# DistilCamemBERT : une distillation du modèle français CamemBERT

Cyrile Delestre, Abibatou Amar

# DistilCamemBERT : une distillation du modèle français CamemBERT

Cyrile Delestre*[1] et Abibatou Amar†[1]

[1]Crédit Mutuel Arkéa, 32 Rue Mirabeau, 29480 Le Relecq-Kerhuon, France

## Abstract

Les modélisations de *Natural Language Processing* (NLP) modernes à base de structure *Transformer* représentent l'état de l'art en terme de performances sur des tâches très diversifiées. Cependant ces modélisations sont complexes et représentent plusieurs centaines de millions de paramètres pour les plus modestes d'entre elles. Ce constat peut nuire à leur adoption au niveau industriel rendant la mise à l'échelle sur une infrastructure raisonnable et/ou conforme aux responsabilités sociétales et environnementales difficile. Dans ce but nous présentons dans cet article une modélisation permettant de réduire drastiquement le coût calculatoire d'un modèle français très connu (CamemBERT), tout en préservant de bonnes performances.

**Mots-clef**: Distillation, CamemBERT, Transformers, NLP.

## 1 Introduction

C'est en juin 2017, que l'article fondateur de tout un pan des modélisations NLP est publié par Google, introduisant la structure Transformer [1], et en juin 2018 que la première modélisation réalisée par OpenAI reprenant cette structure est publiée sous le nom de *Generative Pre-Traning* (GPT), utilisant une variante auto-régressive [2]. Puis, c'est en octobre de cette même année que Google introduit la modélisation *Bidirectional Encoder Representations from Transformers* (BERT) [3]. Finalement, en juillet 2019, Meta (anciennement Facebook) introduit une variante sur la tokenization et la stratégie d'apprentissage offrant de meilleures performances sur les tâches de l'état de l'art, *Robustly Optimized BERT pretraining Approach* (RoBERTa) [4]. Le français n'est pas en reste avec deux modélisations. En mai 2020, une nommée FlauBERT [5], est adaptée de BERT sur un corpus français et en juillet 2020, CamemBERT [6] est quant à elle une adaptation de RoBERTa. Ces modélisations, bien que très performantes, sont très coûteuses en temps d'inférence. C'est dans l'optique de réduire cette contrainte que HuggingFace, en octobre 2019, introduit DistilBERT [7], une stratégie de distillation de la modélisation BERT afin de diviser le temps d'inférence par deux en réduisant la taille de la structure du réseau tout en limitant les pertes de performances sur les tâches de NLP. Cette stratégie d'apprentissage a donné toute une série de modélisations Distil* dans diverses langues, mais malheureusement pas en Français. C'est pour cette raison que nous introduisons aujourd'hui DistilCamemBERT, une distillation du modèle CamemBERT.

Dans une première partie nous aborderons les principes d'attention des réseaux Transformers afin de faciliter la description du principe de distillation sur ce type de modélisation dans une seconde partie. Puis, finalement, dans la troisième et dernière partie nous nous pencherons sur l'étude de performance de notre modèle DistilCamemBERT au travers de divers benchmarcks permettant de le confronter à d'autres types de modélisation dans des utilisations typiques.

## 2 Transformers

Dans cette section nous allons introduire la structure Transformer. Pour des raisons de simplicité nous omettrons volontairement les normalisations et les boucles résiduelles afin de faciliter l'explication et alléger les écritures. Nous définirons par la matrice $\mathbf{H} \in \mathbb{R}^{d_\mathbf{h} \times n}$ la représentation latente des tokens en sortie d'une couche Transformer où $d_\mathbf{h}$ représente la dimension la-

*cyrile.delestre@arkea.com

†amar.abibatou@gmail.com

tente des tokens et $n$ le nombre de tokens que compose la/le phrase/paragraphe/texte en entrée de la modélisation. Ainsi en sortie du $l$-ième Transformer nous avons $\mathbf{H}_l = [\mathbf{h}_{l,1}, \ldots, \mathbf{h}_{l,n}]$.

