

**Université de Montréal**

**Détection de mots manquants**

par

**Joss Rakotobe**

Département d'informatique et de recherche opérationnelle  
Faculté des arts et des sciences

Rapport de stage présenté en vue de l'obtention du grade de  
Maître ès sciences (M.Sc.)  
en Apprentissage Automatique

May 1, 2021



# Université de Montréal

Faculté des arts et des sciences

---

Ce rapport de stage intitulé

## Détection de mots manquants

présenté par

**Joss Rakotobe**

a été évalué par un jury composé des personnes suivantes :

*Ioannis Mitliagkas*

---

(président-rapporteur)

*Ioannis Mitliagkas*

---

(directeur de recherche)

*Nom du membre de jury*


---

(membre du jury)



## Résumé

---

 Le logiciel Antidote ([www.antidote.info](http://www.antidote.info)) permet de corriger des textes en français et en anglais. Il détecte plusieurs milliers de types d'erreurs. L'entreprise derrière ce fameux logiciel, Druide Inc., aimerait ajouter de nouveaux types de correction à l'aide de modules basés sur l'apprentissage profond. Dans le cadre de ce stage, le but est de créer un modèle capable de détecter s'il manque un mot dans une phrase, et de trouver le mot manquant. Plusieurs approches ont été testées, les résultats des derniers modèles étant très prometteurs atteignant jusqu'à 85% de bonnes réponses pour la tâche de détecter s'il manque un mot ou non, 68% de précision et 60% de rappel pour la tâche de prédire le mot qui manque.

*Mots clés: Apprentissage automatique, Apprentissage profond, Apprentissage machine, Traitement de texte, Détection de mot manquants, Encodeur, Transformers, BERT [1], CamemBERT [6], RoBERTa [5], Distillation, Compression de modèle [7, 8], Réseaux de neurones, Intelligence artificielle.*



# Table des matières

---

<b>Résumé</b> .....	5
<b>Liste des tableaux</b> .....	9
<b>Liste des figures</b> .....	11
<b>Liste des sigles et des abréviations</b> .....	13
<b>Remerciements</b> .....	15
<b>Introduction</b> .....	17
<b>Chapitre 1. Preuve de concept</b> .....	19
1.1. À la recherche du modèle pour la tâche .....	20
1.1.1. Modèle de base des encodeurs: BERT .....	21
1.1.2. Variantes .....	22
1.2. Entraînement à la tâche .....	23
1.2.1. Création des données .....	23
1.2.2. Décomposition de la tâche .....	23
1.2.3. Choix des paramètres .....	25
1.2.4. Premiers résultats .....	25
<b>Chapitre 2. Vers une intégration dans Antidote</b> .....	29
2.1. Distillation .....	29
2.2. Fusion des modèles .....	30
2.2.1. Première méthode: combiner les encodeurs .....	31
2.2.2. Deuxième méthode: combiner les têtes .....	32
2.3. Autres variantes d'encodage .....	33
2.3.1. Troncature .....	33
2.3.2. MobileBert .....	33
2.4. Analyse de la performance et Optimisation .....	34

2.4.1. Métriques.....	34
2.4.2. Optimisation.....	34
<b>Chapitre 3. Développement et Futures directions.....</b>	<b>37</b>
3.1. Développement du modèle.....	37
3.2. Futures améliorations.....	37
3.2.1. Amélioration des données d'entraînement.....	37
3.2.2. Amélioration du modèle.....	38
<b>Conclusions.....</b>	<b>39</b>
<b>Références bibliographiques.....</b>	<b>41</b>



## Liste des tableaux

---

1.1	Un exemple de jetonisation avec le jetoniseur associé à CamemBERT. La phrase comporte un mot manquant avant "un".....	25
1.2	Exactitude des modèles sur les trois tâches.....	26
2.1	Comparaison des scores des modèles DistilCamemBERT3 et CamemBERT.....	30
2.2	Comparaison des scores des modèles DistilCamemBERT3 et CamemBERT3.....	33
2.3	Derniers résultats obtenus pour les modèles.....	35



## Liste des figures

---

1.1	Visualisation avec BertViz ( <a href="https://github.com/jessevig/bertviz">https://github.com/jessevig/bertviz</a> ). À gauche, les attentions de la première tête de la première couche d'un Transformer. À droite, celles de la première tête de la deuxième couche.....	20
1.2	Corrélation entre la taille du modèle et sa performance sur GLUE.....	21
1.3	Chaîne de traitement pour la détection de mots manquants. Les modèles sont en bleu foncé. Le chemin en rouge est celui suivi si on ne détecte pas de mot manquant.....	24
1.4	Exactitude des classificateurs de phrases en français pendant 4 époques: CamemBERT (en rouge) et FlauBERT (en bleu) sur l'ensemble de validation....	26
1.5	Comparaison de la vitesse des modèles en français et d'un modèle distillé: CamemBERT (en rouge), FlauBERT (en bleu) et DistilBERT en vert.....	27
2.1	Comparaison de la vitesse des modèles DistilCamemBERT3 (en rouge) et CamemBERT (en bleu).....	30
2.2	Comparaison de la vitesse d'entraînement des modèles DistilCamemBERT3 (en rouge) et CamemBERT (en bleu) sur une même machine.....	31
2.3	Première méthode de combinaison des modèles.....	32
3.1	Trois exemples de phrases visualisées avec Captum. Dans le premier exemple, il s'agit de la phrase complète. Ici, c'est le premier et le dernier mot qui ont eu la plus grande importance dans la décision du modèle de prédire qu'il ne manque pas de mot. Dans le deuxième exemple, on a enlevé le terme "il a prédit qu'il manque un mot à cause du terme "BBC". Dans le troisième exemple, il a prédit qu'il manque un mot à cause de la partie "BBC two examples that".....	38



## Liste des sigles et des abréviations

---

TAL	Traitement Automatique de la Langue
BERT	Représentation encodée bi-directionnelle à partir d'un Transformer, de l'anglais Bidirectional Encoder Representation from Transformers
GLUE	General Language Understanding Evaluation
BPE	Byte Pair Encoding



## Remerciements

---

Tous mes remerciements à mes deux superviseurs qui m'ont encadré tout au long de mon stage:

- Jasmin Lapalme, chef de projet logiciel chez Druide informatique, qui grâce à ses connaissances, autant dans le domaine du traitement de texte que dans celui de l'apprentissage machine, a su me diriger dans la bonne direction dès les premiers jours. Tout au long de mon stage, il a toujours été quelqu'un sur qui je pouvais compter si quelque chose venait à me bloquer, si j'avais des questions générales ou si j'avais des idées à proposer. Honnêtement, je ne pouvais pas mieux tomber!

- Akram Erraqabi, doctorant au Mila, qui a été très impliqué pendant nos rencontres. Nos discussions m'ont notamment aidé à réfléchir plus en profondeur sur les manières d'aborder les divers problèmes et à trouver de nouvelles approches intéressantes qui auront porté leurs fruits.





# Introduction

---

**Druide Informatique** ([www.druide.com](http://www.druide.com)) est une entreprise québécoise spécialisée dans le domaine de la **rédaction informatique**. Fondée en 1993, elle est notamment connue dans toute la francophonie à travers son logiciel de correction grammaticale appelé **Antidote** ([www.antidote.info](http://www.antidote.info)), mais aussi pour son logiciel d'apprentissage de la dactylographie Tap'Touche ([www.taptouche.com](http://www.taptouche.com)). De plus, elle propose un service WebElixir ([www.webelixir.net](http://www.webelixir.net)), pour veiller à la qualité des sites Internet.

**Antidote** permet de corriger des textes en français et en anglais. Il est utilisé par plus d'un million de personnes dans toute la francophonie, via les plateformes Mac, Windows, Linux, Iphone et Ipad. Grâce à sa capacité à détecter des milliers de types d'erreurs, il est la référence en matière de logiciel de correction de texte en français, prisé par toute personne cherchant à vérifier scrupuleusement un corpus.

Étant constamment en quête d'amélioration, *Druide* se tourne maintenant vers l'apprentissage automatique pour ajouter de nouvelles capacités à son logiciel principal. En particulier, le but de ce stage est de développer un modèle basé sur l'apprentissage profond, capable de détecter s'il manque un mot dans une phrase, et de proposer la bonne correction, le cas échéant. De plus, ce modèle devrait être développé avec comme objectif final une intégration future dans Antidote.

