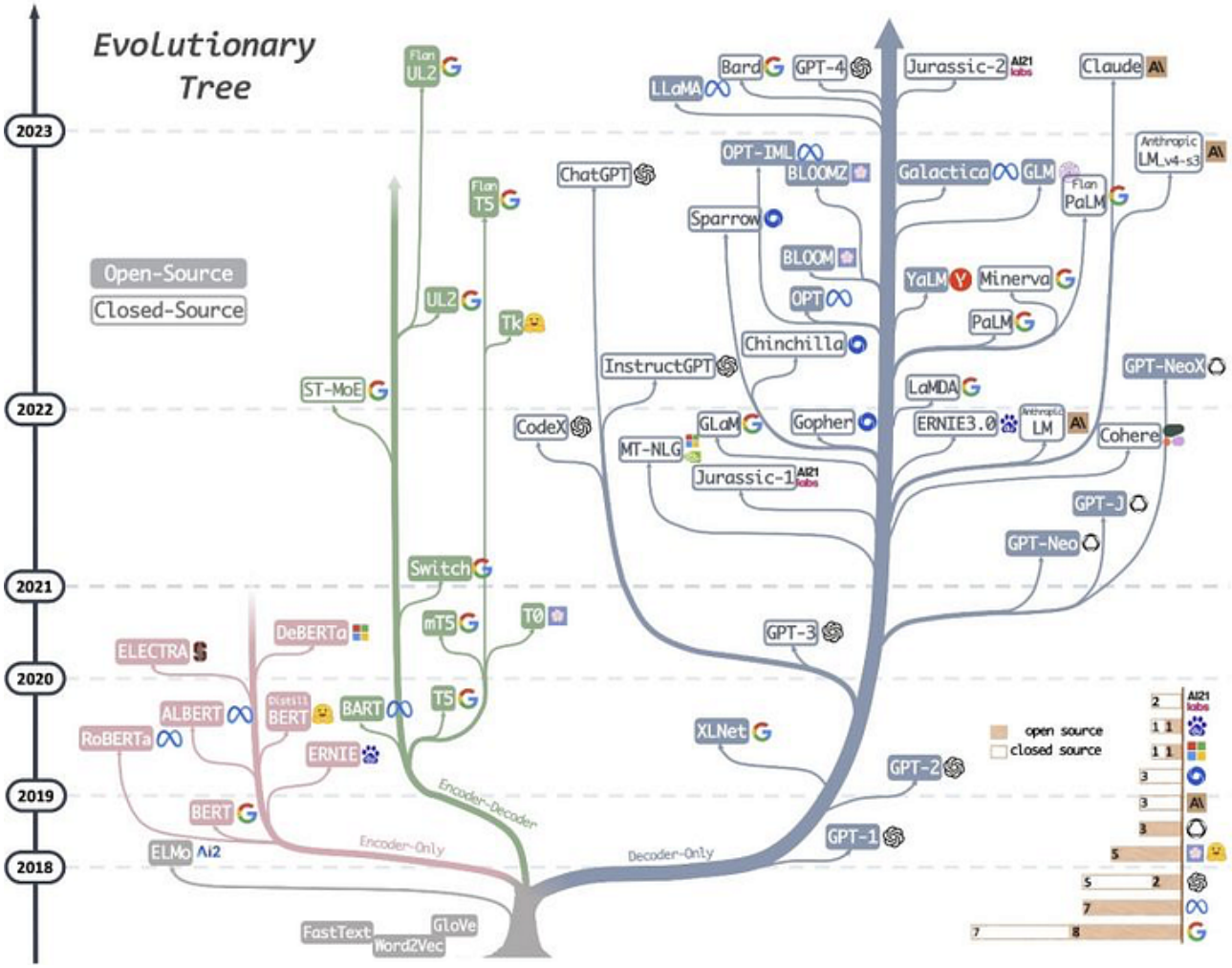
Fonctionnement LLM

L'intelligence artificielle générative (GenAI), notamment ChatGPT, capte l'attention de tous. Les grands modèles de langage (LLM) basés sur des transformateurs, formés à grande échelle sur une grande quantité de données non étiquetées, démontrent la capacité de se généraliser à de nombreuses tâches différentes. Pour comprendre pourquoi les LLM sont si puissants, nous approfondirons leur fonctionnement dans cet article.



