Interaction retardée dans l'encodeur du Transformer pour répondre efficacement aux questions dans un domaine ouvert

Wissam Siblini*, Mohamed Challal*
Charlotte Pasqual*

*Worldline Lyon
wissam.siblini@worldline.com

Résumé. La tâche de question-réponse sur un large corpus de documents (par exemple Wikipedia) est un défi majeur en informatique. Bien que les modèles de langage basés sur le Transformer tels que Bert aient montré une capacité à surpasser les humains pour extraire des réponses dans des petits passages de texte pré-sélectionnés, ils souffrent de leur grande complexité si l'espace de recherche est beaucoup plus grand. La façon la plus répandue de faire face à ce problème consiste à ajouter une étape préliminaire de recherche d'information pour filtrer fortement le corpus et ne conserver que les passages pertinents. Dans cet article, nous proposons une solution plus directe et complémentaire qui consiste à modifier l'architecture des modèles à base de Transformer pour permettre une gestion plus efficace des calculs. Les modèles qui en résultent sont compétitifs avec ceux d'origine et permettent, en domaine ouvert, une accélération significative des prédictions et parfois même une amélioration de la qualité de réponse.

1 Introduction

Depuis la parution du papier *Attention is all you need* de Vaswani et al. (2017), le modèle du Transformer n'a cessé de gagner en popularité dans le domaine du traitement automatique du langage naturel (TALN). Il est entraîné en deux étapes, une première de pré-entraînement coûteuse en temps et en données permettant de construire une représentation contextualisée des mots d'un vocabulaire, puis une seconde d'affinage moins onéreuse qui le spécialise dans une tâche de TALN bien précise. C'est de cette façon que Bert (Devlin et al., 2019), partie encodeur de l'architecture du Transformer, a montré une performance impressionnante sur de nombreuses tâches difficiles de compréhension du langage et a séduit un très large public par sa polyvalence et sa facilité d'utilisation. Des récentes variantes comme Albert (Lan et al., 2019) vont jusqu'à produire des réponses de meilleure qualité que l'humain.

Nous nous intéressons dans ce papier à la tâche de recherche d'une réponse à une question utilisateur dans un grand ensemble de documents textuels (par exemple, dans les millions de pages de l'encyclopédie Wikipedia). Les modèles de langage comme Bert sont efficaces sur une sous-tâche appelée Extractive Question Answering (eQA) dont le jeu de données de référence est SQuAD (Rajpurkar et al., 2016) : étant donnée une paire question-document, le but est de localiser la réponse dans le document. Mais sur notre tâche cible, appelée Open Domain

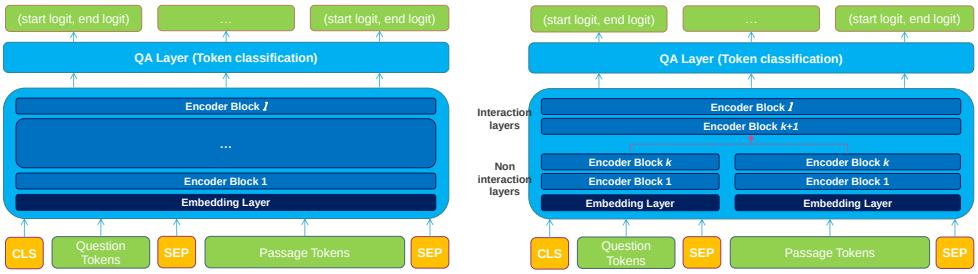
Question Answering (ODQA), le problème est plus complexe car pour chaque question l'espace de recherche est beaucoup plus grand. Les modèles basés sur le Transformer nécessitant un temps non négligeable pour traiter une seule paire question-paragraphe, ils ne peuvent évidemment pas en gérer des millions en temps réel. La solution la plus courante consiste alors à combiner le Transformer avec des techniques de recherche d'information (RI ou IR en anglais) (Manning et al., 2008) pour pré-sélectionner un sous ensemble de p documents pertinents et ainsi réduire les calculs. Une telle combinaison a fait ses preuves dans BertSerini (Yang et al., 2019) où le très répandu moteur d'indexation Lucene (reposant sur BM25) (Robertson et al., 1995) pour la partie IR a été combiné avec Bert pour la partie eQA. Nous proposons ici d'aborder la question de la complexité sous un angle plus direct et complémentaire qui consiste à modifier l'architecture du modèle d'eQA afin que de nombreux calculs puissent être effectués plus efficacement. Plus précisément, nous considérons un mécanisme d'interaction retardée dans les couches du Transformer, consistant à appliquer certaines opérations uniquement dans les derniers blocs. Nous implémentons ce mécanisme pour Bert et Albert, pour obtenir des variantes que nous appellerons DilBert (*Delaying Interaction Layers in Bert*) et DilAlbert. Nous étudions leur comportement d'abord pour la sous-tâche d'eQA (SQuAD) et montrons qu'ils sont compétitifs par rapport aux modèles de base. Ensuite, pour la tâche cible d'Open Domain Question Answering, nous montrons théoriquement la réduction de complexité induite par la proposition et confirmons empiriquement une accélération des calculs d'un ordre de grandeur sur GPU ou CPU. Enfin, nous évaluons les modèles sur le jeu de données d'ODQA de référence (OpenSQuAD, Chen et al. (2017)) en les combinant avec Lucene pour la partie IR comme Yang et al. (2019). Bien que DilBert (resp. DilAlbert) se comporte légèrement moins bien que Bert (resp. Albert) face à un unique paragraphe pertinent (SQuAD), il peut le surpasser dans le cadre ODQA (OpenSQuAD) lorsqu'il doit sélectionner la bonne réponse sur plusieurs paragraphes. Le code pour reproduire l'intégralité des expériences du papier, ainsi qu'un module intéractif de question-réponse avec Wikipédia, est fourni [1].