En général, l'architecture d'un modèle en traitement automatique de la langue se décompose en deux grandes parties: le choix de la base du modèle, appelée aussi l'encodeur, qui permet de donner une représentation tensorielle<sup>1</sup> à chaque mot, et le choix de la tête du modèle, qui est une partie très spécifique à la tâche, dans notre cas un classificateur qui va nous dire s'il manque un mot dans une phrase ou non.

---

<sup>1</sup>Un *tenseur* est un tableau de nombres à une ou plusieurs dimensions. Les vecteurs et les matrices sont les appellations courantes pour les cas à une dimension et à deux dimensions respectivement.

Notre premier objectif consistait à démontrer la faisabilité du problème grâce aux dernières avancées en apprentissage profond. Dans le premier chapitre, nous ferons une présentation de la méthodologie utilisée pour la création des données d’entraînement ainsi que des modèles utilisés. Puis, nous montrerons que ce type de problème est bel et bien solvable par des modèles d’apprentissage profond et nous visualiserons le genre de représentation apportée.

Dans un deuxième temps, nous nous pencherons sur la partie optimisation des modèles. Nous verrons les différentes approches qui ont été tentées pour faire face aux défis de l’entreprise. Enfin, nous terminerons par l’exploration d’autres directions que l’entreprise québécoise pourrait essayer pour se rapprocher de ses attentes.

# Chapitre 1

---

## Preuve de concept

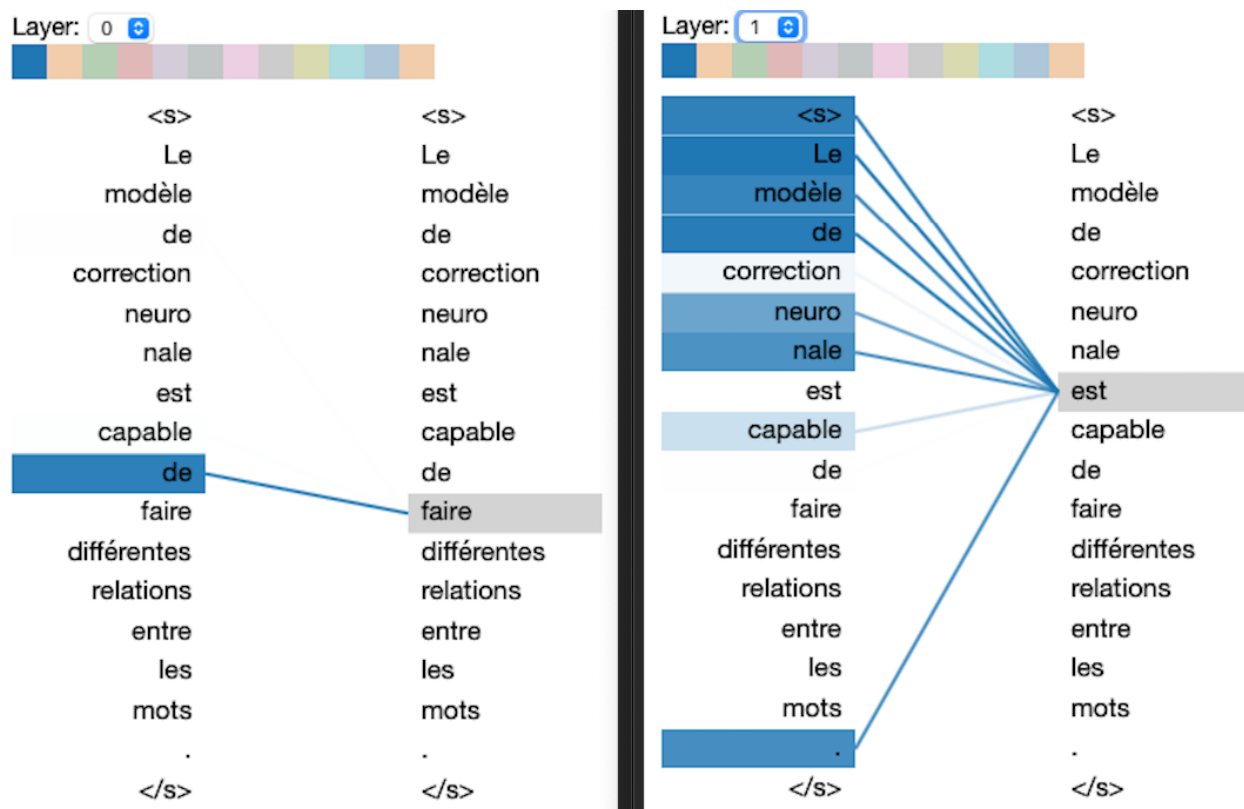
Notre premier objectif est de montrer qu'un modèle basé sur l'apprentissage machine est capable de détecter s'il manque un mot dans une phrase.

Les modèles de traitement automatique de la langue connaissent une avancée fulgurante ces dernières années. Dernièrement, une famille de modèles appelée les **Transformers** [9] se bouscule entre eux pour atteindre de nouveaux sommets dans l'état de l'art. Ce sont des modèles d'apprentissage profond basés sur des mécanismes d'attention. Un mécanisme d'attention est un module qui produit une représentation pour chaque entrée du modèle en se basant sur toutes les entrées. Ce module apprend à quel point chaque entrée doit faire *attention* aux autres entrées pour avoir la meilleure représentation. En regardant les poids des attentions, on peut parfois voir quel type de relation le module a essayé d'apprendre et à quel point chaque élément est lié aux autres par cette relation. Par exemple, on retrouve souvent de simples relations du type « mot - mot suivant », mais aussi des relations plus avancées du type « sujet - verbe » (voir figure 1.1).

Afin de pouvoir évaluer la capacité d'un modèle à comprendre l'anglais, deux références ont été adoptées par la grande communauté de pratiquants du traitement automatique de la langue. Ces références sont connues sous les noms de **GLUE** (**G**eneral **L**anguage **U**nderstanding **E**valuation) [11] et **SuperGLUE** [10], regroupant plusieurs tâches demandant parfois un niveau de compréhension élevé de la langue. Certains Transformers sont déjà capables de faire mieux que la base de référence humaine<sup>1</sup>, atteignant des scores impressionnants face aux tâches les plus dures de GLUE. En regardant ces résultats, tout porte à croire que ce type de modèle devrait être capable de résoudre notre tâche, à savoir apprendre à détecter si des phrases sont complètes ou non, tâche qui semble *à priori* moins difficile pour l'humain que certaines des tâches de GLUE et SuperGLUE.

---

<sup>1</sup><https://gluebenchmark.com/leaderboard>.  
<https://super.gluebenchmark.com/leaderboard>



**Fig. 1.1.** Visualisation avec BertViz (<https://github.com/jessevig/bertviz>). À gauche, les attentions de la première tête de la première couche d'un Transformer. À droite, celles de la première tête de la deuxième couche.

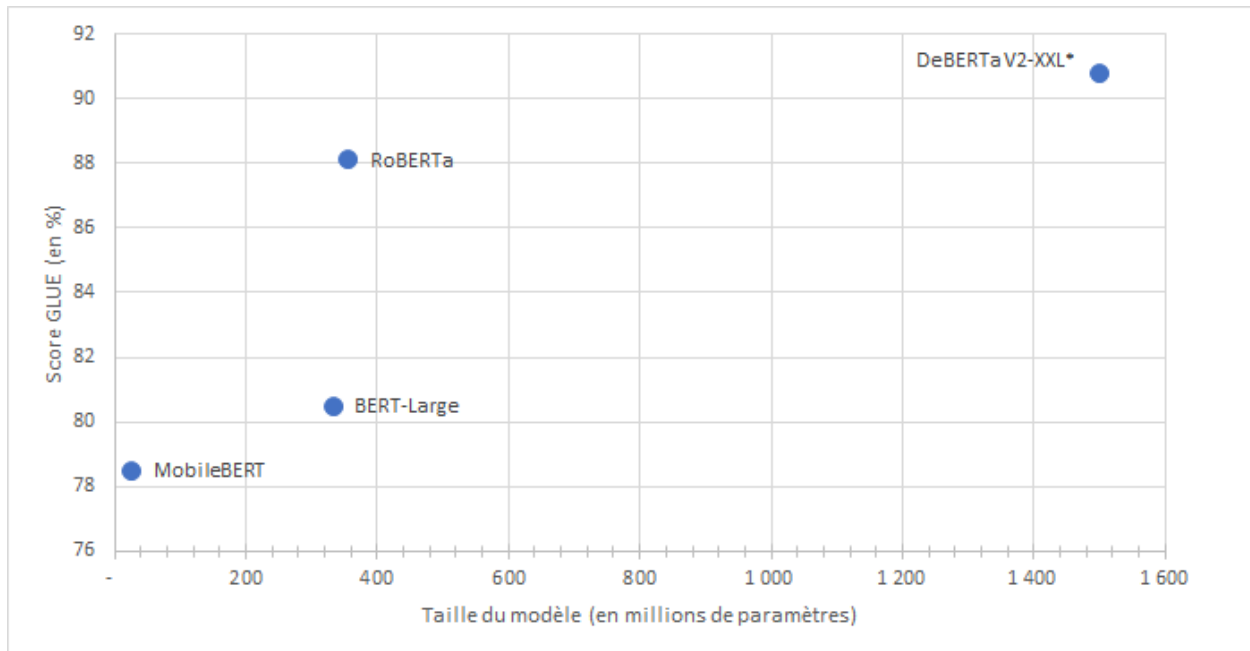
## 1.1. À la recherche du modèle pour la tâche

