Rapport Stage L3

Machine-learning models to study how children develop in their ability to take turns in conversations

Martin Cuingnet et Abdellah Fourtassi, équipe TALEP du LIS

August 22, 2024

Voici mon rapport de stage de L3 réalisé du 3 juin 2024 au 26 juillet 2024 dans l'équipe TALEP au LIS à Luminy, encadré par Abdellah Fourtassi.

Ce travail porte sur la compréhension et la modélisation des tours de parole, une modélisation de la structure conversationelle, servant de cadre théorique à ce rapport. Nous nous intéressons en particulier à la perception de la conversation et à la prise de parole chez l'enfant.

Ce travail détaille l'application et la modification du modèle des transformers sur les données conversationnelles des datasets ChiCo et ChiCa pour prédire les changements de tours de parole, en comparant les performances entre adultes et enfants. Cette analyse utilise la modalité textuelle (les paroles échangées dans la conversation) à laquelle a été greffé la modalité visuelle (celle du regard dans la discussion)

L'architecture des transformers a par la suite été modifiée pour y inclure la modalité du regard, situant cette étude à l'intersection du machine learning, de l'interprétabilité et de la psychologie. Led modèle développés restent néanmoins limités par la taille réduite des datasets utilisés et le manque de résultats d'interprétabilité.

Malgré ces limitations, le modèle a révélé des différences entre adultes et enfants, et soulignant l'importance du regard dans la communication. Les perspectives d'amélioration incluent l'augmentation de la taille des datasets et l'amélioration de l'interprétabilité du modèle.

Ce rapport contient **13** pages effectives.

Table des matières

1. Contexte

Nous allons dans ce travail utiliser la modélisation d'une conversation décrite dans [1]. Ici, une *conversation* est décomposée en différents *tours de paroles*. Informellement, un tour de parole constitue une période de temps pendant laquelle un interlocuteur prend la parole. Même si d'autres personnes peuvent intervenir pendant un tour de paroles (sous la forme d'onomatopées, de courtes phrases ou de mouvements de têtes par exemple), la conversation est menée par l'interlocuteur responsable du tour.

En utilisant cette modélisation, chaque interaction de la part d'un intervenant peut être rangée dans deux catégories (deux *canaux*).

- Le **Main Channel** : L'intervention est effectuée comme élément principal d'un tour de parole : c'est-à-dire réalisé par l'interlocuteur responsable du tour en cours
- Le **Back Channel** : L'intervention est mineure, verbale ou non-verbale et effectuée pendant le tour de parole d'un autre intervenant.

Voici un exemple illustré de tours de paroles ainsi que des Main Channel et des Back Channel associés :

Alice : Salut !

Bob : Salut !

Alice : Tu ne sais pas la dernière ?

Bob : Non ?

Alice : Aujourd'hui le facteur n'est pas passé !

Bob : Non !

Alice : On m'a même raconté qu'il ne passera jamais ! Quelle histoire !

Bob : [Hochement de tête]

Alice : Et moi qui attendais du courrier... [Lève les yeux au ciel]

L'alternance des couleurs représente la succession des tours de paroles avec en **gras le Main Channel** et en *italique le Back Channel*.

Fig. 1. – Succession de tours de paroles

Comme on peut le voir dans Fig. 1, les interventions de Bob à la fin de la discussion peuvent-être catégorisées comme Back Channel du fait qu'elles soient mineures et très courtes.

Mais cette structure conversationnelle sous forme de tours, très répandue dans la société occidentale, n'est pas totalement maîtrisée par des jeunes enfants, comme montré par [2]. On va donc ici s'intéresser à l'acquisition de ces compétences conversationnelles ainsi qu'à l'état actuel de ces capacités.

Une telle étude a déjà été réalisée dans le passé. L'étude [3], menée par mon maître de stage et qui a servi de socle de travail durant le stage, exprime quantitativement des différences dans les capacités conversationnelles entre l'adulte et l'enfant. Cette étude s'attelle, en utilisant un modèle de Machine Learning, à la prédiction de channel pour un interlocuteur dans une conversation à deux intervenant (soit adulte/adulte, soit adulte/enfant).

Le modèle entraîné peut alors être utilisé pour prédire les changement de tours entre enfant et adulte, et le score d'un tel modèle peut alors quantifier la propension de l'enfant à lui-même prédire les tours de paroles.

Pour cela, le modèle utilisé dans [3] est un LSTM (Long-Term Short-Term Memory), un modèle séquentiel utilisé pour reconnaître, à partir d'un contexte de deux secondes de conversation, le canal d'un interlocuteur. Pour cela, le modèle prend en paramètre plusieurs modalités (la hauteur de la voix, le type de mots prononcés, la direction du regard,...) qui sont alors concaténées dans la couche d'entrée.

2. Compréhension du modèle LSTM

Dans le cadre de [3], les données utilisées (pour l'entraînement et le test) sont les données du dataset ChiCo (détaillé dans [4]), un dataset contenant des conversations Zoom adulte/adulte et adulte/enfant dont les features principales (hauteur de la voix, regard, transcription textuelle,...) ont été extraites au prélables.

Le modèle peut alors prendre en entrée la totalité ou une partie des features extraites et réaliser des prédictions à partir de celles-ci.

Comme indiqué précédemment, le modèle LSTM est un modèle séquentiel (il va appliquer la même opération sur chaque élément de l'entrée séquentielle en gardant en mémoire les résultats des opérations précédentes jusqu'à obtenir le résultat du modèle). Le modèle LSTM va donc prendre en entrée les différentes features concaténées arrangées séquentiellement et va chercher à prédire le channel du premier intervenant (0 pour le Back Channel et 1 pour le Main Channel).

L'implémentation ainsi que l'entraînement du modèle a été réalisée en Python en utilisant le module `pytorch`, module que j'ai utilisé pendant toute la durée du stage.

Une explication en profondeur de l'architecture LSTM se trouve au Chapitre 10.1.

3. Une nouvelle approche : les transformers

Après avoir repris et remanié légèrement le code utilisé dans [3] (voir https://github.com/abhishek-agrawal94/BC-MC-Prediction°), j'ai réussi à reproduire les résultats du papier.

Cependant, j'ai repéré quelques point qui m'ont fait me questionner sur l'optimalité de la démarche utilisée pour obtenir ces résultats.

Tout d'abord, le modèle ne semble pas utiliser la modalité verbale à son plein potentiel : dans [3], au lieu d'utiliser le transcript de la conversation, le LSTM ne s'intéresse qu'à la classe grammaticale des mots. Si dans ChiCo il y a la phrase « je pense à un animal », le LSTM lui verra « pronom verbe conjonction déterminant nom ». Or, il me semble raisonnable de supposer que la sémantique exacte des phrases à un impact non négligeable sur la structure des tours.

De plus, le LSTM est un modèle qui commence à être dépassé dans le domaine du Natural Language Processing, surtout concernant l'aspect verbal (aucun des meilleurs modèles actuels pour faire de la prédiction de texte ne sont des LSTM).

Je vais donc tenter d'appliquer un autre type de modèle à ce problème, quitte à me concentrer sur la modalité verbale dans un premier temps : les transformers.

4. Comprendre et implémenter les transformers

La première version des modèles de transformers tels qu'on les connaît aujourd'hui (c'est à dire basés uniquement sur le mécanisme d'attention) a été introduit dans [5].

