack-LSTM. The Stack-LSTM output the sequence of transition operations needed given the current buffer, the stack, the output

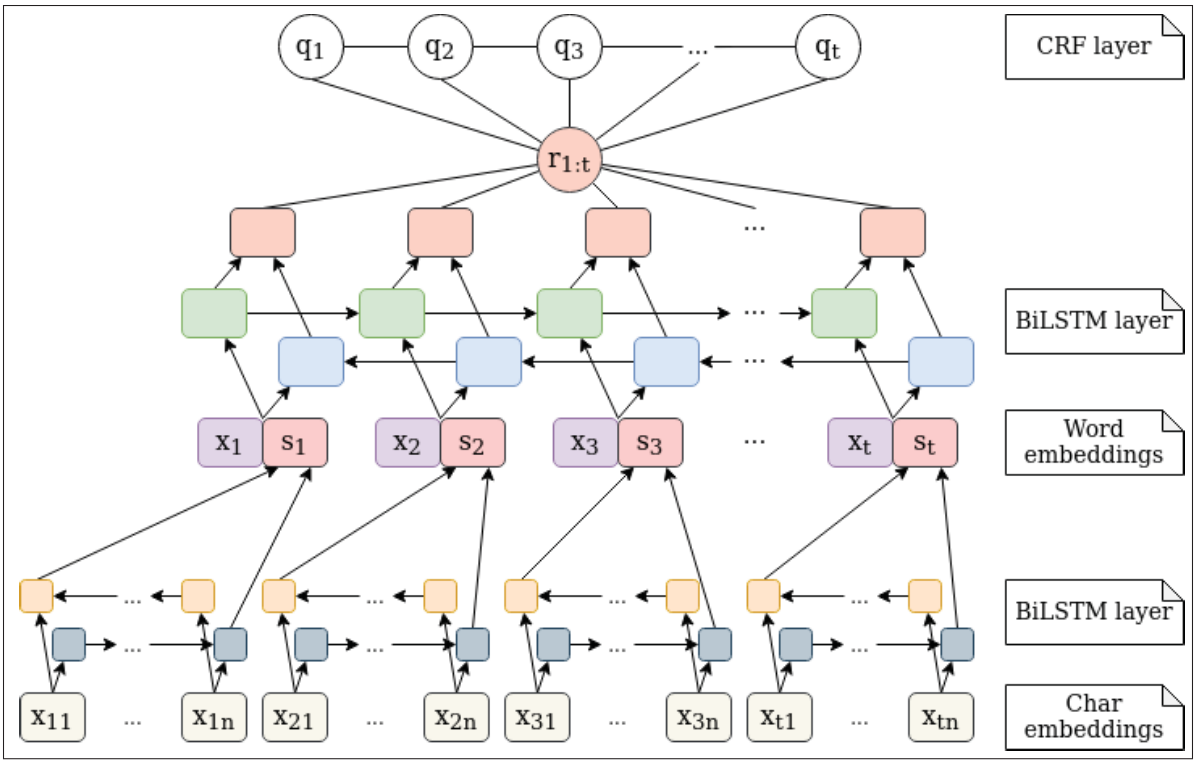


Figure 2.3 BiLSTM-CRF with word and character embeddings model architecture
Taken from Lample *et al.* (2016)

and the history of operations. Finally, it is worth mentioning that each segment in the output stack is represented with a single embedding vector computed by a BiLSTM.

Instead of using a BiLSTM to model character-level information, Chiu & Nichols (2016) used a simple CNN architecture with one convolution layer followed by a max pooling. Then, they concatenate the CNN outputs and the word embeddings before feeding them to the BiLSTM. The resulting model is a BiLSTM-CNN and Figure 2.5 visually summarizes its architecture.

Table 2.3 shows the F1 score of all model on CoNLL-2003 test set. All of these models was trained with and without additional data. We include as additional data everything that is not part of the CoNLL-2003 dataset like pretrained embeddings, dictionnaries, gazetteers, lexicons, etc.

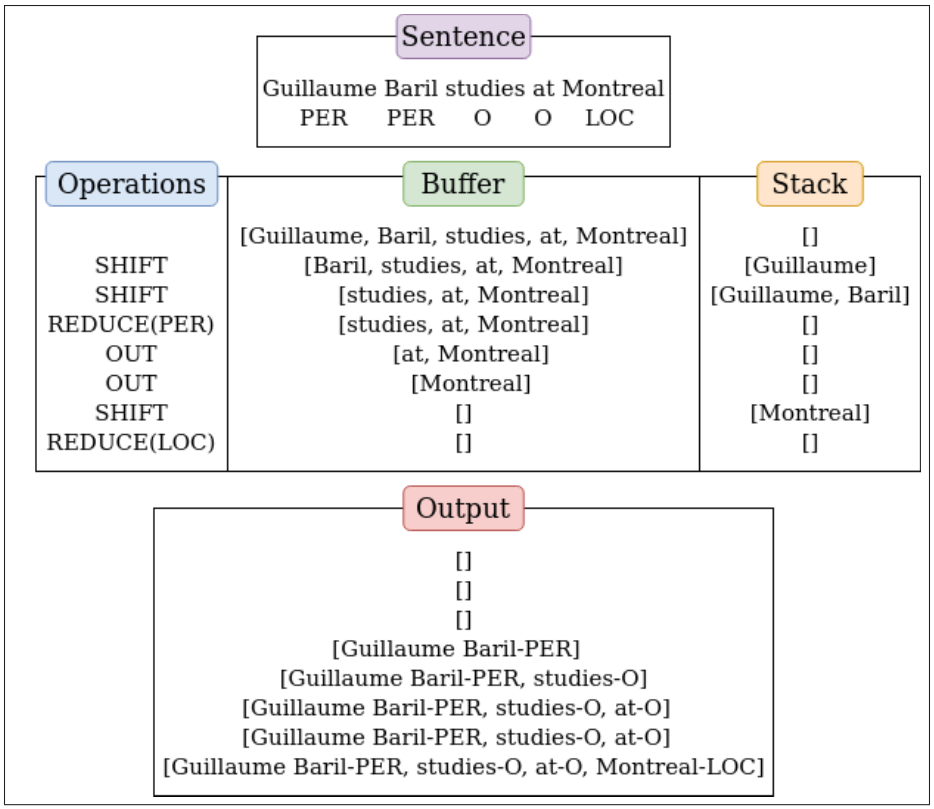


Figure 2.4 Example of a transition sequence with a Stack-LSTM
Adapted from Lample *et al.* (2016)

The best model without any additional data is the Word-level BiLSTM-CRF by Huang *et al.* (2015) with a F1 score of 84.26 and the best model using additional data is the Word+character-level BiLSTM-CNN by Chiu & Nichols (2016) with a F1 score of 91.62.

We think the Word+character-level BiLSTM-CRF model did worst than the Word-level BiLSTM-CRF model because it is bigger and its capacity led to overfitting.

### 2.3.2 Transformer-based models

As we know, Transformers firstly described in (Vaswani *et al.*, 2017) uses an encoder-decoder architecture. But, depending on the application, we may only need the encoder or the decoder. In the literature, the bidirectional Transformer is often referred as the encoder and the unidirectional Transformer is often referred as the decoder (Devlin *et al.*, 2019).

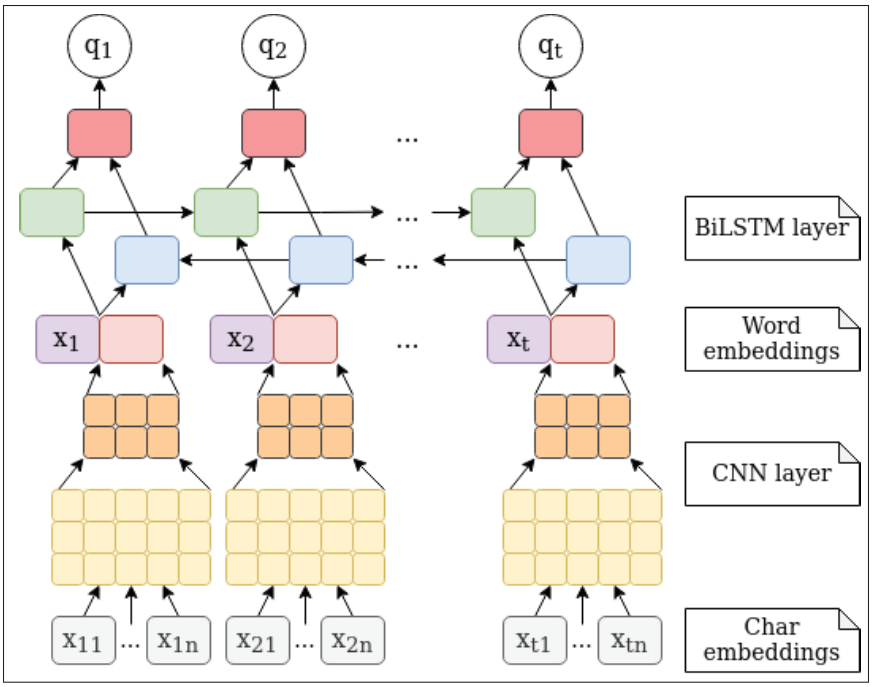


Figure 2.5 BiLSTM-CNN with word and character embeddings model architecture
Taken from Chiu & Nichols (2016)

Table 2.3 LSTM-based model F1 scores on CoNLL-2003 test set with no additional data (NAD) and with additional data (AD)

| Model | NAD | AD |
|---|---|---|
| Word LSTM (Huang *et al.*, 2015) | 79.82 | 83.74 |
| Word BiLSTM (Huang *et al.*, 2015) | 81.11 | 85.17 |
| Word BiLSTM-CRF (Huang *et al.*, 2015) | **84.26** | 90.10 |
| Word+Char BiLSTM-CRF (Lample *et al.*, 2016) | 83.63 | 90.94 |
| Word Stack-BiLSTM (Lample *et al.*, 2016) | 80.88 | 90.33 |
| Word+Char BiLSTM-CNN (Chiu & Nichols, 2016) | 83.38 | **91.62** |

One of the most popular Transformer-based language representation model is BERT, which stands for Bidirectional Encoder Representations from Transformers (Devlin *et al.*, 2019). The authors originally created two models with different hyperparameters as described in Table 2.4. Also, all FFNs has a hidden layer size of $4D$ and BERT uses WordPiece embeddings (Wu *et al.*, 2016) with a vocabulary of 30 000 tokens as input and output representations.

Table 2.4 BERT Hyperparameters

| **Model** | **Layers** $N$ | **Hidden layers size** $D$ | **Self-attention heads** $H$ |
|---|---|---|---|
| Base | 12 | 768 | 12 |
| Large | 24 | 1024 | 16 |

This model is trained in two steps. Firstly, BERT is pre-trained on a lot of data using two unsupervised tasks :

- **Masked LM** (MLM) consisting of masking some WordPiece tokens at random and predicting these masked words. This task trains the model to understand context in a sentence.
- **Next Sentence Prediction** (NSP) consisting of feeding two sentences to the model and predicting if sentence A follows sentence B or not. This task trains the model to understand relation between multiple sentences.

The resulting model is a language model. Then, the second step is to fine-tune BERT on specific tasks with labelled data. It does not require as much resources as the first step. The authors fine-tuned BERT on a NER dataset named CoNLL-2003 (Tjong Kim Sang & De Meulder, 2003) and the results are displayed in Table 2.5. CoNLL-2003 consists of news stories with four entity types : organizations, persons, locations and miscellaneous.

Table 2.5 BERT results on CoNLL-2003
after fine-tuning

| **Model architecture** | **Dev F1 score** | **Test F1 score** |
|---|---|---|
| Base | 96.4 | 92.4 |
| Large | 96.6 | 92.8 |

BERT has been improved by many authors. Important improvements has been made by Liu *et al.* (2019) and they named it RoBERTa, which stands for a Robustly Optimized BERT pretraining Approach. Firsly, these authors improve the Masked LM task by using dynamic masking. In other words, the masked tokens of each sentences change in every epoch. Also, they found that removing the NSP task matches or improves performance. Then, they used a Byte-Pair encoding (BPE) tokenizer (Sennrich *et al.*, 2016) with a larger vocabulary size of 50 000 tokens.

Additionally, they found that BERT was undertrained during the pre-training step. To improve this, they trained their model longer with a bigger batch size. Finally, they used longer input sequences, so the model can learn long-range dependencies.

Table 2.6 Comparison between BERT and RoBERTa base models

| Model | Training data | Pre-Training tasks | Total parameters | Tokenizer | Masking strategy |
|---|---|---|---|---|---|
| BERT | 13 GB | NSP & MLM | 110 M | WordPiece 30k | Static |
| RoBERTa | 160 GB | MLM | 125 M | BPE 50k | Dynamic |

Now, we will explore two French language models based on BERT and RoBERTa. The first one is called FlauBERT (Le *et al.*, 2020) and the second one is called CamemBERT (Martin *et al.*, 2020).

**FlauBERT** is very similar to RoBERTa. The only difference is the corpus used to train the model. They collected their data from three main sources: one corpus from the WMT19 shared task (Li *et al.*, 2019), one corpus from the OPUS collection (Tiedemann, 2012) and three datasets from the Wikimedia projects. After preprocessing, the training corpus was 71 GB in size.

**CamemBERT** is also very similar to RoBERTa, but there is three main differences. Firstly, it uses SentencePiece tokenization (Kudo & Richardson, 2018) which is an extension of BPE and WordPiece. This tokenizer with a vocabulary size of 32 000 tokens does not require pre-tokenization. Secondly, they use whole-word masking during the MLM task. In other words, they will not mask sub-tokens created by the tokenizer, but the whole original word. Nevertheless, the authors state that it does not impact the performance on lower level tasks like NER. Finally, they trained their model with the French part of the OSCAR corpus (Ortiz Suárez *et al.*, 2019) consisting of 138 GB of raw text.

Both models are very similar. As we can see in Table 2.7, they obtained very similar score in almost all experiences done by Le *et al.* (2020).

Table 2.7 Results (in %) on various tasks with FlauBERT and CamemBERT base models

| Task | FlauBERT | CamemBERT |
|---|---|---|
| Text classification (Accuracy) | 93.22 | 93.38 |
| Paraphrasing (Accuracy) | 89.49 | 90.14 |
| Inference (Accuracy) | 80.6 | 81.2 |
| Verb Disambiguation (F1-score) | 43.92 | 50.02 |
| Noun Disambiguation (F1-score) | 54.74 | 56.06 |

## 2.4 End-to-end models

There are many drawbacks to using a pipeline that independently trains the FA and NER models. Firstly, the error does not propagate from one component to the other, which means we cannot train both models effectively. Secondly, existing NER models are not robust to the noisy output of ASR models (Serdyuk *et al.*, 2018), degrading the overall performance of the pipeline. End-to-end models try to mitigate those drawbacks by optimizing all components as one model (Lugosch *et al.*, 2019), removing the unnecessary intermediate output of each components, thus reducing the overall size and complexity of the model (Ghannay *et al.*, 2018). In this section, we explore different end-to-end model for Spoken Language Understanding and some end-to-end NER models from speech.

### 2.4.1 End-to-end spoken language understanding

Spoken Language Understanding (SLU) systems usually perform three tasks: domain classification, intent detection and slot filling (Tur & De Mori, 2011). Altough it is not the same as NER from speech systems, the ideas behind it is similar. Also, it is important to note that NER on speech data is a SLU task. First SLU systems were pipeline applying ASR on the audio, followed by a Natural Language Understanding (NLU) model applied on the ASR output. Nowadays, there is a lot of research toward end-to-end SLU models to mitigate the drawbacks of a pipeline.

One of the first research on this subject was done by Serdyuk *et al.* (2018). Their model focuses on two tasks: domain classification and intent detection. They used an encoder-decoder

architecture where the encoder is a multilayer bidirectional Gated Recurrent Unit (BiGRU) network. Their best model uses 4 BiGRU layers. The encoder uses log-Mel filter bank features and the output is given to the decoder. The decoder has three layers: a max-pooling layer along the time axis, a FNN layer and a softmax layer to calculate the probability of intents or domains. They were able to show that their end-to-end model require way less parameters compared to a pipeline, but they got worse performance.

Then, Lugosch *et al.* (2019) proposed a public dataset for SLU named FSC and a pretrained model focusing only on the intent detection task. Their model has three modules: a phoneme module, a word module and an intent module. The first two modules are SincNet layers (Ravanelli & Bengio, 2018) pretrained on the Librispeech dataset (Panayotov *et al.*, 2015) to predict phonemes or words. Then, there is the last module which is very similar to the model developed by Serdyuk *et al.* (2018). After the pretraining phase, all modules are fine-tuned on the SLU task. They obtained an accuracy of 97.2% on their dataset, but they did not compare their results with existing models. Thus, it is hard to determine if their pretrained model is good or not.