Arbre évolutif LLM

Formellement, un modèle de langage de décodeur uniquement est simplement une distribution conditionnelle `p(xi|x1···xi−1)` sur les prochains jetons `xi` dans des contextes donnés `x1 · · · xi−1`. Une telle formulation est un exemple de processus de Markov, qui a été étudié dans de nombreux cas d'utilisation. Cette configuration simple nous permet également de générer jeton par jeton de manière autorégressive.
Avant notre plongée en profondeur, je dois souligner les limites de cette formulation pour atteindre *l'intelligence artificielle générale* (*AGI*). La pensée est un processus non linéaire, mais notre appareil de communication, la bouche, ne peut parler que de manière linéaire. Le langage apparaît donc comme une séquence linéaire de mots. C'est un début raisonnable pour modéliser un langage avec un processus de Markov. Mais je soupçonne que cette formulation peut capturer complètement le processus de réflexion (ou AGI). D'un autre côté, la pensée et le langage sont interdépendants. Un modèle de langage suffisamment fort peut encore démontrer une certaine sorte de capacité de réflexion, comme le montre GPT4. Dans ce qui suit, examinons les innovations scientifiques qui font que les LLM apparaissent *intelligemment* .

Transformer

Il existe de nombreuses façons de modéliser/représenter la distribution conditionnelle `p(xi|x1···xi−1)`. Dans les LLM, nous tentons d'estimer cette distribution conditionnelle avec une architecture de réseau neuronal appelée Transformer. En fait, les

réseaux de neurones, en particulier une variété de réseaux de neurones récurrents (RNN), ont été utilisés dans la modélisation du langage depuis longtemps avant Transformer. Les RNN traitent les jetons de manière séquentielle, en conservant un vecteur d'état qui contient une représentation des données vues avant le jeton actuel. Pour traiter le `n`-ème jeton, le modèle combine l'état représentant la phrase jusqu'au jeton `n-1`avec les informations du nouveau jeton pour créer un nouvel état, représentant la phrase jusqu'au jeton.`n`. Théoriquement, les informations d'un jeton peuvent se propager arbitrairement loin dans la séquence, si à chaque instant l'état continue de coder des informations contextuelles sur le jeton. Malheureusement, le problème du gradient de disparition laisse l'état du modèle à la fin d'une longue phrase sans informations précises et extractibles sur les jetons précédents. La dépendance des calculs de jetons sur les résultats des calculs de jetons précédents rend également difficile la parallélisation des calculs sur le matériel GPU moderne.

Ces problèmes ont été résolus par des mécanismes d'auto-attention dans Transformer . Transformer est une architecture de modèle qui évite la récurrence et s'appuie entièrement sur un mécanisme d'attention pour établir des dépendances globales entre l'entrée et la sortie. La couche d'attention peut accéder à tous les états précédents et les peser selon une mesure de pertinence apprise, fournissant ainsi des informations pertinentes sur les jetons éloignés. Il est important de noter que les Transformers utilisent un mécanisme d'attention sans RNN, traitant tous les jetons simultanément et calculant les pondérations

d'attention entre eux dans des couches successives. Étant donné que le mécanisme d'attention utilise uniquement des informations sur les autres jetons des couches inférieures, il peut être calculé pour tous les jetons en parallèle, ce qui entraîne une vitesse d'entraînement améliorée.

