Compréhension et génération de langage naturel fondées sur graphes de connaissances

Timothée Bachy

Compréhension et génération de langage naturel fondées sur graphes de connaissances

Timothée
BACHY

Sous la direction de Gwénolé LECORVÉ & Quentin BRABANT

Entreprise: ORANGE

UFR LLASIC
Département I3L

Mémoire de master 2 mention Sciences du Langage - 60 crédits

Parcours : Industries du Langage

Année universitaire 2021-2022

Université Grenoble Alpes

Compréhension et génération de langage naturel fondées sur graphes de connaissances

Timothée BACHY

Sous la direction de Gwénolé LECORVÉ & Quentin BRABANT

Entreprise: ORANGE

UFR LLASIC
Département I3L

Mémoire de master 2 mention Sciences du Langage - 60 crédits

Parcours : Industries du Langage, option professionnelle

Année universitaire 2021-2022

Remerciements

Je tiens à remercier tout d'abord mes deux tuteurs de stages, Gwénolé et Quentin pour tout ce qu'ils m'ont apporté au cours de ces 5 mois en termes de connaissances et de savoir-faire. Un grand merci également pour leur patience, leur pédagogie et toutes les discussions sympathiques et décontractées lors des pauses déjeuner.

Je remercie de même toute l'équipe NADIA (permanents, doctorants et alternants) plus généralement, je n'y ai fait que des rencontres très positives et me suis senti accueilli très chaleureusement et amicalement dès mon arrivée.

Enfin, un grand merci à ma copine et à ma famille de m'avoir soutenu et entouré malgré les difficultés de la distance.

DÉCLARATION ANTI-PLAGIAT

1. Ce travail est le fruit d'un travail personnel et constitue un document original.
2. Je sais que prétendre être l'auteur d'un travail écrit par une autre personne est une pratique sévèrement sanctionnée par la loi.
3. Personne d'autre que moi n'a le droit de faire valoir ce travail, en totalité ou en partie, comme le sien.
4. Les propos repris mot à mot à d'autres auteurs figurent entre guillemets (citations).
5. Les écrits sur lesquels je m'appuie dans ce mémoire sont systématiquement référencés selon un système de renvoi bibliographique clair et précis.

PRENOM : Timothée

NOM : BACHY

DATE : Le 30/07/2022

Sommaire

Table des matières

Introduction

J'ai effectué mon stage de fin d'études chez Orange, au sein de sa sous-entité Orange Innovation et plus précisément parmi les membres de l'équipe NADIA, à Lannion (en Bretagne). Ce stage a duré 5 mois, du 7 mars au 5 août. Il s'est déroulé sous la tutelle de deux chercheurs de l'équipe, Gwénolé LECORVE et Quentin BRABANT. Orange est l'un des leaders mondiaux de la télécommunication et le premier en Europe, avec environ 142 000 employés répartis dans 26 pays d'Europe, d'Afrique, d'Amérique du Nord et du Sud. Orange Innovation est la division orientée recherche d'Orange. Intégrant près de 5000 collaborateurs dont 3700 ingénieurs, elle est répartie sur 19 sites dont Lannion, mais aussi Châtillon, Grenoble, Caen ou Rennes en France, ainsi que d'autres à l'étranger comme à Londres, Madrid ou Tokyo. Au sein de ce vaste univers de recherche, l'équipe NADIA (pour NAtural DIAlogue) est l'une de celles qui se chargent de la recherche en Traitement du Langage Naturel, avec une spécialité orientée dialogue. On y travaille entre autres sur la catégorisation de textes, l'analyse d'opinion, l'analyse de conversation et les systèmes de dialogue humain-machine

Le sujet de mon stage était de mener des recherches pour déterminer si l'on pouvait entraîner un modèle capable de traiter à la fois la tâche de compréhension du langage naturel et celle de génération de langage naturel. Ce stage prenait place dans un contexte de recherche plus vaste qui cherche à simplifier la mise en place de la chaîne de traitement d'un système de questions-réponses basé sur des graphes de connaissances, pour éviter de cumuler les modules mono-tâche le long de cette chaîne en proposant un seul modèle capable de toutes les exécuter. La finalité de ce sujet de recherche serait une amélioration potentielle des services de questions-réponses proposés par Orange.

Dans ce rapport de stage, je vais présenter mon travail, en commençant par donner un peu de contexte et définir les tâches de compréhension et de génération de langage naturel, les représentations formelles du langage manipulées dans le cadre de ces tâches ainsi que les métriques utilisées pour évaluer la performance des modèles effectuant ces tâches. Je présenterai aussi quelques modèles préexistants pour ces tâches. Dans un second temps, je rentrerai dans le détail de mon travail à proprement parler, c'est-à-dire mes lectures d'articles, les outils auxquels je me suis formé et/ou que j'ai utilisés, les modèles et corpus, les différentes expériences menées et les problèmes rencontrés. Finalement, je commenterai les résultats que j'ai obtenus lors des différentes séries d'évaluations pour chacune des tâches sur chacun des corpus, avant de conclure avec les futurs travaux qui seront menés sur le sujet en poursuite de ce stage et mon bilan personnel.

Partie 1

-

Contexte et définition des tâches

Chapitre 1. Graphes de connaissances

Il convient de commencer par présenter correctement les objets que j'ai été amené à manipuler au cours de mon stage. En particulier ce qu'on appelle les représentations formelles, utilisées dans les deux tâches de compréhension et de génération.

1.1. Définition

Un graphe de connaissance est un objet abstrait qui a pour vocation de donner un aperçu d'un domaine de connaissance. Ce domaine est constitué par des entités qui entretiennent des relations entre elles. Dans un graphe de connaissances, chaque entité est unique, mais plusieurs entités peuvent entretenir la même relation. La figure 1[1] donne un exemple de graphe de connaissances. Les entités sont représentées par les nœuds du graphe et les relations par des arcs étiquetés avec le nom des propriétés.

Cette formalisation des connaissances permet de s'abstraire de l'ambiguïté du langage naturel (choix du vocabulaire, syntaxe, etc.) et ouvre l'exploitation des connaissances aux techniques de traitements de graphes (sujet non étudié durant le stage).

Parmi les représentations possibles d'un graphe, celle que j'ai utilisée s'appuie sur sa décomposition en un ensemble de triplets.

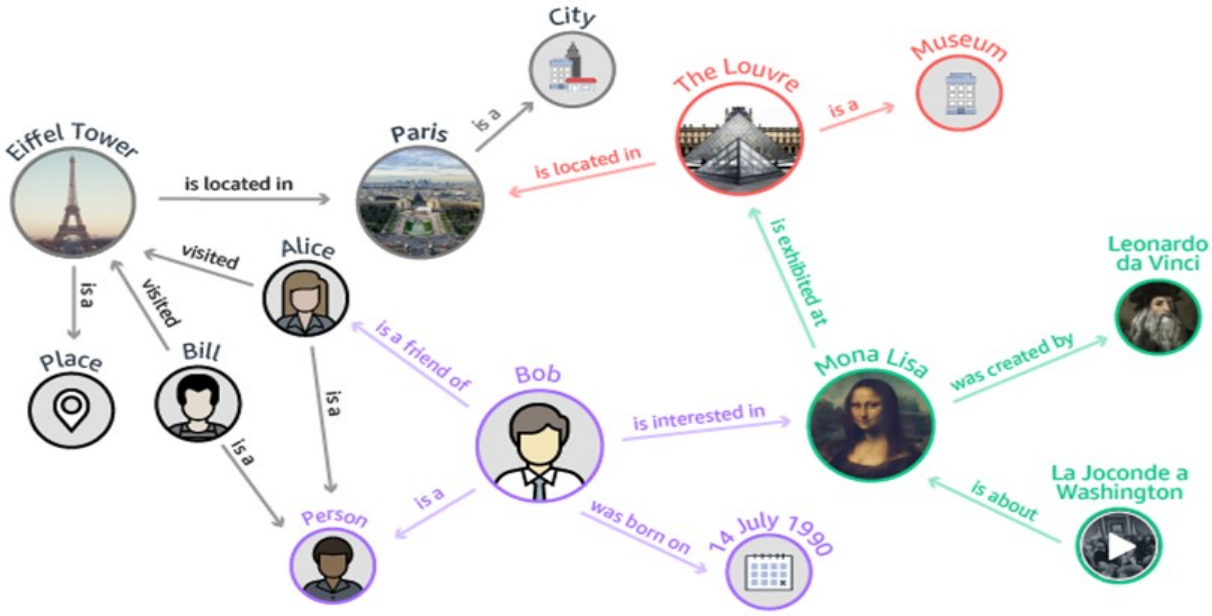

Figure 1: Un exemple de graphe de connaissances

1 Source : https://aws.amazon.com/fr/neptune/knowledge-graphs-on-aws/

1.2 Triplet RDF

Le *Ressource Description Framework* (RDF) ou Cadre de Description des Ressources en français est un format de graphe de connaissance utilisé pour décrire les ressources accessibles en ligne. L'élément fondamental de ce type de graphe est un objet appelé triplet, composé des 3 éléments suivants :

- Un **sujet**, par exemple : Le_Louvre

- Un **prédicat**, par exemple : IsLocatedIn

- Un **objet**, par exemple : Paris

Ce qui donne le triplet : (Le_Louvre, IsLocatedIn, Paris)

Dans l'exemple du graphe de connaissances donné plus haut, un triplet correspond par exemple au nœud intitulé « Le Louvre » (le sujet), à l'arc intitulé « Is located in » qui part de ce nœud, et enfin à l'entité « Paris », destination de l'arc. Notre graphe peut ainsi être décomposé de manière exhaustive en une liste de triplets, chaque entité pouvant apparaître dans plusieurs triplets. Les flèches correspondent toujours à un prédicat, c'est à dire la relation qui unit le sujet et son objet. Une entité peut apparaître comme sujet ou objet dans n'importe quel nombre de triplets de l'ensemble.