2 Travaux connexes

La tâche de question-réponse intéresse le monde de la recherche depuis très longtemps (Woods et WA, 1977) avec l'objectif de créer des moteurs de recherche intelligents pour parcourir des bases de documents non structurés à très grande échelle comme Wikipedia (Chen et al., 2017; Yang et al., 2019; Lee et al., 2018). Ce type de système est souvent conçu avec plusieurs couches qui sélectionnent successivement une portion de texte de plus en plus petite et précise. Cela commence généralement par une étape de **recherche d'information ad-hoc** avec un modèle *retriever* qui identifie les documents pertinents pour la question considérée. Ensuite, un algorithme conçu pour l'**extractive Question Answering**, appelé *reader*, identifie la portion de texte contenant la réponse dans les passages pré-sélectionnés.

La **recherche d'information ad-hoc** consiste généralement à : (i) appliquer un modèle d'encodage à tous les documents, (ii) encoder la question également, (iii) appliquer un modèle de scoring qui évalue la pertinence de chaque paire question-document à partir de leur encodage, et enfin (iv) trier les documents en conséquence (Manning et al., 2008). Pour l'encodage, les propositions vont de stratégies basées sur le vocabulaire et la fréquence telles que le sac-de-

1. https://github.com/wissam-sib/dilbert



(a) Modèle d'origine (b) Modèle avec interaction retardée

FIG. 1: Architecture générale, avec l blocs, des modèles de langage basés sur le Transformer (gauche) pour la tâche d'eQA et proposition avec interaction retardée (droite). Dans le deuxième cas, les k premiers blocs sans interaction sont appliqués indépendamment à la question et au paragraphe et les $l - k$ dernières couches d'interaction sont appliquées à l'ensemble. Le terme *start logit* (resp. *end logit*) fait référence à un score prédit par le modèle indiquant à quel point il est envisageable pour chaque token d'être le début (resp. la fin) de la réponse.

mots ou TF-IDF, aux plongements simple comme word2vec ou fortement contextualisés avec Bert (Devlin et al., 2019; MacAvaney et al., 2019). Pour le scoring, le choix le plus populaire est une mesure fixe de distance (cosinus ou euclidienne) mais d'autres stratégies comme les réseaux siamois ont été testées. Dans la littérature récente, MacAvaney et al. (2019) ont testé plusieurs combinaisons entre des techniques d'encodage contextualisé (Elmo et Bert) et des modèles de scoring (DRMM, KNRM, PACRR). En affinant Bert et en exploitant son encodage d'une façon originale, les auteurs ont obtenu les meilleurs résultats sur plusieurs jeux de données de référence. A côté de cela, il existe une méthode de base mais éprouvée appelée BM25 (Robertson et al., 1995), reposant sur un calcul de similarité pondérée par les statistiques de TF-IDF, qui reste jusqu'à aujourd'hui très compétitive avec les méthodes neuronales (Lin, 2019) ; elle est extrêmement rapide et bénéficie de décennies d'ingénierie fine, mise en œuvre par exemple dans l'implémentation de Lucene (Yang et al., 2018).

Pour la tâche d'**extractive Question Answering**, les algorithmes à l'état de l'art aujourd'hui sont basés sur la partie encodeur du Transformer tels que Bert (Devlin et al., 2019) ou Albert (Lan et al., 2019). Ces derniers ont progressivement remplacé l'usage des RNN (réseaux de neurones récurrents) dans la discipline en corrigeant deux problèmes, la disparition du gradient et la difficulté de paralléliser les calculs. Ils sont composés d'une couche de plongement en entrée, suivie d'une succession de l blocs d'encodeurs (qui implémentent un mécanisme d'auto-attention) et enfin d'une couche de sortie. Pour l'eQA, ils prennent en entrée une séquence de texte correspondant à la concaténation de la question et du passage de texte associé. La séquence entière est tokenisée et un caractère spécial [CLS] (resp. [SEP]) est ajouté au début (resp. entre la question et le passage et à la fin). Ensuite, les variables associées (identifiants de token, de segment, de position) sont fournies au modèle qui prédit deux probabilités pour chaque token du passage d'être le début et la fin de la plage de réponse (Figure 1a).

Les algorithmes pour **question-réponse en domaine ouvert** combinent **recherche d'information** et **eQA**. Un exemple emblématique est DrQA (Chen et al., 2017), un chatbot développé par Facebook capable de répondre aux questions en effectuant une recherche sur l'in-

tégralité du Wikipedia anglais en temps réel. Il repose sur un retriever combinant *TF-IDF* et *similarité du cosinus* qui sélectionne 5 documents pertinents et sur un reader de type *RNN multi-couches*. DrQA a été suivi d'autres propositions dans la littérature qui ont pu surpasser sa qualité de réponse en travaillant notamment sur le retriever : utilisation d'un re-ranker de paragraphe (Lee et al., 2018) où d'un retriever "minimal" sélectionnant directement des portions de texte petites mais pertinentes (Min et al., 2018). Entre-temps, Bert a vu le jour et Yang et al. (2019) ont saisi l'opportunité de proposer une première implémentation réussie en combinant le modèle de langage affiné sur SQuAD simplement avec un retriever basé sur Lucene. Cette combinaison a notamment obtenu la meilleure performance sur le banc d'expérience de référence OpenSQuAD (Chen et al., 2017). Depuis lors, quelques propositions (Wang et al., 2019; Xie et al., 2020) se sont concentrées sur une sélection plus efficace des passages et une mise en commun des réponses pour améliorer encore la performance. Dans toutes ces approches, la question du passage à l'échelle est toujours abordée du point de vue du retriever. Pourtant, le cadre de l'ODQA offre de nombreuses opportunités d'accélérer le reader (e.g. Bert), ce qui est la principale motivation derrière ce travail.