La première représentation $\mathbf{H}_0$ en entrée de la première couche de Transformer est la concaténation des *word embeddings* sommés avec l'encodeur de position qui permet d'ajouter *une texture* au mot en fonction de sa position dans la phrase. Il est à noter que le *positional encoder* est une fonction prédéfinie dans l'article original, alors que ce sont des coefficients appris par le réseau durant la phase d'entraînement pour leurs déclinaisons en BERT et RoBERTa.

Ainsi la $l$-ième couche Transformer peut être écrite comme suit :

$$\mathbf{H}_l = \text{Transformer}_l(\mathbf{H}_{l-1}),\ l \in [\![1, L]\!] \tag{1}$$

Les réseaux de neurones de type Transformers sont en réalité des structures relativement simples qui s'articulent autour d'une mémoire fonctionnelle. C'est-à-dire une mémoire qui permet de préserver la dérivation des fonctions composées rétro-propageant le gradient dans le réseau. C'est cette partie qui constitue la mécanique d'attention (*Head Attention*).

Ainsi, à chaque couche, une représentation d'une requête (*Query*), d'une clef (*Key*) et d'une valeur (*Value*) sont déterminées par les transformations linéaires suivantes :

$$\mathbf{Q}_{j,l}(\mathbf{H}) = \mathbf{W}^Q_{j,l}\mathbf{H} + \mathbf{1}_{1\times n} \otimes \mathbf{b}^Q_l \tag{2}$$

$$\mathbf{K}_{j,l}(\mathbf{H}) = \mathbf{W}^K_{j,l}\mathbf{H} + \mathbf{1}_{1\times n} \otimes \mathbf{b}^K_l \tag{3}$$

$$\mathbf{V}_{j,l}(\mathbf{H}) = \mathbf{W}^V_{j,l}\mathbf{H} + \mathbf{1}_{1\times n} \otimes \mathbf{b}^V_l \tag{4}$$

avec $\mathbf{W}^k_{j,l} \in \mathbb{R}^{\frac{d_\mathbf{h}}{J}\times d_\mathbf{h}}\ \forall k \in \{Q, K, V\}$, $J$ est le nombre de *head attention* mis en parallèle à chaque couche d'attention (resp. $j$ est le $j$-ième), $\mathbf{1}_{1\times n}$ est un vecteur ligne de $n$ 1 et $\otimes$ est le produit de Kronecker.

Ainsi, plus une requête va concorder avec une clef par relation de colinéarité, plus la valeur émise associée à cette clef sera importante. Cette mécanique est modélisée de cette manière :

$$\text{Head}_{j,l}(\mathbf{H}) = \mathbf{V}_{j,l}\ \text{softmax}\left(\frac{\mathbf{Q}^T_{j,l}\mathbf{K}_{j,l}}{\sqrt{d_\mathbf{h}/J}}\right)^T \tag{5}$$

où l'opérateur softmax est calculé par rapport à la dimension des clefs (dernière dimension), permettant de répartir la probabilité des requêtes sur l'ensemble des clefs, et le rapport $\sqrt{d_\mathbf{h}/J}$ peut être vu comme un facteur de limite d'échelle permettant de maîtriser la dynamique du produit scalaire ($\frac{\mathbf{q}^T\mathbf{k}}{\sqrt{d_\mathbf{h}/J}} \xrightarrow[d_\mathbf{h}/J\to\infty]{\mathcal{L}} Z \sim \mathcal{N}(0,1)$) afin d'améliorer la rétro-propagation du gradient dans l'opérateur softmax (car des grandes valeurs entraînent de petits gradients).

A l'aide d'une telle mécanique il est facile de comprendre que deux éléments peuvent interagir entre eux sans être à proximité, *a contrario* des méthodes de détection d'attention utilisant des réseaux de neurones récurrents, comme c'est le cas pour ELMo [8]. De même l'interaction peut être indifféremment causale ou non vis-à-vis de la séquence de tokens, ce qui justifie le terme *bidirectionnal* dans l'acronyme BERT.