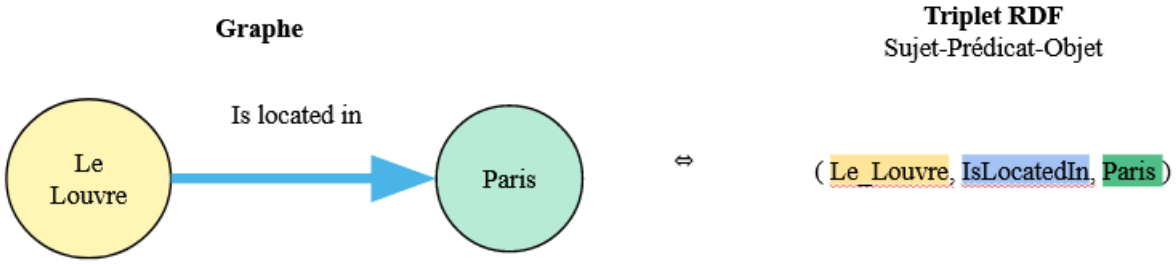

Figure 2: Illustration de la correspondance entre représentation graphique et sous forme de triplets d'un graphe de connaissances

Le triplet, ou plutôt les ensembles de triplets ont constitué un des deux types de langage formel utilisés lors de mon stage. Le second est le format de requête SPARQL

1.3 Langage de requête SPARQL

Pour trouver la réponse à une question dans une base de données, nous employons le langage de requête appelée SPARQL (pour SPARQL Protocol and RDF Query Language). C'est le langage de requête standard pour les bases de données de type RDF. Ses requêtes exploitent l'objet triplet RDF, précédemment introduit, de la manière suivante :

```
SELECT ?m
WHERE {
        ?m     isA    Monument
        ?m     isLocatedIn Paris
        ?m     startsWith  'L'
} ORDER ASC . LIMIT 5
```

La première partie de la requête indique le type d'action que l'on veut mener. Ici on cherche à sélectionner toutes les entités « `?m` » qui rempliront les contraintes. Le seconde partie de la requête qui commence avec un `WHERE`, liste entre deux accolades toutes les contraintes que doivent valider les entités dans la base pour être sélectionnées. On peut remarquer aisément que ces contraintes sont des triplets « à trou ». Dans notre exemple c'est le sujet de chaque triplet que l'on cherche. Après la partie `WHERE`, on peut retrouver un nombre variable de prérequis supplémentaires donnés sous la forme de tâches à exécuter sur les résultats trouvés, comme un décompte, un tri ordonné, etc. Ici nous demandons au système de ranger les réponses trouvées par ordre alphabétique croissant et de ne prendre que les 5 premières.

Maintenant que les objets formels ont été présentés, nous allons introduire les deux tâches qui les manipulent : la compréhension du langage naturel et la génération du langage naturel.

Chapitre 2. Le NLU et le NLG

2.1. Les systèmes de Questions-Réponses

Les deux tâches sur lesquelles j'ai travaillé, la compréhension et la génération de langage naturel, prenaient place dans le cadre d'un projet de recherche plus grand sur les systèmes de questions-réponses. Ces systèmes sont composés d'une chaîne de traitement modulaire complexe, qui effectue d'autres tâches que ces deux-ci. Un schéma explicatif est donné ci-dessous :

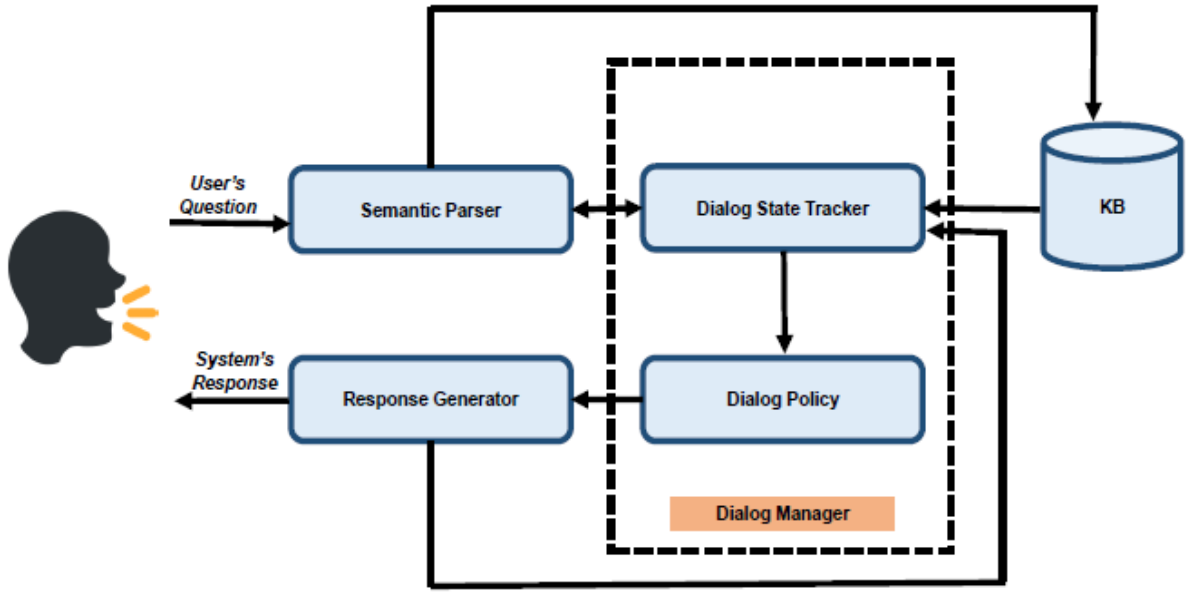

Figure 3: Schéma d'un exemple de système de questions-réponses

Dans ce système, la base de connaissance (le cylindre à droite intitulé « KB » pour *Knowledge Base*) contient des informations relatives au domaine ou à la tâche pour laquelle le système est destiné. Pour faire des requêtes dans sa base, le système utilise une représentation formelle du langage (voir chapitre précédent), différente de la représentation naturelle qui est celle de l'utilisateur de ce système. Un étape de traduction entre les deux est donc nécessaire, dans un sens comme dans l'autre. Ce sont les tâches de compréhension et de génération qui s'en occupent. Dans le schéma, les blocs « *Semantic Parser* » et « *Response Generator* » correspondent respectivement à la compréhension et à la génération. Enfin, selon le type d'acte de dialogue, la représentation formelle manipulée par le système sera différente. Ainsi les ensembles de triplets RDF seront utilisés pour les phrases affirmatives, tandis que les les requêtes SPARQL seront employées lorsque l'énoncé est une question.

2.2. La compréhension du langage naturel

Dans le cadre de mon stage, la compréhension du langage naturel (*Natural Language Understanding*, NLU) est définie comme la tâche pour un système qui consiste à passer d'un texte en langage naturel à son équivalent en langage formel, c'est-à-dire dans notre cas un ensemble de triplets RDF ou une requête SPARQL si le texte est une question. Par équivalent, il faut comprendre que le langage formel en sortie doit contenir toute l'information nécessaire à la compréhension du message de l'utilisateur afin de pouvoir lui répondre de manière pertinente. L'information nécessaire correspond au contenu sémantiquement significatif du message, sans lequel il ne pourrait être compris. Le système n'a pas besoin de conserver certains termes, comme les prépositions, les articles ou les copules par exemple, afin de comprendre le sens du message, bien que ces éléments soient indispensables aux humains pour avoir des conversations naturelles et fluides.

Par exemple, si la phrase en entrée est :

« Paris is the capital of France and has a population of 2,140,526 »

Le système doit extraire un ensemble de deux triplets, car il y a deux informations dans cette phrase :

{ (Paris , capitalOf , France) , (Paris , nbInhabitants , 2140526) }

Si la phrase est une question :

« Which city is the capital of France and has a population of 2,140,526 ? »

Le système doit construire la requête SPARQL suivante, qui contient deux contraintes dans sa partie WHERE:

```
SELECT ?c WHERE {
        ?c      capitalOf       France .
        ?c      nbInhabitants   2140526
}
```

2.3. La génération de langage naturel

La génération de langage naturel (*Natural Language Generation*, NLG) est en réalité la tâche inverse de la précédente. La génération consiste à passer d'un ensemble de triplets ou d'une requête SPARQL à son équivalent en langage naturel, c'est-à-dire un texte d'une ou plusieurs phrases. Ce texte doit contenir toutes les informations présentes dans la représentation formelle d'origine. Dans le cas d'un système de questions-réponses, c'est la tâche qui permet, une fois la réponse trouvée dans le graphe de connaissances de la transmettre de manière intelligible et naturelle à l'être humain.

Il convient cependant d'apporter ici quelques précisions supplémentaires. Une différence fondamentale entre langage formel et langage naturel est la diversité. En effet, le langage formel, triplets ou requête SPARQL, est très rigide, et ceci afin d'être interprétable et exploitable par une machine. Ainsi, contrairement au langage humain, son vocabulaire est limité et l'ambiguïté n'est pas permise. Cela a une conséquence qui a eu une importance dans mon travail, c'est qu'il peut exister plusieurs manières de verbaliser des triplets en langage naturel. Par exemple :

Le triplet : (Le_Louvre, isLocatedIn, Paris)

Peut être verbalisé (au moins) de ces trois manières :

- « Le Louvre se trouve à Paris »

- « Le Louvre est situé à Paris »

- « Paris abrite Le Louvre »

De plus, les triplets ne sont pas ordonnés au sein de leur ensemble, contrairement aux informations dans une phrase en langage naturelle, qui doivent forcément se suivre les unes les autres dans le temps de l'énonciation. C'est donc au modèle de choisir dans quel ordre les verbaliser, sachant que si tout ordre peut être grammaticalement juste et interprétable, tous ne sont pas aussi naturels et plausibles les uns que les autres.

Maintenant que les deux tâches ont été présentées, il est nécessaire de se demander comment nous pouvons évaluer les performances d'un système capable d'effectuer l'une de ces tâches, ou les deux.

Chapitre 3. Les métriques d'évaluation pour le NLU et le NLG

Afin d'évaluer la performance des modèles qu'on entraîne, il est nécessaire de leur faire passer des séries de tests lors desquelles on utilise des métriques propres à chaque tâche et qui permettent de comparer les modèles entre eux en mesurant la qualité de leurs sorties face à des références.

3.1. Les métriques de NLU

Dans le cas de la tâche de NLU, la sortie et la référence ont soit la forme d'un ensemble de triplets soit la forme d'une requête SPARQL. Pour comparer une sortie et une référence sous forme d'ensembles de triplets, les métriques sont au nombre de trois : La précision, le rappel et le F-score. Prenons un exemple de génération à partir d'un texte en entrée (texte qui n'est pas présenté car il n'est pas impliqué dans les métriques) :

Figure 4: Exemple de prédictions et références pour du NLU sur des triplets RDF

La **précision** correspond au nombre de triplets correctement prédits (c'est-à-dire présents à la fois dans les triplets prédits et les triplets de référence) sur le nombre de triplets prédits au total par le modèle. Dans l'exemple ci-dessus, le seul triplet correctement prédit est (Paris, capitalOf, France) et le modèle a généré trois triplets, donc la précision est de 1/3 ≈ 0,33.