3 Retarder les couches d'interaction

Dans l'ODQA, l'ensemble des documents est quasi-statique. Si q questions sont posées et qu'il y a p documents [2] dans lesquels faire la recherche, un reader basé sur le Transformer effectuera des prédictions sur toutes les paires possibles ($q \times p$). Non seulement cela est très coûteux, mais les calculs doivent être effectués de manière interactive. Plus précisément, dans les couches d'encodeur, le mécanisme d'attention rend la représentation des éléments (tokens) des documents dépendant de la question, empêchant alors de pré-calculer des représentations intermédiaires pour les documents avant de connaître la question. Nous avons donc envisagé une modification structurelle des modèles existants pour à la fois réduire le nombre de calculs et rendre possible certains pré-traitements. L'idée clé est de "retarder" l'interaction entre la question et le document. Aussi, nous nous limitons à (i) des changements qui ne sont pas spécifiques à un modèle en particulier afin que la proposition puisse être appliquée à un maximum de modèles de langage basés sur l'architecture du Transformer et (ii) à ne pas introduire de nouveaux poids dans les modèles pour pouvoir bénéficier des poids pré-entraînés existants.

Le principe de l'approche proposée est schématisé sur la Figure 1b. Nous considérons une entrée fractionnée entre les deux segments (après le premier séparateur [SEP]) . Notons $s_q \in \mathbb{R}^{n_q \times d_e}$ (resp $s_p \in \mathbb{R}^{n_p \times d_e}$) le segment question (resp. paragraphe) en sortie de la couche de plongement (bloc "Embedding layer" sur la Figure), avec n_q la taille de la séquence qui correspond à la question comprenant le token [CLS] et le premier token [SEP], n_p la taille du passage comprenant le dernier token [SEP], et d_e la dimension de l'espace de plongement (512 en général). Notons E_j le j-ième bloc d'encodeur, paramétré par un ensemble de poids θ_j. Nous appliquons les k premiers blocs d'encodeur indépendamment sur les deux segments pour obtenir $s'_q = E_k \circ ... \circ E_1(s_q)$ et $s'_p = E_k \circ ... \circ E_1(s_p)$, que nous concatenons $s' = \text{concat}(s'_q, s'_p)$ avant d'appliquer les $l-k$ derniers blocs d'encodeur pour obtenir $s'' = E_l \circ ... \circ E_{k+1}(s')$ qui passera finalement dans la couche de sortie ("QA Layer"). Nous appellerons les k premiers blocs les couches sans interaction et les $l-k$ derniers blocs les couches d'interaction.

2. Dans la suite, on utilisera soit le terme document, soit le terme paragraphe, soit le terme passage.

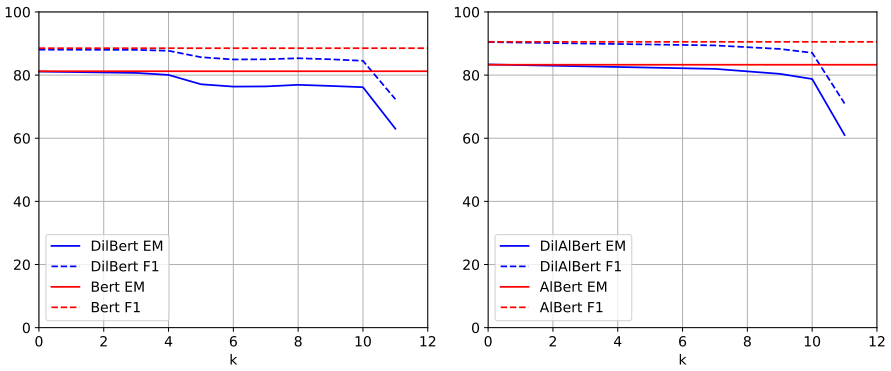
L'impact de l'hyperparamètre k est étudié dans la section expérimentale. Notons que les poids θ_k, ..., θ_1 impliqués dans le calcul de s'_q sont les même que ceux impliqués dans le calcul de s'_p ou encore pour calculer $E_k \circ ... \circ E_1(\text{concat}(s_q, s_p))$ dans le modèle d'origine. On les initialise avec les mêmes valeurs au départ (avec les poids du modèle pré-entraîné) mais on peut soit les faire évoluer de façon indépendante, soit les partager entre la partie question et la partie paragraphe. On choisira la deuxième option car elle n'entraîne pas d'augmentation du nombre de poids par rapport au modèle d'origine. Notre proposition est codée en Python et basée sur les implémentions de la bibliothèque *transformers*, version 2.7.0 (Wolf et al., 2019).