Le texte d'entrée est analysé en jetons par un tokeniseur de paires d'octets, et chaque jeton est converti en un vecteur d'intégration. Ensuite, les informations de position du jeton sont ajoutées à l'intégration. Les blocs de construction du transformateur sont des unités d'attention de produit scalaire à l'échelle. Lorsqu'une phrase est transmise dans un modèle de transformateur, des poids d'attention sont calculés simultanément entre chaque jeton. L'unité d'attention produit des intégrations pour chaque jeton dans le contexte qui contiennent des informations sur le jeton lui-même ainsi qu'une combinaison pondérée d'autres jetons pertinents, chacun pondéré par son poids d'attention.
Pour chaque unité d'attention, le modèle de transformateur apprend trois matrices de poids ; la requête pondère WQ, la clé pondère WK et la valeur pondère WV. Pour chaque jeton i , l'intégration du mot d'entrée est multipliée par chacune des trois matrices de poids pour produire un vecteur de requête qi, un vecteur clé ki et un vecteur de valeur vi. Les poids d'attention sont un produit scalaire entre qi et kj, mis à l'échelle par la racine carrée de la dimension des vecteurs clés et normalisés via softmax. La sortie de l'unité d'attention pour le jeton i est la somme pondérée des vecteurs de valeur de tous les jetons,

pondérée par l'attention du jeton i à chaque jeton j . Le calcul de l'attention pour tous les jetons peut être exprimé sous la forme d'un grand calcul matriciel :
?
Un ensemble de matrices (WQ, WK, WV) est appelé tête d'attention, et chaque couche de transformateur possède plusieurs têtes d'attention. Avec plusieurs têtes d'attention, le modèle peut calculer différentes pertinences entre les jetons. Les calculs pour chaque tête d'attention peuvent être effectués en parallèle et les sorties sont concaténées et projetées vers la même dimension d'entrée par une matrice WO.
Dans un codeur, il existe un perceptron multicouche (MLP) entièrement connecté après le mécanisme d'auto-attention. Le bloc MLP traite en outre chaque codage de sortie individuellement. Dans le cadre codeur-décodeur (par exemple pour la traduction), un mécanisme d'attention supplémentaire est inséré entre l'auto-attention et le MLP dans le décodeur pour extraire les informations pertinentes des codages générés par les codeurs. Dans une architecture avec décodeur uniquement, cela n'est pas nécessaire. Quelle que soit l'architecture du codeur-décodeur ou du décodeur uniquement, le décodeur ne doit pas utiliser la sortie actuelle ou future pour prédire une sortie, la séquence de sortie doit donc être partiellement masquée pour empêcher ce flux d'informations inverse, ce qui permet la génération de texte autorégressive. Pour générer jeton par jeton, le dernier décodeur est suivi d'une couche softmax pour produire les probabilités de sortie sur le vocabulaire.

Mise au point supervisée

Le GPT avec décodeur uniquement est essentiellement un algorithme de pré-entraînement non supervisé (ou auto-supervisé) qui maximise la probabilité suivante :

où k est la *taille de la fenêtre contextuelle* . Bien que l'architecture soit indépendante des tâches, GPT démontre que des gains importants en matière d'inférence en langage naturel, de réponse aux questions, de similarité sémantique et de classification de texte peuvent être réalisés par un pré-*entraînement génératif* d'un modèle de langage sur un corpus diversifié de texte non étiqueté, suivi d'un apprentissage *discriminatif. mise au point* sur chaque tâche spécifique.
Après avoir pré-entraîné le modèle avec l'objectif ci-dessus, nous pouvons adapter les paramètres à la tâche cible supervisée. Étant donné un ensemble de données étiqueté C, où chaque instance consiste en une séquence de jetons d'entrée, x1, . . . , xm, avec une étiquette y. Les entrées passent par le modèle pré-entraîné pour obtenir le hlm d'activation du bloc de transformateur final, qui est ensuite introduit dans une couche de sortie linéaire ajoutée avec les paramètres Wy pour prédire y :

En conséquence, nous avons la fonction objectif suivante :

De plus, il est utile d'inclure la modélisation du langage comme objectif auxiliaire car elle améliore la généralisation du modèle supervisé et accélère la

convergence. Autrement dit, nous optimisons l'objectif suivant :

La classification du texte peut être directement affinée comme décrit ci-dessus. D'autres tâches, comme la réponse à des questions ou l'implication textuelle, ont des entrées structurées telles que des paires de phrases ordonnées ou des triplets de document, de question et de réponses. Étant donné que le modèle pré-entraîné a été formé sur des séquences de texte contiguës, il nécessite quelques modifications pour s'appliquer à ces tâches.