GLUE nous permet d'avoir une idée globale des meilleurs modèles en Traitement Automatique de La langue (TAL). Cependant, chaque modèle possède ses forces et ses faiblesses; il n'est pas rare de voir un modèle plus performant dans une tâche, et moins performant dans une autre. En particulier, bien qu'il ne s'agisse pas de notre préoccupation primaire, l'objectif final de l'entreprise reste de pouvoir intégrer ce modèle dans le logiciel Antidote. À cette fin, il sera important que le modèle final soit assez rapide lors de l'inférence<sup>2</sup>.

La modélisation de la langue naturelle est très complexe et les modèles ont besoin d'un nombre important de paramètres, en général plusieurs centaines de millions pour entrer dans le classement de GLUE<sup>3</sup> (voir figure 1.2). Malheureusement, à cause de leur taille, ces modèles sont très lents à l'inférence (plus de paramètres implique, en général, plus de

<sup>2</sup>Étant donné que la vitesse d'inférence du modèle varie d'un appareil à un autre, on utilisera la vitesse actuelle d'Antidote sur un même appareil comme référence.

<sup>3</sup>Le top du classement SuperGLUE est présentement atteint par l'équipe de Microsoft, avec un modèle appelé **DeBERTa** de près de 1,5 milliard de paramètres.



**Fig. 1.2.** Corrélacion entre la taille du modèle et sa performance sur GLUE.

calculs) et il y a donc un compromis à faire sur le nombre de paramètres de notre modèle (voir figure 1.2).

### 1.1.1. Modèle de base des encodeurs: BERT

Les Transformers ont atteint des records dans les tâches courantes du traitement de la langue. En effet, ce système d'attention permet de donner une très bonne représentation contextuelle à chaque mot. Cette avancée a été marquée par un Transformer nommé **BERT** (**B**idirectional **E**ncoder **R**epresentation from **T**ransformers). Le modèle original a été développé par Google en 2018 et comprend deux versions: les modèles BERT<sub>BASE</sub> et BERT<sub>LARGE</sub> qui comprennent respectivement 12 couches et 24 couches, chacune étant composée de 768 unités de large, divisées en 12 têtes d'attention multiples. On distingue trois majeures parties de l'encodeur:

- le **jeteur** qui sépare le texte en une liste de sous-mots, appelés les **jetons**, et qui associe à chaque sous-mot un entier qui le caractérise;
- le **module d'intégration principale** qui s'occupe de donner une représentation vectorielle à chaque jeton en fonction de sa valeur et de sa position (à ce stade, chaque jeton est représenté quasi indépendamment du contexte);
- les **modules d'attention** qui permettent d'approfondir la représentation de chaque jeton en prenant en compte le contexte dans lequel il se trouve (les autres jetons); chaque tête d'attention apprend une tâche particulière qui s'apparente parfois à un lien sémantique entre les jetons (voir figure 1.1).

Un des gros avantages de BERT est qu'il est déjà préentraîné, c'est-à-dire que la représentation vectorielle qu'il procure aux phrases est générale et peut être utilisée comme base de représentation pour n'importe quelle tâche. L'entraînement à une tâche spécifique est alors beaucoup plus rapide.

### 1.1.2. Variantes

En plus du modèle BERT, nous avons essayé une variante en anglais, et deux autres variantes en français.

- **RoBERTa** [5] est une version optimisée de manière plus **Robuste** par rapport à la version originale de **BERT**. On note deux différences clés entre les deux modèles:
  - la jetonisation par Byte Pair Encoding (BPE): RoBERTa utilise un vocabulaire de cinquante mille mots contre trente mille pour BERT, et la jetonisation des mots se fait au niveau de l'octet au lieu d'être faite au niveau du caractère comme pour celle de BERT. Cela permet à RoBERTa de pouvoir jetoniser n'importe quelle phrase sans jamais avoir besoin d'utiliser de jeton "inconnu".
  - la méthode utilisée pour préentraîner RoBERTa ainsi que le choix des hyperparamètres sont plus optimisés (voir [5] pour plus de détails).

Certes, on constate une légère augmentation du nombre de paramètres de vingt millions, à cause que la jetonisation se base sur un vocabulaire plus grand. Cependant, les expérimentations sur GLUE confirment la robustesse de RoBERTa par rapport à BERT, avec une différence plus notable au niveau des scores, comparé à la différence du nombre de paramètres (voir figure 1.2).

- **CamemBERT** [6], dont le nom utilise un jeu de mots avec le fameux fromage de la Normandie, pour marquer qu'il s'agit d'un encodeur en français. Ce modèle a été préentraîné de manière similaire à RoBERTa à la différence qu'il utilise une jetonisation SentencePiece[3].
- **FlauBERT** [4] pour **F**rench **l**anguage **u**nderstanding via **BERT**, est une autre version française du modèle RoBERTa, avec un jetoniseur Moses[2].

Pour les modèles en anglais, nous avons utilisé des versions distillées de BERT et RoBERTa, contenant 6 couches, qui sont disponibles publiquement. Pour le français, étant donné que nous n'avions pas trouvé de modèle distillé, nous avons tronqué les modèles en gardant les 6 premières couches des modèles de base. Nous reviendrons plus en détail sur la distillation dans la section 2.1, mais l'avantage de ces modèles est qu'ils sont deux fois plus petits et beaucoup plus rapides à entraîner que les modèles de base.

## 1.2. Entraînement à la tâche

### 1.2.1. Création des données

Afin de pouvoir développer notre modèle d'apprentissage profond, *Druide* a collecté un corpus contenant plusieurs milliards de mots provenant de sources très variées. Pour adapter le corpus à la tâche de détection de mots manquants, il était essentiel de filtrer les séquences du corpus pour qu'il ne reste que des phrases complètes. On a fait appel à la librairie *SpaCy* (<https://spacy.io/>), spécialisée dans le traitement de texte. Cette librairie comprend des modèles de traitement automatique de la langue capables de déterminer de manière assez précise la classe grammaticale d'un mot, ainsi que sa dépendance par rapport au reste de la phrase. On a ainsi pu extraire un corpus contenant que des phrases complètes, c'est-à-dire, commençant par une majuscule, se terminant par un point, un point d'interrogation ou un point d'exclamation, et possédant un sujet, un verbe et un complément d'objet. De plus, nous nous limitons aux phrases qui ne sont pas trop longues, avec moins de 128 mots. Lors de la jetonisation, nous tronquons le nombre de jetons en prenant seulement les 128 premiers (incluant toujours les jetons de début et de fin de phrase) et en rembourrant la liste pour obtenir les 128 jetons s'il en manque. Avoir une taille fixe est essentiel pour pouvoir entraîner le modèle par lot de plusieurs phrases à la fois.

### 1.2.2. Décomposition de la tâche

Le problème de détection de mots manquants se décompose naturellement en une chaîne de traitement composé de trois grandes étapes: détecter s'il manque un mot, et le cas échéant, à quel endroit dans la phrase il se trouve et enfin trouver de quel mot il s'agit. Nous avons décidé de suivre ces étapes en construisant un modèle différent pour chaque étape. La chaîne de traitement complète est schématisée dans la figure 1.3. Chaque modèle est composé de manière assez simple: un encodeur de la famille BERT avec par dessus un classificateur adapté à la tâche.