L'interaction retardée réduit la complexité des k premiers blocs. Plus précisément, le mécanisme d'auto-attention dans le modèle d'origine nécessite un nombre de calculs proportionnel à n_s^2 où $n_s = n_q + n_p$ est la longueur de la séquence en entrée. Par conséquent, la proposition réduit la complexité de $O(n_q^2 + n_p^2 + 2n_p \times n_q)$ à $O(n_q^2 + n_p^2)$. Cette réduction peut toutefois paraître négligeable en pratique, parce que le paragraphe est beaucoup plus long que la question, et que les opérations intra-bloc sont parallélisables. Le gain majeur en complexité prend place dans le cadre de l'ODQA. Notons $C \times n_s^2$ la complexité d'un seul bloc d'encodeur. Dans ce cas, l'encodeur complet a une complexité de $l \times C \times n_s^2$ pour une prédiction, et donc $l \times C \times q \times p \times n_s^2$ pour traiter un ensemble de q questions et p paragraphes (pour simplifier, nous supposons ici que les paragraphes sont découpés de sorte que les paires question-paragraphe correspondent à la longueur n_s). Avec l'interaction retardée, les calculs pour chaque question (resp. paragraphe) dans les k premiers blocs n'ont pas besoin d'être reproduits pour chaque paragraphe (resp. question). Par conséquent, la complexité devient $k \times C \times (q \times n_q^2 + p \times n_p^2) + (l - k) \times C \times q \times p \times n_s^2$. Même si p et q sont seulement de l'ordre de 10^2, le terme quadratique avec $p \times q$ domine et la diminution de complexité tend vers le rapport $\frac{l-k}{l}$ entre le nombre de couches d'interaction dans la variante proposée et le nombre total de couches dans le modèle d'origine. Notons que puisque la base de document est statique, on peut ignorer le temps de traitement des paragraphes dans le régime permanent (terme $k \times C \times p \times n_p^2$), qui pourra être effectué lors d'une étape d'initialisation au départ.

4 Étude expérimentale

Nous effectuons ici des expériences sur le challenge de référence pour la tâche de question-réponse en domaine ouvert : OpenSQuAD. Nous commençons par la partie extractive Question Answering associée, basée sur SQuAD v1.1 (Rajpurkar et al., 2016) qui contient 100 000 paires question-paragraphe (environ 10% sont dans l'ensemble de développement). Nous considérons deux modèles comme base. Le premier est Bert car c'est le plus utilisé aujourd'hui et également celui avec le plus grand choix de poids pré-entraînés [3]. Et, pour tester notre framework dans une situation challengeante, le second est Albert car il a la spécificité d'utiliser les mêmes poids dans ses l blocs d'encodeurs (les poids θ_j sont les mêmes pour tout j, $j \in [|1, k|]$ **et** $j \in [|k+1, l|]$) et cela pourrait avoir un impact inattendu sur le mécanisme que nous explorons. Nous appelons notre variante DilBert (resp. DilAlbert) qui signifie "Delaying Interaction Layers in Bert" (resp. Albert). Nous commençons par évaluer l'impact de l'interaction retardée, en fonction de l'hyperparamètre k, pour la sous-tâche d'eQA. Ensuite, nous analysons l'accélération GPU/CPU puis la qualité de réponse dans un cadre d'ODQA.

3. `https://huggingface.co/models`



(a) Bert et Dilbert (b) Albert et DilAlbert

FIG. 2: Evolution de la correspondance exacte (EM) et du score F1 (F1) des variantes Dil sur l'ensemble de développement de SQuAD v1.1 par rapport au nombre de couches sans interaction k. Les lignes horizontales représentent la performance du modèle d'origine.

Qualité de réponse pour l'extractive Question Answering Pour la tâche d'eQA, nous affinons Bert, DilBert, Albert et DilAlbert sur l'ensemble d'entraînement de SQuAD v1.1. Nous considérons la version base de chacun de ses modèles (12 blocs d'encodeurs) et comme point de départ, les poids pré-entraînés anglais : *bert-based-uncased* pour Bert et DilBert et *albert-base-v2* pour Albert et DilAlbert. L'entraînement est effectué avec le script *run_squad.py* de la librairie *transformers* en utilisant les hyperparamètres par défaut : 2 epochs, une longueur de séquence d'entrée n_s de 384, une taille de batch de 12 et un pas d'apprentissage de 3e-5. Les modèles sont évalués sur l'ensemble de développement (dev) de SQuAD v1.1 avec les deux métriques officielles de la compétition (la correspondance exacte EM et le score F1). Pour DilBert et DilAlbert, nous faisons varier l'hyperparamètre k de 0 à $l-1$ et analysons l'impact sur les performances (Figure 2). Nous pouvons d'abord observer que sans interaction retardée ($k=0$), notre implémentation de DilBert (resp. DilAlbert) fournit bien les mêmes résultats que Bert (resp. Albert). Ensuite, à mesure que k augmente, la performance reste très compétitive et ne diminue significativement que pour $k=11=l-1$. Pour $k=l$, il n'y a plus d'interaction, donc les *start logits* et *end logits* associés aux tokens du paragraphe ne dépendent pas de la question et la performance chute alors considérablement (EM ≈ 13, non rapporté sur la figure). A titre d'information, les résultats des variantes Dil avec k couches sans interaction sont nettement supérieurs à une variante du modèle d'origine auquel on aurait complètement supprimé k blocs : par exemple, un Bert dans lequel on ne conserve que 2 couches, entraîné selon le même protocole, obtient une correspondance exacte de seulement 26,4 et un score F1 de 36,2. Notons que l'expérience est réalisée ici sur SQuAD v1.1 car OpenSQuAD en est dérivé. Néanmoins, les variantes proposées s'appliquent également à SQuAD v2.0. Par exemple, Albert, DilAlbert$_{k=6}$ et DilAlbert$_{k=10}$ obtiennent respectivement un F1-Score de 81,4, 80,3 et 72,4 sur ce dernier. Il y a une légère différence entre le comportement de DilBert et DilAlbert. Alors que le second semble avoir une évolution régulière par rapport à k, le premier a deux plateaux pour k dans $[0,4]$ et dans $[5,10]$ et une diminution plus notable entre les deux. En nous inspirant des travaux de Clark et al. (2019), notre hypothèse est que, puisque chaque bloc