Implication textuelle : concaténer les séquences de jetons de prémisse p et d'hypothèse h, avec un jeton de délimiteur ($) entre les deux.
Similitude : il n'y a pas d'ordre inhérent entre les deux phrases comparées. Par conséquent, la séquence d'entrée contient les deux ordres de phrases possibles (avec un délimiteur entre les deux) et traite chacun indépendamment pour produire deux représentations de séquence, qui sont ajoutées élément par élément avant d'être introduites dans la couche de sortie linéaire.
Réponse aux questions et raisonnement de bon sens :chaque échantillon a un document contextuel z, une question q et un ensemble de réponses possibles {ak}. GPT concatène le contexte du document et la question avec chaque réponse possible, en ajoutant un jeton de délimiteur entre les deux pour obtenir [z;q;$;ak]. Chacune de ces séquences est traitée indépendamment puis normalisée via une

couche softmax pour produire une distribution de sortie sur les réponses possibles.

Transfert Zero-Shot (alias Meta Learning)

Bien que GPT montre que le réglage fin supervisé fonctionne bien sur des ensembles de données spécifiques à une tâche, pour obtenir de bonnes performances sur une tâche souhaitée, il faut généralement affiner un ensemble de données de milliers à des centaines de milliers d'exemples spécifiques à cette tâche. Fait intéressant, GPT2 démontre que les modèles de langage commencent à apprendre plusieurs tâches sans aucune supervision explicite, conditionnées par un document et des questions (appelées invites). Apprendre à effectuer une seule tâche peut être exprimé dans un cadre probabiliste par l'estimation d'une distribution conditionnelle `p(output|input)`. Puisqu'un système général doit être capable d'effectuer de nombreuses tâches différentes, même pour la même entrée, il doit dépendre non seulement de l'entrée mais également de la tâche à effectuer. Autrement dit, il devrait modéliser `p(output|input, task)`. Auparavant, le conditionnement des tâches était souvent mis en œuvre au niveau architectural ou au niveau algorithmique. Mais le langage offre un moyen flexible de spécifier les tâches, les entrées et les sorties sous la forme d'une séquence de symboles. Par exemple, un exemple de formation en traduction peut être écrit sous la forme d'une séquence `(translate to french, english text, french text)`. En particulier, GPT2 est conditionné sur un

contexte de paires d'exemples du format `english sentence = French sentence`puis après une invite finale de`english sentence =`nous échantillonnons à partir du modèle avec un décodage gourmand et utilisons la première phrase générée comme traduction.
De même, pour induire un comportement de résumé, GPT2 ajoute le texte `TL;DR:`après l'article et génère 100 jetons avec un échantillonnage aléatoire Top-k avec k = 2, ce qui réduit la répétition et encourage des résumés plus abstraits qu'un décodage gourmand. De même, un exemple de formation en compréhension écrite peut être écrit sous la forme `(answer the question, document, question, answer)`.
Notez que le transfert sans tir est différent de l'apprentissage sans tir dans la section suivante. Dans le transfert zéro-shot, « zéro-shot » signifie qu'aucune mise à jour du gradient n'est effectuée, mais cela implique souvent de fournir des démonstrations de temps d'inférence au modèle (par exemple, l'exemple de traduction ci-dessus), et il ne s'agit donc pas vraiment d'un apprentissage à partir de zéro exemple. .
Je trouve un lien intéressant entre cette approche de méta-apprentissage et la sémantique de Montague, qui est une théorie de la sémantique du langage naturel et de sa relation avec la syntaxe. En 1970, Montague formulait son point de vue :
Il n'y a à mon avis aucune différence théorique importante entre les langues naturelles et les langues artificielles des logiciens ; en effet, je considère qu'il est possible de comprendre la syntaxe et la sémantique des deux types de langages

avec une seule théorie naturelle et mathématiquement précise.
Philosophiquement, le transfert zéro-shot et la sémantique de Montague traitent le langage naturel de la même manière que *le langage de programmation* . Les LLM capturent la tâche via les vecteurs d'intégration dans une approche de boîte noire. Cependant, nous ne savons pas exactement comment cela fonctionne réellement. En revanche, les caractéristiques les plus importantes de la sémantique de Montague sont son adhésion au principe de compositionnalité — c'est-à-dire que la signification du tout est fonction des significations de ses parties et de leur mode de combinaison syntaxique. Cela peut être une approche pour améliorer les LLM.