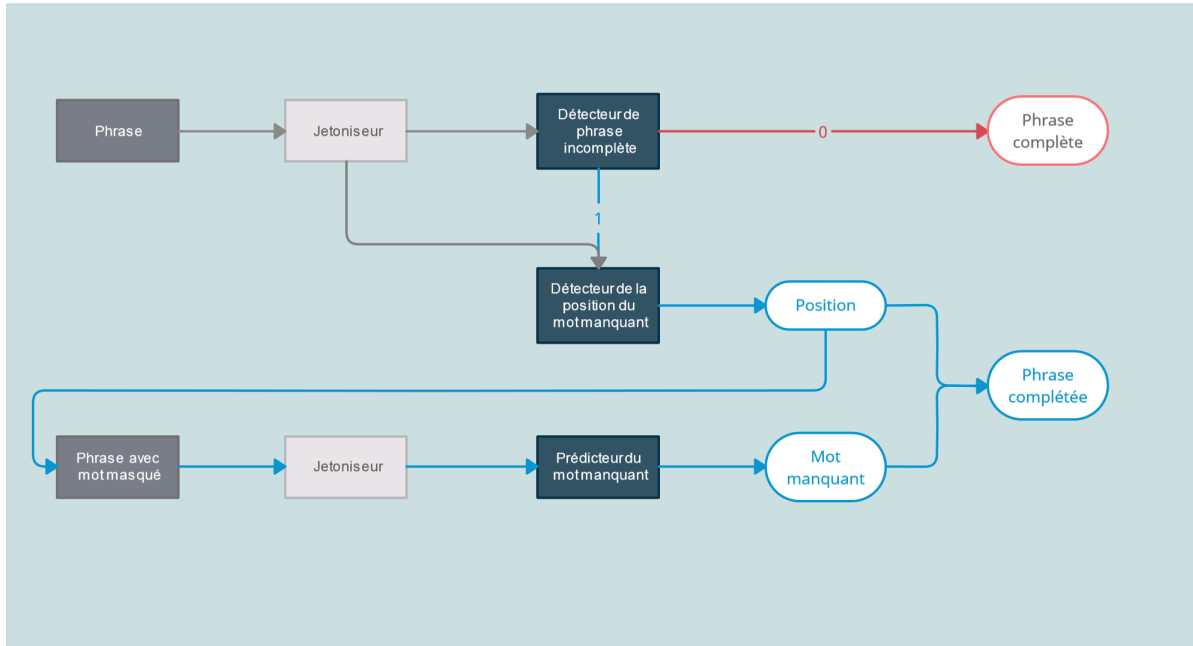
- **Détection de phrases incomplètes (Dét. Incomplet<sup>4</sup>)**

Dans un premier temps, nous cherchons à déterminer si un modèle d'apprentissage profond est capable de détecter s'il manque un mot dans une phrase. Pour cela, nous utilisons un simple classificateur binaire de phrase par dessus un encodeur de la famille BERT.

Le corpus se compose alors de 50% de phrases complètes et de 50% de phrases auxquelles on a enlevé un mot. Le mot enlevé a été choisi aléatoirement avec une distribution relative à sa classe grammaticale attribuée par SpaCy. Pour chaque

---

<sup>4</sup>Nom de la sous-tâche pour référence dans les tableaux des résultats



**Fig. 1.3.** Chaîne de traitement pour la détection de mots manquants. Les modèles sont en bleu foncé. Le chemin en rouge est celui suivi si on ne détecte pas de mot manquant.

phrase, l'étiquette associée est tout simplement un nombre binaire: 1 s'il manque un mot, 0 sinon.

- **Détection de la position d'un mot manquant (Dét. Position)**

Après avoir montré que ce type de modèle est capable de détecter lorsqu'il manque un mot ou non dans une phrase, on aimerait pouvoir détecter à quelle position il manque un mot. Pour cela, nous utilisons un classificateur binaire de mot par dessus un encodeur de la famille BERT.

Le corpus est identique à celui de la tâche précédente. En revanche, pour chaque phrase, l'étiquette associée est un vecteur binaire de longueur 128, prenant pour valeur 1 en position  $i$  s'il manque un mot avant le jeton  $i$ , 0 sinon (voir 1.1).

- **Prédiction du mot manquant (Préd. Mot)**

Pour cette dernière partie, on a utilisé un classificateur de mot masqué par dessus l'encodeur BERT. Cette tâche fait déjà partie du préentraînement des modèles de la famille BERT. Nous savions donc déjà qu'ils sont capables de résoudre cette partie, ne laissant en suspens que les résultats atteints.

Pour cette tâche, le corpus n'est constitué que de phrases incomplètes. De plus, nous indiquons la position du mot manquant en ajoutant un jeton "masque" là où il manque un mot. L'étiquette est alors le nombre associé au premier jeton tiré de la jetonisation du mot. Comme exemple, avec CamemBERT, si le mot manquant est



Phrase	Ceci un exemple.						
Jetons	<s>	Ceci	un	exemple	.	<\s>	Rembourrage
Ids des jetons	4	15	18	114	42	2	1, ..., 1
Étiquettes	0	0	1	0	0	0	0, ..., 0

**Tableau 1.1.** Un exemple de jetonisation avec le jetoniseur associé à CamemBERT. La phrase comporte un mot manquant avant "un".

*jetoniseur*, il sera décomposé en trois jetons: [je, ton, iseur]. L'étiquette sera alors 50, le numéro associé au jeton *je*.

### 1.2.3. Choix des paramètres

Pour le choix des hyperparamètres, nous nous sommes basés sur les valeurs originales trouvées dans la littérature pour les paramètres de l'optimiseur [1], et sur les capacités des machines pour la taille des lots et le nombre de jetons par phrase. Ainsi, nous avons utilisé:

- 128 jetons par phrase pour les premiers entraînements;
- un maximum de phrases par lot<sup>5</sup>;
- un optimiseur AdamW pour la descente du gradient, avec un taux d'apprentissage de  $2 \cdot 10^{-5}$ , un epsilon de  $1 \cdot 10^{-8}$ , une décroissance du poids de 0,01 et les paramètres  $(\beta_1, \beta_2) = (0,9; 0,999)$ ;
- étant donné qu'il ne s'agit que de la phase *Preuve de Concept*, nous avons limité l'entraînement à 4 époques avec un arrêt plus tôt lorsqu'on constate une dégradation de l'exactitude<sup>6</sup> du modèle sur l'ensemble de validation ou une non-amélioration sur une longue période sur l'ensemble d'entraînement.

### 1.2.4. Premiers résultats

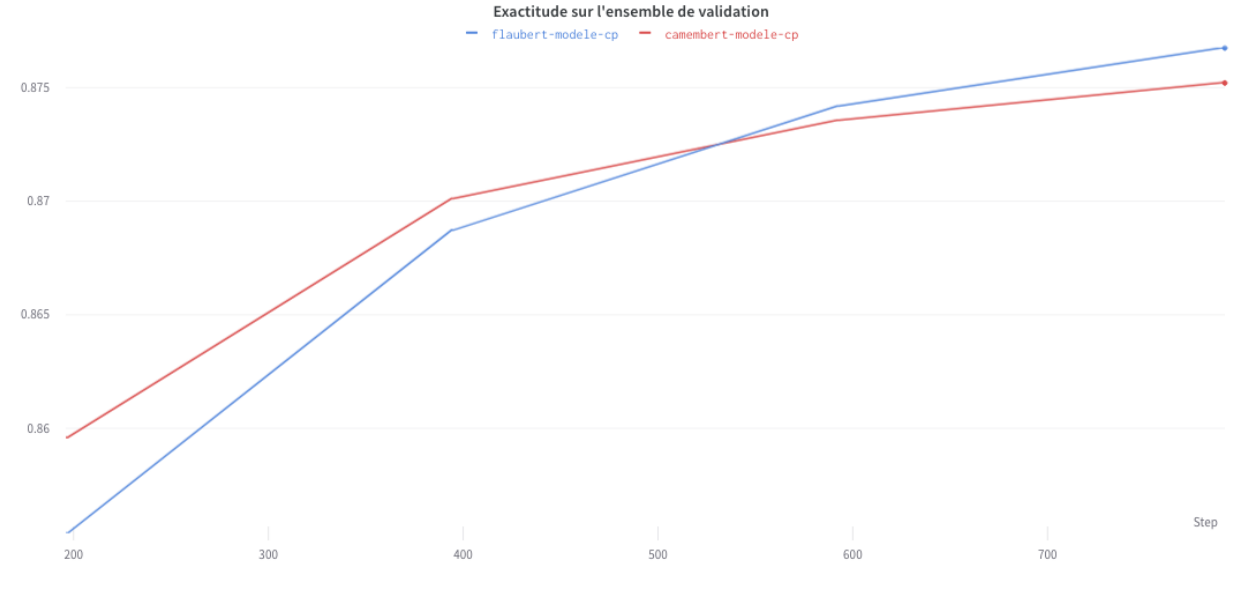
Pour toutes les tâches, les Transformers arrivent à avoir d'assez bons résultats (voir tableau 1.2). Évidemment, il faut jauger ces résultats relativement à la tâche: imaginez qu'on utilise des prédictions aléatoires, le détecteur de phrases incomplètes aurait un score de 50%, le détecteur de positions de mots manquants aurait un score de  $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{256} = \frac{129}{512}$  et le prédicteur de mot manquant aurait une exactitude de  $\frac{1}{\text{Taille}_{\text{vocabulaire}}} < \frac{1}{30000}$ . Ainsi, deviner exactement quel mot il manque est un problème beaucoup plus complexe que déterminer la position du mot manquant. On remarque aussi que les modèles en français sont nettement plus performant que ceux en anglais. Une partie de cette différence s'explique par le fait que des modèles distillés ont été utilisés en anglais. Bien que ce type de modèle soit connu pour