Comme sous-entendu précédemment avec la valeur $J$, plusieurs mécaniques d'attention sont déployées en parallèle à chaque couche de Transformer, c'est ce qui est nommé le *Multi-Head Attention*. Il s'agit de la simple concaténation des différentes attentions qui sont remélangées linéairement afin de revenir à une signification homogène de représentation latente par token :

$$\begin{aligned}\text{MultiHead}_l(\mathbf{H}) = \mathbf{W}^{out}_l &\left[\text{Head}^T_{1,l}, \ldots, \text{Head}^T_{J,l}\right]^T \\ &+ \mathbf{1}_{1\times n} \otimes \mathbf{b}^{out}_l\end{aligned} \tag{6}$$

avec $\mathbf{W}^{out}_l \in \mathbb{R}^{d_\mathbf{h}\times d_\mathbf{h}}$ .

Finalement la couche de Transformer se concrétise par un dernier mélange non-linéaire nommé réseau *feed-forward* :

$$\text{Transformer}_l(\mathbf{H}) = FF_l(\text{MultiHead}_l(\mathbf{H})) \tag{7}$$

avec

$$\begin{aligned}FF_l(\mathbf{H}) = \mathbf{W}^{forward}_l &\text{ReLu}(\mathbf{W}^{feed}_l\mathbf{H} + \mathbf{1}_{1\times n} \otimes \mathbf{b}^{feed}_l) \\ &+ \mathbf{1}_{1\times n} \otimes \mathbf{b}^{forward}_l\end{aligned} \tag{8}$$

où $\mathbf{W}^{feed}_l \in \mathbb{R}^{E\times d_\mathbf{h}}$, $\mathbf{W}^{forward}_l \in \mathbb{R}^{d_\mathbf{h}\times E}$ et généralement $E > d_\mathbf{h}$.

En prenant en compte les couches de normalisation il est possible de déterminer le nombre de paramètres présents dans le réseau à l'aide de l'opération suivante[1] :

$$p = L(4d^2_\mathbf{h} + 2d_\mathbf{h}E + 9d_\mathbf{h} + E) + d_\mathbf{h}(|\mathcal{W}| + |\mathcal{E}| + 2) \tag{9}$$

où $|\mathcal{W}|$ est la taille du dictionnaire et $|\mathcal{E}|$ est le nombre maximum de tokens possible pour le *positional encoder*. Par exemple, pour CamemBERT, $|\mathcal{W}| = 32'005$ et $|\mathcal{E}| = 514$. A l'aide de ces éléments il est facile de décrire la structure CamemBERT qu'on peut résumer dans le tableau 1.

[1] La dernière couche de *pooling* n'a pas été prise en compte dans le calcul.

# 3 La distillation

Le domaine du *Knowledge Distillation* (KD) est une discipline relativement récente introduite en 2006 [9] afin de contrer des modélisations de plus en plus complexes. C'est en 2015 [10] que l'approche est généralisée et l'intérêt est démontré sur des modèles de classification d'image (qui à l'époque était le domaine où l'on trouvait les plus gros réseaux de neurones, donc les plus coûteux). Le principe est très facile à formaliser : le but est d'entraîner un modèle moins coûteux en nombre de paramètres (dans le lexique du domaine KD il s'agit du modèle **étudiant**) en s'aidant d'un modèle de référence (modèle **professeur**).

C'est en 2019 [7] que la société HuggingFace introduit pour la première fois une technique de distillation sur un modèle de type Transformer, alias BERT. Il existe aujourd'hui d'autre approches qui ont été appliquées à ce type de modélisation (MobileBERT [11], MiniLM [12] en sont deux exemples), mais DistilBERT est le premier à en montrer l'intérêt.