Also, Radfar *et al.* (2020) created a new SLU model with Transformers because they allow parallelization and they require less parameters to achieve the same or better performance compared to RNNs. Their model focuses on all tasks. The encoder uses low frame rate log Short-Time Fourier Transform (STFT). Then, the decoder takes as input the encoder output and the predictions of the last utterance and predict the domain, the intent and the slot. They trained their model, Serdyuk *et al.* (2018) model and Lugosch *et al.* (2019) model on FSC to compare them (see Table 2.8).

Lastly, Qian *et al.* (2021) also used Transformers. They leveraged pretrained ASR and NLU Transformers to create their own model. They obtained state-of-the-art accuracy on the FSC dataset (see Table 2.8).

Table 2.8 Comparaison of different end-to-end SLU model on the FSC dataset

| Model | Accuracy (in %) |
|---|---|
| GRU (Serdyuk *et al.*, 2018) | 95.3 |
| Pretrained SincNet+GRU (Lugosch *et al.*, 2019) | 97.2 |
| Transformer (Radfar *et al.*, 2020) | 97.6 |
| Pretrained Transformer (Qian *et al.*, 2021) | **99.7** |

### 2.4.2 End-to-end named entity recognition from speech

One task that SLU systems can perform is NER. To the best of our knowledge, the only work related to end-to-end NER from speech model is from Ghannay *et al.* (2018). There is also a paper who tries to use the same architecture on english data (Yadav *et al.*, 2020), but it had nothing more to the original paper.

Ghannay *et al.* (2018) used an architecture very similar to DeepSpeech 2 (Amodei *et al.*, 2015). Their model has two 1D invariant convolution layers, six BiLSTM layers, a lookahead convolution layer, an FNN layer and a softmax layer (see figure 2.6). The character sequence $c$ outputted by the model is composed of the alphabet and nine named entity tags (see Table 2.9).

Table 2.9 Nine named entity tags outputted by the Deep RNN model

| Tag symbol | Signification |
|---|---|
| [ | Beginning of a person |
| ( | Beginning of a function |
| { | Beginning of an organisation |
| $ | Beginning of a location |
| & | Beginning of a production |
| % | Beginning of an amount |
| # | Beginning of a time period |
| ) | Beginning of an event |
| ] | End of an entity |

Because audio recordings with named entity annotations are very rare, the authors proposed two strategies to compensate the lack of data during training. The first strategy consisted of

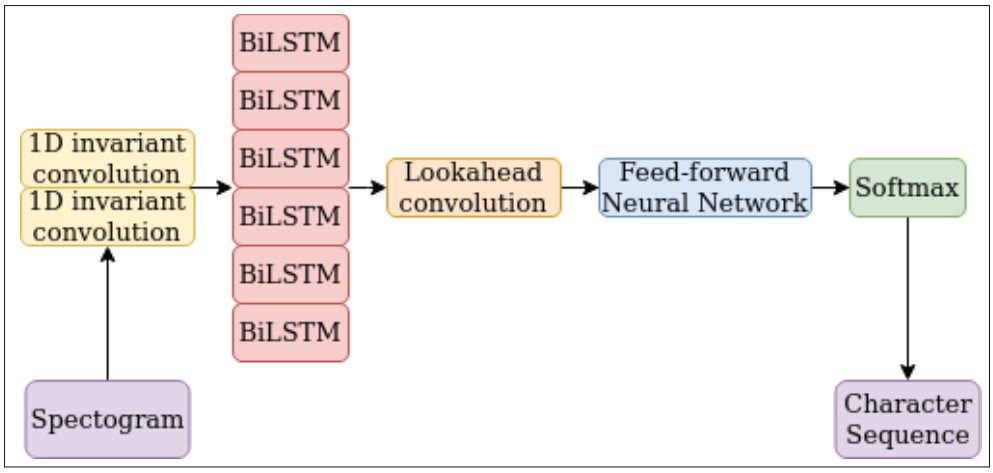


Figure 2.6 Deep RNN architecture of the end-to-end NER from speech model
Taken from Ghannay *et al.* (2018)

using a multi-task learning approach. Firstly, they trained the model only on the ASR task and after the training, they reinitialized the softmax layer to take into consideration the nine named entity tags. Then, they retrained the model with the audio recordings with NER annotations. The second strategy consisted of inscreasing the amount of data by annotating audio recording without NER annotations with a NER system dedicated to text data. These new annotations was then used in the training phase of the end-to-end model.

They trained and evaluated their models on a corpus named DeepSUN. This corpus combines four French corpora composed of audio recordings from radio and television stations: ESTER 1, ESTER 2, ETAPE, and Quaero.

Their end-to-end model was better than the pipeline to detect named entities in an example. However, the performance of the end-to-end model to extract the named entity values was worse than the pipeline. In other words, the end-to-end model was better to determine if there is a named entity in a sentence, but it could not say which words are part of that entity. Table 2.10 shows the performance of the end-to-end model, and Table 2.11 shows the performance of the pipeline.

Table 2.10 End-to-end model performance on the test set

| **Detection** | | | **Extraction** | | |
|---|---|---|---|---|---|
| Precision | Recall | F1 score | Precision | Recall | F1 score |
| **0.76** | **0.63** | **0.69** | 0.49 | 0.41 | 0.47 |

Table 2.11 Pipeline performance on the test set

| **Detection** | | | **Extraction** | | |
|---|---|---|---|---|---|
| Precision | Recall | F1 score | Precision | Recall | F1 score |
| 0.74 | 0.58 | 0.65 | **0.57** | **0.45** | **0.50** |

Yadav *et al.* (2020) obtained a F1 score of 0.906 on their dataset. They stated that they got better performance than Ghannay *et al.* (2018) mainly because the Word Error Rate (WER) of their model was better. Indeed, they got a WER of 2.72% compared to the 19.96% of Ghannay *et al.* (2018). So, improving the ASR WER improves the overall performance of the end-to-end model.

## 2.5 Summary

In summary, we explored many models used by both pipeline components and existing end-to-end models.

# CHAPTER 3

# METHODOLOGY

In this section, we present our methodology. We begin by introducing our proposed approach. Then, we present the algorithms used in our experiments to find the best components for the pipeline. After that, we introduce the corpora. Finally, we define our evaluation metrics and how we aligned the prediction of our FA models with the gold standard annotations.

## 3.1 Proposed approach

Our proposed approach to anonymize French audio recordings consists of a pipeline with two components. The first component is the FA algorithm aligning the transcription with the audio and the second component is the NER model finding the named entities in the transcription. Then, we replace the audio inside the boundaries of the named entities with silences, outputting a redacted audio recording. The architecture of the pipeline is shown in Figure 3.1.

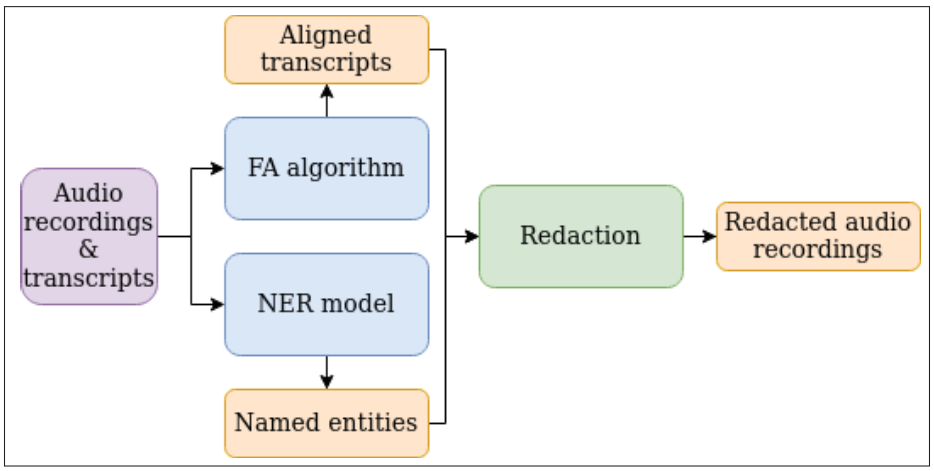


Figure 3.1 Pipeline architecture overview

Following what we have seen on end-to-end models, we decided to carry out our experiments only with a pipeline for multiple reasons. Firstly, ASR is out of scope for this research project. We assume that the transcript of the audio recording already exists and that we only need to apply FA to align each word with its corresponding audio signal. So, having an end-to-end models

which only applies FA and NER seems excessive. Secondly, there is no accessible pretrained end-to-end NER from speech model in French at the moment. It would take an enormous amount of data and a lot of computing power to train SLU models from scratch, and we do not have these resources. Finally, end-to-end models did not perform that much better compared to pipelines. Indeed, they were better in some cases, but they got the same performance at best for the most part.

The research is done in two stages. In the first stage, we find the best FA algorithm and NER model. Then, in the second stage, we create the pipeline by combining the FA and NER algorithms.

## 3.2 Algorithms explored

This section enumerates the algorithms explored in the first stage to select the best components for the pipeline. We start with the FA algorithms and then present the NER algorithms.

### 3.2.1 Forced alignment algorithms

We compare both algorithms explored in Section 2.2 with their default hyperparameter values (see Table 3.1). The first algorithm is MFA (version 2.0.0a4), developed by McAuliffe *et al.* (2017), which is based on Kaldi. We use the pretrained acoustic model *french_prosodylab* and its associated dictionary.

The second algorithm, which is based on Julius, is SPPAS (version 3.7) developed by Bigi (2015). Such as MFA, we use one of the pretrained acoustic model named *fra* and its dictionary.

Table 3.1 Hyperparameters of each FA models

| **Algorithm** | **Beam size** | **Audio frequency (kHz)** |
|---|---|---|
| MFA | 100 | 8 |
| SPPAS | 1000 | 16 |

We do not use Cepstral Mean and Variance Normalization (CMVN) because it can degrade the model performance when the utterances are short (Prasad & Umesh, 2013), which is our case.

### 3.2.2 Named entity recognition algorithms

We choose to evaluate two different LSTM-based models for our experiments. The first model is the Word-level BiLSTM-CRF (Huang *et al.*, 2015) because it has the highest F1 score on the CoNLL-2003 dataset without any additional data. Furthermore, in French, there are not many resources compared to English. Thus, we might not be able to use pretrained embeddings or gazetteers. Therefore, this model might yield the best results.

The second LSTM-based model is the Word+character-level BiLSTM-CRF (Lample *et al.*, 2016) for two reasons. Firstly, as we said earlier, this architecture has proven good performance with little domain-specific knowledge. Because there are not many resources in French, this aspect is very important because it allows us to use a model trained on a large amount of data in another context which may not be exactly the same domain. Secondly, this model has the second-best F1 score with additional and without additional data. We want to know if the character-level information from this model will be more useful in French than it was in English because, indeed, it did not help for the CoNLL-2003 NER task.

For both LSTM-based models, we use the code created by Guillaume Lample[4] because it allowed us to reproduce both architectures easily (see Table 3.2). Notice that we use the same layer size for both Word-level BiLSTMs because it does not significantly affect model performance (Huang *et al.*, 2015). Also, both models will be trained with Stochastic Gradient Descend (SGD).

Table 3.2 Hyperparameters of each LSTM-based models

| Algorithm | Char layer size | Word layer size | Dropout | SGD Learning rate | Epochs |
|---|---|---|---|---|---|
| Word-level | N/A | 100 | 0.5 | 0.005 | 15 |
| Word+char-level | 25 | 100 | 0.5 | 0.005 | 30 |

[4] Available at https://github.com/glample/tagger

We also choose to evaluate one Transformer model, CamemBERT (Martin *et al.*, 2020), in this stage for multiple reasons. First, this model is more popular than FlauBERT, so many researches use CamemBERT for their experiments, making it easier to compare results and improve the model. Also, as we can see in Table 2.7 in Section 2.3.2, CamemBERT performs similarly or better than FlauBERT in all tasks. Finally, CamemBERT was pre-trained on more data than FlauBERT. Thus, it should increase performance on downstream tasks (Liu *et al.*, 2019).

We use the base model available with Huggingface with almost all the default hyperparameters except for those present in Table 3.3.

Table 3.3 Hyperparameters of CamemBERT

| **Batch size** | **Learning rate** | **Number of epochs** |
|---|---|---|
| 16 | 0.00005 | 5 |

## 3.3 Corpora

This section introduces the corpora related to our objectives and which is used in our experiments. First, we start with the speech corpus used to evaluate the pipeline and for training and evaluating the FA models. Then, we present the corpus used to train and assess the NER models.

### 3.3.1 Speech corpus

The speech corpus used is the Nijmegen Corpus of Casual French (NCCFr) developed by Torreira *et al.* (2010). This corpus contains 36 hours of transcribed conversations. Also, there are 46 different speakers from multiple regions of France.

We could not use this corpus directly to evaluate FA algorithms because we needed word boundaries. So, we had to annotate this corpus manually to meet our needs. Firstly, we split each corpus interval (see Figure 3.2). Each interval corresponds to a phrase of a single speaker.

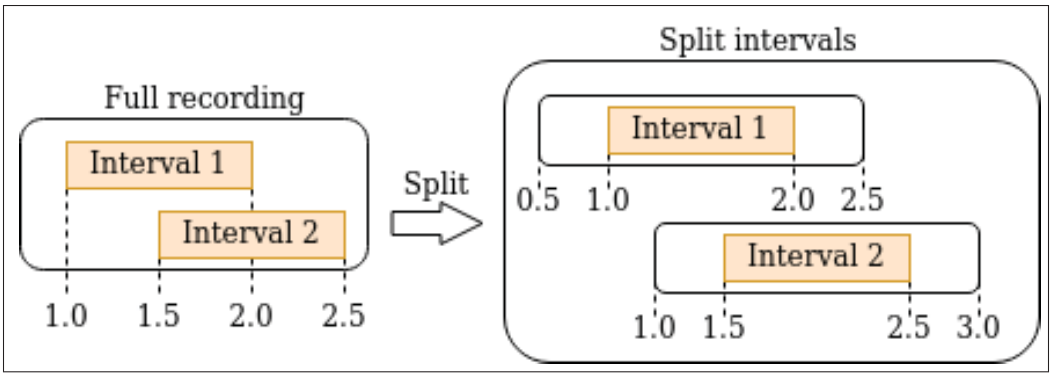


Figure 3.2 Illustration of the splitting of each phrase of the NCCFr corpus with time relative to the full recording

Then, we randomly chose phrases potentially containing named entities and we annotated 381 seconds by hand with Praat, developed by Boersma & Weenink (2020). We can see one example in Figure 3.3 where we defined each word's boundaries and annotated each named entity.

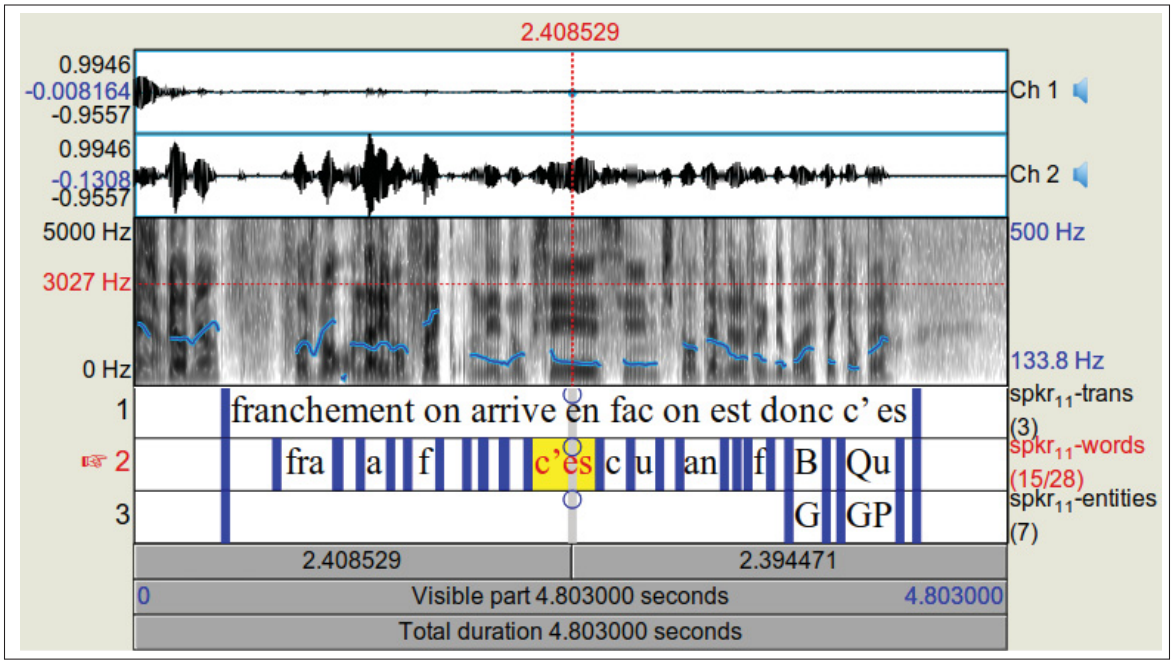


Figure 3.3 Example of one phrase annotated word by word with Praat

The first tier *trans* contains the full phrase, the second tier *words* has each word boundary and the last tier *entities* contains the type of each named entity present in the phrase.