<sup>5</sup>Un algorithme créé par moi-même est lancé pour trouver la taille maximale de lots qui peut être utilisé pour chaque modèle. On atteint jusqu'à 800 phrases en utilisant deux cartes graphiques en parallèles.

<sup>6</sup>Pour tous les modèles, on définit l'exactitude sur un ensemble comme étant le nombre de phrases ayant une parfaite prédiction divisé par le nombre total de phrases de l'ensemble.

Exactitude sur l'ensemble de validation (en %)				
Langue	Anglais*		Français	
Modèle	BERT	RoBERTa	CamemBERT	FlauBERT
Dét. Incomplet	78,5	82,0	87,5	87,7
Dét. Position	74,4	78,0	84,3	82,09
Préd. Mot	40,3	48,7	56,8	58,7

**Tableau 1.2.** Exactitude des modèles sur les trois tâches.

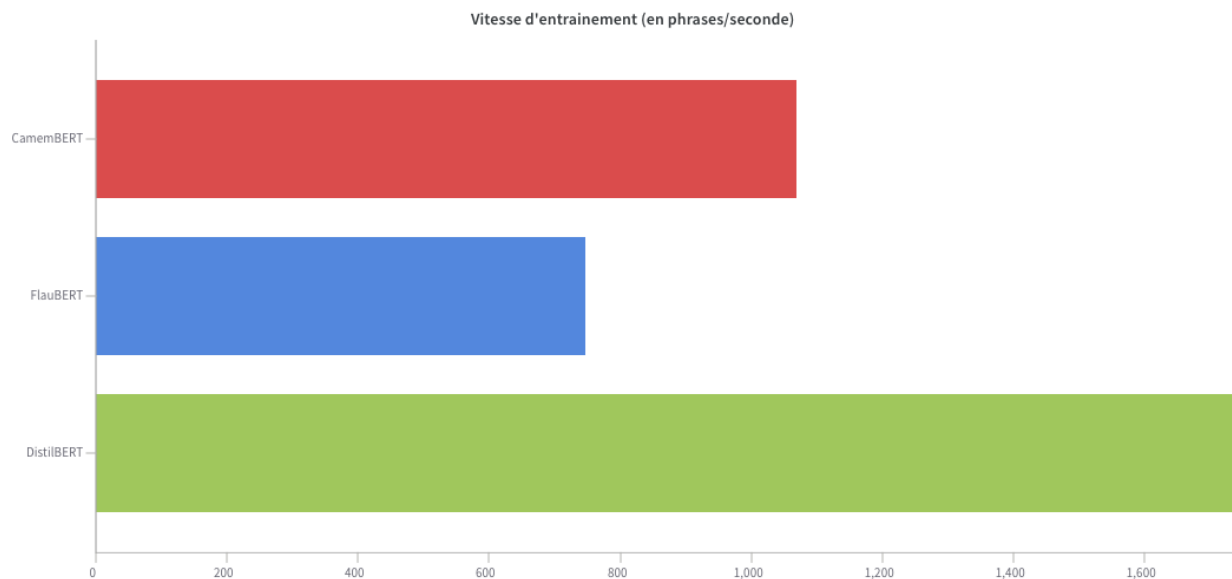


**Fig. 1.4.** Exactitude des classificateurs de phrases en français pendant 4 époques: CamemBERT (en rouge) et FlauBERT (en bleu) sur l'ensemble de validation.

être légèrement moins performant que l'original, la différence observée porte à croire qu'il y a autre chose qui entraîne cet écart, probablement des données qui sont plus propres en français.

On retrouve, comme mentionné dans l'article de FlauBERT[4], une forte compétition entre FlauBERT et CamemBERT, avec pour l'instant, aucun des deux qui ne fait l'unanimité. Cependant, on préférera CamemBERT qui a une taille de vocabulaire deux fois plus petite que celle de FlauBERT, ce qui le rend aussi plus rapide (voir figure 1.5).

Pour les modèles en anglais, l'encodeur RoBERTa est nettement supérieur sur toutes les tâches. Pour la suite, on n'utilisera plus DistilBERT pour la détection en anglais.



**Fig. 1.5.** Comparaison de la vitesse des modèles en français et d'un modèle distillé: CamembERT (en rouge), FlauBERT (en bleu) et DistilBERT en vert



## Chapitre 2

---

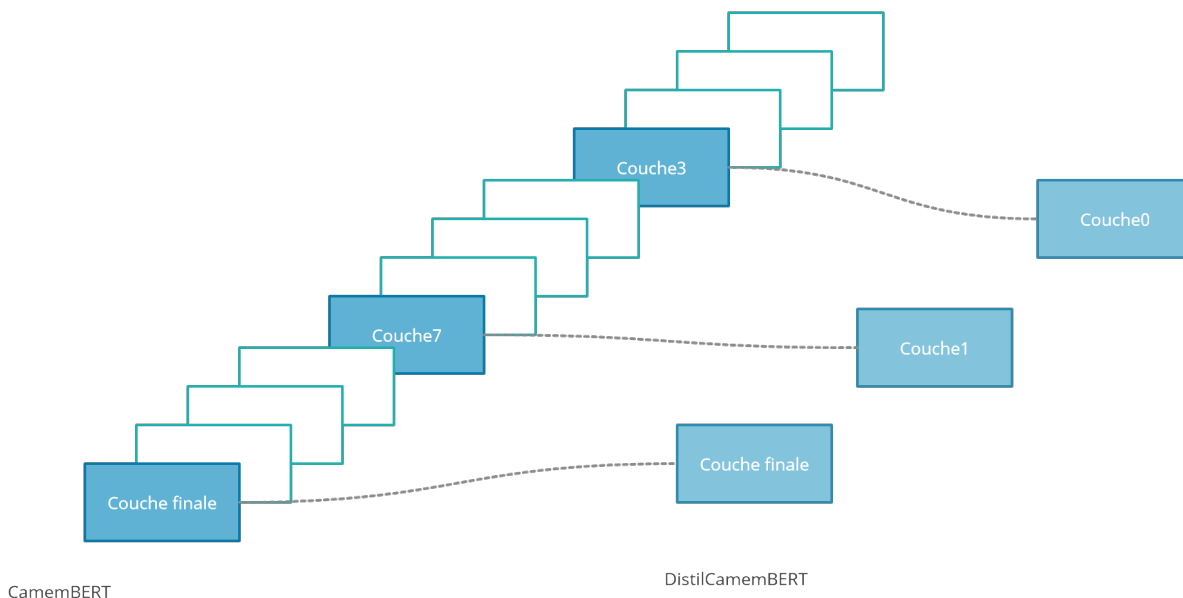
### Vers une intégration dans Antidote

Maintenant que nous avons démontré la capacité des Transformers dans la détection de mots manquants, notre prochain objectif est de former un modèle pouvant être intégré dans Antidote. Cela requiert principalement des améliorations sur la vitesse d'inférence et la taille du modèle. Dans cette optique, on a réduit le nombre de jetons à 64 par phrase, ce qui semble déjà assez raisonnablement grand. Dans ce chapitre, nous montrons les différentes approches qui ont été tentées pour améliorer ces points.