Je vais ici détailler le fonctionnement d'un transformer similaire, mais se basant sur la self-attention et non la cross-attention (le transformer du papier original était conçu pour la traduction machine et non la prédiction de texte) :

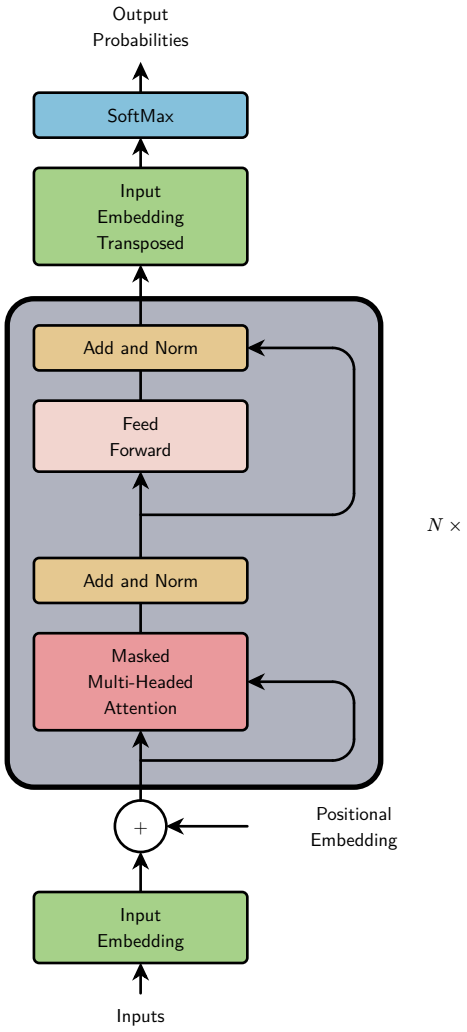


Fig. 2. – Architecture générale d'un transformer auto-attentif [5]

L'objectif du modèle est donc, en parant d'une chaîne de caractère de prédire le « mot » (en réalité le token) suivant. Pour ce faire, chaque mot (token) du texte va être représenté par un vecteur et chacun de ces vecteurs vont communiquer entre eux pour faire évoluer la sémantique respective de chaque mot au sein de la phrase, jusqu'à ce que le vecteur associé au dernier mot du texte encapsule la sémantique du mot à prédire, permettant de déterminer ce mot.

4.1. Tokenization

Comme ce que j'ai laissé suggéré précédemment, en interne, un transformer n'utilise pas des mots mais des ***tokens***. Un token représente souvent un sous-mot et à chaque token est associé un entier.

Voici un exemple de phrase ainsi que le codage associé :

Hello this string was tokenized !!
[15496, 428, 4731, 373, 11241, 1143, 37867, 10185]

Ce codage est obtenu en utilisant le tokenizer de GPT-2, que j'ai utilisé depuis la bibliothèque `transformers` [6]. Cet tokenizer étant une application de l'algorithme BPE (*Byte-Pair Encoding*) détaillé dans [7], et appliqué sur le corpus utilisé pour l'entraînement de GPT-2. Le code associé à chaque token sera alors utilisé pour obtenir le vecteur le représentant dans l'embedding (voir Chapitre 4.2)

4.2. Plongement lexical (embedding)

On va maintenant chercher à associer un vecteur à chaque token capturant sa sémantique. Ce vecteur (vivant dans un espace à 12 288 dimensions), va être déterminé pendant la phase d'entraînement, comme tout les autres poids. C'est l'étape de *plongement lexical*.

Ces vecteurs encapsulent la sémantique des tokens en hors de leur contexte (celui-ci va être pris en compte dans la phase d'attention) : un exemple de cette sémantique capturée est le calcul vectoriel entre tokens. Par exemple, on a la relation :

$$\overrightarrow{\text{woman}} - \overrightarrow{\text{man}} + \overrightarrow{\text{king}} \simeq \overrightarrow{\text{queen}}$$

4.3. Attention

Après avoir converti chaque token en vecteur de sémantique, on va ensuite affiner leur signification afin d'encapsuler le contexte les précédant. Pour cela, on va transformer ces vecteurs avec le mécanisme d'attention, qui va affiner à chaque passe la sémantique réelle de chaque mot une fois le contexte pris en compte. Ainsi, à la fin de plusieurs passes au travers du mécanisme d'attention (avec à chaque passe des matrices de poids différentes), le vecteur associé au dernier token de la phrase encapsulera la sémantique du token à prédire.

Une explication détaillée de ce processus est disponible au Chapitre 10.2.

4.4. Probabilités

Finalement, après que les vecteurs soient passés par tous les blocs, on va produire, à partir du vecteur associé au dernier token de la séquence, une distribution de probabilité sur l'ensemble des tokens possibles. Pour cela, on va multiplier une matrice de poids par ce vecteur, la matrice d'unembedding, ce qui donnera un vecteur ayant pour dimension le nombre de tokens possible.

Enfin, il suffit d'appliquer à nouveau l'algorithme soft-max (voir [8] et Chapitre 10.2) pour obtenir la distribution de probabilités sur l'ensemble des tokens, ce qui permet d'alors réaliser une prédiction en choisissant le token avec la plus haute probabilité.

5. Entraîner un transformer

En pratique, la manipulation et l'entraînement de tels modèle se fait avec le module `pytorch`, une bibliothéque optimisée pour la manipulations de tenseurs, et en particulier la bibliothèque `transformers` [6], qui permet de manipuler plus facilement ces modèles.

Pour pouvoir utiliser un transformer dans le cadre d'une application de prédiction de tours de paroles, une première idée serait de directement entraîner un transformer vierge sur nos données d'entraînement. Or, il y a plusieurs problèmes à cela :

Premièrement, les données d'entraînement sont insuffisante pour un entraînement *ex nihilo* (le transcript du corpus ChiCo [4] est de l'ordre du Ko) contre les centaines, voir milliers, de gigas

nécessaires à un entraînement complet.
De plus, la puissance de calcul disponible n'est pas suffisante (plus de 3 millions de dollars ont été nécessaires pour entraîner GPT-3)
On peut également souligner l'impact écologique d'un entraînement d'une telle ampleur.

L'idée est donc de partir d'un base déjà existante, des modèles *pré-entraînés* comme GPT-2, qu'on va ensuite légèrement modifier pour les adapter à nos données : on va les *finetuner*.

Je vais donc réutiliser les même mécanismes qui ont été utilisés pour entraîner ces modèles, c'est-à-dire la minimisation classique d'un coût par descente de gradient et l'utilisaion de la back propagation, mais sur les données plus restreintes et personnalisées du dataset ChiCo.

6. Application des transformers aux problèmes des tours de paroles

Après avoir expliqué le fonctionnement et le mécanisme d'entraînement d'un transformer, on cherche maintenant à utiliser cette architecture dans le cas particulier des Large Language Model (LLM) pour prédire les tours de paroles.

Or, contrairement au LSTM qui prédisait directement le channel (MC ou BC) du locuteur, un LLM, lui ne peut que prédire le token suivant une certaine liste de token et non faire directement de la classification : il faut donc adapter la formulation du problème.

6.1. Formalisation adaptée à la prédiction de texte

En m'inspirant de [9], je vais donc formaliser le problème en introduisant un token spécial dans le modèle et le dataset : le token `<|turnshift|>`. Ce token va signifier le changement de tour de parole dans la conversation.

Ainsi, en modifiant le transcript original de [4], on obtient, en reprenant l'exemple de Fig. 1, un texte sous cette forme :

```
salut
<|turnshift|>
salut
<|turnshift|>
tu ne sais pas la dernière
<|turnshift|>
non
<|turnshift|>
aujourd'hui le facteur n'est pas passé on m'a même raconté qu'il ne passera jamais
quelle histoire et moi qui attendais du courier [lève les yeux au ciel]
```

(On remarquera la suppression des majuscules et de la ponctuation, afin de faire la prédiction sur des éléments purement vocaux, afin de pouvoir appliquer le modèle sur un transcript généré de manière automatique)

Avec ce formalisme, prédire le channel (c'est à dire prédire le changement de tour), revient à prédire l'apparition du token `<|turnshift|>`. Ainsi, pour chaque suite de tokens, on obtient la probabilité de changement de tour en calculant la probabilité associé au token `<|turnshift|>`.

Ainsi, pour évaluer le modèle, on ramène cette probabilité d'apparition à un choix binaire décidé par un threshold arbitraire : si $\mathbb{P}($ `<|turnshift|>` $) \geq \text{threshold}$ alors on prédit la fin du tour.

6.2. Méthode d'évaluation du modèle

Pour tester les performances du modèle, on va, à partir de données de tests, donner au modèle un grand nombre de phrases d'un certain nombre de tokens et lui demander si le prochain token est un `<|turnshift|>` ou non et comparer cette prédiction avec le token suivant réel.

Le dataset original est composé de 18 conversations (9 Adulte/Adulte et 9 Enfant/Adulte) et le jeu de test est composé de deux de ces conversation. Le modèle est ainsi évalué par validation croisée à 2 blocs (leave-2-out cross validation) : on entraîne plusieurs modèle avec une répartition train dataset/test dataset différente.

Ma première approche a été, à partir du jeu de test, de tester toute les phrases possibles (en reprenant l'exemple précédent, à partir de la phrase `the sky is blue`, on obtient les tests `the`, `the sky` et `the sky is`).