### 3.3.2 Text corpus

We used the French corpus for NER and Relation Extraction of financial news (FrenNER) developed by Jabbari *et al.* (2020). They manually annotated 130 news articles from forty daily French newspapers. The corpus contains 6736 entities with 26 different types.

Before using this dataset, we had to clean it. We replaced some characters like *\t* or *\u2009* with spaces, removed multiple spaces, and removed leading determiner from named entity annotations.

Then, we modified or deleted some entity types for two reasons. First, either the entity was irrelevant for us, or there were few entities of a certain type. Therefore, we only kept the Currencies, the Locations, the Money Amounts, the Organizations and the Persons, and we modified these entities to fit into one of the five kept types :

- **Shareholderships** and **Financing** contained a lot of Money Amount and Currencies. So, we decided to change some of them into Money Amount and Currencies and we deleted the others because they were irrelevant (*d'actionnaires, entre au capital, lever, investis, etc.*).
- **Geopolitical Entities** were converted into Locations and Organizations if applicable. Otherwise, they were deleted because they were irrelevant (*les gouvernants, etat, g20, etc.*).
- **World Regions**, **Countries**, **Local Regions**, and **Cities** were all converted into Locations because it was not necessary to split them into sub-types.
- **Agents**, **Associations**, **Medias** and **Compagnies** were all converted into Organizations for the same reason as the previous point.

Finally, we split the news articles into sentences for a total of 4424 sentences. These sentences were randomly split into ten groups and used in a 10-fold cross-validation scheme. We decided to do this step at this point because we wanted to have the same groups for each model. Thus, the comparison between each model would be more accurate than generating new groups for each of them.

## 3.4 Evaluation

In this section, we present the algorithms used to evaluate the models. First, we begin with the evaluation metrics of the FA models. Then, we explain how we aligned the prediction of our FA models with the gold standard. After that, we define the evaluation metrics of the NER models. Finally, we describe the metrics to evaluate the final pipeline.

### 3.4.1 Forced alignment evaluation

The evaluation of a FA model is done by calculating the accuracy on the corpus $C$ defined as

$$\text{Accuracy} = \frac{1}{|C|} \sum_{p,g \in C} \sum_{p_i, g_i \in p,g} \delta_t(p_i, g_i) \tag{3.1}$$

where each pair $p$ and $g$ are the model prediction and the gold standard, respectively. Also, $p_i$ and $g_i$ represents the $i^{\text{th}}$ word boundary and $\delta_t$ is a function returning 1 if $p_i$ is correctly aligned given a tolerance $t$.

Two $\delta_t$ functions were used. Let's define $p_i^0$ the time when the $i^{\text{th}}$ word starts and $p_i^1$ the time when the $i^{\text{th}}$ word ends with the same notation for $g_i$. The first function $\delta_t^s$, referenced as *std*, is the absolute difference between the prediction and the gold standard defined as

$$\delta_t^s(p_i, g_i) = \begin{cases} 1 & \text{if } |p_i^0 - g_i^0| \leq t \;\&\; |p_i^1 - g_i^1| \leq t \\ 0 & \text{else} \end{cases} \tag{3.2}$$

The second function $\delta_t^o$, referenced as *outer*, is very similar to the first one, but the tolerance is applied only inside the boundaries. We use this function because when anonymizing audio, the error is less important if we remove more information than not enough. This function is useful for comparing the accuracy with the *std* function and seeing if most of the errors are inside or

outside the boundaries. It is defined as

$$\delta_t^o(p_i, g_i) = \begin{cases} 1 & \text{if } p_i^0 \leq g_i^0 + t \;\&\; g_i^1 - t \leq p_i^1 \\ 0 & \text{else} \end{cases} \tag{3.3}$$

Figure 3.4 shows how the functions work. If the prediction $p^0$ is in the green area and $p^1$ is in the blue area, then the word is considered correctly aligned.

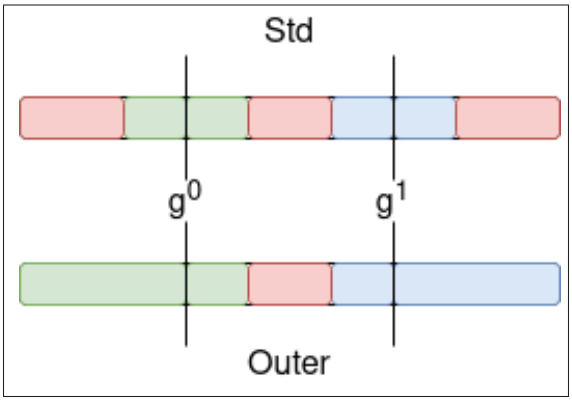


Figure 3.4 Visual representation of the alignment functions *std* and *outer* used to evaluate the FA algorithms.

### 3.4.2 Association between the alignment prediction and the gold standard

Before calculating the accuracy with eq. (3.1), we need to associate each prediction $p_i$ with its corresponding gold standard annotation $g_j$. Then, we explain the four scenarios that might happen when associating the annotations (see algorithm 3.1).

Firstly, if we have an exact match between one $p_i$ and one $g_j$, we can determine with eq. (3.2) or (3.3) if it is correctly associated.

Sometimes, predictions or gold standard associations can be composed of more than one word. So, the second scenario manages the case when the gold standard is made of multiple words. For example, we could have the gold standard annotation *Aujourd'hui*, but the algorithm split

Algorithm 3.1 Prediction and gold standard association algorithm

**Input:** Prediction $p$, gold standard $g$ and evaluation function $\delta_t$
**Output:** Accuracy $\alpha$

$\alpha, t, i, j \leftarrow 0$
$l \leftarrow []$
**while** $i < |p|$ **do**
    **if** *match*$(p_i, g_j)$ **then**
        **if** $\delta_t(p_i, g_j)$ **then**
            $\alpha \leftarrow \alpha + 1$
        $j \leftarrow j + 1$
    **else if** *multipleWords*$(g_j)$ & *partial*$(p_i, g_j)$ **then**
        $k \leftarrow i$
        **while** $i < |p| - 1$ & *partial*$(p_{k+1}, g_j)$ & *!match*$(p_{i:k}, g_j)$ **do**
            $i \leftarrow i + 1$
            $k \leftarrow k + 1$
        **if** $\delta_t(p_{i:k}, g_j)$ **then**
            $\alpha \leftarrow \alpha + 1$
        $j \leftarrow j + 1$
    **else if** *multipleWords*$(p_i)$ & *partial*$(p_i, g_j)$ **then**
        $k \leftarrow j$
        **while** $k < |g| - 1$ & *partial*$(p_i, g_{k+1})$ & *!match*$(p_i, g_{j:k})$ **do**
            $j \leftarrow j + 1$
            $k \leftarrow k + 1$
        **if** $\delta_t(p_i, g_{j:k})$ **then**
            $\alpha \leftarrow \alpha + 1$
        $j \leftarrow j + 1$
    **else**
        removeOld($l$)
        **for** $l_k \in l$ **do**
            **if** *match*$(l_k, g_j)$ **then**
                $l$.remove($l_k$)
                **if** $\delta_t(l_k, g_j)$ **then**
                    $\alpha \leftarrow \alpha + 1$
                $j \leftarrow j + 1$
                break
        $l \leftarrow p_i$
    $i \leftarrow i + 1$

it into *Aujourd* and *hui*. If we start with $k = i$, we will increment $k$ to obtain $p_{i:k}$. The new annotation $p_{i:k}$ starts at $p_i^0$, ends at $p_k^1$ and is composed of every words between $p_i$ and $p_k$. It will be repeated as long as there is a partial match between $p_{k+1}$ and $g_j$ and as long as there is not an exact match between $p_{i:k}$ and $g_j$. Then, we can determine if $p_{i:k}$ and $g_j$ is correctly associated.

The third scenario manages the case where the prediction is made of multiple word. For example, we could have the prediction on the word *c'est* while having a gold standard annotation for *c'* and *est*. We use the same logic as the second scenario, but instead of merging the predictions, we will merge the gold standard. Then, we will determine if $p_i$ and $g_{j:k}$ is correctly associated.

Fourthly, $p_i$ might not match with $g_j$ at all. The algorithm might not find an association for every word or the annotations might not have the same word order as the gold standard. The last scenario manages this case by having a list of unassociated predictions $l$. First, we remove old annotations in $l$ because we do not want a prediction from the end matching a gold standard at the beginning. Then, we will check if we have an exact match between one word in the list $l_k$ and $g_j$. If we have a match, we will determine if $l_k$ and $g_j$ is correctly aligned. Finally, we add the current unmatched prediction $p_i$ to $l$.

### 3.4.3 Named entity recognition evaluation

There are three evaluation metrics for NER models. The first metric is the precision. As we can see in eq. (3.4), it corresponds to the proportion of real entity found among all the predictions made.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.4}$$

The second metric is the recall. As we can see in eq. (3.5), it corresponds to the proportion of total entities found.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.5}$$

The last metric is the F1 score, which is the harmonic mean of the precision and the recall. It is defined as

$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.6}$$

Now, let's define what we count as a true positive (*TP*), a false positive (*FP*) and a false negative (FN) :

- ***TP*** is when the model predict the right type of a named entity.
- ***FP*** is when the model predict a named entity when there is none.
- ***FN*** is when the model does not find a named entity.

Also, if the model does not predict the right type of a named entity, we count it as a *FP* and also as a *FN*.

In our pipeline, finding the type of an entity is not as important as it is in a classic NER task. It is a piece of useful information, but it is more important to find entities than to classify them. So, in order to evaluate only the ability of a model to find entities, we decided to calculate the precision, the recall, and the F1 score without counting if the predicted type is suitable. In other words, if the model does not predict the right type of a named entity, we will not count it as a *FP* or as a *FN*. We refer to those metrics as No Type Error (NTE).

### 3.4.4 Pipeline evaluation

The evaluation of the final pipeline is similar to the evaluation of FA and NER models. It uses elements from both methods. We use the same precision, recall and F1 score used to evaluate NER models (see Section 3.4.3), but we calculate *TP*, *FP* and *FN* using the *outer* function $\delta_t^o$ (see eq. (3.3)) where $t$ is the tolerance.

Let's define $K$ as the set of predictions made by the model with its corresponding gold standard. The first variable *TP* is equal to the number of named entities found and correctly aligned. More formally, *TP* is defined as

$$TP = \sum_{p,g \in K} \sum_{p_i,g_i \in p,g} \delta_t^o(p_i, g_i) \tag{3.7}$$

The second variable *FN* is equal to the number of named entities found and not correctly aligned plus the number of gold standard annotations without any corresponding prediction. Since we represent the absence of prediction for a certain gold standard as $p_i^0 = 0$ and $p_i^1 = 0$, the *outer* function will always return 0 in that case. Thus, we can define *FN* as

$$FN = \sum_{p,g \in K} \sum_{p_i, g_i \in p,g} 1 - \delta_t^o(p_i, g_i) \tag{3.8}$$

Finally, the last variable *FP* is equal to the number of predictions without any corresponding gold standard annotation. In a similar vein, the absence of a gold standard for a certain prediction is represented as $g_i^0 = 0$ and $g_i^1 = 0$.

$$FP = \sum_{p,g \in K} \sum_{p_i, g_i \in p,g} g_i^0 + g_i^1 == 0 \tag{3.9}$$

## 3.5 Summary

In summary, we presented both corpora: a speech corpus named NCCFr (Torreira *et al.*, 2010) and a text corpus for NER called FrenNER (Jabbari *et al.*, 2020). Also, we presented the evaluation methods and the algorithms used in our experiments.

# CHAPTER 4

# STAGE ONE: CHOOSING THE PIPELINE COMPONENTS

In this section, we show the experiments carried out in order to choose the best models for the components of the final pipeline. Firstly, we choose the best FA algorithm and then, we choose the best NER model. Lastly, we summarize the final outcomes of this stage.

## 4.1 Choosing the best forced alignment algorithm

The first experiment consists of three phases and the goal is to choose the best algorithm for our final pipeline. Firstly, we align the corpus (see Section 3.3.1) with both algorithms. The next two phases analyze this alignment.

Secondly, we evaluate multiple tolerances in order to see their effect on the accuracy. Based on this, we choose a tolerance for the last phase.

Finally, we check if the examples adequately represent the dataset by observing the evolution on the *outer* accuracy when adding more examples to our sample. In other words, we want to make sure the model is robust. In total, we have a dataset of 85 examples. We will start with a sample of five examples, evaluate it, add five more examples, evaluate it, etc. For each evaluation, we calculate the accuracy of the whole sample and we also create ten random sub-samples. Then, we calculate the accuracy on each sub-sample and plot the mean and the variance.

### 4.1.1 Results

In this section, we analyze the alignment of the corpus done by both algorithms.

#### 4.1.1.1 Tolerance comparison phase

We can see in Figure 4.1 the accuracy of each model with both evaluation functions according to the tolerance. Table I-1 shows the same information. For SPPAS, there is a difference in

accuracy of less than 1% between the *std* and the *outer* functions starting with a tolerance of 0.60 seconds. For MFA, it starts with a tolerance of 0.20 seconds. Also, with the *outer* function, SPPAS has less than 1% increase in accuracy over the previous tolerance starting at 0.50 seconds. For MFA, it is even better at 0.30 seconds.

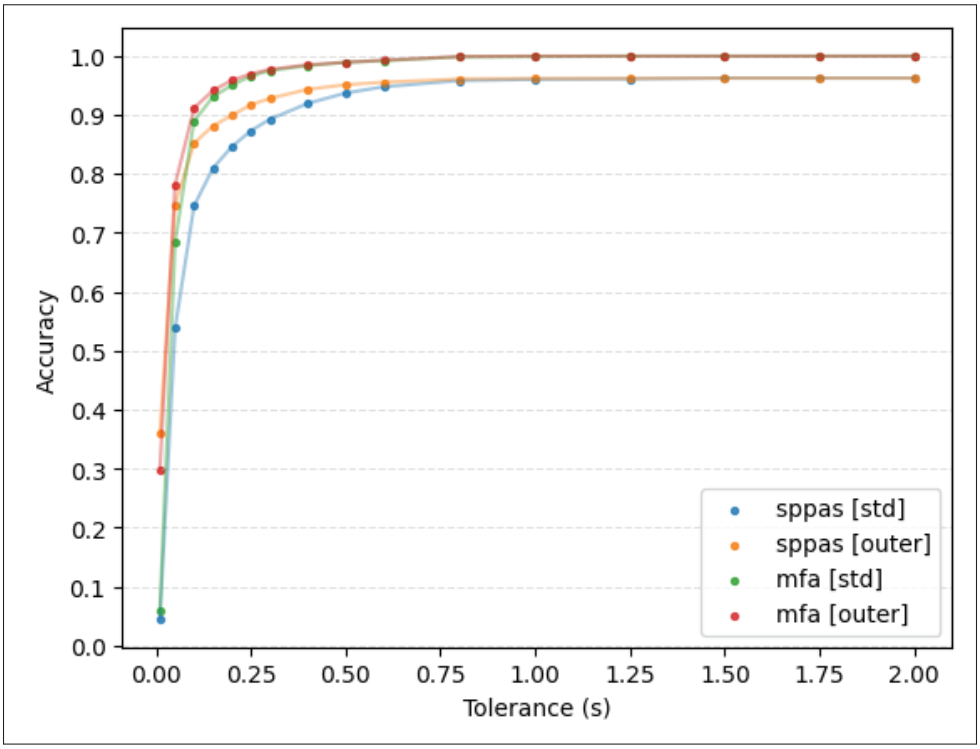


Figure 4.1 Evolution of the accuracy of FA models at different tolerances

In French, the oral flow is around 200 words per minute (Rist, 1999) or approximately three words per second. Based on Goodenough-Trepagnier & Frankston (1978), the average number of syllables per word is around 1.25. So, we have about four syllables per second, which means each syllable takes an average of 0.25 seconds. Also, named entities are typically composed of multiple syllables. Thus, we decided to use a tolerance of 0.25 seconds for the last phase, which is equal to one syllable.