Le **rappel** correspond au nombre de triplets correctement prédits sur le nombre de triplets dans la référence. Dans l'exemple ci-dessus, la référence contient deux triplets, le rappel est donc de 1/2 soit 0,5.

Enfin, **le F-score** est une métrique qui permet de rendre compte de la performance globale du modèle en s'appuyant à la fois sur la précision et le rappel. Cette métrique est un meilleur indicateur car il est important qu'un modèle génère le plus possible de bonnes prédictions (précision) mais également qu'il soit exhaustif par rapport à la référence (rappel). Le calcul de cette métrique correspond à une moyenne harmonique, on a donc ici :

F-Score = 2 x (précision x rappel) / (précision + rappel)

F-Score = 2 x (1/3 x 1/2) / (1/3 + 1/2) = 2/5 = 0,4.

3.2. Les métriques de NLG

A l'inverse du NLU, lorsqu'on évalue une tâche de NLG, on compare un texte en langage naturel généré par le modèle, à partir d'un ensemble de triplets ou d'une requête SPARQL, à un texte de référence pour cet ensemble/cette requête. Il existe de nombreuses métriques de NLG, qui peuvent être réparties en trois catégories : les métriques lexicales (basées sur les n-grammes), les métriques basées sur les plongements (*embeddings*) et les métriques humaines. Ne seront présentées ici que celles que j'ai été amené à utiliser.

Les métriques lexicales cherchent à retrouver des n-grammes de mots[2] présents à la fois dans la prédiction et la référence. Un n-gramme est en fait une série de *n* mots consécutifs où *n* est un entier naturel positif. Dans l'exemple de la figure 5, la prédiction et la référence ont beau ne pas être identiques, il est possible de retrouver un 4-gramme, c'est-à-dire une suite de 4 mots, « is the capital of » dans les deux.

La métrique **BLEU** (*BiLingual Evaluation Understudy*) correspond à la précision que nous avons évoquée plus haut, mais sur des n-grammes de mots donc, plutôt que des triplets.

La métrique **METEOR** (*Metric for Evaluation of Translation with Explicit ORdering*) correspond à peu près au F-score précédemment évoqué. Elle s'appuie sur le score BLEU et une autre métrique ROUGE (que je n'ai pas utilisée, mais qui correspond au rappel). Elle est pondérée en faveur du rappel :

$$METEOR = 10 * (BLEU * ROUGE) / (ROUGE + 9 * BLEU)$$

La métrique **TER** (*Translation Error Rate*) correspond au nombre de modifications (additions, substitutions ou délétions de mots) qu'il est nécessaire d'appliquer à la prédiction du modèle pour atteindre la référence. Contrairement aux autres métriques, un score plus faible indique une meilleure performance car moins de modifications sont nécessaires.

Le **ChrF** (*Character F-Score*) correspond à un F-score proche dans l'idée du METEOR, mais se calcule sur n-grammes de caractères plutôt que de mots.

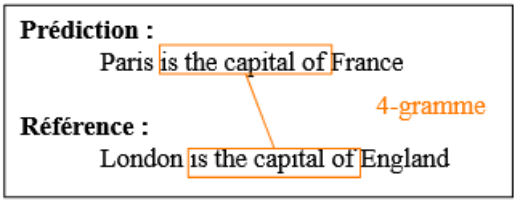

Figure 5: Exemple de 4-gramme dans une prédiction et une référence pour du NLG

2 Le mot n'a pas une définition unique, elle dépend du contexte et de l'outil qui a été utilisé pour tokeniser le texte. Ici nous considérerons pour simplifier que c'est une suite de caractères précédée et suivie d'espaces.

Les métriques basées sur les embeddings fonctionnent en s'appuyant sur les représentations numériques vectorielles des mots, appelées embeddings, plutôt que sur les mots eux-mêmes. Ces représentations sont fournies par des modèles. Les comparaisons entre embeddings sont sensées être plus fiables et plus proches du jugement humain que celles entre mots car elles prennent en compte le contexte et la proximité sémantique que peuvent avoir des mots différents.

Le **BERTScore** donne une précision, un rappel et un F-score issus du calcul de la similarité cosinus des embeddings contextuel que le modèle BERT (*Bidirectionnal Encoder Representations from Transformers*) est capable de générer.

BLEURT (*BiLingual Evaluation Understudy with Representations from Transformers*) est basée sur BERT également et lui ressemble beaucoup. Elle est sensée être meilleure car elle bénéficie d'une phase de fine-tuning du modèle BERT sur le domaine de données à évaluer avant évaluation.

Enfin, **les métriques humaines** constituent la dernière catégorie de métriques pour le NLG. Ici, la prédiction du modèle est comparée à sa référence par un être humain, qui l'évalue selon plusieurs critères, dont nous donnons ici une liste non exhaustive :

- **La grammaticalité de la phrase**, est-ce que les règles de syntaxe de la langue sont respectées ?

- **La sens de la phrase**, est-ce que la sémantique des mots fonctionne ? La phrase veut-elle dire quelque chose ?

- **L'aspect naturel**, la plausibilité, est-ce que cette phrase paraît crédible ? Aurait-elle pu être écrite ou prononcé en contexte réel par un être humain ?

- **La couverture**, est-ce que toutes les informations contenues dans la référence (et donc dans la représentation formelle de départ) sont trouvables dans la prédictions ?

- **Le bruit**, y a-t-il des informations en trop dans la phrase, qui ne sont pas dans la référence ?

- **La justesse**, est-ce que les rôles de sujet, prédicat ou objet de chaque élément dans les triplets se retrouve bien dans la phrase générée ?

On pourrait citer bien d'autre métriques humaines, comme l'ordre des informations par exemple, la liste ne peut donc pas être exhaustive.

Chapitre 4. Quelques modèles de NLU et/ou NLG préexistants

L'ensemble des modèles que je présente ici sont issus de mes lectures d'articles scientifiques et sont trouvables dans la bibliographie en fin de rapport. Je ne présente que les trois modèles avec lesquels nous avons comparé nos performances (voir plus dans la partie Résultats). Je ne rentre pas dans le détail de ces modèles mais pointe plutôt leurs apports innovants en termes de NLU et/ou NLG.

4.1. P² : Plan and Pretrain[3]

Ce modèle de NLG a la particularité d'utiliser un planificateur R-GCN (*Relational-Graph Convolutional Network*) entraîné à déterminer l'ordre le plus naturel dans lequel les informations des triplets devraient apparaître dans le texte généré. Dans la figure 6, on peut voir que les deux triplets présents dans le graphe n'ont pas d'ordre naturel. C'est le planificateur qui se charge de linéariser ce graphe avant que le modèle de type transformer T5 ne génère le texte à proprement parler. Ici, l'ordre choisi est chronologique (date de création avant date de dernière version), ce qui est assez naturel et instinctif pour un être humain.

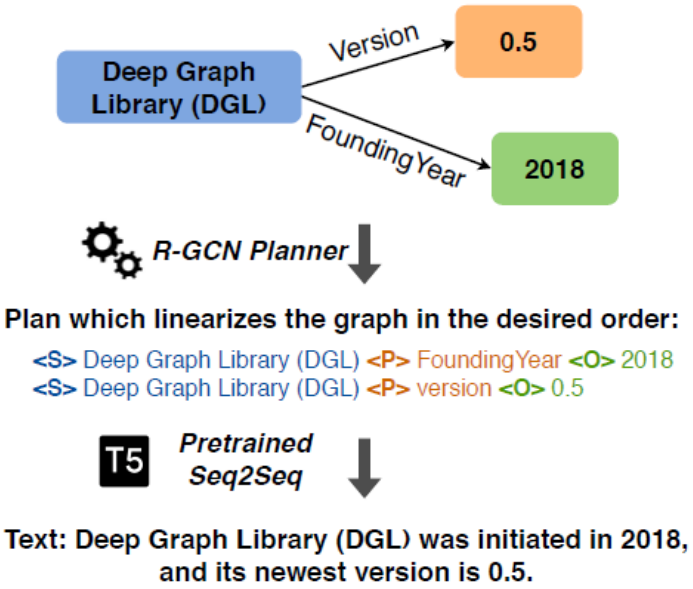

Figure 6: Schéma du modèle P²

3 Guo, Qipeng, et al. « P² : A Plan-and-Pretrain Approach for Knowledge Graph-to-Text Generation ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 100-106, https://aclanthology.org/2020.webnlg-1.10

4.2. bt5: Bilingual T5[4]

Le modèle bT5 réalise les deux tâches, NLG et NLU et ce dans deux langues, anglais et russe. Comme on peut le voir dans le schéma ci-dessous, il peut prendre en entrée des triplets et générer du texte en anglais ou en russe (NLG) ou bien prendre du texte anglais ou russe et générer des triplets (NLU). Il est important de noter que les triplets sont toujours en anglais. Sa particularité est qu'il a été entraîné en plus du NLU et du NLG à de la traduction automatique entre les deux langues sur un corpus additionnel de nouvelles d'actualité bilingues. Cette capacité à passer d'une langue à l'autre a amélioré ses performances dans les autres tâches de NLU et NLG.