Or le problème de cette approche vient du fait que dans la grande majorité des cas, il faut prédire un token différent d'un `<|turnshift|>` : ainsi, un modèle ne prédisant jamais de `<|turnshift|>` obtiendra un très bon score (avec cette approche, un modèle ne prédisant jamais de `<|turnshift|>` obtenais un score de 90%)

Pour corriger ce biais, je teste donc le modèle sur un nombre égal de cas où il faut prédire `<|turnshift|>` (vrai positifs) et de cas où il ne faut pas le prédire (vrais négatifs). Le score final est donc $s = \frac{\mathrm{VP} + \mathrm{VN}}{2}$ avec VP le taux de vrais positifs et VN le taux de vrais négatifs.

Ainsi, un modèle ne prédisant jamais de `<|turnshift|>` comme un modèle prédisant un `<|turnshift|>` à coup sûr obtiendra un taux de réussite de 50%.

6.3. ChiCo

6.3.1. Expériences

J'ai réalisé plusieurs expériences à partir du dataset ChiCo avec différents modèles et des variations méthodologiques.

6.3.1.1. GPT2-large

Comme expliqué au Chapitre 5, on va, pour créer le modèle de prédiction de `<|turnshift|>`, partir d'un modèle pré-entraîné, ici gpt2-large.

Pour respecter le formalisme décrit au Chapitre 6.1 et pouvoir évaluer le modèle à la manière du Chapitre 6.2,j'ai remis en forme le dataset (de la manière présentée au Chapitre 6.1) et j'ai modifié l'architecture du transformer.
J'ai d'abord rajouté le token `<|turnshift|>` dans le vocabulaire du modèle (j'ai rajouté une colonne dans la matrice d'embedding). Ainsi, pendant l'étape de finetuning, le modèle va également apprendre la sémantique du `<|turnshift|>`.

Le finetuning fait usage du code présent au Chapitre 10.5.1 et utilise le modèle gpt2-large préentrainé disponible dans la bibliothèque `transformers` de [6].

6.3.1.2. GPT2-large sur le corpus en anglais

J'ai conservé la même architecture qu'auparavant mais cette fois-ci j'ai traduit le dataset ChiCo en anglais en utilisant la librairie python `GoogleTranslate`.

6.3.1.3. Claire-7B

Ce modèle, presque 10 fois plus gros que GPT2-large, est une version finetunée sur des données conversationnelles francophones du modèle Falcon7B (voir [10]).

Ainsi, en comparaison à GPT2, modèle quasiment monolingue (seulement 10MB de données était en français sur les 40000Mb totaux d'après [11]), on pourrait s'attendre à ce que Claire7B (entraîné sur 400M de mots en français selon https://huggingface.co/OpenLLM-France/Claire-7B-0.1°) ait des meilleurs résultats en comparaison.

Le modèle étant 10x plus important en taille, je ne disposais pas des ressources nécessaires sur le cluster du labo pour le finetuner sur ChiCo. Je vais donc employer une technique différente du finetuning, le *prompting* : je ne modifie pas les poids du modèle, je donne seulement le texte à compléter au modèle avec le plus grand contexte possible.

6.3.1.4. Résultats

	taille du contexte	75 TOKENS
langue du dataset	**threshold** / **modèle**	5 %
FR	gpt2-large (774M)	84.2 %
EN	gpt2-large (774M)	88.0%
FR	Claire-7B (7B)	79%

Tableau 1. – Taux de réussite des différents modèle selon les hyperparamètres

Voir Chapitre 7.1 pour une analyse des résultats obtenus.

6.4. ChiCa

Après la création du dataset Chico, Abdellah Fourtassi et son équipe se sont attelés à la création d'un nouveau dataset, le dataset **ChiCa**. Ce dernier est également un rassemblement de conversations adulte/enfant. Mais, contrairement, à ChiCo, ces conversations sont en face à face. De plus, l'aspect multimodal du dataset est renforcé par rapport à ChiCo, car en plus des modalités classiques, chaque intervenant était muni de lunettes traquant son regard.

J'ai donc tenté de reproduire les résultats obtenus sur ChiCo sur ce nouveau dataset. Mais contrairement aux expérimentation réalisées avec ChiCo, je vais ici tenter d'inclure la modalité visuelle dans le modèle, c'est-à-dire, prendre en compte les regards des interlocuteurs pendant la conversation (ie le *gaze*).

6.4.1. Prise en compte de la modalité visuelle

Pour chaque mot prononcé, on va noter si oui ou non la personne ayant prononcé le mot regardais son interlocuteur et donner cette information au modèle.

D'après la méta-analyse [12] sur le rôle du regard dans la prise de tour de paroles, l'intervenant a tendance a détourner le regard de l'interlocuteur en début de parole et à le regarder en fin de parole. En donnant au modèle cette information supplémentaire et au vu de la corrélation positive qu'il semble y a voir entre le changement de tour et le regard, on s'attend donc à voir une amélioration de ses performances avec ces données supplémentaires.

Ainsi, chaque mot (token) prononcé sera associé à une information binaire : a-il été prononcé en regardant l'interlocuteur ou non ?

On va incorporer cette information du regard dans chaque token en modifiant l'architecture du modèle. Pour cela, j'introduis une couche supplémentaire que j'appelle le *gaze embedding*. À la manière du *positional embedding*, qui consiste à rajouter à l'embedding de chaque token un vecteur encapsulant sa position dans la séquence, on va rajouter à chaque *gazed token* (un token prononcé quand l'intervenant regarde son interlocuteur) un vecteur spécifique : le *gaze vector*.
Vous pouvez voir une représentation visuelle de cette architecture à la Fig. 7 de l'annexe.

Pour identifier un token gazed d'un token non-gazed, je vais étendre la matrice d'embedding : pour chaque `token` dans la matrice d'embedding, on va ajouter `~token~` qui sera sa version gazed. Au début de l'entraînement, on aura également $\overrightarrow{\text{token}} = \overrightarrow{\sim\text{token}\sim}$ et $\overrightarrow{\text{gaze\_vector}} = \vec{0}$ de sorte qu'un token gazed et non-gazed possèdent la même sémantique originelle.

6.4.2. Manipulation du dataset

Il va maintenant falloir formater le dataset pour être sous la forme acceptée par notre modèle. L'exemple de Fig. 1 devient alors (en imaginant la direction des regards d'Alice et Bob) :

```
~salut~
<|turnshift|>
~salut~
<|turnshift|>
tu ne sais pas ~la~ ~dernière~
<|turnshift|>
~non~
<|turnshift|>
aujourd'hui le facteur n'est pas passé ~on~ ~m'a~ ~même~ ~raconté~ ~qu'il~ ~ne~
~passera~ ~jamais~ non mais vraiment c'est honteux ~et~ ~moi~ ~qui~ ~attendais~ du
courier [lève les yeux au ciel]
```

Mais, problème, dans le dataset original, les seules informations auquel on a accès sont :

- la position du regard à chaque seconde
- la positon du visage à chaque seconde
- le mot prononcé à chaque seconde

Il va donc falloir joindre les différentes tables pour savoir si pour tel mot, la direction du regard est alignée avec la position du visage, permettant de savoir si le mot est gazed ou non.

6.4.3. Gaze Embedding

Avec l'architecture proposée, en notant E la matrice d'embedding, la formule pour calculer l'embedding final d'un token s'écrit :

$$\overrightarrow{\text{token}} = E[\text{token}_{\text{id}}] + \overrightarrow{\text{positional\_embed}}[\text{position}_{\text{token}}]$$

$$\overrightarrow{\text{token\_gazed}} = E[\text{token\_gazed}_{\text{id}}] + \overrightarrow{\text{gaze\_vector}} + \overrightarrow{\text{positional\_embed}}[\text{position}_{\text{token\_gazed}}]$$

6.4.4. Premier test

Pour tester l'architecture proposée dans un contexte simplifié, je vais extrapoler le résultat de [12] à l'extrême en supposant que le seul et unique token gazed pour chaque tour de parole est le dernier token de chaque tour.
Ainsi, il y aura une corrélation totale entre la présence d'un token gazed et la présence d'un `<|turnshift|>` au token suivant. J'ai donc entraîné un modèle sur ces données de test pour voir si le modèle pouvait correctement généraliser ce pattern.

Après avoir mis en forme les données et finetuné le modèle customisé de gpt2, j'ai obtenu un score de 98.2 % avec le code du Chapitre 10.5.2, ce qui montre que l'architecture proposée est suceptible de généraliser la corrélation entre regard et tours de paroles.