		Bert	DilBert$_{k=10}$	DilBert$_{k=11}$	AlBert	DilAlBert$_{k=10}$	DilAlBert$_{k=11}$
GPU	NI Q		1.0 (1e-3)	0.9 (8.2e-4)		1.2 (1.2e-3)	1.4 (1.3e-3)
	NI P		0.8 (8e-4)	1.1 (1e-3)		1.0 (1.0e-3)	1.1 (1.0e-3)
	I Q-P	117.9 (9.8e-4)	17.0 (8.5e-4)	9.9 (9.9e-4)	142.2 (1.2e-3)	22.4 (1.1e-3)	13.1 (1.3e-3)
	Total	117.9	18.8	11.9	142.2	24.6	15.6
	Accélération	x1	x6.3	x9.9	x1	x5.8	x9.1
CPU 10-threads	NI Q		3.5 (3.5e-3)	3.9 (3.6e-3)		4.1 (4.1e-3)	4.8 (4.3e-3)
	NI P		12.0 (1.2e-2)	13.34 (1.2e-2)		16.7 (1.7e-2)	17.6 (1.6e-2)
	I Q-P	2602.6 (2.2e-2)	258.7 (1.3e-2)	126.7 (1.3e-2)	2597.6 (2.2e-2)	346.8 (1.7e-2)	176.3 (1.7e-2)
	Total	2602.6	274.2	144.0	2597.6	367.7	198.7
	Accélération	x1	x9.5	x18.1	x1	x7,1	x13.1

TAB. 1: Temps nécessaire (s) pour rechercher la réponse à 100 questions dans 100 passages avec Bert, Albert, DilBert et DilAlbert. «NI Q» (resp. «NI P») mesure le temps pour traiter les q questions (resp. les p passages) dans les couches sans interaction, et «I QP» pour traiter les $p \times q$ paires dans les couches d'interaction. Nous rapportons entre parenthèses un temps normalisé par rapport au nombre de blocs (l, k ou $l-k$) et d'entrées à traiter (p, q ou $p \times q$).

de Bert a ses propres poids par opposition à Albert, les têtes d'attention pré-entraînées dans la sixième couche pourraient avoir une composante utile pour la tâche d'eQA ciblée.

Accélération des calculs en domaine ouvert Pour analyser l'accélération induite par l'interaction retardée, nous considérons un cadre simple de question-réponse en domaine ouvert où $q = 100$ questions sont posées et le reader (par exemple Bert) recherche les réponses dans une base de $p = 100$ passages sélectionnés. Nous mesurons le temps total nécessaire à Bert, Albert, DilBert et DilAlbert pour traiter toutes les paires question-paragraphe (Tableau 1). Pour les variantes Dil, nous détaillons le temps de traitement indépendant des questions et des paragraphes dans les couches sans interaction et de traitement dépendant des paires question-paragraphe dans les couches d'interaction. Nous considérons deux valeurs de k : 10 car cela permet de préserver une majeure partie de la performance eQA (Figure 2) et $l-1 = 11$, la plus grande valeur possible. Les expériences sont menées sur un serveur bull avec un GPU Nvidia Tesla V100 et un processeur Intel Xeon Gold 6132 (2,6-3,7 GHz) à 5 cœurs / 10 threads.

Les résultats empiriques confirment les intuitions de l'analyse de complexité. Premièrement, comme les calculs intra-blocs sont parallélisables et qu'un GPU a une grande puissance de parallélisation, on peut observer sur GPU que le temps de traitement par bloc et par séquence d'entrée est approximativement toujours le même (environ 1e-3) alors qu'il dépend davantage de la longueur de la séquence sur CPU (environ 4e-3 pour la question d'une longueur de $n_q \approx 16$ et 1,3e-2 pour la concaténation question-paragraphe d'une longueur de $n_s = 384$). Deuxièmement, dans le cadre de l'ODQA, où les réponses à plusieurs questions (q) sont recherchées dans plusieurs paragraphes (p), les calculs sur des couples question-passage prennent la majeure partie du temps car ils sont proportionnels au nombre de paires $p \times q$, largement supérieur à p et q. Étant donné que les modèles d'origine ont $l = 12$ couches d'interaction et que les variantes Dil n'ont que $l-k$ couches d'interaction (1 ou 2), il en résulte que le facteur d'accélération dans l'encodeur qui implémente l'interaction retardée est de l'ordre de $\frac{l}{l-k}$ pour un grand nombre de questions et de passages. On observe donc empiriquement une accélération d'environ un ordre de grandeur pour les variantes Dil.

Qualité des réponses en domaine ouvert Outre l'accélération, nous vérifions ici si l'interaction retardée a un impact sur la qualité des réponses sur le challenge OpenSQuAD (Chen et al., 2017). L'objectif et de répondre aux 10570 questions de l'ensemble de développement de SQuAD v1.1, mais sans connaître le paragraphe associé contenant la réponse. Les algorithmes doivent alors effectuer la recherche dans l'ensemble de 5 075 182 articles du Wikipédia anglais. Puisque notre papier se concentre spécifiquement sur la partie reader d'un pipeline d'ODQA classique (retriever + reader), nous analysons principalement l'impact de celui-ci sur les performances globales. Nous choisirons donc une approche éprouvée de la littérature s'appuyant sur un reader avec une architecture de Transformer, en remplaçant ce dernier par notre variante. La baseline BertSerini (Yang et al., 2019) semble une candidate parfaite pour notre expérience, avec une performance très compétitive et l'emploi de Bert comme reader. Son implémentation n'étant pas disponible publiquement, nous détaillons la nôtre ci-dessous.