Apprentissage en contexte

GPT3 montre que la mise à l'échelle des modèles de langage améliore considérablement les performances en quelques tâches, indépendamment des tâches. GPT3 spécialise davantage la description en « zero-shot », « one-shot » ou « quelques-shots » en fonction du nombre de démonstrations fournies au moment de l'inférence : (a) « apprentissage en quelques coups » ou apprentissage en contexte où nous autorisons autant de démonstrations que le permet la fenêtre contextuelle du modèle (généralement 10 à 100), (b) « apprentissage ponctuel », où nous autorisons une seule démonstration, et © apprentissage « zéro-shot », où aucune démonstration sont autorisés et seule une instruction en langage naturel est donnée au modèle.

Pour un apprentissage en quelques étapes, GPT3 évalue chaque exemple de l'ensemble d'évaluation en tirant au hasard K exemples de l'ensemble d'entraînement de cette tâche comme conditionnement, délimités par 1 ou 2 nouvelles lignes selon la tâche. K peut être n'importe quelle valeur comprise entre 0 et la valeur maximale autorisée par la fenêtre contextuelle du modèle, qui est nctx = 2048 pour tous les modèles et correspond généralement à 10 à 100 exemples. Des valeurs plus élevées de K sont généralement meilleures, mais pas toujours.
Pour certaines tâches, GPT3 utilise également une invite en langage naturel en plus (ou pour K = 0, à la place) des démonstrations. Pour les tâches qui impliquent de choisir un achèvement correct parmi plusieurs options (choix multiples), l'invite comprend K exemples de contexte plus un achèvement correct, suivis d'un exemple de contexte uniquement, et le processus d'évaluation compare la probabilité modèle de chaque achèvement.
Sur les tâches qui impliquent une classification binaire, GPT3 donne aux options des noms plus sémantiquement significatifs (par exemple « Vrai » ou « Faux » plutôt que 0 ou 1), puis traite la tâche comme un choix multiple.
Sur les tâches avec complétion de forme libre, GPT3 utilise la recherche de faisceaux. Le processus d'évaluation note le modèle en utilisant le score de similarité F1, BLEU ou la correspondance exacte, en fonction de ce qui est standard pour l'ensemble de données concerné.

La taille du modèle compte (jusqu'à présent)

La capacité du modèle de langage est essentielle au succès de l'apprentissage indépendant des tâches et son augmentation améliore les performances de manière log-linéaire entre les tâches. GPT-2 a été créé comme une mise à l'échelle directe de GPT-1, avec son nombre de paramètres et la taille de son ensemble de données augmentés d'un facteur 10. Mais il peut effectuer des tâches en aval dans un paramètre de transfert zéro - sans aucun paramètre ni architecture. modification.

GPT3 utilise le même modèle et la même architecture que GPT2, à l'exception de l'utilisation d'une alternance de modèles d'attention denses et dispersés localement dans les couches du transformateur.

Taille du modèle

Sur TriviaQA, les performances de GPT3 augmentent progressivement avec la taille du modèle, ce qui suggère que les modèles linguistiques continuent d'absorber des connaissances à mesure que leur capacité augmente. Les performances en un seul coup et en quelques coups apportent des gains significatifs par rapport au comportement sans coup.