6.4.5. Résultats Données Réelles

J'applique désormais l'architecture décrite précédemment aux données de ChiCa mises en formes :

	taille du contexte	75 TOKENS
langue du dataset	**threshold** / **modèle**	10 %
FR	gpt2 (117M)	86.3 %
FR	gpt2-large (774M)	88.1%

Tableau 2. – Taux de réussite de différents modèles sur ChiCa avec gaze

Voir Chapitre 7.1 pour une analyse des résultats obtenus.

7. Analyse des résultats proposés

7.1. ChiCo

7.1.1. De l'importance du contexte

Dans la conversation comme dans les transformers, le contexte a une grande importance. Une phrase ou un mot peut couvrir une sémantique tout différente selon le contexte antérieur.

J'ai donc tenté de vérifier cette assertion en testant les performances sur différentes tailles de contexte :

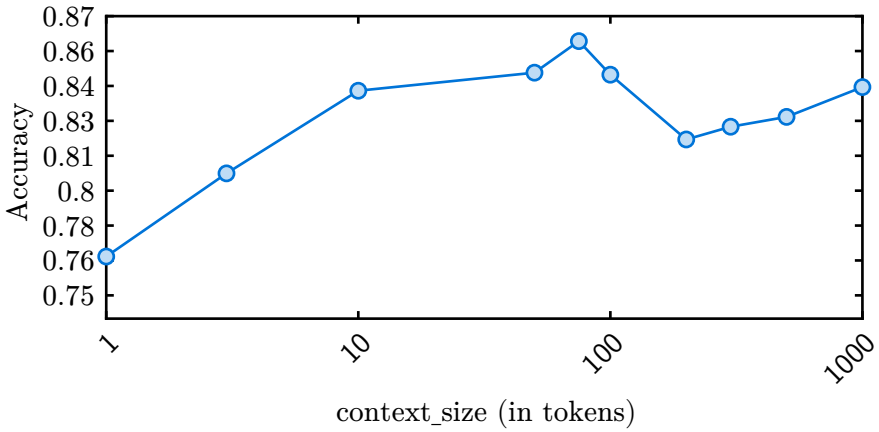


Fig. 3. – Réussite du modèle basé sur GPT2-large en fonction de la taille du contexte

Le résultat obtenu est surprenant,le taux de réussite n'est pas strictement croissant en fonction de la taille du contexte, même si la variation est assez légérere pour qu'elle puisse être assimilée à du bruit.

Pour tester l'impact réel du contexte sur le modèle, j'ai menée une étude qualitative sur quelques phrases de tests, qui j'ai demandé au modèle de prédire pour différents contextes :

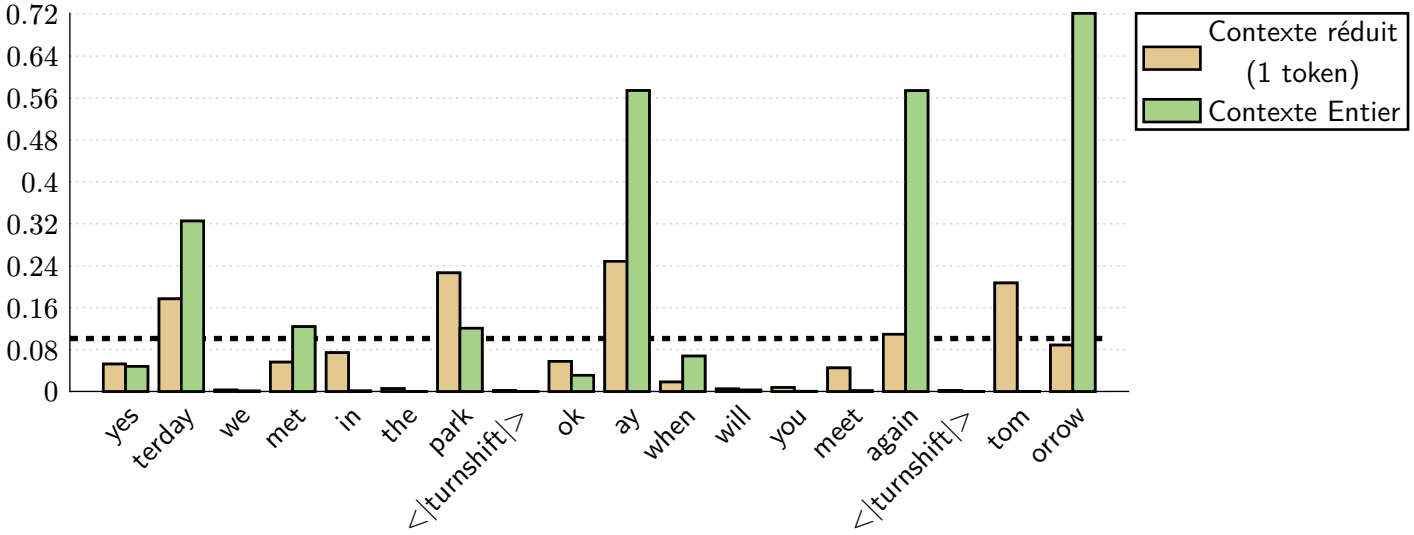


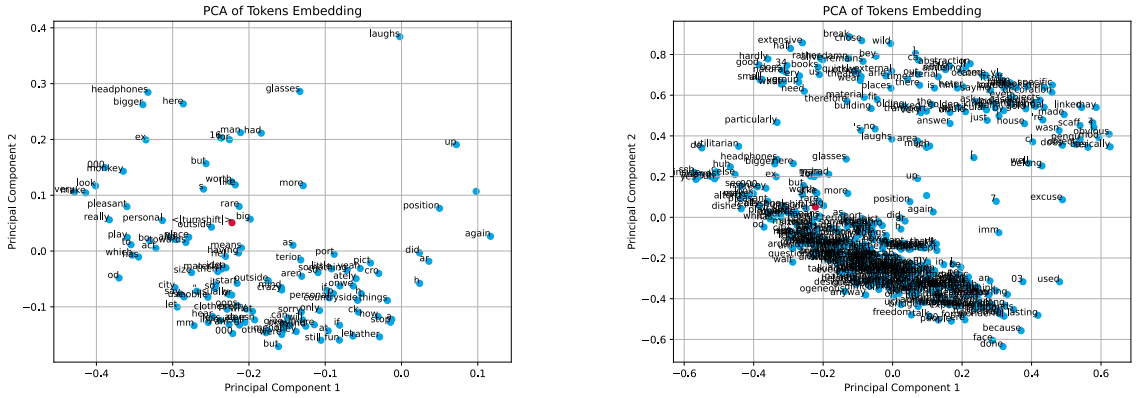
Fig. 4. – Prédiction de changement de tour sur une phrase (en pointillé le threshold d'acceptation de changement de tour du modèle)

On voit que les prédictions faites sont radicalement différentes d'un contexte à l'autre, montrant que le modèle ne comprend pas les phrases de la même manière.

7.1.2. Interprétation des prédictions

Ce modèle étant utilisé comme socle théorique pour différencier adulte et enfant, il serait intéressant de savoir les raisons de telle ou telle prédictions.
Or, un des gros problèmes dans la manipulation de grands modèles de langages est leur taille les rendant peu interprétables.
J'ai donc ici essayé d'utiliser des techniques du domaine de l'interprétabilité mécanique (voir [13]) pour tenter d'interpréter les prédiction du modèles.

Pour commencer, j'ai tenté de voir si des patterns pouvaient émerger en regardant seulement les vecteurs d'embedding de chaque token et en faisant un PCA (Principal Componnent Analysis, voir [14]), principalement pour voir quels sont les tokens les plus proche de `<|turnshift|>` (en rouge) :



(a) Graphe zoomé (b) Graphe dézoomé

Fig. 5. – PCA des vecteurs d'embedding pour GPT2-large finetuné sur ChiCo anglophone (en rouge le vecteur `<|turnshift|>`)

Même si la structure des des données est intéressante (2 zones semblent se dégager), les mots autour de `<|turnshift|>` semblent totalement décorrélées et s'apparentant à du bruit.
J'ai également tenté de produire un résultat similaire avec le modèle GazeGPT entraîné sur ChiCa.

Mais les résultats obtenus s'apparentent encore plus à du bruit. Vous pourrez malgré tout retrouver cette figure ici : Fig. 8.

Le problème avec cette méthodologie est qu'elle n'explore qu'une toute petite partie de la complexité d'un transformer. En effet, les poids de la matrice d'embedding représente moins de 0.03% des poids totaux.