Nous appliquons d'abord un prétraitement pour indexer correctement Wikipedia avec Lucene en utilisant la librairie python pyserini, version 0.9.4.0 (Yang et al., 2018). Comme les documents sont parfois longs et couvrent plusieurs sous-sujets, ils sont difficiles à traiter en tant que tels pour le retriever et le reader (Yang et al., 2019; Lee et al., 2018). Nous commençons par les diviser en paragraphes selon deux stratégies : (1) en utilisant le double saut de ligne comme délimiteur. Cela conduit à un nombre de paragraphes final d'environ 29,1 millions comme Yang et al. (2019) et (2) d'une longueur fixe de 100 mots avec fenêtre glissante de 50 mots comme Wang et al. (2019). Nous obtenons des résultats finaux relativement identiques avec les deux techniques mais conservons la deuxième qui se démarque légèrement en résultats absolus. Les paragraphes divisés sont ensuite traités avec le script *index* de pyserini qui construit les index inversés nécessaires au retriever. On instancie ensuite un retriever SimpleSearcher de pyserini qui utilise BM25 avec les index pré-calculés pour renvoyer, pour chaque question, les p paragraphes avec le meilleur score s_{bm25}. On considérera les valeurs $p = 29$ et $p = 100$ comme dans BertSerini. Cette étape de pré-sélection prend moins de 0,15s sur CPU avec la configuration matérielle décrite dans la section précédente. Ensuite, le reader est appliqué à chaque paragraphe et produit un *start logit* s_s et un *end logit* s_e pour chaque token. Pour agréger les résultats et obtenir une prédiction finale, nous étudions deux variantes : une première qui exclut le score du retriever et sélectionne la plage de texte avec le score de reader $s_r = \frac{s_s+s_e}{2}$ le plus élevé parmi tous les paragraphes, et une seconde qui l'inclut en remplaçant s_r par $\mu s_r + (1-\mu)s_{bm25}$. Une validation croisée sur un sous-ensemble de la partie apprentissage de SQuAD v1.1 montre que, pour des valeurs allant 0,1 à 0,9 par pas de 0,1, la valeur $\mu = 0,5$ conduit à une performance finale optimale, pour les readers considérés (Bert, DilBert$_{k=10}$, Albert et DilAlbert$_{k=10}$ affinés sur SQuAD v1.1). Nous ajoutons la version multilingue de Bert (mBert) pour ouvrir des perspectives dans d'autres langues. Nous évaluons le pipeline d'ODQA avec la correspondance exacte (EM) et le score F1 (F1) entre le texte prédit à partir du dump de Wikipedia et la réponse attendue.

Évidemment, la tâche ici est plus difficile que l'eQA puisque le paragraphe contenant la réponse n'est pas fourni. La partie retriever sélectionnera p paragraphes qui pourraient ne pas contenir la réponse et d'autres difficultés relatives à l'évaluation apparaissent (voir section Discussion). Pour donner une idée de l'impact de la partie retriever, nous calculons le pourcentage de questions pour lesquelles la réponse attendue apparaît au moins une fois dans les p paragraphes sélectionnés (colonne R du tableau 2). Comme la méthodologie de pré-découpage de paragraphes et la version de pyserini n'ont pas été détaillées par Yang et al. (2019), nous

avons pu obtenir les mêmes performances de retriever R que Bertserini pour $p = 29$ mais des performances inférieures pour $p = 100$.

Modèle	EM		F1		R
	sans	avec	sans	avec	
DrQA (Chen et al., 2017)	27.1	-	-	-	77.8
Par. R. (Lee et al., 2018)	-	28.5	-	-	83.1
MINIMAL (Min et al., 2018)	-	34.7	-	42.5	64.0
Yang et al. (2019) :					
Bertserini ($p = 29$)	-	36.6	-	44.0	75.0
Bertserini ($p = 100$)	-	38.6	-	46.1	85.8
Nos résultats :					
Bert ($p = 29$)	37.4	43.5	44.1	50.9	74.6
DilBert ($p = 29$)	38.8	42.0	46.7	50.4	74.6
Bert ($p = 100$)	31.8	44.3	38.0	51.8	82.4
DilBert ($p = 100$)	36.5	43.2	43.8	51.6	82.4
mBert ($p = 29$)	34.4	41.9	40.6	49.1	74.6
DilmBert ($p = 29$)	38.5	42.8	45.8	50.5	74.6
mBert ($p = 100$)	28.3	42.3	34.1	49.3	82.4
DilmBert ($p = 100$)	34.7	43.8	41.7	51.6	82.4
Albert ($p = 29$)	40.6	44.1	47.0	51.3	74.6
DilAlbert ($p = 29$)	39.7	43.6	47.7	51.8	74.6
Albert ($p = 100$)	36.2	44.9	42.1	52.1	82.4
DilAlbert ($p = 100$)	36.8	44.8	43.8	52.9	82.4

TAB. 2: Performance de notre pipeline ODQA sur OpenSQuAD. Les résultats sans utilisation du score du retriever (sans) et avec (avec) sont indiqués.