## 3.1 Structure de l'étudiant

Distil* a pour objectif de réduire le nombre de couches Transformers du modèle d'origine par un facteur 2. La structure est résumée dans la tableau 1. Pour l'initialisation de l'entraînement, les paramètres de la partie *word embedding* et *positional encoding* sont copiés ainsi que les paramètres d'une couche Transformer sur deux. Ceci permet de réduire la distance entre le minimum local objectif et celui du modèle de référence.

| modèle | $L$ | $d_{\mathbf{h}}$ | $J$ | $E$ | $p$ |
|---|---|---|---|---|---|
| CamemBERT | 12 | 768 | 12 | 3'072 | 110M |
| DistilCamemBERT | 6 | 768 | 12 | 3'072 | 67,5M |

Table 1: Description du réseau CamemBERT et DistilCamemBERT.

## 3.2 Fonctions coût

Dans cette partie, nous allons lister les différentes fonctions coût utiles à l'apprentissage de la modélisation. Afin d'aider à la compréhension et l'explication des différentes fonctions coûts qui composent la fonction coût d'apprentissage, nous avons réalisé la figure 1 qui illustre les éléments de la structure Transformer utilisée que ce soit pour le modèle professeur (CamemBERT) ou étudiant (DistilCamemBERT).

Nous posons $\mathcal{O}$ l'ensemble des observations

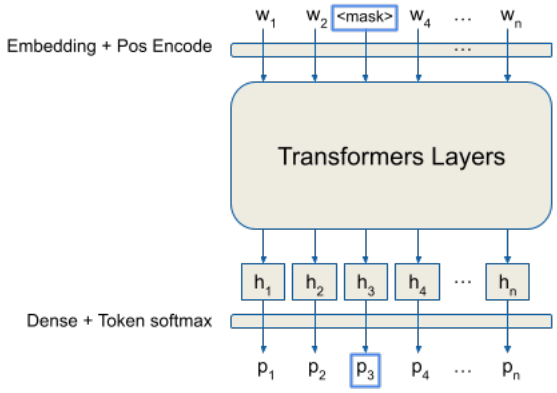


Figure 1: Schéma simplifié d'un réseau Transformer dans le cadre de DistilCamemBERT. Le cadrage bleu permet d'illustrer la correspondance entre un token masqué et la distribution de probabilité sur le dictionnaire correspondant pour une tâche de MLM.

d'apprentissage et $\mathcal{M}$ l'ensemble des tokens masqués. De même, l'exposant $s$ de $x^s$ fait référence à la partie étudiante et $t$ de $x^t$ au modèle professeur.

**Soft label :** cette fonction coût caractérise les méthodes de distillation sur les modèles de classification. Il s'agit d'exploiter la densité de probabilité sur l'ensemble du dictionnaire du modèle professeur et de rapprocher le comportement étudiant de celui-ci. La minimisation de la divergence de Kullback-Leibler sur chaque token permet de garantir un comportement semblable :

$$SoftLabelLoss = \frac{1}{\sum_{k\in\mathcal{O}} n_k} \sum_{k\in\mathcal{O}} \sum_{0\leq i<n_k} D_{KL}(p^s_{k,i} \| p^t_{k,i}) \tag{10}$$

Le principal reproche de cette fonction coût est de focaliser l'apprentissage du modèle élève sur la tâche d'estimation des tokens manquants (tâche MLM pour *Masked Language Model*). Elle ne témoigne pas du fait que la modélisation est générique et que, par *downstream*, le modèle s'adapte à des tâches très variées.

**Similarité Cosine :** a contrario, la dernière couche cachée représentant la dernière représentation latente des tokens est située en amont du classifieur. Une contrainte sur la colinéarité entre étudiant et professeur est donc appliquée :

$$CosineLoss = \frac{1}{\sum_{k\in\mathcal{O}} n_k} \sum_{k\in\mathcal{O}} \sum_{0\leq i<n_k} \frac{(\mathbf{h}^s_{k,i})^T \mathbf{h}^t_{k,i}}{\|\mathbf{h}^s_{k,i}\| \|\mathbf{h}^t_{k,i}\|} \tag{11}$$

**Tâche MLM :** il s'agit de la principale tâche d'entraînement du modèle professeur, elle est considéré

comme étant générique. Il s'agit également d'une partie de l'apprentissage en autonomie pour le modèle étudiant, car ne s'appuyant pas sur les résultats du modèle professeur :

$$MLMLoss = \frac{-1}{\sum_{k \in \mathcal{O}} |\mathcal{M}_k|} \sum_{k \in \mathcal{O}} \sum_{i \in \mathcal{M}_k} p^w_{k,i} \log(p^s_{k,i}) \tag{12}$$

où $p^w \in \{0; 1\}$ est le *hard label* du mot masqué.