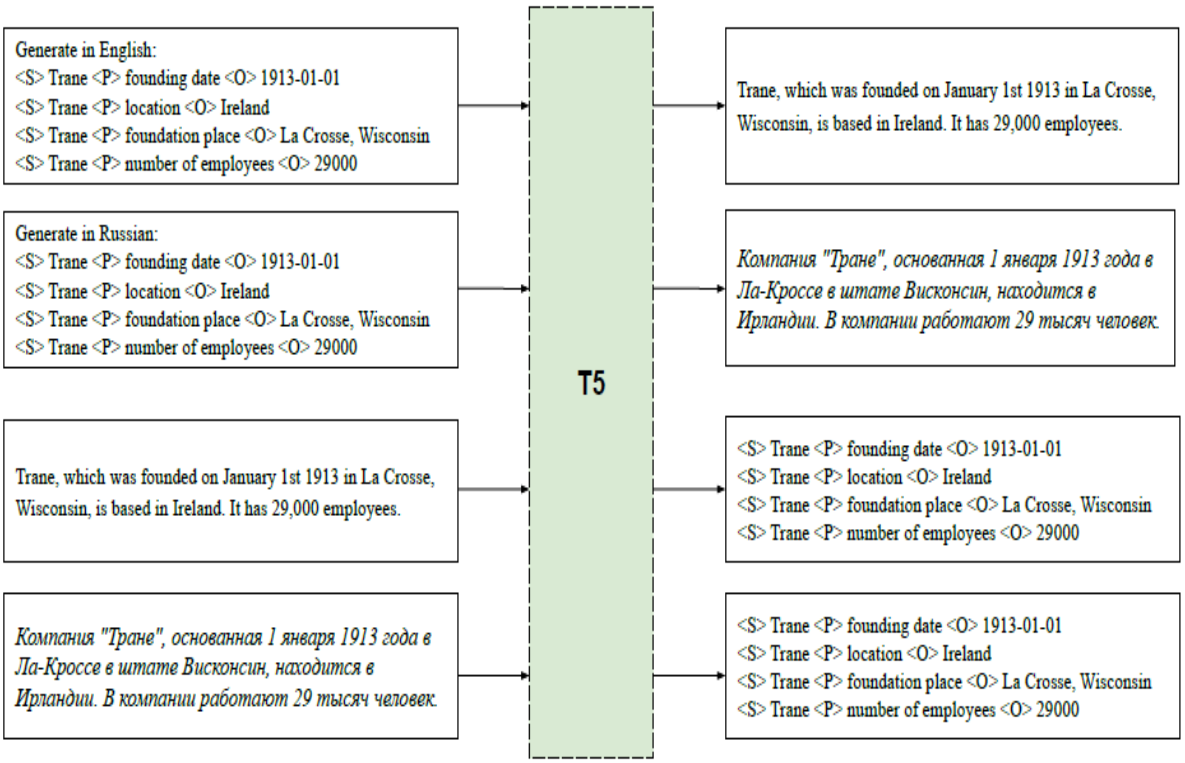

Figure 7: Schéma du modèle bT5

4 Agarwal, Oshin, et al. « Machine Translation Aided Bilingual Data-to-Text Generation and Semantic Parsing ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 125-130, https://aclanthology.org/2020.webnlg-1.13

4.3. CycleGT[5]

Ce modèle est également bi-tâche (à la fois NLU et NLG) mais a la particularité de proposer une approche non supervisée. Cela signifie que lors de son entraînement, le modèle ne disposait que des triplets pour le NLG et que du texte naturel pour le NLU. Impossible donc de comparer ses prédictions à une référence. Les deux tâches ont été entraînées de manière cyclique, c'est-à-dire que pour la tâche de NLG, le modèle commence par passer d'un texte en langage naturel vers des triplets (NLU), puis, à partir de ces mêmes triplets, cherche à revenir au texte d'origine (NLG). C'est entre ce nouveau texte généré et le texte d'origine que la fonction de coût est calculée. Le NLU est entraîné de la même manière mais en commençant par les triplets. Comme nous le verrons dans les résultats, les scores de ce modèle sont plus bas que ceux des autres modèles étant donné la difficulté supplémentaire que se sont rajoutés ses concepteurs. Cependant, cette approche a le mérite de tenter de répondre à une problématique très concrète aujourd'hui : dans le monde réel, les données ne sont souvent pas aussi bien préparées et complètes que lors d'un challenge.

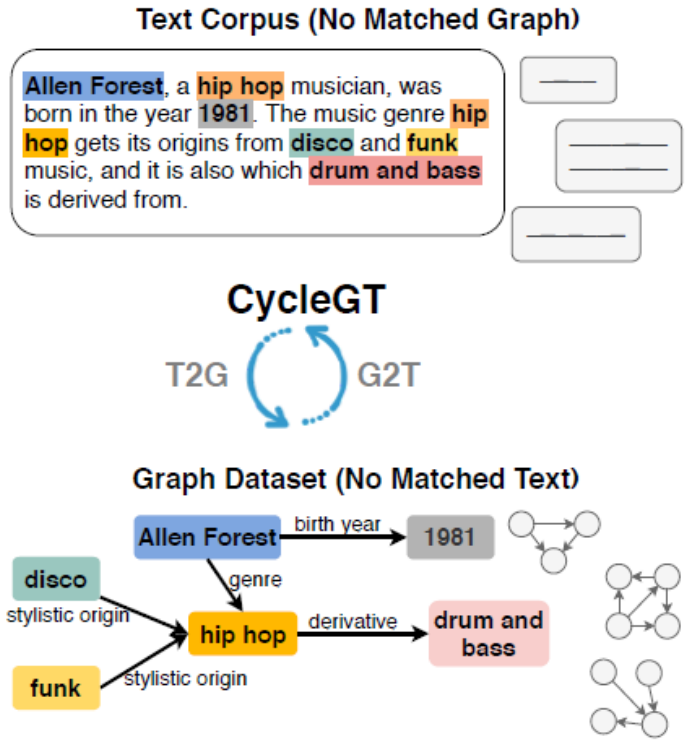

Figure 8: Schéma du modèle CycleGT

5 Guo, Qipeng, et al. « CycleGT : Unsupervised Graph-to-Text and Text-to-Graph Generation via Cycle Training ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 77-88, https://aclanthology.org/2020.webnlg-1.8

Maintenant que les représentations formelles fondées sur les graphes de connaissances (RDF et SPARQL) ont été présentées, ainsi que les deux tâches qui les exploitent (NLU et NLG) et les métriques utilisées pour évaluer la performance des modèles qui les effectuent, la partie suivante détaillera comment mon travail s'est organisé autour de celles-ci, à travers l'entraînement et l'évaluation de modèles.

Partie 2

-

Mes travaux et expériences

Chapitre 5. Lectures et Auto-Formation

Mon premier mois de stage a été consacré à de la lecture d'articles scientifiques sur l'état de l'art en matière de NLU et NLG, puis à de l'autoformation à quelques outils que j'allais être amené à utiliser.

5.1. Lectures d'articles

Toutes les références des articles que j'ai lus sont trouvables dans la bibliographie en fin de rapport. Ces lectures m'ont permis d'une part de compléter mes connaissances généralistes sur les concepts clés de mon stage, tels que la génération et la compréhension de langage naturel, les systèmes de questions réponses, les graphes de connaissances, etc. non seulement en termes de définition mais aussi d'historique et d'état de l'art dans ces domaines. Elles m'ont également permis de me familiariser avec quelques modèles, plus ou moins semblables à ce que je devais chercher à mettre en place, comme les trois modèles que je viens de présenter dans la partie précédente par exemple. Mais aussi avec des challenges, comme WebNLG, des corpus (E2E, LCQuAD, et beaucoup d'autres). Au delà de ça, ces lectures m'ont appris certains aspects récurrents dans les articles et dans la méthode de recherche scientifique en général, comme la définition d'un contexte et la mise en perspective du sujet par rapport aux travaux des pairs et l'apport qu'il peut représenter, la présentation du type de modèle et des corpus utilisés, les méthodes d'évaluations, les métriques et les manières de présenter les résultats, etc.

5.2. Auto-formation

Étant donné que je manquais de pratique et que j'allais être amené à utiliser des outils que je ne connaissais pas, j'ai dû me former à PyTorch d'une part, une bibliothèque Python qui permet de faire de l'apprentissage neuronal et aux bibliothèques `transformers` et `datasets` d'HuggingFace (HF) d'autre part, qui permettent de charger, de préparer et d'utiliser facilement des modèles, des métriques et des corpus depuis une plateforme en ligne. Le tutoriel que j'ai suivi et qui a servi de base au code de mon script d'entraînement et d'évaluation peut être trouvé à l'adresse suivante : https://huggingface.co/course/chapter0/1?fw=pt

Chapitre 6. Environnement et outils de travail, Modèles et Corpus

Je décris ici succinctement mon environnement de travail et les outils de développement que j'ai utilisés, les deux modèles de type transformer BART et T5 et les deux corpus WebNLG et LCQuAD.

6.1 Environnement et outils de travail

C'est à partir de l'ordinateur qu'Orange avait mis à ma disposition dans mon bureau que je travaillais en SSH (*Secure Shell*) sur l'une des deux machines équipées de cartes GPU (*Graphics Processing Unit*) de l'équipe NADIA. Ces GPU permettent de faire des calculs matriciels, ce qui est nécessaire quand on entraîne des modèles d'une certaine taille sur des volumes de données importants. Je travaillais soit directement en terminal de commande via Putty et l'éditeur de texte Vim, soit dans des *notebooks* via un serveur Jupyter.

Tout mon code se trouve dans un unique script Python, capable de réaliser à la fois l'entraînement et l'évaluation des modèles. Les modèles une fois entraînés sont stockés dans un répertoire « `models` » et les résultats de leurs évaluations dans un répertoire « `results` ». J'ai également créé un fichier « `requirements.txt` » qui permet de connaître la liste de toutes les bibliothèques que j'avais installées dans mon environnement virtuel Conda pour pouvoir le reproduire facilement et éviter les soucis de compatibilité.

En termes de bibliothèques Python, j'ai utilisés les suivantes :

- ***argparse***, afin de pouvoir donner des paramètres au script via la ligne de commande (comme le nom du modèle à utiliser, les tâches à entraîner/évaluer, les corpus à utiliser, …)

- ***datasets***, pour charger des corpus et des métriques depuis la plateforme d'HF.

- ***transformers***, pour charger des modèles et des tokeniseurs de la plateforme d'HF.

- ***torch***, pour les classes `Dataset` et `DataLoader`

- ***accelerator***, pour tirer parti d'avoir plusieurs cœurs GPU à disposition

- ***json***, pour lire le corpus LCQuAD, que j'avais au format JSON en local

- ***nltk***, pour tokeniser les prédictions et références, nécessaire pour certaines métriques

- ***numpy*** pour calculer facilement des moyennes (scores de certaines métriques)

6.2. Modèles : BART et T5

Toutes les expériences d'entraînement et d'évaluation sur les différentes tâches et corpus ont été réalisées avec deux modèles pré-entraînés de type transformer différents, BART et T5, connus pour être parmi les meilleurs de l'état de l'art à l'heure où ce rapport est rédigé.

Avant de présenter ces deux modèles, il est nécessaire de définir quelques termes :

Un **modèle Seq2Seq** (séquence à séquence) prend en entrée une séquence de mots (une ou plusieurs phrases) et génère une autre séquence de mots en sortie. Il le fait à l'aide d'un réseau de neurones dit récurrent qui traduit chaque mot un par un en prenant en compte les mots précédents déjà traduits.