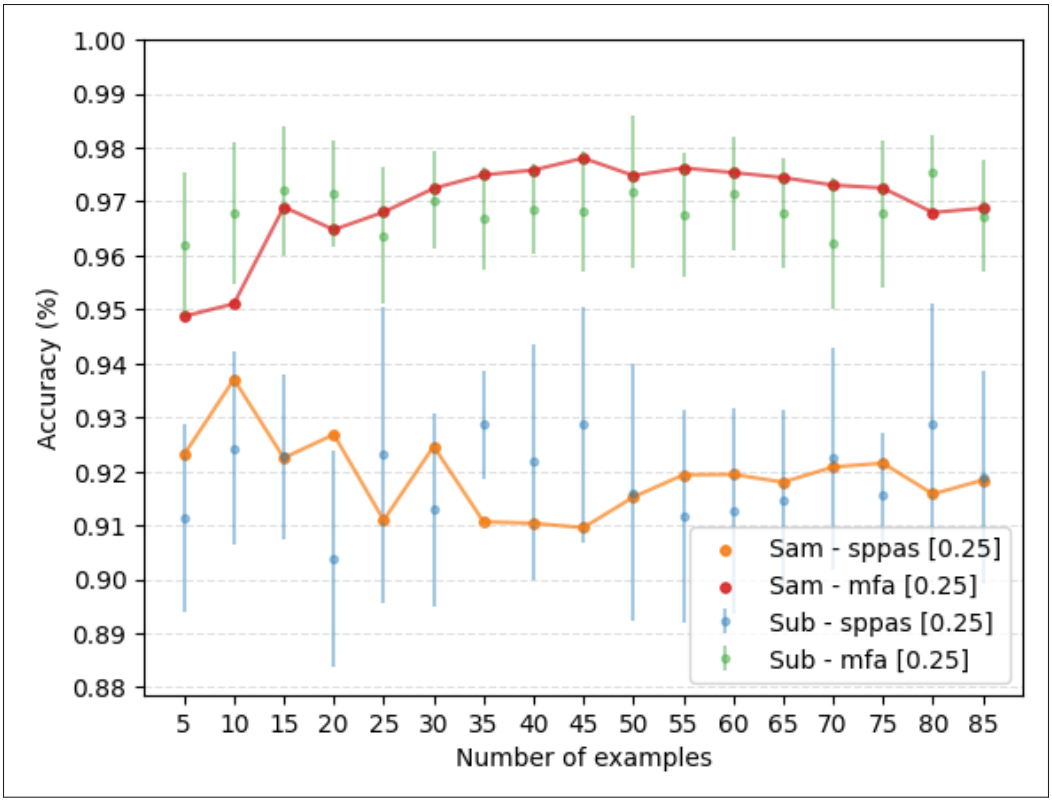


Figure 4.2 Evolution of the *outer* accuracies of FA models on different sample (sam) and sub-samples (sub) sizes with a tolerance of 0.25 seconds

#### 4.1.1.2 Robustness verification phase

Lastly, we can see in Figure 4.2 and Table I-2 the result of the last phase. The accuracy of both models is constant when adding more examples. For SPPAS, it is around 0.92, and for MFA, it is around 0.97. Therefore, we can conclude that we have enough examples to measure the alignment well. Also, is it worth mentioning that SPPAS is twice as fast as MFA to annotate the 85 examples. Indeed, SPPAS takes 9.14 seconds while MFA takes 20.66 seconds.

Based on our results, we choose to keep the MFA algorithm for our final pipeline because it has better accuracy than SPPAS. Furthermore, since the pipeline is offline, we think it is better to minimize the error of the first component rather than minimize the time it takes.

### 4.1.2 Discussion

In Section 2.2, we saw that MFA had an accuracy of 0.97 with a tolerance of 100 ms on the Buckeye corpus and that SPPAS had an accuracy of 0.99 on a spontaneous French corpus with a tolerance of 80 ms. In our experiment, we are not close to these results. Indeed, with a tolerance of 100 ms, SPPAS had a *std* accuracy of 0.05 and MFA was at 0.06. There could be four sources of error.

First of all, we used the pretrained models without any fine-tuning. This could explain the poor performance on this specific corpus.

Secondly, the main source of error in our corpus is the annotator. The author of this work did it without having great knowledge of the annotation task and some words were difficult to annotate, for examples :

- *Petit humain* : it was difficult to split the sounds *i* and *u*.
- *Hummmm* : it was difficult to know exactly where to add the boundaries when the word was stretched for a few seconds.

So, maybe the annotations are not precise enough. This could explain the low accuracy.

Also, another source of error could come from the alignment algorithm between the gold standard annotations and the prediction described in Section 3.4.2. We used ad hoc tests to verify the output of the algorithm. So, there could be potential bugs where the words are not correctly aligned. Thus, associating wrong words together and giving inappropriate boundaries.

Finally, the last source of error could be in the manipulation of the data. Indeed, we did a lot of manipulations to transform the NCCFr audio into examples usable by both SPPAS and MFA.

That being said, even if we assume a perfectly annotated corpus, our experiments shown worse performance compared to what is expected from the literature. However, we obtained a similar accuracy with a tolerance of 250 ms, which is perfectly fine for our usecase.

## 4.2 Choosing the best named entity recognition model

The second experiment also consists of four phases, and the goal is very similar to the first experiment: we want to choose the best model for our final pipeline. Therefore, the first phase is to train all three models. Then, we choose the best model based on their results on the test set. Thirdly, we try to improve the recall of the best model by adding a confidence threshold. Finally, we use a soft voting algorithm to ensemble all models trained with 10-fold cross-validation.

### 4.2.1 Results

In this section, we train and analyze all three models.

#### 4.2.1.1 Training phase

As explained in Section 3.3.2, we used 10-fold cross-validation to evaluate the robustness of our models. Each table in Appendix 2 shows the metrics mean and standard deviation of each group, and each figure below shows the metrics mean. The metrics are the loss over the training set, and the precision, the recall and the F1 score over the dev set. Figure 4.3, Table I-3, and Table I-4 show the results obtained when training the Word-level BiLSTM-CRF.

Figure 4.4, Table I-6, and Table I-7 show the results obtained when training the Word+char-level BiLSTM-CRF. Finally, Figure 4.5, Table I-9, and Table I-10 show the results obtained when fine-tuning CamemBERT.

#### 4.2.1.2 Evaluation phase

To evaluate our model and decide which one is the best, we compare their performance on the test set. Table 4.1 shows the results of each model. For more details, see Appendix 2. Also, Table 4.2 shows the NTE results obtained on the same models.

Surprisingly, the Word+char-level BiLSTM-CRF performed worst than the Word-level BiLSTM-CRF. Nonetheless, CamemBERT performed better than both BiLSTM-CRF models with a F1

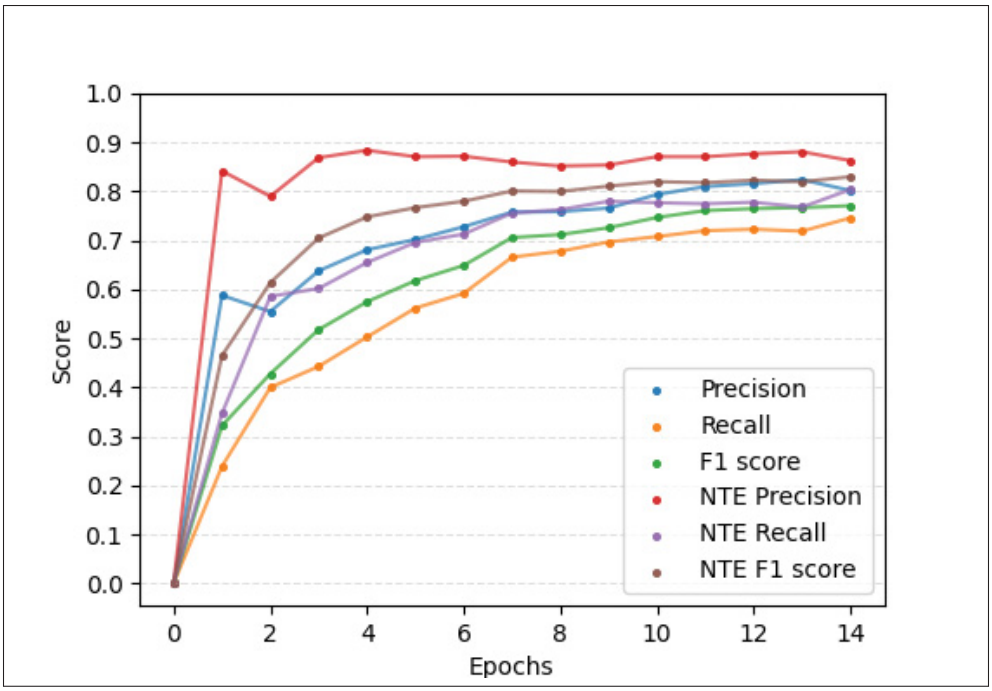


Figure 4.3 Word-level BiLSTM-CRF performance on FrenNER dev set at each epoch

Table 4.1 Performance on FrenNER test set

| Model | Precision | Recall | F1 Score |
|---|---|---|---|
| Word BiLSTM-CRF | 0.808 ± 0.029 | 0.768 ± 0.038 | 0.786 ± 0.013 |
| Word+char BiLSTM-CRF | 0.800 ± 0.051 | 0.726 ± 0.059 | 0.757 ± 0.017 |
| CamemBERT | 0.865 ± 0.014 | 0.885 ± 0.024 | 0.874 ± 0.013 |

score of 0.899. As stated in the previous section, this difference comes from the pre-training phase of the Transformer. Therefore, we decided to choose CamemBERT as our model for the pipeline.

#### 4.2.1.3 Recall improvement phase

Our final objective is to create a pipeline to anonymize call recordings. In our case, it is less severe to remove more information than removing less information. Thus, improving recall while decreasing precision is a little bit better. To be able to control this, we added a confidence

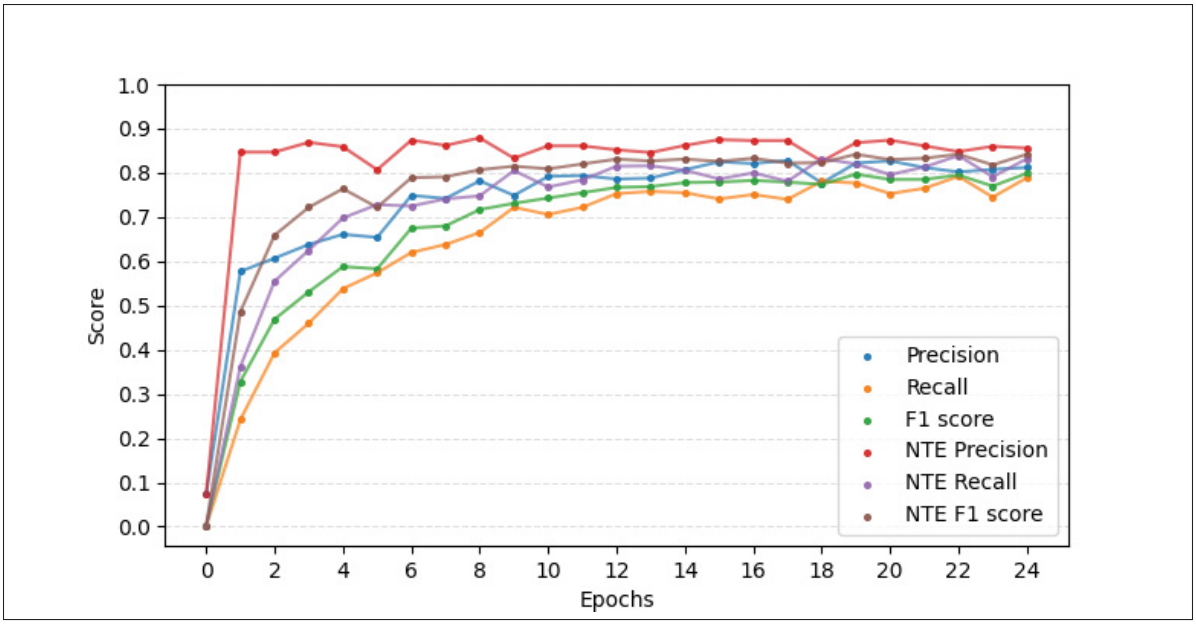


Figure 4.4 Word+char-level BiLSTM-CRF performance on FrenNER dev set at each epoch

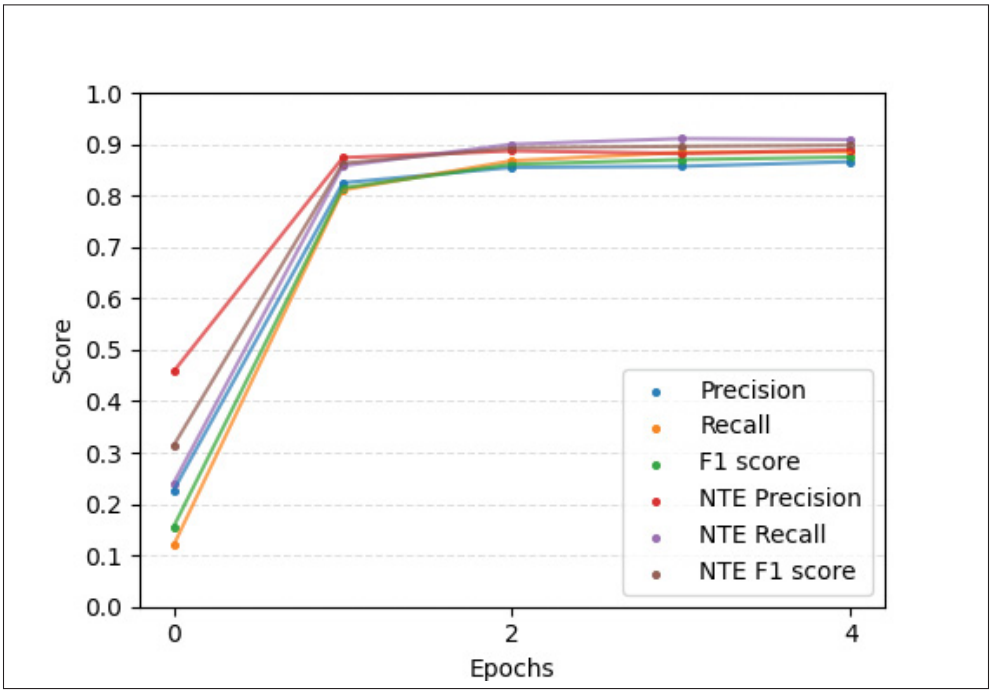


Figure 4.5 Camembert performance on FrenNER dev set at each epoch

Table 4.2 NTE performance on FrenNER test set

| Model | Precision | Recall | F1 Score |
|---|---|---|---|
| Word BiLSTM-CRF | 0.853 ± 0.032 | 0.811 ± 0.038 | 0.830 ± 0.012 |
| Word+char BiLSTM-CRF | 0.858 ± 0.042 | 0.779 ± 0.073 | 0.813 ± 0.022 |
| CamemBERT | 0.889 ± 0.016 | 0.910 ± 0.023 | 0.899 ± 0.013 |

threshold over the "Not an entity" probability. For example, with a confidence threshold of 0.5, if the "Not an entity" label has a probability of less than 0.5, we change it to 0 and apply another softmax to recalculate the probability of each label. Then, we choose the most likely entity type.

Let's see the result of the confidence threshold on CamemBERT over the dev set. Figure 4.6 shows the evolution of performance depending on the confidence threshold. Similarly, Figure 4.7 shows performance change with the NTE metrics depending on the confidence threshold. For more details, see Tables I-12 and I-13.