Finalement la fonction coût d'apprentissage est un mélange linéaire des trois précédentes. Le choix des coefficients a été choisi arbitrairement. L'idée était d'appliquer plus de poids à la tâche d'imitation des *soft label* représentant, dans un contexte de classification, la fonction coût "classique" d'une distillation, puis de garder l'apprentissage MLM en tâche de fond. Pour finir, la fonction est calculée comme suit :

$$\begin{aligned} Loss = 0.5 \times SoftLabelLoss + 0.3 \times CosineLoss \\ + 0.2 \times MLMLoss \end{aligned} \tag{13}$$

# 4 Résultats

Afin d'estimer la performance du modèle nous allons comparer la modélisation à CamemBERT dans 4 tâches types de ce genre de modélisation. En effet, le modèle est optimisé pour le moment pour de l'estimation de tokens cachés qui est une tâche générique. Il est possible via un *fine tuning* d'adapter le modèle facilement à d'autres tâches plus utiles (*downstream task*). Quand c'est possible nous utiliserons des modélisations CamemBERT déjà existantes présentes sur la plateforme de partage HuggingFace-Hub : Analyse de sentiment [14], *Natural Language Inference* [15], *Question-Answering* [16].

Il est difficile d'être exhaustif dans l'évaluation des performances de ce type de modélisation, mais afin d'avoir une bonne vision globale des capacités de DistilCamemBERT nous avons choisit 4 tâches faisant intervenir 4 moyens différents d'utiliser l'information contenue dans le texte. Une vue des différentes approches est schématisée sur la figure 2.

L'ensemble des résultats sur les différentes tâches est disponible dans la tableau 2 où la performance est mesurée avec la métrique f1-score.

## 4.1 Analyse de sentiment

Une tâche de classification de texte vient exploiter le premier token qui est dans le cas de BERT un token de classification, ou dans RoBERTa (CamemBERT) le token <p>. Contrairement à ce que laisse croire le token, ce token n'est utilisé qu'en début d'entrée, il ne sépare ni les phrases, ni les paragraphes si plusieurs devaient être en entrée. Ainsi ce token est neutre et placé toujours au même endroit durant l'apprentissage, il contient donc l'information de l'ensemble du texte. Cette tâche va donc exploiter les informations *intra* texte.

Il s'agit d'une classification bi-classe qui vise à estimer "positif" ou "négatif" des commentaires d'utilisateurs issus du site AlloCiné, donc très orientés critique de films, et d'Amazon, plus orientés produit et qualité de service.

Globalement on peut voir que les deux modélisations donnent des résultats très proches.

## 4.2 Named Entity Recognition

Il s'agit d'une tâche de classification de tokens qui permet d'identifier des éléments d'un texte. A l'instar de la classification de texte, cette tâche exploite les informations *intra* texte. L'application la plus représentative de cette tâche est la détection d'entités nommées (*Named Entity Recognition*) ou simplement tâche NER.

Ici la modélisation reconnaît 4 classes : les personnes, les organisations, les lieux et les références culturelles nommées.

Pour mesurer les performances pour cette tâche, nous avons utilisé la définition du f1-score servant pour les applications de question-réponse, qui mesure la qualité de l'intersection des tokens classés et de ceux issus de la vérité terrain [13]. Sur cette tâche également on peut voir que la performance de la modélisation DistilCamemBERT est proche de celle de son homologue CamemBERT.