#### 2.1. Distillation

La distillation [7] est une technique de compression cherchant à condenser un modèle parent dans un modèle enfant, tout en essayant de garder la même information. Les encodeurs, qu'on a utilisés en anglais dans le chapitre 1, sont des exemples de modèles qui ont été distillés. Pour ces modèles, le parent est un Transformer avec 12 couches qui a été préentraîné, et l'enfant, quant à lui, ne contient que 6 couches. Chaque couche a exactement la même architecture, aussi bien chez le modèle parent que chez le modèle enfant. Le procédé d'apprentissage vise à retrouver chez l'enfant le même encodage final que celui du parent. L'apprentissage se fait par étape. Ainsi, on associe à chaque couche de l'enfant, une couche du parent à partir de laquelle celle de l'enfant apprendra. Les dernières couches des deux modèles doivent être associées entre elles et, idéalement, les couches intermédiaires choisies pour le parent sont espacées de manière régulière afin d'homogénéiser le champ d'apprentissage; on demande souvent que le nombre de couches du parent soit un multiple de celui de l'enfant.

À fortiori, on constate souvent une légère perte de précision du modèle, mais cette perte est largement compensée par un gain en mémoire et en vitesse (comme vu précédemment 1.5). Afin de confirmer que la distillation n'était pas la cause principale de la grande différence de scores observés entre le français et l'anglais dans le chapitre 1, on a effectué la



**Fig. 2.1.** Comparaison de la vitesse des modèles DistilCamemBERT3 (en rouge) et CamemBERT (en bleu)

Exactitude sur l'ensemble de validation (en %)		
Modèle	DistilCamemBERT3	CamemBERT
ClassificateurDeMots	84,2	84,3
PrédicteurDeMots	53,6	56,8

**Tableau 2.1.** Comparaison des scores des modèles DistilCamemBERT3 et CamemBERT

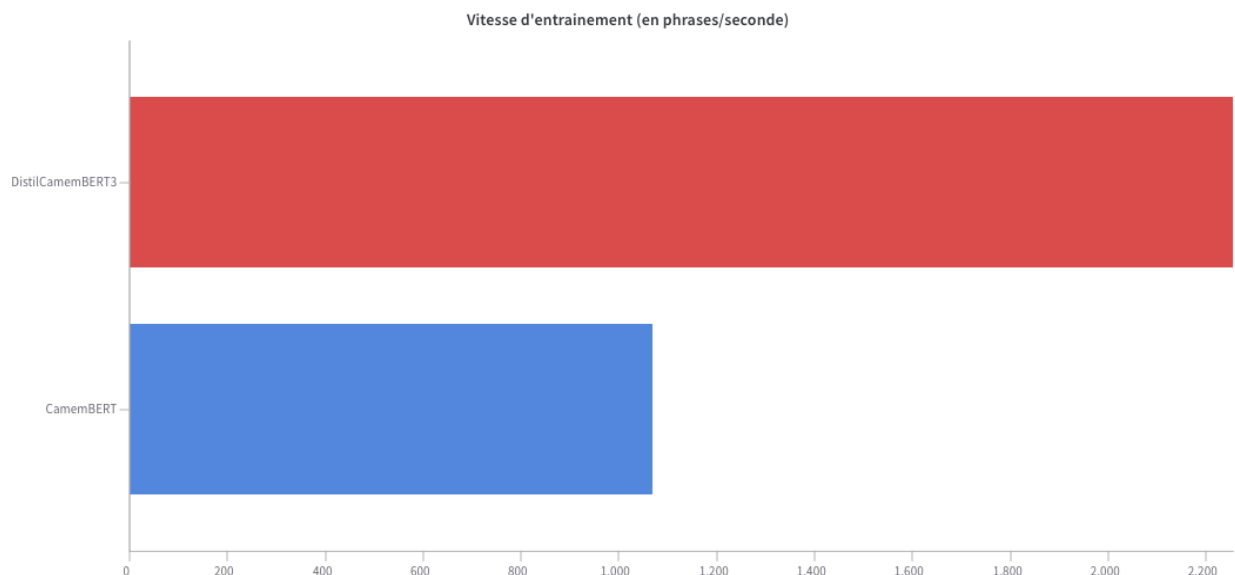
distillation de l'encodeur en français CamemBERT en un Transformers à trois couches, appelé DistilCamemBERT3 (voir figure 2.1). On a répété les étapes des entraînements pour la détection de la position d'un mot manquant et pour la prédiction du mot manquant et nous comparons ses résultats avec ceux du modèle original (voir le tableau 2.1 et la figure 2.2).

Le modèle distillé est entraîné beaucoup plus rapidement et obtient des scores très proches du modèle de base pour ces tâches. Il permet aussi d'utiliser de plus grands lots de phrases, ce qui aide à la généralisation<sup>1</sup>.

## 2.2. Fusion des modèles

Pour le moment, nous avons réussi à résoudre le problème en créant une chaîne de traitement. À l'inférence, dans le cas positif où il manque un mot, on doit utiliser les trois modèles pour pouvoir déterminer le mot qui manque: le classificateur de phrases incomplètes doit

<sup>1</sup>On peut obtenir des effets similaires pour le modèle de base en utilisant la technique d'accumulation de gradients.



**Fig. 2.2.** Comparaison de la vitesse d’entraînement des modèles DistilCamemBERT3 (en rouge) et CamemBERT (en bleu) sur une même machine.

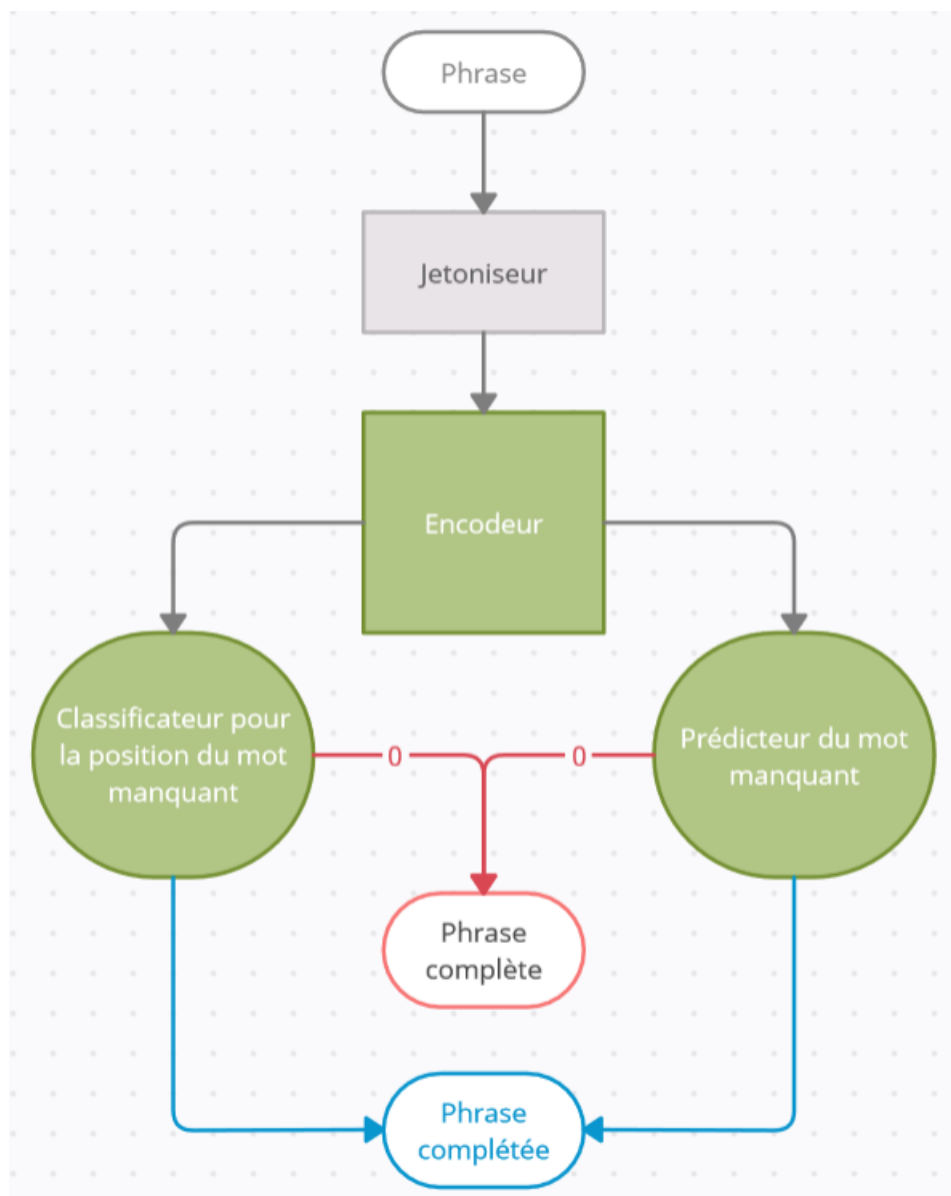
déterminer s’il manque un mot dans la phrase, puis dans le cas positif, un deuxième classificateur doit déterminer la position du mot manquant et enfin, après avoir apposé un jeton masqué à la position du mot manquant, le classificateur de mot va prédire le mot qui manque. La présence de trois modèles différents entraîne trois fois plus d’espace et de temps lors de l’inférence. Pourtant, lorsque nous regardons une phrase avec un mot manquant, que ce soit pour deviner qu’il manque un mot, la position du mot manquant ou le mot qui manque, ce sont les mêmes mots (reliés au potentiel mot manquant) qui importe et auxquels les encodeurs devront faire attention, à une petite différence près. Ainsi, notre intuition nous dit qu’il devrait être possible de combiner les trois modèles en un, en utilisant notamment un seul encodeur.

