ESAIP

Design4Green - Rapport

Équipe 5

LE POTTIER Mathias, HERBRETEAU Mathis, MARTINAT Julien, PASQUIER Samuel
06/11/2025

Résumé du projet

Créer une application Flask exposant POST /summarize qui renvoie un résumé de 10–15 mots d'un texte donné, en réduisant l'empreinte (énergie, temps, mémoire) sans dégrader la qualité. Notre projet est pensé pour recevoir du texte en anglais, nous n'avons pas ré-entraîner le model pour lui faire comprendre le français. Dans la suite de ce projet nous comparons le modèle Low Rank (LoRA) avec le modèle LoRA optimisé.

Optimisation de notre modèle

Quantization dynamique

L'une des principales optimisations appliquées dans ce projet consiste à effectuer une quantization dynamique en INT8 sur le modèle de base EleutherAI/pythia-70m-deduped. La quantization consiste à convertir les poids du modèle, initialement représentés en précision flottante (FP32), vers un format entier 8 bits (INT8). Grâce à cette optimisation on peut s'apercevoir que le temps de calcul a diminué d'environ 30 à 40 % et la qualité du résumé n'a pas été impactée.

Entraînement et adaptation du modèle

Nous avons choisi de réentraîner notre modèle afin de mieux l'adapter à nos usages spécifiques. Pour cela, nous utilisons le jeu de données XSUM, une base dédiée à l'entraînement de modèles de résumé de textes en anglais.

L'entraînement est réalisé sur 2 epochs, avec un taux d'apprentissage de 1×10^{-4}. Nous appliquons 100 étapes d'échauffement, un journal de progression tous les 50 pas, une évaluation toutes les 500 étapes, et une sauvegarde du modèle tous les 500 pas. Ces paramètres assurent un suivi régulier de la performance tout en maintenant une stabilité de l'apprentissage.

Afin de réduire les coûts computationnels et préserver les capacités générales du modèle, nous avons opté pour une adaptation légère via Low-Rank Adaptation (LoRA). Cette technique consiste à introduire de petites matrices à faible rang au sein des couches d'attention, ce qui permet d'ajuster uniquement un nombre limité de paramètres spécifiques à la tâche, sans réentraîner l'ensemble du modèle.

Dans notre configuration, LoRA est paramétré avec un rang de 16, un facteur d'échelle (α) de 32, et un taux de dropout de 0,05.

Ces choix offrent un équilibre entre performance et efficacité numérique, en réduisant la mémoire nécessaire et la consommation énergétique lors de l'inférence, tout en maintenant la qualité du résumé généré.

Prompt Engineering

Une possibilité que nous avons explorée est le Prompt Engineering, nous avons cherché la possibilité de modifier et d'adapter notre Prompt pour avoir de meilleur résultat et rester dans la limite de 15 mots.

Non utilisé

Enfin, dans les méthodes que nous avons testées, mais qui n'ont pas été concluantes, on peut citer le Pruning, qui consiste à retirer certains paramètres pour pouvoir gagner en efficacité sans perdre en qualité. On s'est aperçu que le bénéfice n'était pas suffisant par rapport à la perte de qualité observée. Enfin, on a cherché à expérimenter avec Torch Compile, mais nous n'avons pas mesuré de réel bénéfice donc nous avons décidé de ne pas l'utiliser.

Mesure de nos performances

Le code utilisé pour mesurer la latence de notre inférence utilise la librairie time nous permettant de mesurer le nombre de millisecondes écoulées entre le début et la fin de notre inférence. Nous avons également utilisé codecarbon pour mesurer la consommation lors de la réalisation d'un résumé, cependant, les inférences étant très courtes, les valeurs retournées ne sont que des estimations. La comparaison entre nos deux modèles en devient donc plus compliquée. C'est pourquoi nous avons également conçu un script permettant de générer une latence moyenne et une énergie dépensée moyenne calculées sur la génération de dix résumés sur dix textes de base différents :

	Latence moyenne (ms)	Energie moyenne (Wh)
Modèle Baseline (F32)	661.70	0.000065
Modèle Optimisé (INT8)	339.10	0.000032

On a pu calculer que nous avons une accélération de 51 % et 50.8% d'économie d'énergie entre nos deux modèles.

Limite de notre modèle

Dans les limites que nous avons pu déterminer pour notre application, le modèle n'est pas adapté, même après un fine tuning. En effet, avant l'entraînement, le modèle peine à trouver le sens du texte et à générer une phrase cohérente. Même si après l'entraînement,

le modèle trouve plus facilement un lien entre le texte et le résumé, les résultats ne sont toujours pas satisfaisants, d'autant plus si le texte d'entrée est grand, dû aux faibles nombre de paramètres. Même s'il produit des phrases plus cohérentes et avec moins de répétitions, le sens et un résumé global du texte peuvent parfois lui échapper.

Pistes d’amélioration

Pour améliorer nos performances, notamment en termes de latence et de consommation de mémoire, une de nos meilleures pistes serait le multithreading. On pourrait alors utiliser plusieurs cœurs de notre CPU pour réduire le temps de calcul et diviser la tâche.

Changer de modèle, pour choisir un modèle plus adapté, en effet le modèle imposé, EleutherAI/pythia-70m-deduped, est un modèle avec très peu de paramètres, peu adapté à des textes de grande taille, et n'est pas adapté à résumer un texte sans entraînement préalable. Afin d'avoir un modèle plus adapté, on peut choisir une des versions de Eleuther/pythia avec plus de paramètres allant de 160 millions à 12 milliards de paramètres permettant plus de capacité. Enfin, on peut aussi changer complètement de modèle afin d'avoir une IA plus adaptée.

Sans changer de modèle, on peut aussi envisager d'appliquer un entraînement plus poussé, en lui donnant plus de données d'entraînement, et en l'entraînant pendant plus longtemps avec plus d'epochs ou en lui donnant un learning rate différent. Cela pourrait nous permettre d'avoir un modèle avec une plus grande précision.