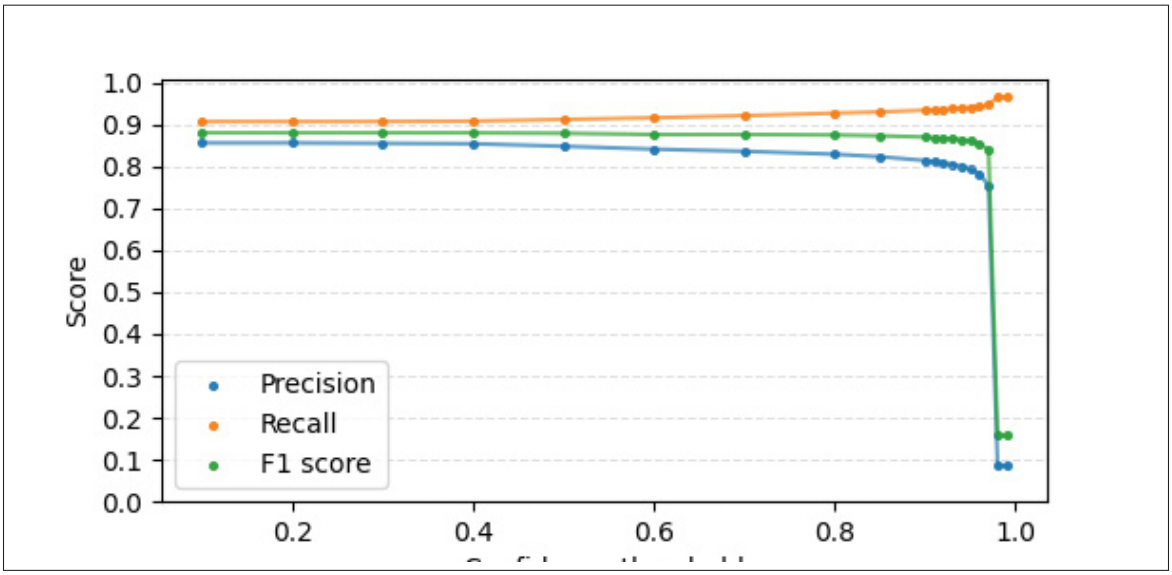


Figure 4.6 CamemBERT performance on FrenNER dev set depending on the confidence threshold

As we can see, the effect of the confidence threshold on the performance is very similar. In Table I-13, we have a NTE recall of 0.974 with a confidence threshold of 0.97. It is very good,

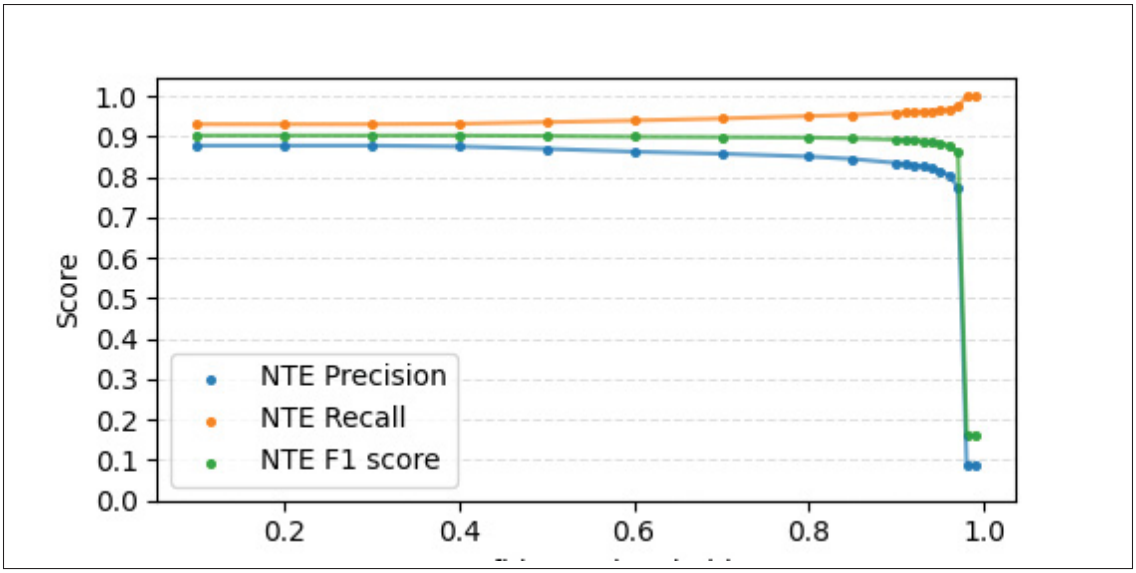


Figure 4.7 CamemBERT NTE performance on FrenNER dev set depending on the confidence threshold

but the precision is down to 0.775. Thus, the NTE F1 score is now at 0.863 instead of 0.903 with no threshold.

Let's choose a threshold that affects the NTE F1 score with a maximum of 0.01. With a threshold of 0.90, we have a NTE F1 score of 0.893, a NTE precision of 0.835, and a NTE recall of 0.959. In other words, we found almost 96% of all entities in our dev set, and around 84% of our predictions are good.

Finally, the model performance on the test set using a confidence threshold of 0.9 is shown in Table 4.3. If we compare with Table 4.1, the model lost 4.4% precision to gain 5.0% recall without affecting the F1 score. Also, the difference between the NTE results are almost the same.

#### 4.2.1.4 Voting ensemble phase

Soft voting is an ensemble algorithm where we average the predicted probabilities of each label. We use this technique because it is recommended for well-calibrated models. After creating this new model, we evaluated it on the whole FrenNER dataset. We expect the performance to be

Table 4.3 CamemBERT performance on FrenNER test set with a confidence threshold of 0.9

| Entity Type | Precision | Recall | F1 Score |
|---|---|---|---|
| Person | 0.939 ± 0.020 | 0.963 ± 0.017 | 0.951 ± 0.016 |
| Currency | 0.848 ± 0.048 | 0.949 ± 0.035 | 0.895 ± 0.039 |
| Location | 0.878 ± 0.032 | 0.940 ± 0.014 | 0.908 ± 0.019 |
| MoneyAmount | 0.818 ± 0.029 | 0.969 ± 0.021 | 0.887 ± 0.019 |
| Organization | 0.759 ± 0.024 | 0.914 ± 0.024 | 0.829 ± 0.017 |
| Total | 0.821 ± 0.015 | 0.935 ± 0.015 | 0.874 ± 0.010 |
| NTE | 0.842 ± 0.016 | 0.960 ± 0.012 | 0.897 ± 0.010 |

higher than those obtained in the previous phases because the ensemble model used this data for training. So, it does not adequately represent the performance of the model. Nonetheless, Tables 4.4 and 4.5 show the results obtained on FrenchNER with the soft voting ensemble.

Table 4.4 CamemBERT soft voting ensemble performance on FrenNER

| Entity Type | Precision | Recall | F1 Score |
|---|---|---|---|
| Person | 0.973 | 0.977 | 0.975 |
| Currency | 0.880 | 0.952 | 0.915 |
| Location | 0.942 | 0.968 | 0.955 |
| MoneyAmount | 0.864 | 0.966 | 0.912 |
| Organization | 0.949 | 0.959 | 0.954 |
| Total | 0.936 | 0.964 | 0.950 |
| NTE | 0.944 | 0.973 | 0.958 |

### 4.2.2 Discussion

As we can see in Figure 4.5, CamemBERT had a significant advantage over the BiLSTM-CRFs. We think it is because it was previously pre-trained on a lot of data. Thus, the model has already learned a good embedding space for French words, which is not the case with BiLSTM-CRFs. In future work, we could retry the experiments with pre-trained word embeddings to see if they improve the performance of BiLSTM-CRFs.

Table 4.5 CamemBERT soft voting ensemble performance on FrenNER with a confidence threshold of 0.9

| Entity Type | Precision | Recall | F1 Score |
|---|---|---|---|
| Person | 0.972 | 0.974 | 0.958 |
| Currency | 0.878 | 0.954 | 0.914 |
| Location | 0.942 | 0.969 | 0.955 |
| MoneyAmount | 0.860 | 0.966 | 0.910 |
| Organization | 0.947 | 0.962 | 0.954 |
| Total | 0.934 | 0.966 | 0.950 |
| NTE | 0.942 | 0.974 | 0.958 |

Also, as we can see in Tables 4.1 and 4.2, the Word+char-level BiLSTM-CRF did worst than the Word-level BiLSTM-CRF. Maybe the Word+char-level model is too large for the amount of data we have, so the model overfits the training set and gives poor results on the test set. In other words, the model could not generalize well on new data. On the other hand, the word-level only model has a lower capacity and thus performs better on our dataset. In future work, we could train all three models on a bigger dataset to see if it solves the overfitting problem.

In addition, we saw that a confidence threshold of 0.9 improves the recall considerably, but it also decreases the precision. This hyperparameter needs to be adjusted depending on the performance of the final pipeline during the second stage.

Finally, we see that the confidence threshold with the soft voting ensemble model does not improve recall. As stated earlier, we think it is because all of this data was used to train the models. Thus, the model is biased, and it memorized almost every named entity. To better evaluate the ensemble model, we could have split the data before creating our groups for cross-validation. However, because we do not have a lot of data and because we evaluate this ensemble model in the next chapter, we decided to keep the maximum amount of data possible for training.

## 4.3 Summary

In summary, we did two experiments to decide which models to use in our final pipeline. Firstly, we chose MFA as our FA algorithm because it has the best accuracy over the two algorithms. Secondly, we chose CamemBERT as our NER model because it was previously pre-trained on a lot of data. Thus, it outperforms other models and is faster to fine-tune new data than LSTM-based models.

# CHAPTER 5

# STAGE TWO: ANONYMIZING AUDIO RECORDINGS

This section shows how to use the pipeline and its results on the speech corpus.

## 5.1 User manual

The architecture of the pipeline is shown in Figure 5.1. The input is given to the FA and NER models. They output the aligned transcripts and the named entities used by the redaction function. Then, the evaluation function compares the redacted audio recordings with the gold annotations in order to evaluate the performance of the pipeline.

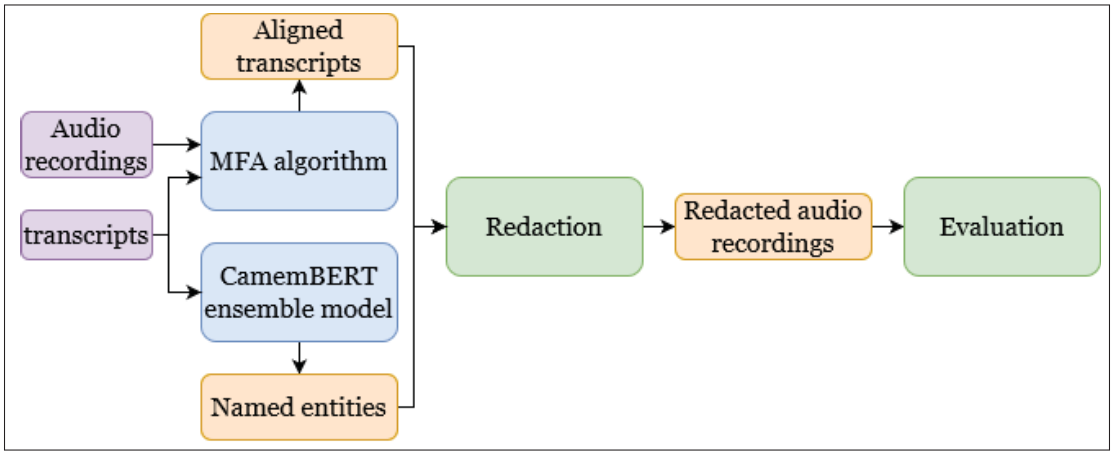


Figure 5.1 Final pipeline architecture overview

In order to make the pipeline easy to use, we decided to create a Docker image with everything needed to anonymize audio recordings. To run the container, you must use command :

```
docker run -it -v <INPUT>:/input
-v <ALIGN>:/align
-v <MODELS>:/ner_models
-v <REDACT>:/redact
-v <GOLD>:/gold
--gpus device=<GPU> <DOCKER>
```

This command has seven parameters :

1. **INPUT**: Path to the directory containing the audio recordings with their transcript.
2. **ALIGN**: Path to the directory used by the FA algorithm to store the alignment between the transcript and the audio.
3. **MODELS**: Path to the directory containing the NER models.
4. **REDACT**: Path to the directory that will contain the redacted audio recordings.
5. **GOLD**: Path to the directory containing the gold annotations.
6. **GPU**: Ids of the GPU to use.
7. **DOCKER**: Name of the container.

The container runs a script that will align the transcript with the audio recording, find all named entities and create a new redacted audio recording for each recording present in **INPUT** (see Figure 5.2).

## 5.2 Results and discussion

We evaluated the pipeline on the 85 examples of the speech corpus (see Section 3.3.1). We used a tolerance $t$ of 0.25 seconds. For more explanations on why we have chosen this tolerance, see Section 4.1.1.1. We also evaluated the pipeline with a confidence of 0.9, as established in Section 4.2.1.3. Table 5.1 shows the performance of the pipeline on the speech corpus. Also, every named entity and its corresponding type of prediction made by the pipeline is available in Tables II-1, II-2, and II-3.

Table 5.1 Pipeline performance on the speech corpus

| Confidence | Recall | Precision | F1 score |
|---|---|---|---|
| None | 0.631 | 0.985 | 0.769 |
| 0.9 | 0.631 | 0.985 | 0.769 |

As we can see, the confidence threshold over the "Not an entity" label has no impact on the performance of our pipeline. It means that the model is always certain that the word it predicted

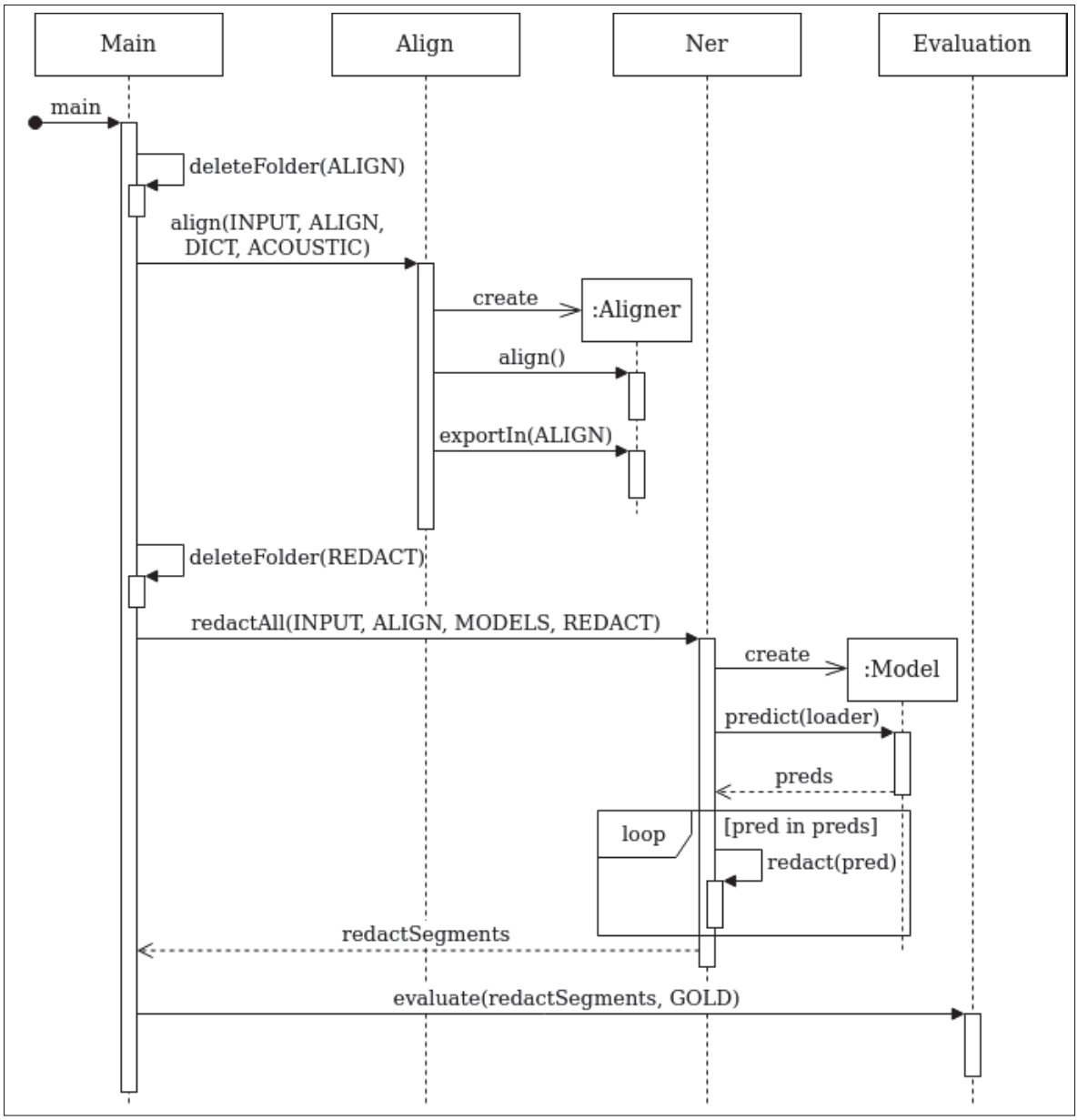


Figure 5.2 Pipeline sequence diagram

as an entity is one. Combined with the fact that the precision is very high and that the recall is very low on the speech corpus, we suspect the speech corpus to be too small.