### 2.2.1. Première méthode: combiner les encodeurs

La première méthode que nous avons essayée pour tenter de combiner les modèles consiste à attacher plusieurs têtes à un même encodeur. Le modèle qui permet de détecter la position d’un mot manquant inclut déjà implicitement un détecteur de phrases incomplètes. En effet, il suffit à ce modèle de prédire la classe zéro pour chaque jeton pour prédire que la phrase est complète.

Ainsi, notre premier modèle combiné possède deux têtes: un détecteur de positions de mots manquants et un prédicteur de mots manquants. La chaîne de traitement devient alors comme suit: pour une phrase donnée, on jetonise la phrase et passe le résultat à l’encodeur. Ce dernier transmet en parallèle son résultat aux deux têtes. La première va déterminer s’il

manque un mot ou pas, et si oui, à quelle position. La deuxième tête s'occupe de prédire quel mot il manque. Dans le cas où il pense qu'il ne manque pas de mot, il prédit la classe 0 qui est associé à un jeton non-utilisé (voir figure 2.3).



**Fig. 2.3.** Première méthode de combinaison des modèles.

### 2.2.2. Deuxième méthode: combiner les têtes

Dans la première méthode utilisée pour combiner les modèles, nous remarquons que le prédicteur n'a pas besoin de connaître la position du mot manquant pour deviner de quel mot il s'agit. De plus, ce dernier est capable de déterminer s'il manque un mot ou non. Nous nous basons sur ça pour construire notre deuxième modèle combiné, qui vise à regrouper les



Exactitude sur l'ensemble de validation (en %)		
Modèle	DistilCamemBERT3	CamemBERT3
ClassificateurDeMots	84,2	83,3
PrédicteurDeMots	53,6	52,8

**Tableau 2.2.** Comparaison des scores des modèles DistilCamemBERT3 et CamemBERT3

deux têtes en une seule. Pour cela, nous utilisons un classificateur à classes multiples, avec autant de classes que de jetons dans le vocabulaire du jetoniseur. Nous associons le jeton 0, qui normalement représente un jeton spécial qui n'est pas utilisé lors de la jetonisation, pour indiquer qu'il ne manque pas de mot avant. Ainsi, pour un exemple donné, son étiquette sera un vecteur  $y$  de taille 64, tel que sa  $i$ -ème composante est définie par:

$$y_i = \begin{cases} k & \text{s'il manque le } k\text{-ème jeton du vocabulaire en avant du } i\text{-ème jeton de l'exemple} \\ 0 & \text{sinon} \end{cases}$$

Ce modèle a été entraîné avec 50% de phrases complètes, et 50% de phrases auxquelles un mot ou une ponctuation a été retiré. Un des avantages de ce modèle est qu'il est capable de prédire pour une phrase plusieurs jetons manquants à la fois.

## 2.3. Autres variantes d'encodage

La liste des possibilités pour l'encodage ne cesse de grandir à chaque année. Nous énumérons ici une autre variante et une méthode simple de réduction de modèle.

### 2.3.1. Troncature

La troncature est tout simplement la suppression des dernières couches des Transformers. Chaque module d'attention apprend une tâche spécifique à partir de la représentation donnée par la couche précédente. Ainsi, enlever un module intermédiaire détruit la chaîne de traitement et oblige un ré-entraînement du modèle via, par exemple, la distillation. Cependant, garder les  $k$  premières couches du modèle permet d'avoir une représentation à partir d'une chaîne de module déjà stable. La représentation devient moins dense mais est quand même bonne (voir tableau 2.2).

### 2.3.2. MobileBert

MobileBERT[8] est à la base un modèle destiné aux appareils à plus faible capacité de calculs, tels que les téléphones intelligents. C'est un encodeur qui a été distillé à partir du BERT<sub>LARGE</sub> en suivant un procédé légèrement différent de DistilBERT à cause de son architecture.

MobileBERT est un Transformer qui a 24 couches, autant que son parent. Cependant, ses

couches sont beaucoup plus fines et contiennent des goulots d'étranglement supplémentaires pour rendre la distillation possible. En effet, pour pouvoir effectuer la distillation d'un encodeur, il faut normalement apparier des couches de l'enfant avec des couches du parent de même dimension. Bien que l'objectif de base semblât concorder avec les besoins de l'entreprise, les résultats obtenus étaient moins bons comparé à DistilRoberta: le modèle tronqué avec 12 couches obtient 80,6% d'exactitude pour la détection de phrases incomplètes, pour une vitesse à l'entraînement de 1142 phrases/seconde.

## 2.4. Analyse de la performance et Optimisation

### 2.4.1. Métriques

Afin de déterminer les forces et les faiblesses des modèles, nous avons ajouté deux autres métriques:

- la **précision** est le nombre de vrais positifs<sup>2</sup> divisé par le nombre de détections positives. Cette mesure permet de déterminer si le modèle est précautionneux: une bonne précision signifie que le modèle ne détectera un positif que quand il est assez sûr de lui.
- le **rappel** est le nombre de vrais positifs divisé par le nombre de positifs de base. Il mesure à quel point le modèle arrive à détecter des cas positifs.

Ces deux mesures sont très importantes et on peut souvent optimiser l'un ou l'autre en jouant avec les hyper-paramètres. Cependant, comme ça n'optimisera pas les deux en même temps, on cherche souvent un juste milieu (comme les F-mesures).

De plus, malgré la combinaison des têtes des modèles, il est toujours possible d'extraire les métriques pour chaque sous-tâche:

- pour la détection de phrases incomplètes, il suffit de regarder si une des classes 0 a une probabilité trop faible;
- pour la détection de la position d'un mot manquant, on peut regarder quelle position a une classe 0 avec faible probabilité;
- pour la prédiction du mot manquant, on peut regarder l'argument maximal à la position trouvée.

### 2.4.2. Optimisation

En notant  $s_{ijk}$  les scores obtenus par le modèle, on calcule les probabilités de la  $k$ -ème classe à la  $j$ -ème position de la  $i$ -ème phrase, en faisant un softmax par rapport à la dernière dimension:  $p_{ij}(k) = \frac{e^{s_{ijk}}}{\sum_k e^{s_{ijk}}}$ . On obtient deux hyper-paramètres:

- pour gérer s'il manque un mot dans la phrase  $i$ :  $p_1 = \max_j(1 - p_{ij}(0))$ ;

---

<sup>2</sup>les positifs de base qui sont détectés par le modèle

Métriques sur l'ensemble de validation (en %)						
	DistilRoberta3			DistilCamemBERT3		
	Exactitude	Précision	Rappel	Exactitude	Précision	Rappel
<b>Dét. Incomplet</b>	78,00	93,55	60,00	84,40	94,87	72,33
<b>Dét. Position</b>	75,67	86,28	55,34	81,56	83,77	66,58
<b>Préd. Mot</b>	62,62	45,47	29,16	68,46	52,54	40,06

**Tableau 2.3.** Derniers résultats obtenus pour les modèles.

- pour contrôler si la meilleure prédiction de mot est acceptable:  $p_2 = \max_{k \geq 1} \max_j (1 - p_{ij}(0)) \cdot p_{ij}(k)$ .

On optimise l'exactitude de la détection de phrases incomplètes sur le corpus de validation, grâce au paramètre  $p_1$ . Puis, on optimise la précision du prédicteur de mots sur le corpus de validation, grâce au paramètre  $p_2$ .