Pour mieux comprendre le « raisonnement » du modèle, j'ai donc cherché un moyen pour voir ce que le modèle « pense » à chaque étape du calcul. Un moyen grossier de faire cela est de simplement appliquer la matrice d'unembedding sur les vecteurs de travail du modèle dans ces couches intermédiaires. Ainsi, en appliquant cette matrice puis un softmax, on obtient, le « token le plus probable » à chaque étape du calcul.
Dans la figure ci-après, on peut voir un exemple où le token avec la probabilité la plus importante est indiqué à plusieurs étapes de calcul (après application de la matrice d'unembedding), avec en haut le token suivant réel :

Layer Number	TER-DAY	WE	MET	IN	THE	PARK	<|TURNSHIFT|>
36	yes	<|turnshift|>	played	and	the	garden	and
32	ゼウス	evening	talked	again	the	garden	and
28	ゼウス	evening	talked	again	Paris	morning	where
24	ゼウス	evening	saw	again	front	morning	where
20	ゼウス	evening	've	with	front	same	where
16	ゼウス	evening	've	amorph	Prague	same	where
12	ゼウス	evening	're	amorph	Prague	same	lands
8	ゼウス	evening	ird	amorph	front	same	keepers
4	terday	evening	ird	ropolis	front	same	Meadows
0	yes	terday	we	met	in	the	park
Inputs	yes	terday	we	met	in	the	park

Tableau 3. – Utilisation de la *logit-lens* sur la phrase « yesterday we met in the park <|turnshift|> »

Comme on peut le voir, cette méthode d'interprétation est loin d'être optimale, les résultats obtenus s'apparentent plus à du bruit qu'à une prédiction réelle. En effet, comme décrit dans [15], la représentation interne des différents embedding au fil des couches est soumises à une *dérive de représentations* : certaines features sont représentées différemment (des sortes de « changement de base ») selon la couche. Or, comme chaque embedding est interprété selon la matrice d'unembedding de la dernière couche (on projette les vecteurs dans la base de la dernière couche), la différence de bases génère des prédictions erronées.

Pour remédier à ce problème, il faut donc interpréter les vecteurs de chaque couche avec la base appropriée. Pour cela, j'ai cherché à utiliser la méthode des *tuned lens* décrite dans [15]. Cette méthode consiste à entraîner, pour chaque couche que l'on souhaite interpréter, des transformations affines, de sorte que la distribution de probabilité résultante après application de ces transformations aux vecteurs de la couche k corresponde le plus possible à la distribution qui est obtenue à la couche finale.

Pour trouver les poids des matrices correspondants à ces transformations, on utilise la même méthode utilisée pour entraîner un transfromer, de la descente de gradient avec de multiples séries de tokens. Mais je n'ai malheureusement pas eu le temps de finaliser cette méthode d'interprétation dans le cas des modèles développés pendant mon stage

7.2. Comparaison Adulte / Enfant

Après avoir obtenu un modèle satisfaisant dans la prédiction des tours de paroles, on va maintenant chercher à comparer les performances du modèle entre les enfants et les adultes.

Pour cela, on va entraîner un modèle spécifiquement sur les conversations adulte/adulte et tester le modèles sur les interventions des enfants dans les conversations adulte/enfant. Il faut donc un moyen de séparer, dans le dataset, les interventions d'adulte et d'enfant.

Pour cela, je remplace le token de `<|turnshift|>` par deux tokens : `<|speaker1|>` et `<|speaker2|>`. `<|speaker1|>` remplace `<|turnshift|>` au début d'une intervention de l'interviewer et `<|speaker2|>` pour l'interviewé.

L'exemple de Fig. 1 devient alors :

```
<|speaker1|>
salut
<|speaker2|>
salut
<|speaker1|>
tu ne sais pas la dernière
<|speaker2|>
non
<|speaker1|>
aujourd'hui le facteur n'est pas passé on m'a même raconté qu'il ne passera jamais non
mais vraiment c'est honteux et moi qui attendais du courier [lève les yeux au ciel]
```

Ainsi, comme dans ChiCo, l'enfant est toujours l'interviewvé, il sera toujours associer au speaker 2, ce qui permettra une comparaison entre adulte (speaker 1) et enfant (speaker 2).

taille du contexte	**75 tokens**
threshold	**5 %**
modèle	**gpt2-large (774M)**
Adulte & Enfant	84.2 %
Adulte	86.2 %
Enfant	81.6 %

Tableau 4. – Taux de réussite de gpt2-large entraîné sur le corpus adulte/adulte en anglais selon le caractère adulte ou enfant de l'intervenant

On observe donc que les interventions des enfants sont plus difficile à prédire que les interventions des adultes. Sachant que le modèle a été entraîné sur des conversations entre adultes, cela montre une différence d'expression entre adulte et enfant, du fait que le modèle a plus de mal à prédire les fin de tours des enfants. On peut donc emmètre l'hypothèse que cette différence provient de l'apprentissage inabouti des enfants sur les coutumes et automatismes de la conversation.

Mais du fait de la faible quantité de donnée, il est difficile de savoir si l'écart de pourcentage est assez important pour être significatif.

7.3. De l'importance du gaze

Après avoir réalisé des tests sur le dataset ChiCa incluant la modalité visuelle du gaze (voir Chapitre 6.4.5), j'ai également appliqué la méthodologie présentée au Chapitre 6.3.1.1 à ChiCa. J'ai obtenu des résultats trés similaire à ceux de ChiCo avec un taux de réussite de 84.1%.

On observe donc qu'en prenant en compte la modalité du gaze, le modèle gagne 4 points de précision par rapport au modèle usant uniquement de la modalité textuelle.

Ce résultat confirme donc la corrélation positive entre regard et prise de parole décrite dans [12] et confirme la pertinence de l'architecture proposée en Fig. 7.

8. Déroulement du stage

J'ai passé une grande partie de mon stage à écrire du code Python et à expérimenter avec différents modèles et hyper-paramètres pour atteindre mes objectifs de prédiction. La majorité du code a été écrit pour l'entraînement des différents modèles et la producction de résultats d'inteprétabilité. Ce stage m'a permis de me familiariser avec l'architecture des Transformers et la bibliothèque python `transformers`. J'ai également appris à utiliser `slurm` pour lancer et gérer des tâches sur le cluster du laboratoire.

Pour donner une idée sur la répartition de mon temps pendant mon stage, vous trouverez un résumé semaine par semaine au Chapitre 10.3.

J'ai pu également participer à la vie du laboratoire en assistant à un séminaire hebdomadaire chaque jeudi.

9. Conclusion

Au travers de ce stage, j'ai pu appliquer des techniques d'entraînement et d'interprétabilité liées aux transformers et au problème de la prédiction de tours de parole. J'ai notamment pu comparer les performances des adultes et des enfants en utilisant un transformer fine-tuné comme base comparative. J'ai également modifié l'architecture habituelle des transformers pour pouvoir les rendre multimodaux en traitant la modalité visuelle du regard, produisant le modèle GPT-Gaze. Enfin, j'ai appliqué des techniques d'interprétabilité pour essayer de mieux comprendre la structure des tours de parole.

Néanmoins, ce travail présente ses limites. Premièrement, la taille réduite des datasets à disposition (de l'ordre de 100 Ko pour ChiCo et ChiCa) diminue la fiabilité des résultats proposés. De plus, le manque de résultats liés à l'interprétabilité des modèles limite également les conclusions sur la capacité conversationnelle adulte/enfant que l'on peut déduire des résultats.

Des pistes d'améliorations seraient la production d'un dataset plus conséquent (qui nécessiterait de récolter de nouvelles données in vivo) et surtout de produire plus de résultats d'interprétabilité. En particulier, produire des résultats sur l'importance du regard chez l'enfant (et non de manière générique enfant/adulte comme cela a été fait) et l'utilisation de la tuned lens, ce qui donnerait une modélisation du chemin de pensée dans le choix de la prise de parole.