Since the training and validation sets both contains the same named entities, it is hard to say if the models overfit the training set. This assumption has been confirmed when evaluating the pipeline on our audio dataset. Indeed, we obtained results comparable to Ghannay *et al.* (2018).

However, we need to keep in mind that their model performs a more complicated task, ASR instead of FA. So, it is normal that their pipeline performance is worse than ours. Like us, their precision is way higher than their recall. While the difference between their precision and recall is around 26%, our difference is about 56%. At first, we thought that the pipeline was not good at finding previously unseen named entities; it could have explained the low recall. In other terms, we thought that the NER model overfits the FrenchNER training set.

However, as shown in Figure 5.3, true positives are not only previously seen entities, and false positive are not only previously unseen entities. Thus, we do not think that the source of error comes from overfitting.

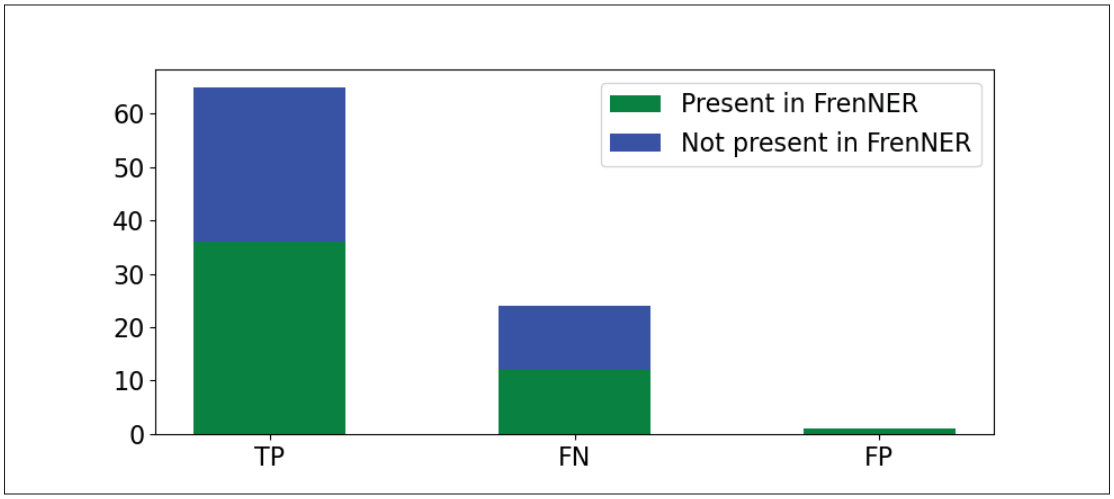


Figure 5.3 Relation between the type of prediction and the presence of the word in FrenchNER

After analyzing the data, we think the error is due to the sentence syntax. Here are two examples of entities successfully found by the pipeline:

1. *ben la **Lazio** c' est le hum le club de **Mussolini**.*
2. *et ils montraient justement un un chef de **Sochaux** et un chef de de **Lyon**.*

As we can see, the sentence syntax is good even though there are filling words or stuttering. Now, let see some examples where the pipeline could not find the entities :

1. *parce qu' après enfin moi moi les autres questions **Al Gore** son prix **Nobel** de la paix je m' en fous*.
2. *hum le club le plus raciste quoi de l' **Italie** un des plus racistes en tous cas*.

In both examples, the syntax is not good. When we read the sentences, it looks like two merged sentences. In these situations, the model cannot detect named entities, and it is expected since the NER model was trained on written text and not on speech.

We found two solutions to improve the performance of the NER component. The first solution would be to develop an algorithm to split sentences into phrases when the syntax is not good, so that the input will be more similar to written text. Also, another solution would be to train the NER model directly on speech with NER annotations, so that the model will learn examples with that kind of noise. Improving this component would enhance the pipeline's overall performance since there is no error due to the FA component.

Finally, it is important to note the source of error during this experiment. As we said in Section 4.1.2, the interval annotations may not be perfect. Therefore, even though the impact of this error while evaluating the pipeline is smaller than while we were evaluating the FA algorithm, it is not negligible.

## 5.3 Summary

In summary, we created a Docker image which is a simple way to use the whole pipeline. We have shown how to use that image and the required parameters. Then, we tested our pipeline on our speech corpus, and we obtained a F1 score of 76.9%. This Proof of Concept (PoC) has promising results and shows that anonymizing audio recording is a feasible task.

# CONCLUSION AND RECOMMENDATIONS

In this thesis, we wanted to create a proof of concept to know if it is possible to anonymize French audio by removing named entities from it. Our proof of concept consists of a pipeline with two components : a forced alignment algorithm followed by a named entity recognition model.

Our first intermediate goal was to determine the best French FA algorithm to align each word with the audio recording. We compared two forced alignment algorithms on our speech corpus with word-level boundaries in French and chose to keep MFA because it had the best accuracy. We did force alignment instead of automatic speech recognition since we did not have the resources to train an ASR model.

Then, our second intermediate goal was to train a French NER model. We trained multiple named entity recognition models on a French corpus of financial news: from LSTM-based model using word-level embedding to pre-trained Transformers. We compared them and chose CamemBERT as our named entity recognition model for two reasons. Firstly, when data is limited, the pre-training phase already done helps a lot. Secondly, the model had the best performance on FrenNER. That being said, the ensemble model consisting of ten CamemBERT models could not be evaluated adequately. Since we assumed that the pipeline evaluation would be enough to evaluate the named entity recognition model and the dataset was small. We decided to use it entirely to train the models.

Afterward, we created the pipeline consisting of three steps :

- Aligning the transcription with the audio with the MFA algorithm.
- Finding the named entities in the transcription with our CamemBERT ensemble model.
- Redacting the audio by replacing the named entities with noise.

The pipeline is usable via a Docker image. We evaluated it on the speech corpus that contains named entity annotations. It obtained a F1 score of 76.9% with a precision of 98.5% and a recall of 63.1%.

In future work, we think the first thing to do is improve the training set by creating a more extensive speech corpus with NER annotations. Indeed, one limitation of this research is the size of our speech corpus. Then, the new speech corpus could be used to train the NER model to be more robust to syntax errors and it could be trained to detect custom named entities (e.g. bank account, SIN).

Another path to explore would be to replace forced alignment with automatic speech recognition and to train an end-to-end NER model from speech. Given a good training dataset and enough resources, the performance of the end-to-end model should be better than the pipeline with separately trained components.

Finally, instead of replacing named entities with a silence, we could replace it with a randomly selected named entities of the same type. This will allow us to decrease the quantity of information deleted, while also removing sensitive information.

To conclude, our main objective was to determine if anonymizing French audio automatically by removing named entities was a feasible task. Our pipeline demonstrated that it is indeed a feasible task even if the resources are limited.

## APPENDIX I

## ADDITIONAL EXPERIMENT RESULTS

### 1. Forced alignment

Table-A I-1 Accuracies of FA models at different tolerances

| Tolerance (s) | sppas [std] | sppas [outer] | mfa [std] | mfa [outer] |
|---|---|---|---|---|
| ≤0.01 | 0.045 | 0.360 | 0.060 | 0.298 |
| ≤0.05 | 0.538 | 0.747 | 0.684 | 0.781 |
| ≤0.10 | 0.745 | 0.852 | 0.889 | 0.911 |
| ≤0.15 | 0.809 | 0.881 | 0.930 | 0.941 |
| ≤0.20 | 0.846 | 0.899 | **0.950** | 0.958 |
| ≤0.25 | 0.873 | 0.917 | 0.966 | 0.969 |
| ≤0.30 | 0.892 | 0.928 | 0.975 | **0.977** |
| ≤0.40 | 0.919 | 0.943 | 0.983 | 0.985 |
| ≤0.50 | 0.937 | **0.951** | 0.989 | 0.989 |
| ≤0.60 | **0.947** | 0.955 | 0.992 | 0.992 |
| ≤0.80 | 0.958 | 0.961 | 0.998 | 0.999 |
| ≤1.00 | 0.960 | 0.961 | 0.999 | 0.999 |
| ≤1.25 | 0.961 | 0.961 | 0.999 | 0.999 |
| ≤1.50 | 0.962 | 0.962 | 0.999 | 0.999 |
| ≤1.75 | 0.962 | 0.962 | 0.999 | 0.999 |
| ≤2.00 | 0.962 | 0.962 | 0.999 | 0.999 |

Table-A I-2 *Outer* accuracies of FA models on different sample (sam) and sub-samples (sub) sizes with a tolerance of 0.25 seconds

| Sample size | Sam sppas [outer] | Sub sppas [outer] | Sam mfa [outer] | Sub mfa [outer] |
|---|---|---|---|---|
| 5 | 0.923 | 0.911 ± 0.0174 | 0.949 | 0.962 ± 0.0134 |
| 10 | 0.937 | 0.924 ± 0.0179 | 0.951 | 0.968 ± 0.0131 |
| 15 | 0.922 | 0.923 ± 0.0153 | 0.969 | 0.972 ± 0.0121 |
| 20 | 0.927 | 0.904 ± 0.0201 | 0.965 | 0.971 ± 0.0099 |
| 25 | 0.911 | 0.923 ± 0.0274 | 0.968 | 0.964 ± 0.0127 |
| 30 | 0.925 | 0.913 ± 0.0180 | 0.972 | 0.970 ± 0.0091 |
| 35 | 0.911 | 0.929 ± 0.0100 | 0.975 | 0.967 ± 0.0095 |
| 40 | 0.910 | 0.922 ± 0.0218 | 0.976 | 0.969 ± 0.0084 |
| 45 | 0.910 | 0.929 ± 0.0218 | 0.978 | 0.968 ± 0.0112 |
| 50 | 0.915 | 0.916 ± 0.0237 | 0.975 | 0.972 ± 0.0141 |
| 55 | 0.919 | 0.912 ± 0.0196 | 0.976 | 0.968 ± 0.0116 |
| 60 | 0.919 | 0.913 ± 0.0190 | 0.975 | 0.971 ± 0.0105 |
| 65 | 0.918 | 0.915 ± 0.0166 | 0.974 | 0.968 ± 0.0102 |
| 70 | 0.921 | 0.922 ± 0.0206 | 0.973 | 0.962 ± 0.0121 |
| 75 | 0.922 | 0.916 ± 0.0112 | 0.972 | 0.968 ± 0.0137 |
| 80 | 0.916 | 0.929 ± 0.0224 | 0.968 | 0.975 ± 0.0070 |
| 85 | 0.918 | 0.919 ± 0.0196 | 0.969 | 0.967 ± 0.0104 |

## 2. Named entity recognition

### 2.1 Word-level BiLSTM-CRF

Table-A I-3 Word-level BiLSTM-CRF performance on FrenNER dev set at each epoch

| Epoch | Loss | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 0 | 9.518 ± 0.069 | 0.0 ± 0.0 | 0.0 ± 0.0 | 0.0 ± 0.0 |
| 1 | 6.781 ± 0.205 | 0.588 ± 0.093 | 0.239 ± 0.094 | 0.322 ± 0.06 |
| 2 | 4.475 ± 0.06 | 0.555 ± 0.173 | 0.4 ± 0.096 | 0.428 ± 0.111 |
| 3 | 3.588 ± 0.036 | 0.638 ± 0.048 | 0.443 ± 0.073 | 0.518 ± 0.048 |
| 4 | 3.088 ± 0.06 | 0.681 ± 0.059 | 0.503 ± 0.058 | 0.575 ± 0.041 |
| 5 | 2.712 ± 0.033 | 0.702 ± 0.037 | 0.562 ± 0.085 | 0.618 ± 0.05 |
| 6 | 2.411 ± 0.034 | 0.728 ± 0.051 | 0.592 ± 0.059 | 0.649 ± 0.029 |
| 7 | 2.18 ± 0.026 | 0.758 ± 0.041 | 0.666 ± 0.06 | 0.706 ± 0.018 |
| 8 | 1.991 ± 0.032 | 0.759 ± 0.064 | 0.678 ± 0.059 | 0.712 ± 0.023 |
| 9 | 1.839 ± 0.032 | 0.766 ± 0.063 | 0.697 ± 0.059 | 0.726 ± 0.027 |
| 10 | 1.69 ± 0.025 | 0.794 ± 0.035 | 0.708 ± 0.036 | 0.747 ± 0.019 |
| 11 | 1.58 ± 0.037 | 0.81 ± 0.047 | 0.72 ± 0.036 | 0.761 ± 0.018 |
| 12 | 1.496 ± 0.055 | 0.816 ± 0.032 | 0.723 ± 0.038 | 0.765 ± 0.015 |
| 13 | 1.4 ± 0.032 | 0.824 ± 0.028 | 0.719 ± 0.04 | 0.767 ± 0.023 |
| 14 | 1.31 ± 0.019 | 0.802 ± 0.054 | 0.746 ± 0.04 | 0.771 ± 0.015 |

Table-A I-4 Word-level BiLSTM-CRF NTE performance on FrenNER dev set at each epoch

| Epoch | Loss | NTE-Precision | NTE-Recall | NTE-F1 Score |
|---|---|---|---|---|
| 0 | 9.518 ± 0.069 | 0.0 ± 0.0 | 0.0 ± 0.0 | 0.0 ± 0.0 |
| 1 | 6.781 ± 0.205 | 0.842 ± 0.098 | 0.347 ± 0.142 | 0.465 ± 0.095 |
| 2 | 4.475 ± 0.06 | 0.79 ± 0.224 | 0.586 ± 0.166 | 0.614 ± 0.145 |
| 3 | 3.588 ± 0.036 | 0.869 ± 0.044 | 0.602 ± 0.088 | 0.705 ± 0.043 |
| 4 | 3.088 ± 0.06 | 0.884 ± 0.033 | 0.655 ± 0.081 | 0.748 ± 0.045 |
| 5 | 2.712 ± 0.033 | 0.871 ± 0.039 | 0.696 ± 0.101 | 0.767 ± 0.057 |
| 6 | 2.411 ± 0.034 | 0.872 ± 0.041 | 0.713 ± 0.079 | 0.78 ± 0.038 |
| 7 | 2.18 ± 0.026 | 0.86 ± 0.045 | 0.756 ± 0.072 | 0.801 ± 0.026 |
| 8 | 1.991 ± 0.032 | 0.852 ± 0.055 | 0.763 ± 0.073 | 0.8 ± 0.021 |
| 9 | 1.839 ± 0.032 | 0.854 ± 0.043 | 0.78 ± 0.078 | 0.811 ± 0.028 |
| 10 | 1.69 ± 0.025 | 0.871 ± 0.031 | 0.777 ± 0.044 | 0.82 ± 0.02 |
| 11 | 1.58 ± 0.037 | 0.871 ± 0.038 | 0.775 ± 0.048 | 0.818 ± 0.016 |
| 12 | 1.496 ± 0.055 | 0.877 ± 0.023 | 0.778 ± 0.051 | 0.823 ± 0.023 |
| 13 | 1.4 ± 0.032 | 0.881 ± 0.026 | 0.768 ± 0.035 | 0.82 ± 0.014 |
| 14 | 1.31 ± 0.019 | 0.863 ± 0.047 | 0.804 ± 0.056 | 0.83 ± 0.02 |

Table-A I-5 Word-level BiLSTM-CRF performance on FrenNER test set

| Entity Type | Precision | Recall | F1 Score |
|---|---|---|---|
| Person | 0.876 ± 0.031 | 0.844 ± 0.051 | 0.859 ± 0.035 |
| Currency | 0.878 ± 0.025 | 0.918 ± 0.058 | 0.897 ± 0.036 |
| MoneyAmount | 0.816 ± 0.051 | 0.851 ± 0.062 | 0.831 ± 0.033 |
| Location | 0.841 ± 0.074 | 0.812 ± 0.050 | 0.823 ± 0.047 |
| Organization | 0.756 ± 0.038 | 0.678 ± 0.080 | 0.710 ± 0.035 |
| Total | 0.808 ± 0.029 | 0.768 ± 0.038 | 0.786 ± 0.013 |
| NTE | 0.853 ± 0.032 | 0.811 ± 0.038 | 0.830 ± 0.012 |