## 4.3 Natural Language Inference

Première tâche de *cross-encoding* qui exploite les relations entre deux textes en entrée, elle est également connue sous le nom de *Recognizing Textual Entailment* (RTE). Le premier texte est appelé prémisse et le second hypothèse. La tâche peut être décrite de la manière suivante :

$$P(pre = c \in \{contradiction, implication, neutre\} | hyp) \tag{14}$$

avec respectivement *pre* et *hyp* pour prémisse et hypothèse. Cette tâche a la particularité de pouvoir créer des classifieurs "zero-shot", c'est-à-dire des classifica-

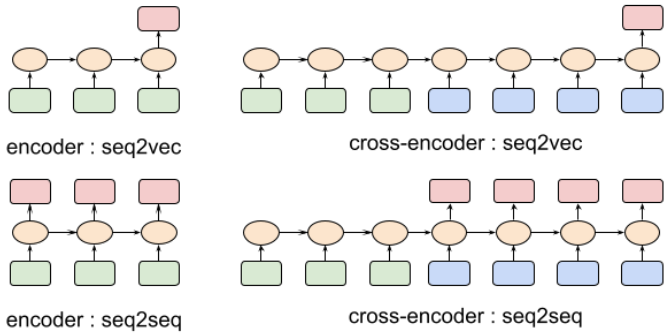


Figure 2: Différents types de structure classiques qu'on peut rencontrer en NLP. Les cases vertes (resp. bleues) représentent les tokens du premier texte (resp. second texte), les ovales représentent le processus de transformation d'attention et les cases en rouge la sortie du modèle qu'on souhaite exploiter.

tions sans entraînement :

$$P(hyp = i \in \mathcal{C}|pre) = \frac{e^{P(pre=implication|hyp=i)}}{\sum_{j\in\mathcal{C}} e^{P(pre=implication|hyp=i)}} \tag{15}$$

avec $\mathcal{C}$ l'ensemble des classes possibles.

Il s'agit indéniablement d'une tâche plus complexe car elle exploite des informations *extra* texte. On constate une perte de performance, bien que celle-ci ne soit pas dramatique.

### 4.4 Question-Answering

La dernière tâche testée est celle de question-réponse correspondant à une structure de *cross-encoding* de tokens labellisés. L'entrée se compose de deux textes, le premier est le "contexte" et le second la "question". L'objectif est d'estimer le début et la fin de la réponse à la question dans le contexte via la labellisation des tokens. Au même titre que la tâche précédente, le modèle doit exploiter les relations *extra* texte.

A l'instar de la tâche NER, la définition du f1-score utilisée est celle servant à mesurer les performances sur ce type de tâche [13]. Cette tâche est très complexe et on constate une perte significative par rapport au modèle de référence.

## 5 Conclusions

Avec une modélisation 2 fois plus petite il a été constaté que le temps d'inférence est bien 2 fois plus court.

Le modèle dans une approche *encoder* fonctionne très bien et possède un très bon comportement, voire même un comportement un peu meilleur que la modélisation CamemBERT classique. Cette dernière remarque est toutefois à prendre avec des pincettes, la différence de performance n'étant pas significative et peut-être due à une différence de "qualité" du *fine tuning* du modèle pré-entraîné.

Le modèle DistilCamemBERT montre ses limites dans les modélisations de type *cross-encoding* avec une perte de 5 points notée sur la tâche de NLI et 15 points de perte sur la tâche de *question-answering*. Les performances n'en demeurent pas moins bonnes. Mais là où, dans une modélisation de type *encoder*, le gain est sans compromis (pas de perte de performance notable + gain en temps d'inférence), ici il s'agira d'un compromis entre temps d'exécution/performance que l'utilisateur final devra choisir en fonction de son objectif, de ses besoins et de ses contraintes.

Ces résultats ne sont pas surprenants. En effet, les modélisations à base d'un simple *encoder* vont plutôt se concentrer sur la compréhension et la structure interne d'une phrase ou d'un texte, là où les *cross-encoder* vont plus se concentrer sur la compréhension et la relation entre deux textes. On peut mettre en relation ce constat avec les analyses qui ont déjà était menées sur l'influence des couches [17]. En effet, la compréhension des mots (morphosyntaxique et syntaxique) se fait plutôt dans les couches basses de la modélisation (relativement proches du *word embedding*), alors que plus on monte dans des couches élevées, plus la modélisation acquiert une compréhension du texte (relation entre mots, entre phrases, etc.). Ainsi les structures avec un nombre de couches important sont plus disposées aux problématiques de *cross-encoding*.