10. Annexes

10.1. Explication du modèle LSTM

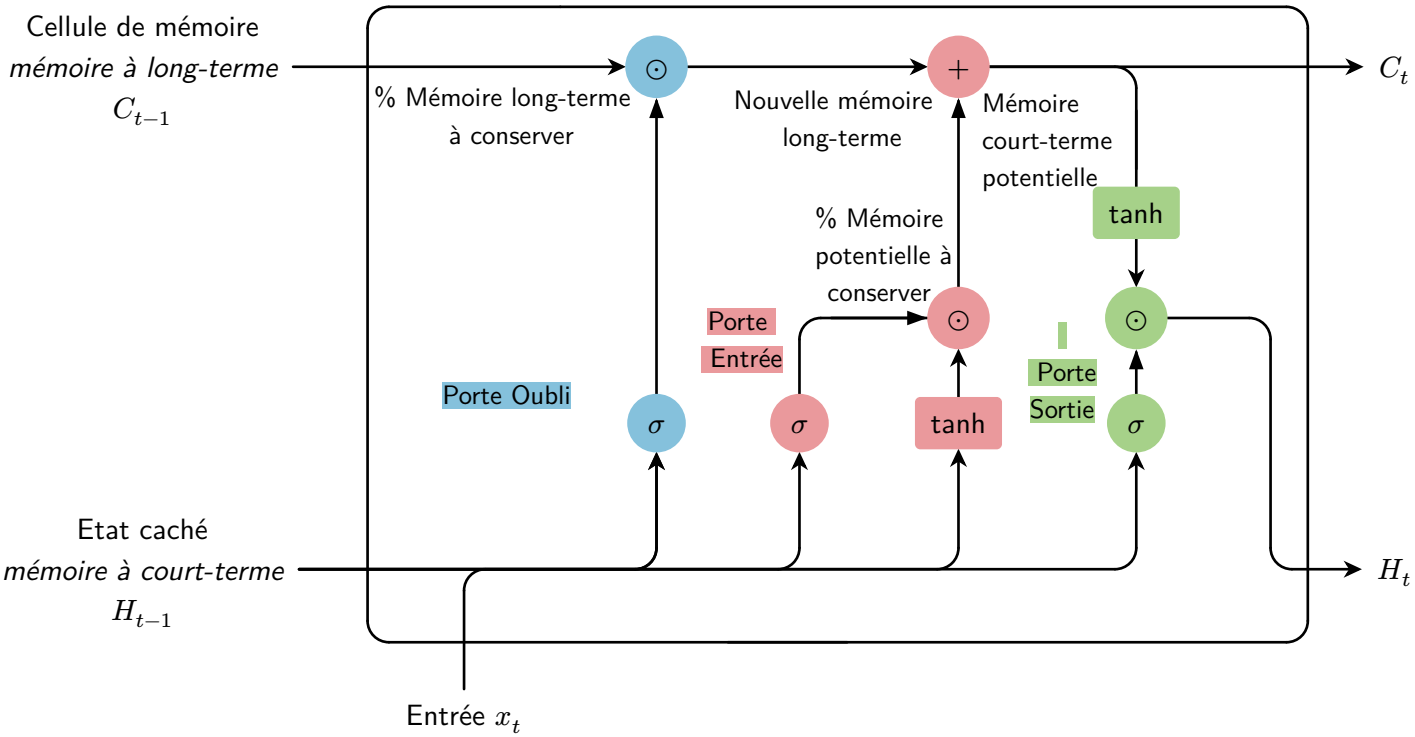


Fig. 6. – Architecture d'un LSTM

Contrairment à son ancêtre, le modèle de Réseau Neuronal Récurrent simple, qui avait une mémoire trés courte et dont l'entraînement était souvent instable (c'est le problème du *vanishing /explosing gradient*), le LSTM, lui, introduit une mémoire à long-terme afin de modèliser des phénomènes prenant place sur un temps plus long.

Cette architecture permet (voir Fig. 6) de modéliser la mémoire à court terme par l'*état caché*, trais influencé par les derniers inputs et la mémoire à long-terme par la *cellule de mémoire*, influencée par toute la séquence d'inputs.

Ainsi, en considérant une séquence d'inputs $(x_1, ..., x_t)$, on va répéter le processus décrit par Fig. 6 pour chacun d'entre eux et obtenir successivement les états suivants pour les états cachés $(H_0, ..., H_t)$ et de même pour les cellules de mémoire $(C_0, ..., C_t)$ (avec H_0 et C_0 initialisés à 0)

Ce processus est composé de 3 étapes :

- La *Porte d'Oubli* : Aprés avoir mulitplié H_{t-1} et x_t par des poids, on utilise la fonction sigmoïde sur la somme de ces deux quantités pour obtenir le pourcentage de mémoire à conserver. Ainsi, si $C_{t-1} = 2$ et $\sigma(w_0 H_{t-1} + w_1 x_t) = 0.5$ alors $C_{t-1} = 1$
  Cette porte permet de « faire de la place » dans la mémoire à long-terme
- La *Porte d'Entrée* : Similairement au point précédent, on obtient avec la sigmoïde un pourcentage correspondant au pourcentage de mémoire potentielle à ajouter à la mémoire à long-terme. Cette mémoire se calcule de la même façon mais avec la tangente hyperbolique. Ainsi, à la mémoire à long-terme on ajoute : $\sigma(w_2 H_{t-1} + w_3 x_t)\tanh(w_3 H_{t-1} + w_4 x_t)$.
- La *Porte de sortie* : A l'inverse de la porte précédente, qui décidait comment l'état caché allait influer sur la cellule de mémoire, cette porte décide comment la cellule de mémoire influe sur le nouvel état caché.

En répétant de manière séquentielle ce processus sur les entrées $(x_1, ..., x_t)$, on obtient alors l'output du modèle H_t : dans notre cas, le channel actuel prédis de l'interlocuteur en fonction de $(x_1, ..., x_t)$, le contexte de deux secondes de la conversation.

10.2. Explication du mécanisme d'attention

Pour permettre à chaque token de s'enrichir du contexte, ces derniers pourront questionner les autres tokens et affiner leurs sémantique en fonction des réponses.

Comme indiqué en Fig. 2, ce mécanisme comporte plusieurs *têtes d'attention*, je vais ici le fonctionnement d'une d'entre elle.

Une tête fonctionne en utilisant les poids suivants :

- Une matrice de requêtes (*queries*) : W_Q
- Une matrice de clés (*keys*) : W_K
- Une matrice de valeurs (*values*) : W_V

A chaque token, on va associer une question $\vec{Q} = W_Q\overrightarrow{\text{token}}$ et une clé $\vec{K} = W_K\overrightarrow{\text{token}}$. Pour savoir si un token de clé $\vec{K}$ répond correctement à une question $\vec{Q}$ associée à un autre token, on calcule $\vec{K} \cdot \vec{Q}$: plus la valeur est grande, le mieux le token répond à la question.

Voici un exemple de ce calcul pour la suite de tokens suivante `[today, the sky, is]` :

	$\overrightarrow{\text{today}}$ $\downarrow W_Q$ $\vec{Q}_1$	$\overrightarrow{\text{the}}$ $\downarrow W_Q$ $\vec{Q}_2$	$\overrightarrow{\text{sky}}$ $\downarrow W_Q$ $\vec{Q}_3$	$\vec{\text{is}}$ $\downarrow W_Q$ $\vec{Q}_4$
$\overrightarrow{\text{today}} \underset{W_K}{\rightarrow} \vec{K}_1$	$\vec{K}_1 \cdot \vec{Q}_1$	$\vec{K}_1 \cdot \vec{Q}_2$	$\vec{K}_1 \cdot \vec{Q}_3$	$\vec{K}_1 \cdot \vec{Q}_4$
$\overrightarrow{\text{the}} \underset{W_K}{\rightarrow} \vec{K}_2$	$\vec{K}_2 \cdot \vec{Q}_1$	$\vec{K}_2 \cdot \vec{Q}_2$	$\vec{K}_2 \cdot \vec{Q}_3$	$\vec{K}_2 \cdot \vec{Q}_4$
$\overrightarrow{\text{sky}} \underset{W_K}{\rightarrow} \vec{K}_3$	$\vec{K}_3 \cdot \vec{Q}_1$	$\vec{K}_3 \cdot \vec{Q}_2$	$\vec{K}_3 \cdot \vec{Q}_3$	$\vec{K}_3 \cdot \vec{Q}_4$
$\vec{\text{is}} \underset{W_K}{\rightarrow} \vec{K}_4$	$\vec{K}_4 \cdot \vec{Q}_1$	$\vec{K}_4 \cdot \vec{Q}_2$	$\vec{K}_4 \cdot \vec{Q}_3$	$\vec{K}_4 \cdot \vec{Q}_4$

Or, chose très importante pour l'étape d'entraînement du modèle, on ne souhaite pas qu'un token ait des informations sur des tokens dans le futur (`the` ne doit pas savoir qu'il y a un `is` deux tokens plus loin). On va donc *masquer* les résultats ne respectant pas ce critère (-∞ étant la pire réponse à une question) :