### 2.2 Word+char-level BiLSTM-CRF

Table-A I-6 Word+char-level BiLSTM-CRF performance on FrenNER dev set at each epoch on group

| Epoch | Loss | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 0 | 9.465 ± 0.121 | 0.073 ± 0.218 | 0.001 ± 0.003 | 0.002 ± 0.005 |
| 1 | 6.717 ± 0.279 | 0.577 ± 0.085 | 0.242 ± 0.078 | 0.328 ± 0.06 |
| 2 | 4.47 ± 0.114 | 0.607 ± 0.079 | 0.393 ± 0.062 | 0.469 ± 0.034 |
| 3 | 3.532 ± 0.044 | 0.638 ± 0.042 | 0.46 ± 0.059 | 0.531 ± 0.038 |
| 4 | 3.004 ± 0.046 | 0.661 ± 0.055 | 0.538 ± 0.071 | 0.588 ± 0.04 |
| 5 | 2.65 ± 0.042 | 0.654 ± 0.191 | 0.574 ± 0.07 | 0.583 ± 0.146 |
| 6 | 2.378 ± 0.031 | 0.749 ± 0.056 | 0.62 ± 0.048 | 0.675 ± 0.024 |
| 7 | 2.145 ± 0.036 | 0.742 ± 0.046 | 0.638 ± 0.08 | 0.68 ± 0.038 |
| 8 | 1.959 ± 0.032 | 0.782 ± 0.031 | 0.665 ± 0.034 | 0.717 ± 0.015 |
| 9 | 1.808 ± 0.036 | 0.749 ± 0.063 | 0.722 ± 0.047 | 0.731 ± 0.024 |
| 10 | 1.681 ± 0.04 | 0.792 ± 0.053 | 0.706 ± 0.048 | 0.743 ± 0.021 |
| 11 | 1.564 ± 0.052 | 0.794 ± 0.039 | 0.722 ± 0.037 | 0.755 ± 0.027 |
| 12 | 1.476 ± 0.033 | 0.786 ± 0.032 | 0.753 ± 0.054 | 0.767 ± 0.024 |
| 13 | 1.39 ± 0.037 | 0.788 ± 0.074 | 0.758 ± 0.037 | 0.769 ± 0.023 |
| 14 | 1.3 ± 0.039 | 0.807 ± 0.045 | 0.755 ± 0.041 | 0.778 ± 0.015 |
| 15 | 1.241 ± 0.033 | 0.825 ± 0.047 | 0.741 ± 0.037 | 0.779 ± 0.019 |
| 16 | 1.193 ± 0.037 | 0.821 ± 0.041 | 0.751 ± 0.04 | 0.783 ± 0.017 |
| 17 | 1.131 ± 0.029 | 0.828 ± 0.036 | 0.74 ± 0.052 | 0.779 ± 0.02 |
| 18 | 1.09 ± 0.035 | 0.777 ± 0.075 | 0.781 ± 0.056 | 0.774 ± 0.022 |
| 19 | 1.042 ± 0.037 | 0.822 ± 0.035 | 0.777 ± 0.036 | 0.797 ± 0.013 |
| 20 | 0.988 ± 0.045 | 0.827 ± 0.046 | 0.753 ± 0.058 | 0.785 ± 0.024 |
| 21 | 0.968 ± 0.041 | 0.812 ± 0.062 | 0.765 ± 0.052 | 0.785 ± 0.025 |
| 22 | 0.903 ± 0.029 | 0.802 ± 0.041 | 0.792 ± 0.037 | 0.795 ± 0.021 |
| 23 | 0.885 ± 0.023 | 0.808 ± 0.052 | 0.745 ± 0.098 | 0.77 ± 0.058 |
| 24 | 0.861 ± 0.02 | 0.812 ± 0.047 | 0.789 ± 0.033 | 0.799 ± 0.021 |

Table-A I-7 Word+char-level BiLSTM-CRF NTE performance on FrenNER dev set at each epoch on group

| Epoch | Loss | NTE-Precision | NTE-Recall | NTE-F1 Score |
|---|---|---|---|---|
| 0 | 9.465 ± 0.121 | 0.073 ± 0.218 | 0.001 ± 0.003 | 0.002 ± 0.005 |
| 1 | 6.717 ± 0.279 | 0.847 ± 0.084 | 0.362 ± 0.132 | 0.486 ± 0.1 |
| 2 | 4.47 ± 0.114 | 0.847 ± 0.069 | 0.555 ± 0.11 | 0.659 ± 0.065 |
| 3 | 3.532 ± 0.044 | 0.869 ± 0.051 | 0.625 ± 0.071 | 0.722 ± 0.037 |
| 4 | 3.004 ± 0.046 | 0.859 ± 0.056 | 0.698 ± 0.086 | 0.764 ± 0.041 |
| 5 | 2.65 ± 0.042 | 0.807 ± 0.223 | 0.728 ± 0.119 | 0.722 ± 0.161 |
| 6 | 2.378 ± 0.031 | 0.874 ± 0.046 | 0.725 ± 0.065 | 0.789 ± 0.027 |
| 7 | 2.145 ± 0.036 | 0.862 ± 0.051 | 0.741 ± 0.09 | 0.791 ± 0.039 |
| 8 | 1.959 ± 0.032 | 0.879 ± 0.029 | 0.748 ± 0.045 | 0.807 ± 0.022 |
| 9 | 1.808 ± 0.036 | 0.833 ± 0.059 | 0.805 ± 0.058 | 0.815 ± 0.02 |
| 10 | 1.681 ± 0.04 | 0.861 ± 0.052 | 0.768 ± 0.05 | 0.809 ± 0.012 |
| 11 | 1.564 ± 0.052 | 0.861 ± 0.025 | 0.784 ± 0.041 | 0.82 ± 0.018 |
| 12 | 1.476 ± 0.033 | 0.852 ± 0.039 | 0.815 ± 0.05 | 0.831 ± 0.015 |
| 13 | 1.39 ± 0.037 | 0.846 ± 0.064 | 0.816 ± 0.055 | 0.827 ± 0.016 |
| 14 | 1.3 ± 0.039 | 0.862 ± 0.044 | 0.806 ± 0.047 | 0.831 ± 0.014 |
| 15 | 1.241 ± 0.033 | 0.875 ± 0.037 | 0.786 ± 0.044 | 0.826 ± 0.012 |
| 16 | 1.193 ± 0.037 | 0.873 ± 0.035 | 0.8 ± 0.047 | 0.833 ± 0.014 |
| 17 | 1.131 ± 0.029 | 0.873 ± 0.032 | 0.781 ± 0.059 | 0.822 ± 0.023 |
| 18 | 1.09 ± 0.035 | 0.826 ± 0.071 | 0.832 ± 0.068 | 0.823 ± 0.019 |
| 19 | 1.042 ± 0.037 | 0.868 ± 0.03 | 0.82 ± 0.041 | 0.842 ± 0.012 |
| 20 | 0.988 ± 0.045 | 0.874 ± 0.041 | 0.796 ± 0.063 | 0.83 ± 0.023 |
| 21 | 0.968 ± 0.041 | 0.861 ± 0.049 | 0.813 ± 0.067 | 0.833 ± 0.024 |
| 22 | 0.903 ± 0.029 | 0.848 ± 0.035 | 0.839 ± 0.042 | 0.842 ± 0.016 |
| 23 | 0.885 ± 0.023 | 0.86 ± 0.052 | 0.791 ± 0.095 | 0.818 ± 0.05 |
| 24 | 0.861 ± 0.02 | 0.856 ± 0.044 | 0.832 ± 0.038 | 0.842 ± 0.02 |

Table-A I-8 Word+char-level BiLSTM-CRF performance on FrenNER test set

| Entity Type | Precision | Recall | F1 Score |
|---|---|---|---|
| Person | 0.847 ± 0.065 | 0.801 ± 0.080 | 0.820 ± 0.049 |
| Currency | 0.861 ± 0.042 | 0.927 ± 0.055 | 0.893 ± 0.047 |
| MoneyAmount | 0.799 ± 0.046 | 0.879 ± 0.047 | 0.835 ± 0.030 |
| Location | 0.851 ± 0.066 | 0.775 ± 0.063 | 0.810 ± 0.058 |
| Organization | 0.749 ± 0.073 | 0.606 ± 0.138 | 0.655 ± 0.070 |
| Total | 0.800 ± 0.051 | 0.726 ± 0.059 | 0.757 ± 0.017 |
| NTE | 0.858 ± 0.042 | 0.779 ± 0.073 | 0.813 ± 0.022 |

### 2.3 CamemBERT

Table-A I-9 Camembert performance on FrenNER dev set at each epoch

| Epoch | Loss | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 0 | 6.679 ± 0.239 | 0.227 ± 0.232 | 0.12 ± 0.126 | 0.156 ± 0.162 |
| 1 | 2.794 ± 0.291 | 0.825 ± 0.05 | 0.811 ± 0.06 | 0.815 ± 0.026 |
| 2 | 1.444 ± 0.083 | 0.855 ± 0.03 | 0.868 ± 0.026 | 0.861 ± 0.023 |
| 3 | 1.051 ± 0.043 | 0.857 ± 0.026 | 0.884 ± 0.017 | 0.87 ± 0.015 |
| 4 | 0.896 ± 0.03 | 0.866 ± 0.023 | 0.885 ± 0.019 | 0.875 ± 0.015 |

Table-A I-10 Camembert NTE performance on FrenNER dev set at each epoch

| Epoch | Loss | NTE-Precision | NTE-Recall | NTE-F1 Score |
|---|---|---|---|---|
| 0 | 6.679 ± 0.239 | 0.458 ± 0.458 | 0.241 ± 0.247 | 0.314 ± 0.318 |
| 1 | 2.794 ± 0.291 | 0.874 ± 0.051 | 0.858 ± 0.06 | 0.863 ± 0.019 |
| 2 | 1.444 ± 0.083 | 0.887 ± 0.025 | 0.9 ± 0.023 | 0.893 ± 0.017 |
| 3 | 1.051 ± 0.043 | 0.882 ± 0.025 | 0.911 ± 0.018 | 0.896 ± 0.015 |
| 4 | 0.896 ± 0.03 | 0.889 ± 0.021 | 0.909 ± 0.019 | 0.898 ± 0.013 |

Table-A I-11 CamemBERT performance on FrenNER test set

| Entity Type | Precision | Recall | F1 Score |
|---|---|---|---|
| Currency | 0.861 ± 0.036 | 0.917 ± 0.043 | 0.887 ± 0.031 |
| Location | 0.892 ± 0.035 | 0.918 ± 0.018 | 0.905 ± 0.021 |
| MoneyAmount | 0.843 ± 0.032 | 0.915 ± 0.049 | 0.876 ± 0.028 |
| Organization | 0.831 ± 0.025 | 0.836 ± 0.047 | 0.833 ± 0.027 |
| Person | 0.942 ± 0.020 | 0.949 ± 0.025 | 0.945 ± 0.019 |
| Total | 0.865 ± 0.014 | 0.885 ± 0.024 | 0.874 ± 0.013 |
| NTE | 0.889 ± 0.016 | 0.91 ± 0.023 | 0.899 ± 0.013 |

Table-A I-12 CamemBERT performance depending on the confidence threshold on FrenNER dev set

| Confidence threshold | Precision | Recall | F1 Score |
|---|---|---|---|
| 0.1 | 0.856 ± 0.022 | 0.907 ± 0.016 | 0.88 ± 0.014 |
| 0.2 | 0.856 ± 0.022 | 0.907 ± 0.016 | 0.88 ± 0.014 |
| 0.3 | 0.855 ± 0.022 | 0.907 ± 0.016 | 0.88 ± 0.014 |
| 0.4 | 0.854 ± 0.023 | 0.908 ± 0.016 | 0.88 ± 0.014 |
| 0.5 | 0.848 ± 0.023 | 0.912 ± 0.016 | 0.879 ± 0.016 |
| 0.6 | 0.841 ± 0.027 | 0.916 ± 0.016 | 0.876 ± 0.017 |
| 0.7 | 0.836 ± 0.028 | 0.921 ± 0.016 | 0.876 ± 0.02 |
| 0.8 | 0.829 ± 0.029 | 0.927 ± 0.015 | 0.875 ± 0.021 |
| 0.85 | 0.823 ± 0.028 | 0.93 ± 0.014 | 0.873 ± 0.02 |
| 0.9 | 0.814 ± 0.027 | 0.934 ± 0.013 | 0.87 ± 0.02 |
| 0.91 | 0.811 ± 0.028 | 0.935 ± 0.013 | 0.868 ± 0.02 |
| 0.92 | 0.808 ± 0.029 | 0.936 ± 0.014 | 0.867 ± 0.021 |
| 0.93 | 0.805 ± 0.029 | 0.938 ± 0.013 | 0.866 ± 0.021 |
| 0.94 | 0.8 ± 0.029 | 0.939 ± 0.013 | 0.864 ± 0.021 |
| 0.95 | 0.794 ± 0.03 | 0.941 ± 0.012 | 0.861 ± 0.021 |
| 0.96 | 0.783 ± 0.03 | 0.942 ± 0.011 | 0.855 ± 0.021 |
| 0.97 | 0.754 ± 0.029 | 0.947 ± 0.01 | 0.839 ± 0.02 |
| 0.98 | 0.086 ± 0.005 | 0.965 ± 0.008 | 0.158 ± 0.008 |
| 0.99 | 0.086 ± 0.005 | 0.965 ± 0.008 | 0.158 ± 0.008 |

Table-A I-13 CamemBERT NTE performance depending on the confidence threshold on FrenNER dev set

| Confidence threshold | NTE-Precision | NTE-Recall | NTE-F1 Score |
|---|---|---|---|
| 0.1 | 0.878 ± 0.022 | 0.931 ± 0.014 | 0.903 ± 0.012 |
| 0.2 | 0.878 ± 0.022 | 0.931 ± 0.014 | 0.903 ± 0.012 |
| 0.3 | 0.878 ± 0.021 | 0.931 ± 0.014 | 0.903 ± 0.012 |
| 0.4 | 0.876 ± 0.022 | 0.932 ± 0.015 | 0.903 ± 0.013 |
| 0.5 | 0.87 ± 0.023 | 0.936 ± 0.014 | 0.902 ± 0.014 |
| 0.6 | 0.863 ± 0.026 | 0.94 ± 0.013 | 0.9 ± 0.016 |
| 0.7 | 0.858 ± 0.028 | 0.945 ± 0.014 | 0.899 ± 0.018 |
| 0.8 | 0.851 ± 0.028 | 0.951 ± 0.012 | 0.898 ± 0.018 |
| 0.85 | 0.845 ± 0.028 | 0.954 ± 0.01 | 0.896 ± 0.018 |
| 0.9 | 0.835 ± 0.027 | 0.959 ± 0.011 | 0.893 ± 0.018 |
| 0.91 | 0.833 ± 0.027 | 0.96 ± 0.012 | 0.892 ± 0.019 |
| 0.92 | 0.83 ± 0.028 | 0.961 ± 0.012 | 0.891 ± 0.019 |
| 0.93 | 0.827 ± 0.029 | 0.963 ± 0.011 | 0.89 ± 0.02 |
| 0.94 | 0.821 ± 0.028 | 0.964 ± 0.011 | 0.887 ± 0.019 |
| 0.95 | 0.815 ± 0.029 | 0.966 ± 0.011 | 0.884 ± 0.019 |
| 0.96 | 0.804 ± 0.028 | 0.968 ± 0.01 | 0.878 ± 0.019 |
| 0.97 | 0.775 ± 0.028 | 0.974 ± 0.01 | 0.863 ± 0.018 |
| 0.98 | 0.089 ± 0.005 | 1.0 ± 0.0 | 0.163 ± 0.008 |
| 0.99 | 0.089 ± 0.005 | 1.0 ± 0.0 | 0.163 ± 0.008 |

## APPENDIX II

## ADDITIONAL PIPELINE RESULTS

This is the list of named entities with the type of prediction made (true positive, false negative or false positive).