Les résultats finaux présentent des propriétés intéressantes (Tableau 2). Bien que légèrement moins bonnes pour l'eQA (Figure 2), les variantes avec interaction retardée surpassent presque toujours les modèles d'origine pour l'ODQA lorsque le score du retriever n'est pas utilisé, et parfois avec une marge importante (pour un nombre élevé p de paragraphes candidats). Lorsqu'on utilise le score du retriever, les conclusions sont plus mitigées. En fait, les modèles d'origine (Bert, mBert, Albert) ont tendances à produire des prédictions bruitées avec des scores élevés pour des passages peu pertinents (avec un faible score s_{bm25}) (Yang et al., 2019; Xie et al., 2020). L'interaction retardée agit comme une régularisation similaire au dropout, dans le sens où l'architecture résultante est la même que celle d'origine, sauf que certaines connexions sont supprimées dans les k premiers blocs. Par conséquent, les variantes proposées sont plus sujettes à la généralisation. Lorsqu'on utilise le score du retriever en complément des prédictions du reader, tous les modèles sont capables de se concentrer sur les passages les plus pertinents, ce qui contribue à améliorer la performance globale. Dans ce cas, les modèles d'origine parviennent à rattraper les variantes avec interaction retardée. Les résultats avec les poids multilingues sont légèrement moins bons qu'avec les poids anglais, mais surtout pour le modèle d'origine. La variante DilmBert est toujours meilleure que mBert. Quant aux résultats avec Albert, ils sont proches de ceux de Bert mais un peu meilleurs en général. La meilleure performance (EM : 44,3, F1 : 51,8) obtenue dans cette étude avec un Bert base-english-uncased, simplement affinée pour la tâche d'eQA sur SQuAD v1.1 et combiné avec BM25, est nettement meilleure que celle de Bertserini alors que ce dernier utilise les mêmes blocs de base et a un score IR (R) plus élevé. Cette amélioration est principalement due à de l'ingénierie (découpage des documents, version de Lucene), et est cohérente avec des observations faites dans la littérature. Par exemple Xie et al. (2020) rapportent également des résultats plus élevés pour BertSerini (EM : 41.8, F1 : 49.5 , R : 86,3).

5 Discussion

En abordant le problème de question-réponse en domaine ouvert sous l'angle du reader, nous sommes parvenus à réduire fortement les calculs. La complexité des algorithmes est un enjeu industriel et sociétal majeur car elle contrôle l'efficacité des systèmes en production mais aussi leur empreinte carbone. Les variantes proposées, avec $k = 10$ permettent de diminuer le nombre de blocs d'interaction de 12 à 2 (voire 1) dans les modèles Bert et Albert très utilisés

Question	Réponse attendue	Réponse prédite	EM	F1
When did Zwilling and Karistadt become active at Wittenberg ?	June 1521	mid-1521	0	0
The Los Angeles Angels of Anaheim are from which sport ?	MLB	Major League Baseball	0	0
How many fumbles did Von Miller force in Super Bowl 50 ?	2	two	0	0
What is the AFC short for ?	American Football Conference	Asian Football Confederation	0	0.33
How many graduate students does Harvard have ?	14000	15000	0	0
What position did Newton play during Super Bowl 50 ?	quarterback	QB	0	0

TAB. 3: Exemples de questions, réponses attendues et réponses prédites par la pipeline ODQA sur OpenSQuAD. Les réponses sont ici toutes correctes mais l'évaluation ne le reflète pas.

aujourd'hui, réduisant le coût de 85% (voire 92%). Pour donner un point de comparaison par rapport à l'existant, la célèbre variante de Bert connue sous le nom de DistilBert (Sanh et al., 2019) permet de réduire le coût de 50% (essentiellement en passant de 12 couches à 6 couches). Sur SQuAD v1, DistilBert obtient environ les mêmes EM/F1/vitesse que notre approche DilBert avec $k = 6$. Mais DilBert permet d'aller plus loin (EM/F1 pour $k = 10$ aussi bon que pour $k = 6$ mais beaucoup plus rapide). De plus, notre approche est générique aux architectures basées sur le Transformer et permet de bénéficier des poids de modèles déjà disponibles, évitant alors la nécessité d'un pré-entraînement préalable consommateur de ressources. On peut par exemple l'appliquer à Albert. Et DilAlbert avec $k = 10$ obtient non seulement un meilleur EM/F1 que DistilBert, mais a aussi environ 4x moins de paramètres et est environ 3x plus rapide en domaine ouvert. Enfin, rien n'empêcherai d'appliquer le mécanisme Dil à Distilbert. Et rien n'empêcherai aussi d'exploiter le mécanisme dans la partie retriever également. En effet, dans un pipeline ODQA retriever + reader, nous avons considéré que le reader était basé sur un Transformer car c'est le cas pour la grande majorité des approches d'ODQA publiées actuellement. Dans certaines, le retriever (ou un reranker) est également basé sur un transformer (Wang et al., 2019) que nous pourrions également optimiser avec l'interaction retardée.

En plus d'avoir atteint notre objectif initial d'accélération, les variantes proposées se sont finalement avérées compétitives en termes de qualité de réponse bout-en-bout. En particulier, en surpassant les modèles originaux dans la recherche de réponse dans une grande variété de passages, grâce à une meilleure capacité de généralisation. SQuAD v1.1 étant basé sur quelques centaines d'articles Wikipédia, les modèles de langage en résultant peuvent afficher une bonne performance d'eQA en partie due à une surspécialisation sur certains sujets (Xie et al., 2020). La régularisation induite par notre proposition réduit certes les performances pour le passage le plus pertinent (Figure 2) mais également les comportements inattendus sur les passages non pertinents (Table 2). D'autres solutions plus explicites pour améliorer la généralisation et la gestion multi-passages ont été récemment explorées (Wang et al., 2019; Xie et al., 2020). Elles consistent à utiliser la normalisation globale du score ou des stratégies de "distant supervision" avec des passages non pertinents pendant l'entraînement pour rendre le modèle plus robuste. Ces idées intéressantes n'excluent pas la possibilité d'utiliser DilBert à la place de Bert. Notons au passage que DilmBert obtient une performance comparable à celle de Bert, ce qui permettrait donc sans doute de généraliser le pipeline d'ODQA actuel au français sans données supplémentaires (Siblini et al., 2019).