	$\overrightarrow{\text{today}}$	$\overrightarrow{\text{the}}$	$\overrightarrow{\text{sky}}$	$\vec{\text{is}}$
	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$
	$\vec{Q}_1$	$\vec{Q}_2$	$\vec{Q}_3$	$\vec{Q}_4$
$\overrightarrow{\text{today}} \underset{W_K}{\rightarrow} \vec{K}_1$	$\vec{K}_1 \cdot \vec{Q}_1$	$\vec{K}_1 \cdot \vec{Q}_2$	$\vec{K}_1 \cdot \vec{Q}_3$	$\vec{K}_1 \cdot \vec{Q}_4$
$\overrightarrow{\text{the}} \underset{W_K}{\rightarrow} \vec{K}_2$	-∞	$\vec{K}_2 \cdot \vec{Q}_2$	$\vec{K}_2 \cdot \vec{Q}_3$	$\vec{K}_2 \cdot \vec{Q}_4$
$\overrightarrow{\text{sky}} \underset{W_K}{\rightarrow} \vec{K}_3$	-∞	-∞	$\vec{K}_3 \cdot \vec{Q}_3$	$\vec{K}_3 \cdot \vec{Q}_4$
$\vec{\text{is}} \underset{W_K}{\rightarrow} \vec{K}_4$	-∞	-∞	-∞	$\vec{K}_4 \cdot \vec{Q}_4$

Après cela, on convertit ces nombres en une distribution de probabilité avec l'algorithme du softmax (voir [8]) et on convertit chaque token en son vecteur de valeur associé :

$$\begin{vmatrix} \overrightarrow{\text{today}} \underset{W_V}{\rightarrow} \vec{V}_1 \\ \overrightarrow{\text{the}} \underset{W_V}{\rightarrow} \vec{V}_2 \\ \overrightarrow{\text{sky}} \underset{W_V}{\rightarrow} \vec{V}_3 \\ \vec{\text{is}} \underset{W_V}{\rightarrow} \vec{V}_4 \end{vmatrix}$$

Et ainsi, pour chaque token, on met à jour le vecteur associé :

$$\overrightarrow{\text{sky}} \mathrel{+}= 0 \cdot \vec{V}_1 + 0 \cdot \vec{V}_2 + \text{softmax}\Big(\vec{K}_3 \cdot \vec{Q}_3\Big) \cdot \vec{V}_3 + \text{softmax}\Big(\vec{K}_3 \cdot \vec{Q}_4\Big) \cdot \vec{V}_4$$

Par exemple, la question posée avec W_Q pourrait-être « Suis-je intéressé par le fait d'être bleu ? », la clé W_K « Suis-je bleu ? » et la valeur W_V représenterais la sémantique du bleu.

Et dans ce cas, $\vec{K}_{\text{sky}} \cdot \vec{Q}_{\text{is}}$ aura sûrement une valeur très élevée.

10.3. Répartition de mon temps

- SEMAINE 1 : Lecture du papier de mon encadrant, Compréhension du modèle LSTM
- SEMAINE 2 : Reproduction des résultats du papier de mon encadrant, Compréhension de l'architecture des transformers, Lecture de la littérature sur les tours de parole
- SEMAINE 3 : Conception théorique du modèle de ChiCo, Mise en forme du dataset ChiCo, Implémentation du modèle, Premiers entranements du modèle
- SEMAINE 4 : Entrainement plus poussé du modèle, Entrainement d'un modèle sur ChiCo traduit en anglais, Premiers résultats d'interprétabilité
- SEMAINE 5 : Test avec des LLM plus important en taille comme Claire-7B, Méthode pour évaluer enfants et adultes séparémment, Entrainements et tests du modèle `<|speaker1|>` / `<|speaker2|>`
- SEMAINE 6 : Conception d'une méthode afin d'incorporer le gaze dans l'architetcure du transformer, Mise en forme du dataset ChiCa
- SEMAINE 7 : Implémentation de GazeGPT, Tests et entrainement du modèle
- SEMAINE 8 : Résultats d'interprétabilité sur GazeGPT, Travail de relecture sur le rapport avec mon encadrant

10.4. Figures

10.4.1. Architecture modifiée : GazeGPT

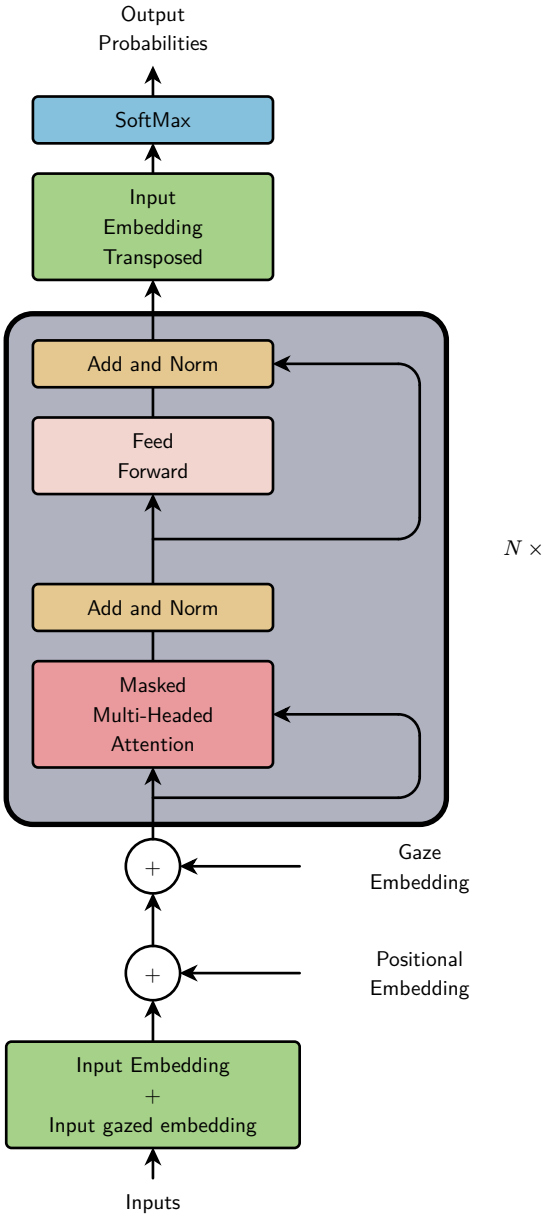


Fig. 7. – Architecture modifiée du transformer

10.4.2. PCA Embedding ChiCa

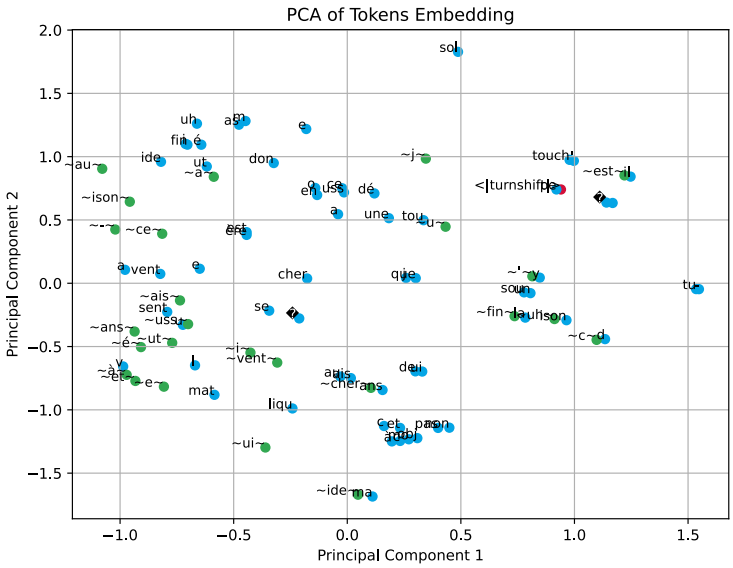


Fig. 8. – Graphe zoomé

10.5. Code

10.5.1. Entrainement du modèle

```python
import torch
import transformers
import numpy as np


device = 'cpu'
if torch.cuda.is_available():
    device = 'cuda'


def fine_tune_model(dataset_path, model, tokenizer, output_dir, progress_bar=True, epochs=3, batch_size=4, block_size=128):

    dataset = transformers.TextDataset(
        tokenizer=tokenizer,
        file_path=dataset_path,
        block_size=block_size
    )

    data_collator = transformers.DataCollatorForLanguageModeling(
        tokenizer=tokenizer,
        mlm=False,
    )

    training_args = transformers.TrainingArguments(
        output_dir=output_dir,
```

```
        overwrite_output_dir=True,
        num_train_epochs=epochs,
        per_device_train_batch_size=batch_size,
        save_steps=10_000,
        save_total_limit=2,
        logging_dir=f"{output_dir}/logs",
        logging_steps=200,
        disable_tqdm = not progress_bar
    )

    trainer = transformers.Trainer(
        model=model,
        args=training_args,
        data_collator=data_collator,
        train_dataset=dataset,
    )

    trainer.train()

    model.save_pretrained(output_dir)
    tokenizer.save_pretrained(output_dir)