Table-A II-1 Pipeline prediction for each named entity (1-30)

| Prediction | Named entity | Present in FrenNER |
|---|---|---|
| TP | Sochaux | Yes |
| TP | Lyon | Yes |
| TP | Argentine | Yes |
| TP | France | Yes |
| TP | Paris | Yes |
| FN | Ardèche | No |
| FN | Al Gore | No |
| FN | Nobel | Yes |
| FN | Zola | No |
| FN | Zola | No |
| FN | Zola | No |
| TP | Marseille | Yes |
| TP | Hollande | Yes |
| TP | Lazio | No |
| TP | Mussolini | No |
| TP | Erénatti | No |
| TP | Loire Atlantique | Partially |
| TP | Le Mans | No |
| FN | Cavanna | No |
| TP | Griselda | No |
| FN | CAPES | No |
| FN | CAPES | No |
| TP | Censier | No |
| TP | Caroline | No |
| FN | CPE | No |
| TP | CAF | No |
| FN | coupe d'Afrique des Nations | No |
| FN | l'ONF | No |
| TP | Liberia | No |
| TP | Nicolas Sarkozy | Yes |

Table-A II-2 Pipeline prediction for each named entity (31-68)

| Prediction | Named entity | Present in FrenNER |
|---|---|---|
| FN | Julie | No |
| FN | UNEF | No |
| TP | Sochaux | Yes |
| TP | Lyonnais | Yes |
| FN | Sorbonne | Yes |
| TP | Paris sept | Yes |
| FN | Censier | No |
| TP | Polonais | Yes |
| TP | Télérama | No |
| TP | Ronaldhino | No |
| FN | Parisiens | Yes |
| FN | Parisiens | Yes |
| FN | Italie | Yes |
| TP | Bordeaux | Yes |
| TP | Bordeaux | Yes |
| FN | CPE | No |
| TP | Boulogne | No |
| TP | Boris | Yes |
| TP | Nouvel Obs | No |
| FN | Paris | Yes |
| TP | Hollande | Yes |
| FN | FSEUL | No |
| TP | Paris | Yes |
| TP | Caen | No |
| FN | Carré | No |
| FN | DRAC | No |
| FN | CRAC | No |
| TP | PSG Macaby Haïfa | No |
| TP | Lens | No |
| FN | Charles | Yes |
| TP | Leguenn | No |
| FN | Fournier | No |
| FN | Roche | Yes |
| TP | France | Yes |
| TP | Hollande | Yes |
| FN | Star Trek | No |
| TP | Irlande | Yes |

Table-A II-3 Pipeline prediction for each named entity (69-106)

| Prediction | Named entity | Present in FrenNER |
|---|---|---|
| TP | Irlande | Yes |
| TP | France | Yes |
| FN | Krups | Yes |
| TP | Po | Yes |
| FN | Sarkozy | Yes |
| FN | Casino | Yes |
| FP | Hollande | Yes |
| TP | Paris | Yes |
| TP | Parisiens | Yes |
| FN | Carré | Yes |
| TP | Maximo Gargia | No |
| TP | quinze dollars | Partially |
| TP | Levi's | No |
| TP | quinze dollars | Partially |
| TP | Sarko | No |
| TP | Brest | Yes |
| TP | Quimper | No |
| TP | Camille | No |
| FN | LRU | No |
| TP | Luxembourg | Yes |
| TP | Thomas Gatlif | Partially |
| TP | Paris | Yes |
| TP | Sarkozy | Yes |
| TP | Jean-Pierre Mader | Partially |
| FN | Macumba | No |
| FN | LRU | No |
| TP | Macdo | No |
| TP | Luxembourg | Yes |
| FN | IUT | No |
| TP | Sorbonne | Yes |
| TP | Paul | Yes |
| TP | Tagada Jones | Partially |
| TP | FN | Yes |
| TP | Paris | Yes |
| TP | breton | Yes |
| TP | Bretagne | Yes |
| FN | PSEUL | No |

## LIST OF REFERENCES


Albawi, S., Mohammed, T. A. & Al-Zawi, S. (2017). Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, pp. 1-6. doi: 10.1109/ICEngTechnol.2017.8308186.

Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J. H., Fan, L., Fougner, C., Han, T., Hannun, A. Y., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A. Y., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J. & Zhu, Z. (2015). Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *CoRR*, abs/1512.02595. Consulted at http://arxiv.org/abs/1512.02595.

Ba, J. L., Kiros, J. R. & Hinton, G. E. (2016). Layer Normalization.

Bahmaninezhad, F., Zhang, C. & Hansen, J. (2018, 06). Convolutional Neural Network Based Speaker De-Identification. doi: 10.21437/Odyssey.2018-36.

Bengio, Y., Simard, P. & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 157-166. doi: 10.1109/72.279181.

Bigi, B. (2015). SPPAS - Multi-lingual Approaches to the Automatic Annotation of Speech. *The Phonetician*, 111-112, 54-69. Consulted at http://www.isphs.org/Phonetician/Phonetician_111-112.pdf#page=54.

Bigi, B. (2016). A phonetization approach for the forced-alignment task in SPPAS. *Human Language Technology. Challenges for Computer Science and Linguistics*, LNAI-9561, 397-410. Consulted at http://link.springer.com/chapter/10.1007%2F978-3-319-43808-5_30.

Bigi, B. & Meunier, C. (2018). Automatic segmentation of spontaneous speech. *Revista de Estudos da Linguagem*, 26(4).

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag.

Boersma, P. & Weenink, D. (2020). Praat: doing phonetics by computer (Version 6.0.37). Consulted at http://www.praat.org.

Cardinal, P. (2013). *Speech recognition on multi-core processors and GPUs*. (Ph.D. thesis, École de Technologie Supérieure).

Chiu, J. P. C. & Nichols, E. (2016). Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4, 357–370.

Cohn, I., Laish, I., Beryozkin, G., Li, G., Shafran, I., Szpektor, I., Hartman, T., Hassidim, A. & Matias, Y. (2019). Audio De-identification: A New Entity Recognition Task. *CoRR*, abs/1903.07037. Consulted at http://arxiv.org/abs/1903.07037.

Dernoncourt, F., Lee, J. Y., Uzuner, Ö. & Szolovits, P. (2017). De-identification of patient notes with recurrent neural networks. *Journal of the American Medical Informatics Association*, 24(3), 596–606.

Devlin, J., Chang, M., Lee, K. & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. doi: 10.18653/v1/n19-1423.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A. & Smith, N. A. (2015). Transition-Based Dependency Parsing with Stack Long Short-Term Memory. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 334–343. doi: 10.3115/v1/P15-1033.

Fang, F., Wang, X. Z., Yamagishi, J., Echizen, I., Todisco, M., Evans, N. W. D. & Bonastre, J.-F. (2019). Speaker Anonymization Using X-vector and Neural Waveform Models. *ArXiv*, abs/1905.13561.

Gales, M. (1998). Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech & Language*, 12(2), 75-98. doi: https://doi.org/10.1006/csla.1998.0043.

Ghannay, S., Caubrière, A., Estève, Y., Camelin, N., Simonnet, E., Laurent, A. & Morin, E. (2018). End-to-end named entity and semantic concept extraction from speech. *2018 IEEE Spoken Language Technology Workshop (SLT)*, pp. 692–699.

Goodenough-Trepagnier, C. & Frankston, R. (1978). Étude sur la distribution des syllabes en français. *Revue québécoise de linguistique*, 7, 43–70.

Graves, A. & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5), 602-610. doi: https://doi.org/10.1016/j.neunet.2005.06.042. IJCNN 2005.

He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation*, 9, 1735-80. doi: 10.1162/neco.1997.9.8.1735.

Huang, Z., Xu, W. & Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging. *CoRR*, abs/1508.01991. Consulted at http://arxiv.org/abs/1508.01991.

Jabbari, A., Sauvage, O., Zeine, H. & Chergui, H. (2020). A French Corpus and Annotation Schema for Named Entity Recognition and Relation Extraction of Financial News. *Proceedings of the 12th Language Resources and Evaluation Conference*, pp. 2293–2299. Consulted at https://aclanthology.org/2020.lrec-1.279.

Kopparapu, S. K. & Laxminarayana, M. (2010). Choice of Mel filter bank in computing MFCC of a resampled speech. *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, pp. 121-124. doi: 10.1109/ISSPA.2010.5605491.

Kudo, T. & Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71. doi: 10.18653/v1/D18-2012.

Lafferty, J. D., McCallum, A. & Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the Eighteenth International Conference on Machine Learning*, (ICML '01), 282–289.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K. & Dyer, C. (2016). Neural Architectures for Named Entity Recognition. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 260–270. doi: 10.18653/v1/N16-1030.

Lancien, M., Côté, M.-H. & Bigi, B. (2020). Developing Resources for Automated Speech Processing of Quebec French. *Proceedings of The 12th Language Resources and Evaluation Conference*, pp. 5323–5328. Consulted at https://www.aclweb.org/anthology/2020.lrec-1.655.

Le, H., Vial, L., Frej, J., Segonne, V., Coavoux, M., Lecouteux, B., Allauzen, A., Crabbé, B., Besacier, L. & Schwab, D. (2020, May). FlauBERT: Unsupervised Language Model Pre-training for French. *Proceedings of The 12th Language Resources and Evaluation Conference*, pp. 2479–2490. Consulted at https://www.aclweb.org/anthology/2020.lrec-1.302.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W. & Jackel, L. (1990). Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*, 2. Consulted at https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

Lee, A. & Kawahara, T. (2009). Recent Development of Open-Source Speech Recognition Engine Julius. *em Proceedings of the 2009 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*.

Lee, A., Kawahara, T. & Doshita, S. (1998). An efficient two-pass search algorithm using word trellis index. *ICSLP*, 98, 1831–1834.

Li, X., Michel, P., Anastasopoulos, A., Belinkov, Y., Durrani, N., Firat, O., Koehn, P., Neubig, G., Pino, J. & Sajjad, H. (2019). Findings of the First Shared Task on Machine Translation Robustness. *CoRR*, abs/1906.11943. Consulted at http://arxiv.org/abs/1906.11943.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692. Consulted at http://arxiv.org/abs/1907.11692.

Liu, Z., Tang, B., Wang, X. & Chen, Q. (2017). De-identification of clinical notes via recurrent neural network and conditional random field. *Journal of Biomedical Informatics*, 75, S34-S42. doi: https://doi.org/10.1016/j.jbi.2017.05.023. Supplement: A Natural Language Processing Challenge for Clinical Records: Research Domains Criteria (RDoC) for Psychiatry.

Lugosch, L., Ravanelli, M., Ignoto, P., Tomar, V. S. & Bengio, Y. (2019). Speech Model Pre-Training for End-to-End Spoken Language Understanding. *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pp. 814–818. doi: 10.21437/Interspeech.2019-2396.

Martin, L., Muller, B., Ortiz Suárez, P. J., Dupont, Y., Romary, L., Villemonte de La Clergerie, É., Seddah, D. & Sagot, B. (2020). CamemBERT: a Tasty French Language Model. *ACL 2020 - 58th Annual Meeting of the Association for Computational Linguistics*. doi: 10.18653/v1/2020.acl-main.645.

McAuliffe, M., Socolof, M., Mihuc, S., Wagner, M. & Sonderegger, M. (2017). Montreal Forced Aligner: Trainable Text-Speech Alignment Using Kaldi. *Proc. Interspeech 2017*, pp. 498-502. doi: 10.21437/Interspeech.2017-1386.

Mohri, M. & Pereira, F. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1), 69-88. doi: https://doi.org/10.1006/csla.2001.0184.

Mohri, M., Pereira, F. & Riley, M. (2008). Speech Recognition with Weighted Finite-State Transducers. In Benesty, J., Sondhi, M. M. & Huang, Y. A. (Eds.), *Springer Handbook of Speech Processing* (pp. 559–584). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-540-49127-9_28.

Neamatullah, I., Douglass, M. M., Lehman, L.-w. H., Reisner, A., Villarroel, M., Long, W. J., Szolovits, P., Moody, G. B., Mark, R. G. & Clifford, G. D. (2008). Automated de-identification of free-text medical records. *BMC Medical Informatics and Decision Making*, 8(1), 32. doi: 10.1186/1472-6947-8-32.

Olah, C. (2015, August). Understanding LSTM Networks. Consulted at https://jonathan-hui.medium.com/speech-recognition-gmm-hmm-8bb5eff8b196.

Ortiz Suárez, P. J., Sagot, B. & Romary, L. (2019). Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures. (Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-7) 2019. Cardiff, 22nd July 2019), 9 – 16. doi: 10.14618/ids-pub-9021.

O'Shea, K. & Nash, R. (2015). An Introduction to Convolutional Neural Networks. *CoRR*, abs/1511.08458. Consulted at http://arxiv.org/abs/1511.08458.

Panayotov, V., Chen, G., Povey, D. & Khudanpur, S. (2015). Librispeech: An ASR corpus based on public domain audio books. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5206-5210. doi: 10.1109/ICASSP.2015.7178964.

Patino, J., Tomashenko, N., Todisco, M., Nautsch, A. & Evans, N. (2021). Speaker Anonymisation Using the McAdams Coefficient. *Interspeech 2021*, pp. 1099-1103. doi: 10.21437/Interspeech.2021-1070.

Pitt, M. A., Dilley, L., Johnson, K., Kiesling, S., Raymond, W., Hume, E. & Fosler-Lussier, E. (2007). Buckeye corpus of conversational speech (2nd release). *Columbus, OH: Department of Psychology, Ohio State University*, 265–270.

Povey, D. & Saon, G. (2006). Feature and model space speaker adaptation with full covariance gaussians. *Interspeech*, pp. 1145–1148.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G. & Vesely, K. (2011). The Kaldi Speech Recognition Toolkit. *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. Consulted at http://infoscience.epfl.ch/record/192584.

Prasad, N. V. & Umesh, S. (2013). Improved cepstral mean and variance normalization using Bayesian framework. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 156-161. doi: 10.1109/ASRU.2013.6707722.

Qian, Y., Bianv, X., Shi, Y., Kanda, N., Shen, L., Xiao, Z. & Zeng, M. (2021). Speech-language pre-training for end-to-end spoken language understanding. *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7458–7462.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257-286. doi: 10.1109/5.18626.

Radfar, M., Mouchtaris, A. & Kunzmann, S. (2020). End-to-End Neural Transformer Based Spoken Language Understanding. *Interspeech 2020*.

Ravanelli, M. & Bengio, Y. (2018). Speaker Recognition from Raw Waveform with SincNet. *2018 IEEE Spoken Language Technology Workshop (SLT)*, 1021-1028.

Rist, C. (1999). 200 mots à la minute : le débit oral des médias. 66–75. doi: 10.3406/colan.1999.2909.

Rosenfelder, I., Fruehwald, J., Evanini, K., Seyfarth, S., Gorman, K., Prichard, H. & Yuan, J. (2014). FAVE (Forced Alignment and Vowel Extraction) Suite Version 1.1.3. doi: 10.5281/zenodo.9846.

Sennrich, R., Haddow, B. & Birch, A. (2016). Neural Machine Translation of Rare Words with Subword Units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725. doi: 10.18653/v1/P16-1162.

Serdyuk, D., Wang, Y., Fuegen, C., Kumar, A., Liu, B. & Bengio, Y. (2018). Towards End-to-end Spoken Language Understanding. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5754-5758.

Sigurðsson, S., Petersen, K. B. & Lehn-Schiøler, T. (2006). Mel Frequency Cepstral Coefficients: An Evaluation of Robustness of MP3 Encoded Music. *ISMIR*, pp. 286-289.

Strunk, J., Schiel, F., Seifart, F. et al. (2014). Untrained Forced Alignment of Transcriptions and Audio for Language Documentation Corpora using WebMAUS. *LREC*, pp. 3940–3947.

Tiedemann, J. (2012). Parallel Data, Tools and Interfaces in OPUS. *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pp. 2214–2218. Consulted at http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

Tjong Kim Sang, E. F. & De Meulder, F. (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pp. 142–147. Consulted at https://www.aclweb.org/anthology/W03-0419.

Torreira, F., Adda-Decker, M. & Ernestus, M. (2010). The Nijmegen Corpus of Casual French. *Speech Communication*, 52(3), 201-212. doi: https://doi.org/10.1016/j.specom.2009.10.004.

Tur, G. & De Mori, R. (2011). Spoken Language Understanding: Systems for Extracting Semantic Information from Speech. *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech*. doi: 10.1002/9781119992691.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u. & Polosukhin, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems*, 30. Consulted at https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wallach, H. M. (2004). Conditional random fields: An introduction. *Technical Reports (CIS)*, 22.

Wu, Y., Schuster, M., andQuoc V. Le, Z. C., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M. & Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, abs/1609.08144. Consulted at http://arxiv.org/abs/1609.08144.

Xuan, G., Zhang, W. & Chai, P. (2001). EM algorithms of Gaussian mixture model and hidden Markov model. *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*, 1, 145-148 vol.1. doi: 10.1109/ICIP.2001.958974.

Yadav, H., Ghosh, S., Yu, Y. & Shah, R. R. (2020). End-to-End Named Entity Recognition from English Speech. *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pp. 4268–4272. doi: 10.21437/Interspeech.2020-2482.

Yadav, V. & Bethard, S. (2019). A Survey on Recent Advances in Named Entity Recognition from Deep Learning models. *CoRR*, abs/1910.11470. Consulted at http://arxiv.org/abs/1910.11470.

Yan, H., Deng, B., Li, X. & Qiu, X. (2019). TENER: Adapting Transformer Encoder for Named Entity Recognition. *ArXiv*, abs/1911.04474.

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., Ragni, A., Valtchev, V., Woodland, P. & Zhang, C. (2015). *The HTK Book (version 3.5a)*.