L'**architecture encodeur-décodeur** des transformers consiste en une phase de conversion des mots d'entrées vers une représentation numérique vectorielle appelée embedding (l'encodage) et une seconde phase qui génère les mots de sortie à partir de l'embedding précédemment produit et du contexte du mot actuel dans la phrase, comme représenté dans le schéma de la figure 9[6].

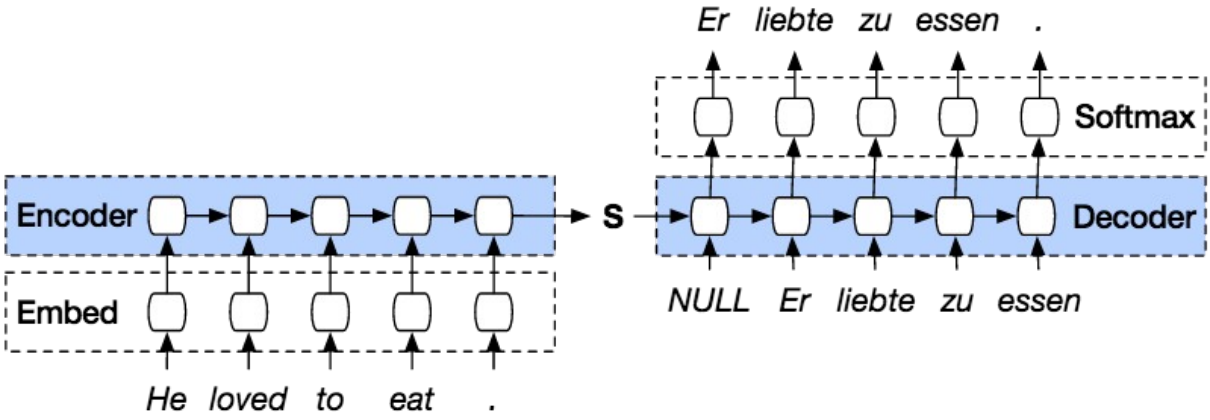

Figure 9: Schéma de l'architecture encodeur-décodeur d'un modèle seq2seq pour un tâche de traduction automatique de l'anglais vers l'allemand

6 Source : https://medium.com/@pierre_guillou/nlp-fastai-sequence-to-sequence-model-seq2seq-38d9984cf271

Utiliser des **modèles pré-entraînés** est très utile car en plus de réduire le temps et les données nécessaires à un entraînement à partir de zéro, ils seront certainement capables de transférer leurs compétences à un nouveau domaine si tant est qu'ils ont été entraînés dans des domaines variés et/ou pas trop différents du nouveau.

Le **fine-tuning** consiste à poursuivre l'entraînement d'un modèle pré-entraîné sur une nouvelle tâche pour adapter ces paramètres plus précisément à celle-ci. C'est ce que j'ai fait lors de mon stage avec les deux modèles pré-entraînés, BART et T5.

BART (*Bidirectional and Auto-Regressive Transformer*) a une architecture classique de traduction séquence à séquence (seq2seq) avec un encodeur bidirectionnel et un décodeur de gauche à droite. Il est pré-entraîné sur une tâche de remplissage de texte à trous dont les phrases ont été mélangées au préalable. Il est connu pour être très bon en génération de langage naturel.

T5 (*Text-To-Text Transfer Transformer*) considère toute tâche linguistique comme une opération texte-à-texte. C'est un modèle encodeur-décodeur, comme BART, qui a été pré-entraîné sur plusieurs tâches à la fois. Il a la particularité de nécessiter l'ajout d'un préfixe devant ses données d'entrée pour savoir quelle tâche il doit exécuter dessus. Par exemple « translate English to German » pour une tâche de traduction de l'anglais vers l'allemand.

6.3. Corpus : WebNLG et LCQuAD

WebNLG[7] est le nom du corpus associé au challenge éponyme. Ce challenge a eu lieu à deux reprises, en 2017 et 2020. La version de 2020 apportait la tâche de compréhension (NLU) en plus de celle de génération (NLG) et une deuxième langue, le russe. Le corpus est composé d'environ 20 000 échantillons fractionnés en 3 dans les proportions suivantes : 70 % pour l'entraînement (fraction nommée « `train` »), 10 % pour l'évaluation durant l'entraînement (fraction nommée « `dev` ») et 20 % pour l'évaluation après entraînement (fraction nommée « `test` »). Chaque échantillon contient un ensemble de 1 à 7 triplets RDF, associé à une ou plusieurs verbalisations en langage naturel, comme on peut l'observer dans l'exemple donné par la figure 9, constitué d'un ensemble de 2 triplets associé à deux références :

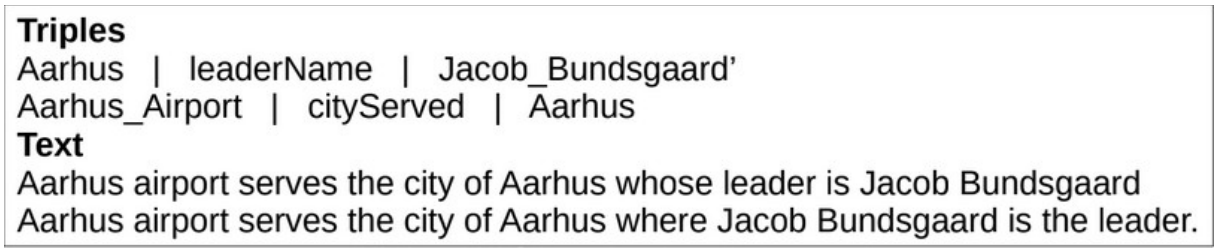

Figure 10: Exemple d'échantillon issu du jeu de données WebNLG

7 Castro Ferreira, Thiago, et al. « The 2020 Bilingual, Bi-Directional WebNLG+ Shared Task: Overview and Evaluation Results (WebNLG+ 2020) ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 55–76, https://aclanthology.org/2020.webnlg-1.7.

LCQuAD[8] (*Largescale Complex Question Answering Dataset*) est un corpus initialement proposé pour un tâche de questions-réponses. Nous utilisons la version 2.0 du corpus, qui contient environ 30 000 échantillons, répartis entre fractions d'entraînement et de tests de la même manière que WebNLG (70 %/10 %/20%). Chaque échantillon est constitué d'une question en langage naturel, avec une version paraphrasée dans certains cas, associée à son équivalent sous la forme d'une requête SPARQL au format attendu par les graphes de connaissances Wikidata ou DBPedia, où les réponses peuvent être trouvées, bien que certaines questions soient volontairement dépourvues de réponses. Voici un exemple d'échantillon :

Requête

```
SELECT DISTINCT ?obj WHERE {
        The-Jungle-Book translator ?obj .
        ?obj instance-of human
}
```

Question

Who is the translator of The Jungle Book ?

Paraphrase de la question

Who has translated The Jungle Book ?

8 Dubey et al. « LC-QuAD 2.0 : A Large Dataset for Complex Question Answering over Wikidata and DBPedia, in Proceedings of the 18th International Semantic Web Conference (ISWC), Springler, 2019, http://jens-lehmann.org/files/2019/iswc_lcquad2.pdf

Chapitre 7. Expériences et problèmes rencontrés

Je décris ici les différentes expériences d'entraînement de modèles que j'ai réalisées au cours de mon stage et l'approche choisie dans la conception du code python. Le choix de mes deux tuteurs de stage était de commencer par entraîner des versions dites « simples » des modèles, c'est-à-dire n'effectuant qu'une seule tâche, sur un seul corpus, afin de servir de point de comparaison avec les modèles plus complexes et ainsi déterminer si ces derniers perdaient en performances ou pas.

7.1. Expériences de base : tâche simple sur corpus unique

Huit modèles simples ont été entraînés. Chaque modèle de base pré-entraîné (BART et T5) a été décliné en quatre versions : Deux sur WebNLG et deux sur LCQuAD, une pour le NLG et une pour le NLU dans les deux cas. Le procédé du script d'entraînement est le suivant :

1) Tout d'abord les arguments de la ligne de commande sont récupérés. Dans l'exemple ci-dessous, le paramètre « `--model` » indique que l'on souhaite utiliser le modèle T5 de base, les paramètres « `--train` » et « `--dev` » indiquent respectivement que l'on veut faire un entraînement de ce modèle et l'évaluer sur la fraction « `dev` » du corpus pendant son entraînement, pour suivre l'évolution de ses progrès. Enfin, le paramètre «`-- webnlg nlg`» indique que l'on souhaite réaliser l'entraînement pour une tâche de NLG sur le corpus WebNLG.

```
python script.py --model t5-base --train --dev --webnlg nlg
```

En suivant cette logique, il est possible de remplacer « nlu » par « nlg », « webnlg » par « lcquad » ou encore « t5-base » par « facebook/bart-base » pour obtenir l'ensemble des huit combinaisons qui nous intéressent.

2) Dans un second temps, le corpus chargé passe par une étape de prétraitement nécessaire car il contient des champs de données et/ou de métadonnées dont nous n'avons pas besoin. Dans le cas de WebNLG, on récupère seulement les ensembles de triplets d'une part et leurs équivalents sous forme de textes en langage naturel d'autre part. Dans le cas de LCQuAD, on récupère seulement la requête SPARQL, la question en langage naturel et si

possible la question paraphrasée. C'est également ici qu'on se charge de rajouter le préfixe de T5 devant les entrées, triplets/requête ou langage naturel, selon la tâche. Nous avons choisi « Convert X to text : » pour le NLG et « Convert text to X » pour le NLU, où X correspond au langage formel du corpus en question (RDF pour WebNLG et SPARQL pour LCQuAD). Ce choix a été motivé par le fait que T5 a déjà vu des préfixes du type « Convert X to Y » lors de son pré-entraînement.

3) Ensuite, il est nécessaire de tokeniser et d'encoder les paires triplets-langage naturel de chaque échantillon pour pouvoir réaliser l'entraînement à proprement parler. En effet, le tokeniseur renvoie un dictionnaire avec un champ « `input_ids` » qui contient la représentation numérique des tokens, attendue par l'objet `DataLoader` qui se charge de faire des paquets d'échantillons. Dans la boucle d'entraînement, pour chacun de ces batchs, le modèle génère des prédictions à partir de l'entrée encodée (les triplets ou la requête SPARQL dans le cas du NLG, le texte en langage naturel dans le cas du NLU), puis calcule l'écart (la fonction de coût ou *loss*) entre ce qu'il a généré et la référence encodée (texte en langage naturel pour le NLG, triplets ou requête pour le NLU) et met à jour ses paramètres afin de réduire cet écart.