De plus, les fréquences des classes n'étant pas du tout balancées entre la classe 0 et les autres classes (il y a autant d'exemple de la classe 0 que de toutes les autres classes réunies), nous avons essayé de jouer avec les poids des classes passés à l'optimisateur. Contre les attentes de la littérature qui prônent de diminuer le poids des classes contenant plus d'exemples, nous sommes tombés sur un cas particulier. En effet, baisser le poids de la classe 0 de sorte à équilibrer l'entraînement a eu pour effet une exactitude nulle. En revanche, augmenter le poids de la classe 0 (fois 3) a augmenté la précision des modèles.

On obtient les résultats du tableau 2.3.



# Chapitre 3

---

## Développement et Futures directions

### 3.1. Développement du modèle

Afin de voir si les modèles obtenus satisfont aux exigences de l'entreprise en termes de vitesse d'inférence, on a consacré une partie du stage au développement du modèle sur Xcode, l'outil principal de développement d'application sur les plateformes d'Apple.

Le développement du modèle se décompose en quatre étapes:

- la conversion du modèle PyTorch vers le format MLModel de CoreML. Cette conversion a pu être effectuée via la librairie CoreMLTools. Durant le stage, la conversion directe d'un modèle PyTorch vers MLModel était encore en version bêta, et plusieurs ajustements ont dû être effectués de notre côté pour réussir à faire cette conversion;
- la jetonisation des phrases qui n'est pas incluse dans le modèle et a dû être codée de zéro pour SentencePiece (le jetoniseur de CamemBERT) et BPE (celui de RoBERTa);
- la prédiction qui n'est pas explicitement incluse dans la sortie du modèle.

Après avoir terminé de développer le modèle sur Xcode<sup>1</sup>, nous avons pu tester sa vitesse sur un Mac sans on a obtenu une vitesse de 58ms/phrasede pour notre modèle, ce qui est encore trop lent pour l'entreprise. C'est pourquoi, nous continuons d'explorer les méthodes permettant d'augmenter la vitesse des modèles, tout en minimisant la perte de capacité.

### 3.2. Futures améliorations

#### 3.2.1. Amélioration des données d'entraînement

Après avoir analysé les métriques par catégorie de mots, nous avons remarqué que plusieurs catégories comme les noms, adjectifs et les adverbes, sont difficiles à prédire. En effet, deux problèmes peuvent se poser par rapport à ces catégories:

---

<sup>1</sup><https://developer.apple.com/xcode/>

- Enlever un mot ne rend pas forcément une phrase incomplète  
ex: en enlevant "belle" de "Elle vend cette belle maison."
- Plusieurs mots peuvent être possibles lorsqu'il manque un mot  
ex: "Il aime les \*\*\*\*\*."  
Déterminer exactement le mot attendu devient alors une tâche impossible sans contexte global.

D'autre part, nous avons constaté que face à des exemples particuliers, les prédictions peuvent générer des problèmes importants qui reviennent souvent dans les modèles de traitement de la langue: les biais<sup>2</sup>.

Pour résoudre ces problèmes, une solution qui s'aligne avec le point précédent pourrait être de filtrer les classes de prédiction, en ne conservant que des mots qui ne peuvent induire de biais.

### 3.2.2. Amélioration du modèle

Afin de visualiser les forces et les faiblesses du modèle, la librairie Captum<sup>3</sup> s'avère très intéressante. Elle permet d'enquêter les problèmes dans l'utilisation pratique du modèle en évaluant la contribution de chaque mot dans sa décision (voir 3.1). Cette enquête pourrait aboutir aux développements complémentaires requis pour améliorer la performance du modèle.

Visualize attributions based on Integrated Gradients

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
neg	neg (0.05)	pos	-1.43	The BBC highlights two examples that illustrate just how much the UK 's energy networks have changed .
pos	pos (0.64)	pos	-0.90	The BBC highlights two examples illustrate just how much the UK 's energy networks have changed .
pos	pos (0.99)	pos	0.08	The BBC two examples that illustrate just how much the UK 's energy networks have changed .

**Fig. 3.1.** Trois exemples de phrases visualisées avec Captum. Dans le premier exemple, il s'agit de la phrase complète. Ici, c'est le premier et le dernier mot qui ont eu la plus grande importance dans la décision du modèle de prédire qu'il ne manque pas de mot. Dans le deuxième exemple, on a enlevé le terme "il a prédit qu'il manque un mot à cause du terme "BBC". Dans le troisième exemple, il a prédit qu'il manque un mot à cause de la partie "BBC two examples that".

<sup>2</sup>Les prédictions peuvent être biaisées selon, par exemple, le genre (masculin, féminin), la religion ou les origines d'un individu.

<sup>3</sup><https://github.com/pytorch/captum>

# Conclusions

---

Ce stage m'a permis de me spécialiser dans le traitement automatique de la langue. J'ai approfondi mes connaissances sur les derniers modèles de l'état de l'art. J'ai pu me familiariser avec plusieurs librairies liées au traitement de texte et à la visualisation. J'ai exploré différentes manières d'obtenir des modèles plus petits pour être capable de les déployer sur des machines à faible capacité de calculs. J'ai pu voir comment déployer des modèles d'apprentissage automatique avec Xcode.

Au niveau de la performance des modèles, les résultats obtenus sont légèrement meilleurs en français qu'en anglais. De manière générale pour les deux modèles, ils étaient capables de détecter de manière assez précise lorsqu'il manque un mot dans une phrase, et où le mot manquant se trouve. Par contre, la prédiction du mot manquant a obtenu des résultats un peu moins satisfaisant, notamment aussi à cause des problèmes de biais. Il faudrait faire des ajustements au niveau des classes de prédiction.





# Références bibliographiques

---

- [1] Jacob DEVLIN, Ming-Wei CHANG, Kenton LEE et Kristina TOUTANOVA : BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.
- [2] Philipp KOEHN, Hieu HOANG, Alexandra BIRCH, Chris CALLISON-BURCH, Marcello FEDERICO, Nicola BERTOLDI, Brooke COWAN, Wade SHEN, Christine MORAN, Richard ZENS, Chris DYER, Ondřej BOJAR, Alexandra CONSTANTIN et Evan HERBST : Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, juin 2007. Association for Computational Linguistics.
- [3] Taku KUDO et John RICHARDSON : SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, novembre 2018. Association for Computational Linguistics.
- [4] Hang LE, Loïc VIAL, Jibril FREJ, Vincent SEGONNE, Maximin COAVOUX, Benjamin LECOUTEUX, Alexandre ALLAUZEN, Benoît CRABBÉ, Laurent BESACIER et Didier SCHWAB : Flaubert: Unsupervised language model pre-training for french. *CoRR*, abs/1912.05372, 2019.
- [5] Yinhan LIU, Myle OTT, Naman GOYAL, Jingfei DU, Mandar JOSHI, Danqi CHEN, Omer LEVY, Mike LEWIS, Luke ZETTMAYER et Veselin STOYANOV : Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.
- [6] Louis MARTIN, Benjamin MÜLLER, Pedro Javier Ortiz SUÁREZ, Yoann DUPONT, Laurent ROMARY, Éric Villemonte de la CLERGERIE, Djamé SEDDAH et Benoît SAGOT : Camembert: a tasty french language model. *CoRR*, abs/1911.03894, 2019.
- [7] Victor SANH, Lysandre DEBUT, Julien CHAUMOND et Thomas WOLF : Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.
- [8] Zhiqing SUN, Hongkun YU, Xiaodan SONG, Renjie LIU, Yiming YANG et Denny ZHOU : Mobilebert: a compact task-agnostic bert for resource-limited devices, 2020.
- [9] Ashish VASWANI, Noam SHAZEER, Niki PARMAR, Jakob USZKOREIT, Llion JONES, Aidan N. GOMEZ, Lukasz KAISER et Illia POLOSUKHIN : Attention is all you need. *CoRR*, abs/1706.03762, 2017.
- [10] Alex WANG, Yada PRUKSACHATKUN, Nikita NANGIA, Amanpreet SINGH, Julian MICHAEL, Felix HILL, Omer LEVY et Samuel R. BOWMAN : Superglue: A stickier benchmark for general-purpose language understanding systems. *CoRR*, abs/1905.00537, 2019.
- [11] Alex WANG, Amanpreet SINGH, Julian MICHAEL, Felix HILL, Omer LEVY et Samuel R. BOWMAN : GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.