```

10.5.2. Test du modèle

```python
import torch
import transformers
import numpy as np
from tqdm import tqdm

# Set the device
device = 'cpu'
if torch.cuda.is_available():
    device = 'cuda'


turnshift_id = 50257


def turnshift_prob(input_tokens, model):
    model.to(device)

    with torch.no_grad():
        outputs = model(input_tokens)
        logits = outputs.logits[:, -1, :]
        softmax_logits = torch.softmax(logits, dim=-1)

    prob = softmax_logits[0, turnshift_id].item()

```

```
    return prob


def compute_score(test_path,model, tokenizer,progress_bar=True, context_size = 100, threshold = 0.05):
    nb_test = 0
    nb_success = 0
    text = ""
    with open(test_path, 'r', encoding='utf-8') as test_file:
        text += test_file.read()
    test_tokens = tokenizer.encode(text, return_tensors='pt').to(device)



    nb_tokens = test_tokens.size(dim=1)
    np_test_tokens = test_tokens.to('cpu').numpy().flatten()
    indexes_turnshift = np.where(np_test_tokens == turnshift_id)[0]
    nb_turnshift = len(indexes_turnshift)


    nb_samples = 10

    if progress_bar:
        pbar = tqdm(total=(nb_samples + 1)*nb_turnshift, desc=f"Score: nan", ncols=100)

    for s in range(nb_samples + 1):
        indexes = np.random.randint(nb_tokens, size=nb_turnshift)
        if s == nb_samples:
            indexes = indexes_turnshift
            nb_test /= nb_samples
            nb_success /= nb_samples
        for i in indexes:
            if i == 0:
                continue
            nb_test += 1
            turnshift_pred = turnshift_prob(test_tokens[:, max(0, i - context_size):i], model) > threshold
            turnshift_groundtruth = test_tokens[0,i].item() == turnshift_id
            nb_success += (turnshift_pred == turnshift_groundtruth)

            if progress_bar:
                pbar.set_description(f"Score: {nb_success / nb_test}")
                pbar.update(1)

    if progress_bar:
        pbar.close()

    return nb_success / nb_test
```

```
71
72
73
```

10.5.3. GPT-Gaze

```python
import torch
from transformers import GPT2Model, GPT2Config, GPT2Tokenizer, GPT2LMHeadModel
import torch.nn as nn
from transformers import GPT2Tokenizer


class GPT2TokenGaze(GPT2LMHeadModel):

    def __init__(self, config):
        super().__init__(config)
        self.gaze_vector = nn.Parameter(torch.randn(config.n_embd))
        self.turnshift_id = 100513
        self.gazed_id = 50257

    def forward(
        self,
        input_ids=None,
        past_key_values=None,
        attention_mask=None,
        encoder_attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        encoder_hidden_states=None,
        labels=None,
        use_cache=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
        end_gaze=False,
    ):
                return_dict = return_dict if return_dict is not None else self.config.use_return_dict
        device = input_ids.device if input_ids is not None else 'cpu'
        self.gaze_vector.data = self.gaze_vector.data.to(device)


        if inputs_embeds is None:
            is_token_gazed = np.where(input_ids.cpu().numpy()[0] >= self.gazed_id, 1, 0)
             is_token_gazed = torch.tensor(is_token_gazed, device=device)

```

```
            inputs_embeds = self.transformer.wte(input_ids)
            inputs_embeds += self.gaze_vector * is_token_gazed.view(-1, 1)

        transformer_outputs = self.transformer(
            input_ids=None,
            past_key_values=past_key_values,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            encoder_hidden_states=encoder_hidden_states,
            encoder_attention_mask=encoder_attention_mask,
            use_cache=use_cache,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )
        hidden_states = transformer_outputs[0]

        if self.model_parallel:
            torch.cuda.set_device(self.transformer.first_device)
            hidden_states = hidden_states.to(self.lm_head.weight.device)

        lm_logits = self.lm_head(hidden_states)

        loss = None
        if labels is not None:
            labels = labels.to(lm_logits.device)
            shift_logits = lm_logits[..., :-1, :].contiguous()
            shift_labels = labels[..., 1:].contiguous()
            loss_fct = nn.CrossEntropyLoss()
                loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.view(-1))

        if not return_dict:
            output = (lm_logits,) + transformer_outputs[1:]
            return ((loss,) + output) if loss is not None else output

        return transformers.modeling_outputs.CausalLMOutputWithCrossAttentions(
            loss=loss,
            logits=lm_logits,
            past_key_values=transformer_outputs.past_key_values,
            hidden_states=transformer_outputs.hidden_states,
            attentions=transformer_outputs.attentions,
            cross_attentions=transformer_outputs.cross_attentions,
        )

    def initialize_duplicated_tokens(self):
```

```
90          original_num_tokens = 50257
91                                                  duplicated_embeddings =
self.transformer.wte.weight.data[:original_num_tokens].clone()
92                          self.transformer.wte.weight.data[original_num_tokens:] =
duplicated_embeddings
93
```

Bibliographie

[1] E. A. J. Sacks Harvey; Schegloff, « A Simplest Systematics for the Organization of Turn-Taking for Conversation », 1974.

[2] E. Baines et C. Howe, « Discourse topic management and discussion skills in middle childhood: The effects of age and task », *First Language*, vol. 30, nº 3–4, p. 508-534, 2010.

[3] A. Agrawal, J. Liu, K. Bodur, B. Favre, et A. Fourtassi, « Development of Multimodal Turn Coordination in Conversations: Evidence for Adult-like behavior in Middle Childhood ». PsyArXiv, mai 2023. doi: 10.31234/osf.io/h8j6x°.

[4] K. Bodur, M. Nikolaus, F. Kassim, L. Prévot, et A. Fourtassi, « ChiCo: A Multimodal Corpus for the Study of Child Conversation », in *Companion Publication of the 2021 International Conference on Multimodal Interaction*, in ICMI '21 Companion. Montreal, QC, Canada: Association for Computing Machinery, 2021, p. 158-163. doi: 10.1145/3461615.3485399°.

[5] A. Vaswani *et al.*, « Attention is All you Need », in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, et R. Garnett, Éd., Curran Associates, Inc., 2017, p. . [En ligne]. Disponible sur: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf°

[6] T. Wolf *et al.*, « Transformers: State-of-the-Art Natural Language Processing », in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, oct. 2020, p. 38-45. [En ligne]. Disponible sur: https://www.aclweb.org/anthology/2020.emnlp-demos.6°

[7] R. Sennrich, B. Haddow, et A. Birch, « Neural machine translation of rare words with subword units », *arXiv preprint arXiv:1508.07909*, 2015.

[8] Wikipedia, « Softmax function — Wikipedia, The Free Encyclopedia ». 2024.

[9] E. Ekstedt et G. Skantze, « TurnGPT: a Transformer-based Language Model for Predicting Turn-taking in Spoken Dialog », in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Online: Association for Computational Linguistics, nov. 2020, p. 2981-2990. doi: 10.18653/v1/2020.findings-emnlp.268°.

[10] E. Almazrouei *et al.*, « Falcon-40B: an open large language model with state-of-the-art performance », 2023.

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, et I. Sutskever, « Language Models are Unsupervised Multitask Learners », 2019.

[12] Z. Degutyte et A. Astell, « The role of eye gaze in regulating turn taking in conversations: a systematized review of methods and findings », *Frontiers in Psychology*, vol. 12, p. 616471, 2021.

[13] L. Bereska et E. Gavves, « Mechanistic Interpretability for AI Safety–A Review », *arXiv preprint arXiv:2404.14082*, 2024.

[14] Wikipedia, « Principal component analysis — Wikipedia, The Free Encyclopedia ». 2024.

[15] N. Belrose *et al.*, « Eliciting latent predictions from transformers with the tuned lens », *arXiv preprint arXiv:2303.08112*, 2023.