4) Une fois l'entraînement terminé, le modèle est sauvegardé dans un répertoire local sous un nom qui permet de se souvenir des tâches et des corpus sur lesquels il a été entraîné.

Exemple : `bart_finetuned-on_webnlg-nlg`

5) Après avoir entraîné un modèle, on peut passer à son évaluation. Elle se fait sur chacune des tâches pour lesquelles le modèle a été entraîné successivement. Ici, il n'est pas nécessaire de tokeniser les références car il n'y a pas de fonction de coût à calculer. Le modèle génère ses prédictions, puis celles-ci sont décodées et comparées aux références à l'aide des métriques que j'ai présentées plus haut : BLEU, METEOR, TER, CHRF et BERTSCORE pour le NLG (et le NLU sur LCQuAD, faute de métrique adaptée pour comparer des requêtes SPARQL), et avec la précision, le rappel et le F-score à l'aide du script d'évaluation officiel du challenge WebNLG pour le NLU sur WebNLG.

6) Tous les résultats sont écrits dans des fichiers dans un répertoire « `results` » dont les noms permettent de retrouver rapidement quel modèle a été évalué et sur quelle tâche.

Exemple : `bart_finetuned-on_webnlg-nlg-nlu_evaluated-on_webnlg-nlu`

Les résultats sont présentés dans la partie 3.

7.2. Expériences complexes : tâches conjointes sur un ou plusieurs corpus

Une fois ces modèles simples entraînés, je suis passé sur les modèles plus complexes, qui constituaient l'objectif de ce stage. Leur entraînement est très similaire à celui des modèles simples, à l'exception de quelques différences que je note ici :

La librairie `argparse` permet de passer plusieurs arguments de corpus et/ou plusieurs tâches pour chaque corpus. Un exemple de ligne de commande pour l'entraînement d'un modèle capable de faire toutes les tâches sur tous les corpus :

```
python script.py --model t5-base --train --webnlg nlg nlu --lcquad nlg nlu
```

Cela permet d'entraîner les 6 combinaisons tâches-corpus restantes : Pour chaque type de modèle (BART et T5) 2 modèles qui font les deux tâches sur un seul corpus et 1 modèle qui fait les deux tâches sur les deux corpus.

J'ai créé une classe `UnionDataset` qui se charge de simuler le fait qu'on aurait un seul grand corpus composé des corpus associés à chaque tâche mis bout à bout. Ainsi, si j'entraîne un modèle sur toutes les tâches et tous les corpus, le programme commence par créer les quatre corpus possibles (WebNLG-nlg, WebNLG-nlu, LCQuAD-nlg et LCQuAD-nlu). La différence entre les corpus de NLG et les corpus de NLU repose sur le fait que les entrées et références sont inversées. C'est lors de la création de l'objet `DataLoader` que tous les échantillons du corpus sont mélangés et ainsi le modèle voit dans chaque batch d'entraînement des échantillons qui peuvent être issus de n'importe quelle combinaison tâche-corpus.

Lors de leur évaluation, les modèles complexes sont évalués successivement sur toutes les combinaisons tâche-corpus qu'ils ont vues lors de leur entraînement. Ainsi, un modèle ayant vu toutes les combinaisons données dans le paragraphe précédent, passera quatre évaluations successives, dont les résultats seront écrits dans quatre fichiers différents.

Il est important d'évoquer ici un erratum concernant l'évaluation du NLU sur WebNLG. Mes tuteurs ont constaté après la fin de mon stage que le script officiel du challenge que j'ai utilisé ne calcule la précision, le rappel et le F-score que sur les éléments des triplets individuellement et non sur les triplets entiers, selon un système de quatre niveaux de tolérance que je détaille dans la partie concernant cette évaluation (chapitre 8.2.)

7.3. Problèmes rencontrés

Lors du développement de mon script d'entraînement, j'ai rencontré plusieurs problèmes, dus principalement au fait que je suis parti du code issu du tutoriel HuggingFace pour une tâche de traduction automatique (ce qui se rapprochait le plus du NLG et du NLU) et que je n'avais pas conçu le code dès le départ pour gérer les cas d'entraînements sur plusieurs tâches et plusieurs corpus à la fois.

Le premier problème majeur a été causé par l'objet *Trainer* d'HuggingFace, utilisé dans leur tutoriel. Cet objet rajoute une couche d'abstraction, sensée faciliter la mise en place de l'entraînement et de l'évaluation d'un modèle en rassemblant dans les paramètres de l'objet tous les éléments de l'entraînement et de l'évaluation à la fois. Cependant, il est difficile de savoir exactement ce que cet objet fait en arrière plan et de modifier son comportement sans avoir au préalable compris toute sa documentation. À cause de ce manque de transparence et de souplesse, j'ai préféré écrire moi-même mes boucles d'entraînement et d'évaluation en PyTorch « pur », ce qui a également eu comme effet bénéfique de me permettre de mieux comprendre et appréhender les différentes étapes et de choisir plus finement le traitement de mes données.

Un autre problème rencontré a été dû à l'utilisation de certaines métriques (ChrF, TER et METEOR) lors de l'évaluation. En effet, ces métriques étaient récupérées sur la plateforme en ligne d'HuggingFace et il semble que leur concepteurs n'avaient pas prévu le cas où, lors d'une évaluation de NLG, il puisse y avoir un nombre variable de textes de références selon les échantillons. J'ai donc créé une fonction autour de ces métriques qui les calcule pour chaque échantillon les uns après les autres et fait une moyenne ensuite, plutôt que sur tous les échantillons d'un coup. De plus, la métrique BLEU ne gérait pas elle même la tokenisation, d'où l'utilisation de la librairie `nltk` à cette fin. Enfin, nous avons écarté la métrique BLEURT qui prenait trop de temps à calculer et dont les scores étaient négatifs car nous ne disposions pas d'un modèle BERT fine-tuné pour nos tâches.

Maintenant que mon environnement de travail et les techniques d'entraînement et d'évaluations des divers modèles sur lesquels j'ai été amené à travailler ont été présentés, la partie suivante consistera en la présentation des résultats de ces expériences sous formes de graphiques, ainsi qu'en l'analyse et le commentaire des performances des modèles dans une perspective comparatiste.

Partie 3

-

Évaluations & Résultats

Chapitre 8. WebNLG

8.1. NLG sur WebNLG

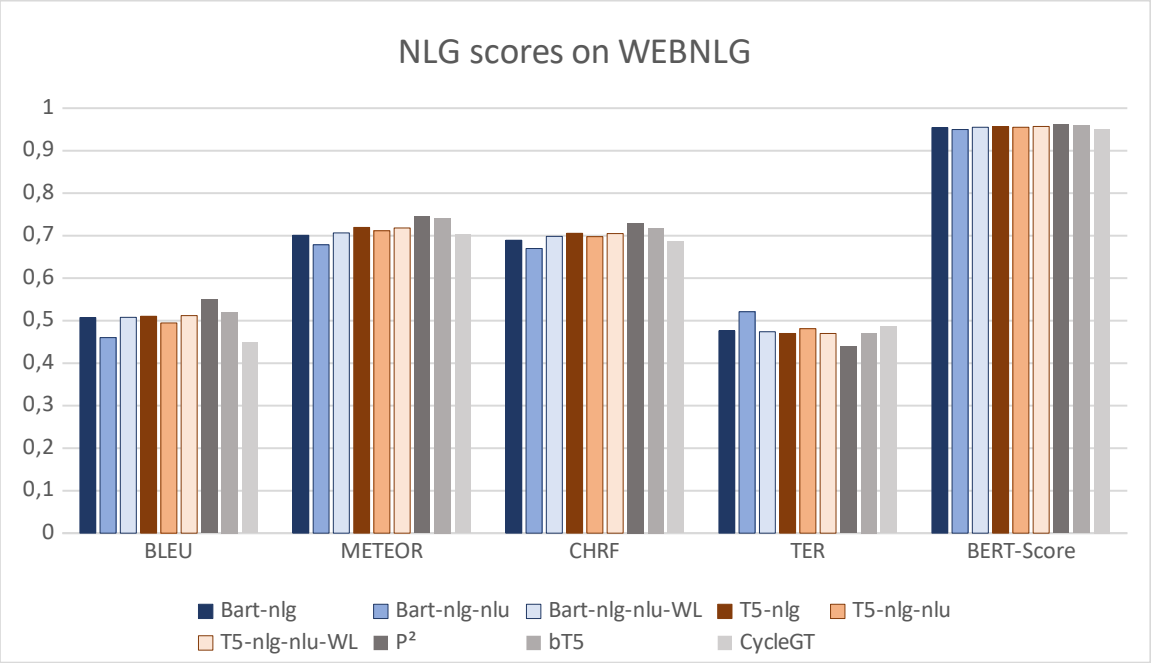

Figure 11: Tableau des scores globaux pour le NLG sur WebNLG

Chacun des 6 modèles ayant été entraîné à faire du NLG sur WebNLG a été évalué avec les 5 métriques de NLG que j'ai présentées plus haut : BLEU, METEOR, CHRF, TER et BERT-score. Ces résultats sont présentés dans la figure 10. Les 3 modèles de type BART sont en bleu et les 3 modèles de type T5 en orange. La nuance de la couleur indique si le modèle a été entraîné soit sur la tâche simple (Bart-nlg, T5-nlg) soit sur les deux tâches jointes (Bart-nlg-nlu, T5-nlg-nlu) soit sur les deux tâches jointes et ce sur les deux corpus (Bart-nlg-nlu-WL, T5-nlg-nlu-WL). WL, pour Webnlg-Lcquad, signifie que les deux corpus ont été mélangés. Ce système de couleur est valable pour tous les graphes suivants. Les 3 colonnes grises correspondent aux 3 participants du challenge WebNLG présentés plus haut (P², bT5 et CycleGT) dont nous avons récupérés les sorties sur le dépôt GitHub du challenge et que nous avons évalués comme nos modèles à des fins de comparaison.

On observe tout d'abord que l'allure de chaque groupe de colonnes est similaire (attention, pour TER un score bas indique une bonne performance). D'autre part, il apparaît également que les modèles double-tâche sur corpus unique sont légèrement moins bons que les modèles simples et les modèles sur corpus joints. Enfin, on peut constater que les modèles double-tâche sur deux corpus ne perdent pas en performance et sont à peu près aussi bons que les participants du challenge.