La qualité des données est importante

Bien que moins discutée, la qualité des données compte également. Les ensembles de données pour les modèles linguistiques se sont rapidement développés. Par exemple, l'ensemble de données CommonCrawl constitue près d'un billion de mots, ce

qui est suffisant pour former des modèles plus volumineux sans jamais mettre à jour deux fois la même séquence. Cependant, il a été constaté que les versions non filtrées ou légèrement filtrées de CommonCrawl ont tendance à avoir une qualité inférieure à celle des ensembles de données plus organisés.
Par conséquent, GPT2 a créé un nouveau web scrape qui met l'accent sur la qualité du document en supprimant tous les liens sortants de Reddit qui ont reçu au moins 3 karma, qui agit comme un indicateur heuristique pour savoir si les autres utilisateurs ont trouvé le lien intéressant, éducatif ou simplement amusant. L'ensemble de données final contient un peu plus de 8 millions de documents pour un total de 40 Go de texte après déduplication et nettoyage heuristique.
De plus, GPT3 a suivi 3 étapes pour améliorer la qualité moyenne des ensembles de données : (1) CommonCrawl filtré en fonction de la similarité avec une gamme de corpus de référence de haute qualité, (2) déduplication floue au niveau du document, au sein et entre les ensembles de données, pour éviter la redondance. et préserver l'intégrité de l'ensemble de validation retenu en tant que mesure précise du surajustement, et (3) ajouté des corpus de référence connus de haute qualité au mélange de formation pour augmenter CommonCrawl et accroître sa diversité.
De même, GLaM développe un classificateur de qualité de texte pour produire un corpus Web de haute qualité à partir d'un corpus brut original plus grand. Ce classificateur est formé pour classer entre

une collection de textes organisés (Wikipédia, livres et quelques sites Web sélectionnés) et d'autres pages Web. GLaM utilise ce classificateur pour estimer la qualité du contenu d'une page Web, puis utilise une distribution Pareto pour échantillonner les pages Web en fonction de leur score. Cela permet d'inclure certaines pages Web de moindre qualité pour éviter les biais systématiques dans le classificateur.

Poids des données et des mélanges dans l'ensemble d'entraînement GLaM

GLaM définit également les poids de mélange en fonction des performances de chaque composant de données dans un modèle plus petit et pour empêcher les petites sources telles que Wikipédia d'être suréchantillonnées.

Chaîne de pensée

Comme indiqué précédemment, la prédiction du prochain jeton n'est pas la même que le processus de réflexion. Il est intéressant de noter que certaines capacités de raisonnement et d'arithmétique des LLM peuvent être débloquées par l'incitation de la chaîne de pensée . Une *chaîne de pensée* est une série d'étapes intermédiaires de raisonnement en langage naturel qui mènent au résultat final. Des modèles de langage suffisamment grands peuvent générer des chaînes de pensée si des démonstrations de raisonnement en chaîne de pensée sont fournies dans les exemples pour des invites en quelques étapes : ⟨entrée, *chaîne de pensée* , sortie⟩. Cependant, pourquoi et comment cela fonctionne ne nous est pas clair.

Apprentissage par renforcement à partir de la rétroaction humaine (RLHF)

L'objectif de modélisation du langage utilisé pour les LLM — prédire le prochain jeton — est différent de l'objectif « suivre les instructions de l'utilisateur de manière utile et sûre ». Ainsi, nous disons que l'objectif de la modélisation du langage est *mal aligné* .

InstructGPT aligne les modèles de langage avec l'intention de l'utilisateur sur un large éventail de tâches en utilisant *l'apprentissage par renforcement à partir de la rétroaction humaine* (RLHF). Cette technique utilise les préférences humaines comme signal de récompense pour affiner les modèles.

Étape 1 : Collectez des données de démonstration et formez une politique supervisée. Les étiqueteurs fournissent des démonstrations du comportement souhaité sur la distribution des invites de saisie. Affinez ensuite un modèle GPT3 pré-entraîné sur ces données à l'aide de l'apprentissage supervisé.

Étape 2 : Collectez des données de comparaison et entraînez un modèle de récompense. Collectez un ensemble de données de comparaisons entre les sorties du modèle, dans lequel les étiqueteurs indiquent quelle sortie ils préfèrent pour une entrée donnée. Entraînez ensuite un modèle de récompense pour prédire le résultat préféré par l'homme.

Étape 3 : Optimisez une politique par rapport au modèle de récompense à l'aide de PPO. Utilisez la sortie du RM comme récompense

scalaire. Affinez la politique supervisée pour optimiser cette récompense grâce à l' algorithme PPO .
Les étapes 2 et 3 peuvent être répétées en continu ; davantage de données de comparaison sont collectées sur la meilleure politique actuelle, qui sont utilisées pour former un nouveau RM, puis une nouvelle politique.