DistilCamemBERT présenté ici est disponible sur la plateforme HuggingFace-Hub en *open-source*. Sous Python, il est très simple de charger le modèle, le code est représenté à la figure 3. Il en va de même pour toutes les *downstream task* présentées, figure 4.

| modèle | accélération | Sentiment (%) | NER (%) | NLI (%) | QA (%) |
|---|---|---|---|---|---|
| CamemBERT | ×1 | 95,74 | 88,93 | **81,68** | **79,57** |
| DistilCamemBERT | ×**2** | **97,57** | **89,12** | 77,48 | 62,65 |

Table 2: Comparaison des performances, mesurées en f1-score, entre le modèle CamemBERT et DistilCamemBERT sur différentes tâches communes à ce type de modélisation.

```
from transformers import (AutoTokenizer,
                          AutoModel)
tokenizer = AutoTokenizer.from_pretrained(
    "cmarkea/distilcamembert-base"
)
model = AutoModel.from_pretrained(
    "cmarkea/distilcamembert-base"
)
model.eval()
```

Figure 3: Chargement du modèle DistilCamemBERT.

```
from transformers import pipeline
task = pipeline(task=..., model=*)
```

Figure 4: Chargement d'une *downstream task*, remplacer * par la tâche associée : "cmarkea/distilcamembert-base-{sentiment, ner, nli, qa}" et le paramètre "task" par le nom de la tâche (se référer à la documentation de HuggingFace).

# References


[1] Vaswani, Ashish and Shazeer, Noam and Parmar, Niki and Uszkoreit, Jakob and Jones, Llion and Gomez, Aidan N and Kaiser, Łukasz and Polosukhin, Illia. "Attention Is All You Need", 31st Conference on Neural Information Processing Systems, 2017.

[2] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. "Improving language understanding by generative pre-training", OpenAI Blog, 2018.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of deep bidirectional transformers for language understanding". In NAACL-HLT, 2018.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar S. Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. "RoBERTa: A robustly optimized bert pretraining approach". ArXiv, abs/1907.11692, 2019.

[5] Hang Le, Loïc Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoît Crabbé, Laurent Besacier and Didier Schwab. "FlauBERT: Unsupervised Language Model Pre-training for French". Proceedings of The 12th Language Resources and Evaluation Conference, 2020.

[6] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de La Clergerie, Djamé Seddah and Benoît Sagot. "CamemBERT: a tasty French language model". Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020.

[7] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". arXiv preprint arXiv:1910.01108, 2019.

[8] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer. "Deep contextualized word representations". NAACL, 2018.

[9] Cristian Buciluǎ, Rich Caruana and Alexandru Niculescu-Mizil. "Model compression". Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, 2006.

[10] Geoffrey Hinton, Oriol Vinyals and Jeff Dean. "Distilling the Knowledge in a Neural Network". CoRR, abs/1503.02531, 2015.

[11] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. "MobileBERT: Task-agnostic compression of bert by progressive knowledge transfer", 2019.

[12] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang and Ming Zhou. "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers". Advances in Neural Information Processing Systems, 2020.

[13] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev and Percy Liang. "SQuAD: 100,000+ Questions for Machine Comprehension of Text". Conference on Empirical Methods in Natural Language Processing, 2016.

[14] Théophile Blard. "tblard/tf-allocine". https://huggingface.co/tblard/tf-allocine, 2020.

[15] Baptiste Doyen. "BaptisteDoyen/camembert-base-xnli". https://huggingface.co/BaptisteDoyen/camembert-base-xnli, 2021.

[16] Etatlab. "etalab-ia/camembert-base-squadFR-fquad-piaf". https://huggingface.co/etalab-ia/camembert-base-squadFR-fquad-piaf, 2020.

[17] Ganesh Jawahar, Benoît Sagot and Djamé Seddah. "What does BERT learn about the structure of language?". ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, 2019.