8.2. NLU sur WebNLG

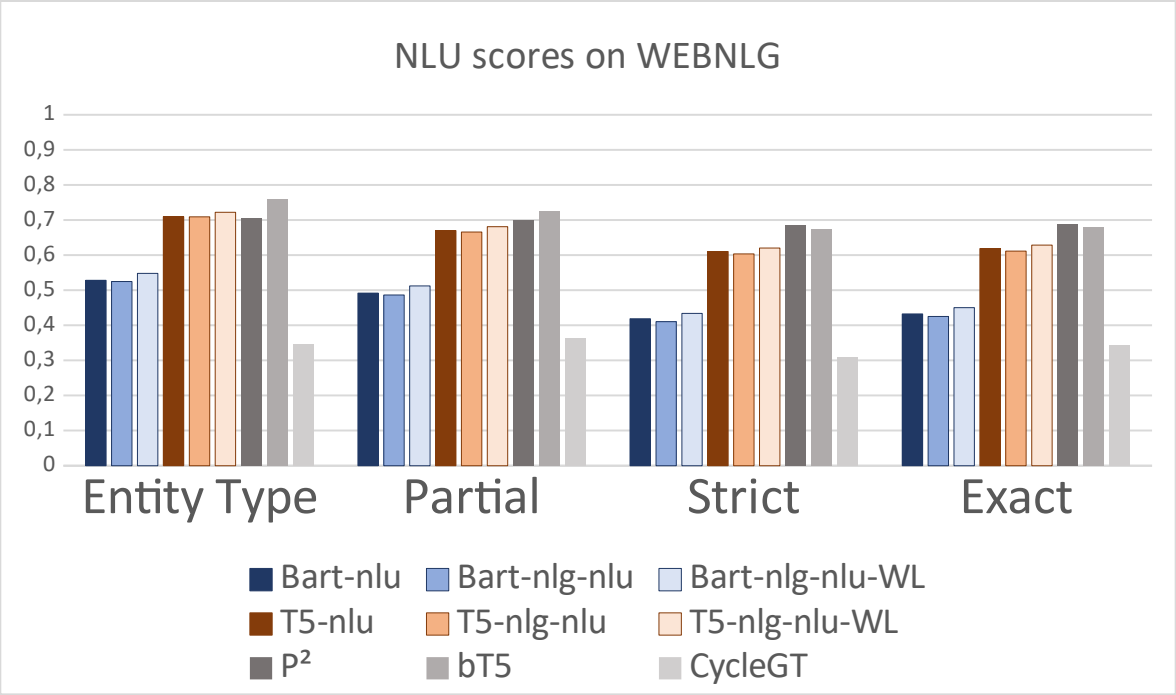

Figure 12: Tableau des scores globaux pour le NLU sur WebNLG

L'évaluation du NLU sur WebNLG a été faite à l'aide du script d'évaluation officiel du challenge. Ce script renvoie des scores de précision, rappel et F-score selon quatre niveaux de tolérance. Chaque niveau de tolérance définit sous quelles conditions un élément d'un triplet prédit est considéré comme correspondant à un élément d'un triplet de la référence :

- **Entity type** signifie que l'élément du triplet correspond en partie et que sa position correspond, par exemple :

Référence : (S Capers , … , …) Prédiction : (S Super Capers , … , …)

- **Partial** signifie que l'élément du triplet correspond en partie mais pas forcément sa position, par exemple :

Référence : (S Capers, … , …) Prédiction : (… , … , O Super Capers)

- **Strict** signifie que l'élément du triplet correspond mais pas forcément sa position, par exemple :

Référence : (S Capers , … , …) Prédiction (… , … , O Capers)

- **Exact** signifie que l'élément du triplet correspond, ainsi que sa position, par exemple :

Référence : (S Capers , … , …) Prédiction (S Capers , … , …)

Bien que très semblables aux résultats du tableau des scores pour le NLG, on peut observer ici que les modèles de type T5 sont nettement supérieurs aux modèles de type BART.

8.3. Évaluation filtrée par nombre de triplets pour le NLG sur WebNLG

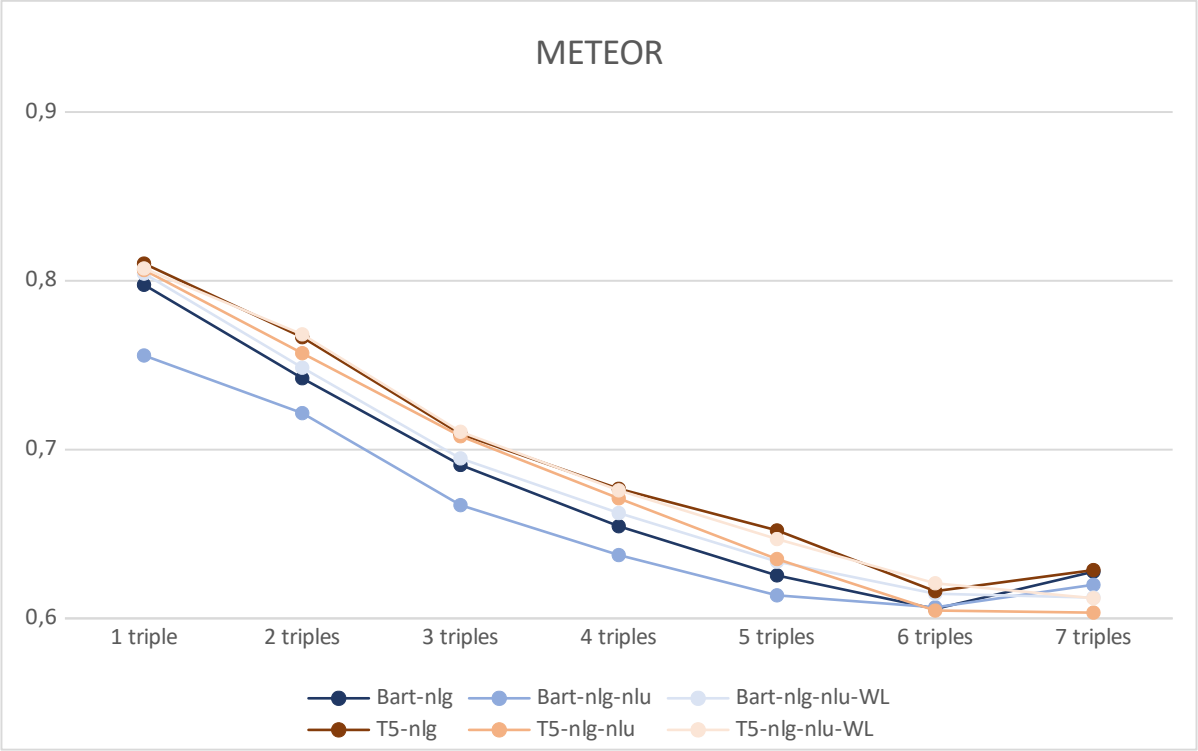

Figure 13: Scores METEOR par nombre de triplets pour le NLG sur WebNLG

Afin de pousser l'analyse des résultats un peu plus loin, nous avons décidé de mener une deuxième série d'évaluation, cette fois-ci en filtrant les échantillons du corpus selon le nombre de triplets des échantillons. Dans WebNLG, le nombre de triplets maximal dans un ensemble étant de 7, il y a donc eu 7 évaluations filtrées à mener.

La question que nous nous posions était de savoir si le nombre de triplets avait un impact sur la performance, et si oui, dans quelle mesure. Notre hypothèse était que oui, la courbe devrait descendre de manière régulière (comme une fonction affine). Pour le NLG, seule la courbe de la métrique METEOR est présentée par la figure 12, les autres courbes étant très similaires et cette métrique étant l'une des meilleures en termes de corrélation avec le jugement humain pour le NLG. Il faut faire attention en observant ces courbes, car l'axe des ordonnées a été zoomé pour pouvoir mieux discerner les courbes de chaque modèle les unes des autres. Il y a donc un biais qui tend à grandir l'écart de score entre les modèles. Cependant, ce qui nous intéresse dans cette évaluation est surtout l'aspect de la pente des courbes.

Nous pouvons donc observer que notre hypothèse semble vérifiée. Les modèles perdent en performance de manière régulière avec l'augmentation du nombre de triplets. La perte est d'environ 0,2 points entre 1 et 6 triplets. Il est intéressant de noter, enfin, que les performances semblent se stabiliser au-delà de 6 triplets, voire de remonter. En poursuite de ce travail, il pourrait être judicieux de vérifier si cette tendance se poursuit au-delà de 7 triplets ou non et trouver pour quelle raison si tel est le cas.

8.4. Évaluation filtrée par nombre de triplets pour le NLU sur WebNLG

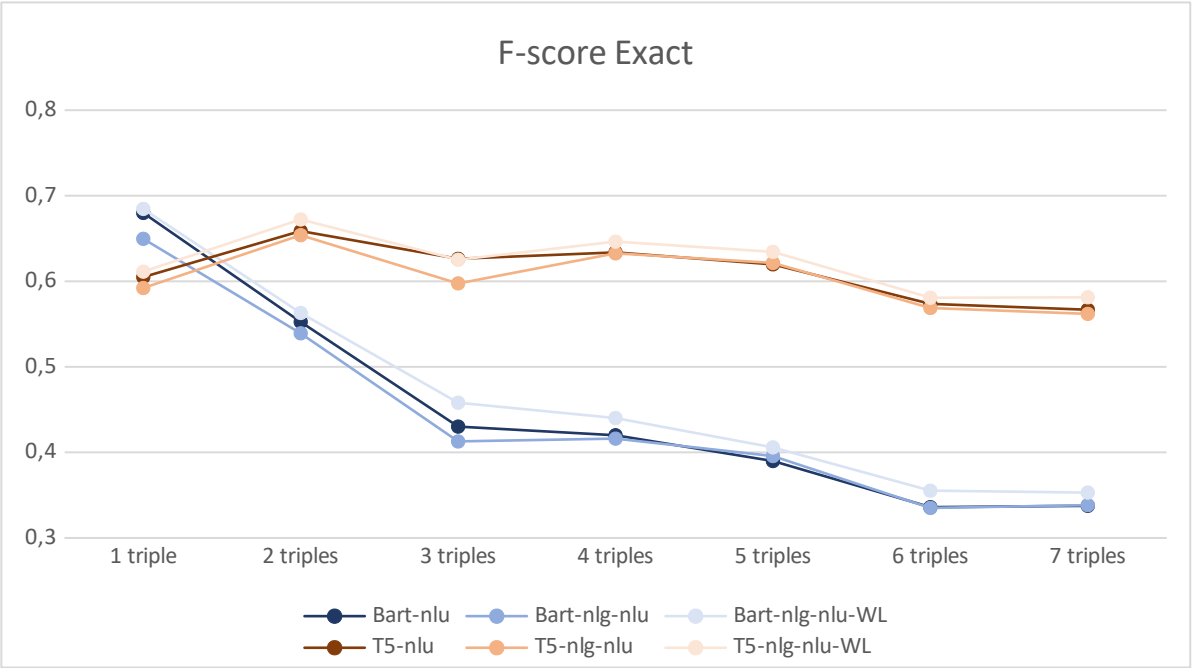

Figure 14: F-Scores par nombre de triplets pour le NLU sur WebNLG

De même, cette série de 7 évaluations filtrées a été menée pour la tâche de NLU. On ne présente ici que la mesure de F-score Exact, étant donnée que les autres niveaux de tolérance sont juste des versions plus relaxées de celle-ci est que les courbes ont la même allure, entre des scores plus élevés.