Un dernier point important à discuter est l'état actuel des meilleurs résultats obtenus sur OpenSQuAD. Bien qu'ils semblent faibles (en particulier par rapport à l'eQA), ils s'expliquent par la complexité supplémentaire de la tâche mais également par des biais dans les métriques associées. Outre le goulot d'étranglement du retriever qui entraîne une perte inévitable de 15%

- 18% pour le reader, il y a des questions dans l'ensemble de développement de SQuAD v1.1 pour lesquelles l'évaluation n'est plus adaptée en domaine ouvert (voir Tableau 3).

6 Conclusion

L'interaction retardée permet une accélération d'environ un ordre de grandeur sur le très célèbre Transformer. L'approche est plutôt générique et peut bénéficier de ressources existantes (architectures, poids pré-entraînés). *A posteriori*, on peut établir un lien avec des propositions récentes comme *Big Bird* (Zaheer et al., 2020) et généraliser à l'idée d'un mécanisme d'attention parcimonieux, qui semble se développer actuellement. Notre étude permet de montrer l'efficacité d'une attention partielle avec un design avantageux pour l'ODQA mais il reste de nombreuses pistes à explorer. L'une d'elles consiste à améliorer l'apprentissage en (i) introduisant des couches sans interaction dès la phase de pré-entraînement et (ii) en suivant les idées de Xie et al. (2020); Wang et al. (2019) pour un affinage supplémentaire orienté multi-passages. Une autre direction est celle de la gestion mémoire dans le cadre de l'ODQA. Plus précisément, stocker des représentations pré-calculées de documents peut devenir coûteux pour les bases de données à grande échelle. Il est donc intéressant d'explorer l'impact de leur compression par quantification, par exemple avec un encodage binaire (Tissier et al., 2019).

Références

Chen, D., A. Fisch, J. Weston, et A. Bordes (2017). Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, pp. 1870–1879.

Clark, K., U. Khandelwal, O. Levy, et C. D. Manning (2019). What does bert look at ? an analysis of bert's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP : Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286.

Devlin, J., M.-W. Chang, K. Lee, et K. Toutanova (2019). Bert : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186.

Lan, Z., M. Chen, S. Goodman, K. Gimpel, P. Sharma, et R. Soricut (2019). Albert : A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv :1909.11942*.

Lee, J., S. Yun, H. Kim, M. Ko, et J. Kang (2018). Ranking paragraphs for improving answer recall in open-domain question answering. *arXiv preprint arXiv :1810.00494*.

Lin, J. (2019). The neural hype and comparisons against weak baselines. In *ACM SIGIR Forum*, Volume 52, pp. 40–51. ACM New York, NY, USA.

MacAvaney, S., A. Yates, A. Cohan, et N. Goharian (2019). Cedr : Contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1101–1104.

Manning, C. D., H. Schütze, et P. Raghavan (2008). *Introduction to information retrieval*. Cambridge university press.

Min, S., V. Zhong, R. Socher, et C. Xiong (2018). Efficient and robust question answering from minimal context over documents. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, pp. 1725–1735.

Rajpurkar, P., J. Zhang, K. Lopyrev, et P. Liang (2016). Squad : 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392.

Robertson, S. E., S. Walker, S. Jones, M. M. Hancock-Beaulieu, M. Gatford, et al. (1995). Okapi at trec-3. *Nist Special Publication Sp 109*, 109.

Sanh, V., L. Debut, J. Chaumond, et T. Wolf (2019). Distilbert, a distilled version of bert : smaller, faster, cheaper and lighter. *arXiv preprint arXiv :1910.01108*.

Siblini, W., C. Pasqual, A. Lavielle, et C. Cauchois (2019). Multilingual question answering from formatted text applied to conversational agents. *arXiv preprint arXiv :1910.04659*.

Tissier, J., C. Gravier, et A. Habrard (2019). Near-lossless binarization of word embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 33, pp. 7104–7111.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, et I. Polosukhin (2017). Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008.

Wang, Z., P. Ng, X. Ma, R. Nallapati, et B. Xiang (2019). Multi-passage bert : A globally normalized bert model for open-domain question answering. *arXiv preprint arXiv :1908.08167*.

Wolf, T., L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, et A. M. Rush (2019). Huggingface's transformers : State-of-the-art natural language processing.

Woods, W. A. et W. WA (1977). Lunar rocks in natural english : Explorations in natural language question answering.

Xie, Y., W. Yang, L. Tan, K. Xiong, N. J. Yuan, B. Huai, M. Li, et J. Lin (2020). Distant supervision for multi-stage fine-tuning in retrieval-based question answering. In *Proceedings of The Web Conference 2020*, pp. 2934–2940.

Yang, P., H. Fang, et J. Lin (2018). Anserini : Reproducible ranking baselines using lucene. *Journal of Data and Information Quality (JDIQ) 10*(4), 1–20.

Yang, W., Y. Xie, A. Lin, X. Li, L. Tan, K. Xiong, M. Li, et J. Lin (2019). End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv :1902.01718*.

Zaheer, M., G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. (2020). Big bird : Transformers for longer sequences. *arXiv preprint arXiv :2007.14062*.

Summary

Transformer-based language models such as Bert suffer from a high complexity in open domain question answering. In this paper, we change their architecture to allow a more efficient management of computations. The resulting variants are competitive with original models and allow a significant speedup in both GPU and CPU.