Mise au point des instructions

Alors que le réglage fin supervisé introduit dans GPT-1 se concentre sur le réglage spécifique à une tâche, T5 est entraîné avec un objectif de maximum de vraisemblance (en utilisant le « forçage de l'enseignant »), quelle que soit la tâche. Essentiellement, T5 exploite la même intuition que le transfert zéro-shot selon laquelle les tâches de PNL peuvent être décrites via des instructions en langage naturel, telles que « *Le sentiment de cette critique de film est-il positif ou négatif ?* » ou « *Traduisez « comment allez-vous » en chinois*. " Pour spécifier quelle tâche le modèle doit effectuer, T5 ajoute un préfixe (texte) spécifique à la tâche à la séquence d'entrée d'origine avant de la transmettre au modèle. De plus, FLANexplore le réglage fin des instructions avec un accent particulier sur (1) la mise à l'échelle du nombre de tâches, (2) la mise à l'échelle de la taille du modèle et (3) le réglage fin des données de chaîne de pensée.
Pour chaque ensemble de données, FLAN compose manuellement dix modèles uniques qui utilisent des instructions en langage naturel pour décrire la tâche pour cet ensemble de données. Alors que la plupart des dix modèles décrivent la tâche d'origine, pour

accroître la diversité, pour chaque ensemble de données, FLAN comprend également jusqu'à trois modèles qui « renversent la tâche » (par exemple, pour la classification des sentiments, nous incluons des modèles demandant de générer une critique de film). Nous ajustons ensuite un modèle de langage pré-entraîné sur le mélange de tous les ensembles de données, avec des exemples dans chaque ensemble de données formatés via un modèle d'instructions sélectionné au hasard pour cet ensemble de données.

Ce que l'on appelle l'ingénierie rapide est essentiellement une ingénierie inverse de la manière dont les données de formation sont préparées pour le réglage fin de l'instruction et l'apprentissage en contexte.

Génération augmentée de récupération (RAG)

En raison du coût et du temps, les LLM dans les utilisations en production sont souvent en retard en termes de fraîcheur des données de formation. Pour résoudre ce problème, nous pouvons utiliser les LLM à la manière de la génération augmentée de récupération (RAG). Dans ce cas d'utilisation, nous ne voulons pas que le LLM génère du texte basé uniquement sur les données sur lesquelles il a été formé, mais nous souhaitons plutôt qu'il incorpore d'une manière ou d'une autre d'autres données externes. Avec RAG, les LLM peuvent également répondre à des questions spécifiques à un domaine (privé). Par conséquent, RAG est également appelé réponse aux questions « à livre ouvert ». LLM + RAG

pourrait être une alternative au moteur de recherche classique. En d’autres termes, il s’agit d’une recherche d’informations avec hallucination.
Actuellement, la partie récupération de RAG est souvent implémentée sous la forme d'une recherche du k-voisin le plus proche (similarité) sur une base de données vectorielle qui contient l'intégration vectorielle de données textuelles externes. Par exemple, DPR formule la formation des encodeurs comme un problème d’apprentissage de métriques. Cependant, il faut remarquer que la recherche d’informations est généralement basée sur la pertinence, ce qui est différent de la similarité. Je m'attends à ce qu'il y ait encore beaucoup d'améliorations dans ce domaine à l’avenir.

Conclusion

Le LLM est un domaine passionnant et connaîtra des innovations rapides. J'espère que cet article vous aidera à comprendre un peu comment cela fonctionne. Outre l'enthousiasme, nous devons également remarquer que les LLM apprennent la langue d'une manière très différente de celle des humains : ils n'ont pas accès au contexte social et perceptuel que les apprenants de langues humaines utilisent pour déduire la relation entre les énoncés et les états mentaux des locuteurs. Ils sont également formés d’une manière différente du processus de réflexion humain. Il pourrait s’agir de domaines dans lesquels améliorer les LLM ou inventer de nouveaux paradigmes d’algorithmes d’apprentissage.