Contrairement à l'évaluation du NLG sur WebNLG, les deux modèles semblent présenter des comportements différents. De plus, certaines particularités dignes de retenir notre attention sont à noter :

1) Les modèles de type BART sont meilleurs que les modèles de type T5 uniquement sur les échantillons d'un seul triplet.

2) Comme vu avec les scores globaux, la différence de performances est plus marquée en NLU qu'en NLG entre les deux types de modèles. Cependant, ces courbes nous apprennent de plus que la perte de performances avec l'augmentation du nombre de triplets est aussi plus importante chez BART. T5 présente une courbe trop chaotique pour parler de perte, alors que celle de BART diminue clairement de presque 0,4 points.

3) On remarque encore qu'à l'instar du NLG, la pente semble s'adoucir avec l'augmentation du nombre de triplets, en tout cas pour les modèles de type BART.

4) Enfin, en plus d'être presque plate, la pente des modèles de type T5 présente une allure plus dentelée qui surprend, comme si le nombre de triplets étaient moins corrélé à la performance chez ce type de modèles.

Chapitre 9. LCQuAD

9.1. NLG sur LCQuAD

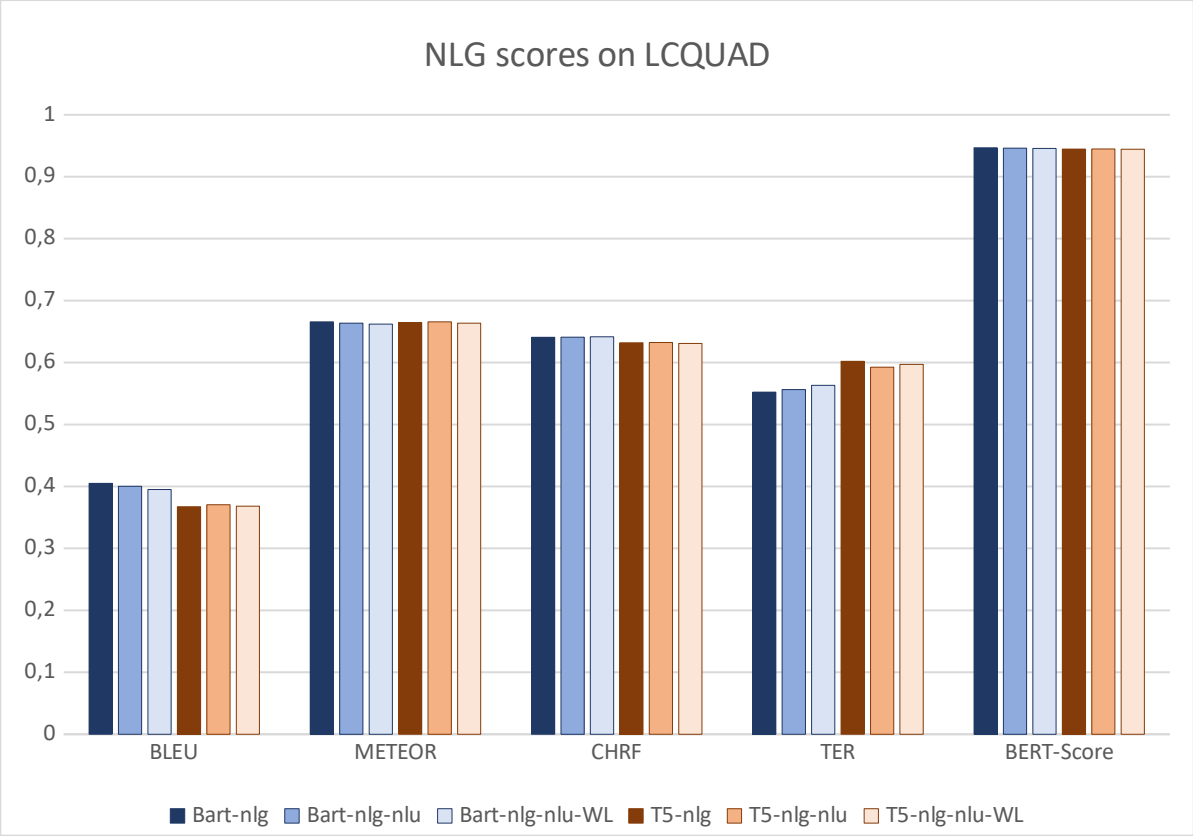

Figure 15: Tableau des scores globaux pour le NLG sur LCQuAD

L'évaluation sur LCQuAD s'est déroulé sensiblement de la même manière que sur WebNLG, à quelques différences près, que j'évoquerai plus loin. Dans le cas du NLG, on peut donc retrouver les 5 mêmes métriques dans le tableau ci-dessus. Les courbes grises sont absentes ici car ils n'ont pas été évalués sur le corpus LCQuAD lors du challenge WebNLG.

La première chose que l'ont peut noter, c'est que les scores sont ici plus resserrés, à la fois entre variantes d'un même modèle et entre les deux types de modèles.

La deuxième chose à remarquer est que l'on n'observe pas ici de perte de performances des modèles double-tâche sur corpus unique par rapport aux deux autres variantes, que l'on avait observée dans le cas du NLG sur WebNLG.

Enfin, on peut constater qu'ici, contrairement au NLG sur WebNLG, ce sont les modèles de type BART qui paraissent légèrement meilleurs.

9.2. NLU sur LCQuAD

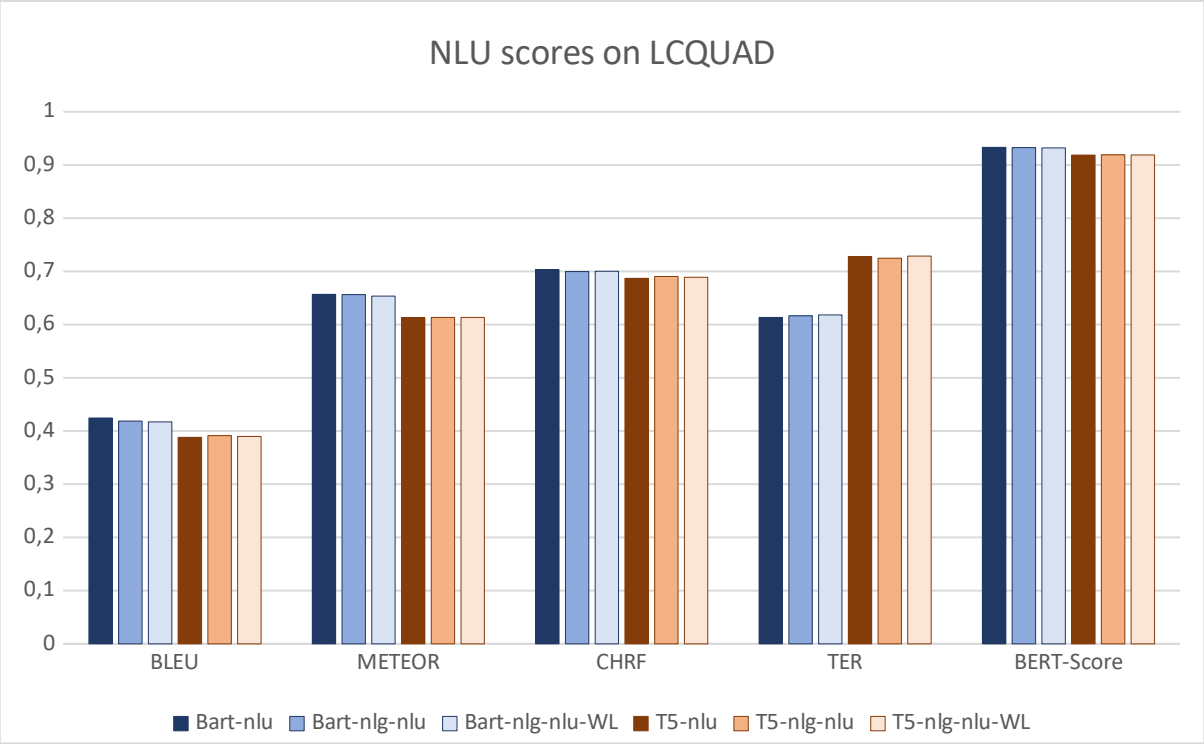

Figure 16: Tableau des scores globaux pour le NLU sur LCQuAD

Par soucis de complétude, l'évaluation a été menée également sur la tâche de NLU pour LCQuAD. Cependant, il est important ici de pointer du doigt le fait que nous n'avions pas de métriques d'évaluation pour cette tâche, n'ayant pas d'équivalent dans la littérature au script du challenge WebNLG pour les requêtes SPARQL. Nous avons donc fait le choix d'utiliser les métriques de NLG qui peuvent donner un vague aperçu de la qualité de la requête générée, en la considérant comme une chaîne de caractères ou tokens en langage naturel. Idéalement, à l'instar du script d'évaluation du challenge WebNLG avec les triplets, il faudrait parser les différents éléments de la requête étant donné qu'ils n'ont pas la même valeur sémantique que dans du texte en langage naturel. Il faut préciser ici que le travail sur LCQuAD s'est fait sur les dernières semaines de mon stage et j'ai donc pu y consacrer moins de temps que sur WebNLG. La résolution de ces problèmes pourrait donc faire partie du travail de thèse qui prendra la suite de mon stage.

Tout ceci étant dit, nous pouvons néanmoins constater que les scores semblent très similaires à ceux de la tâche de NLG sur LCQuAD, avec cependant un écart de performances en faveur des modèles de type BART plus marqué.

9.3. Évaluation filtrée par nombre de contraintes pour le NLG sur LCQuAD

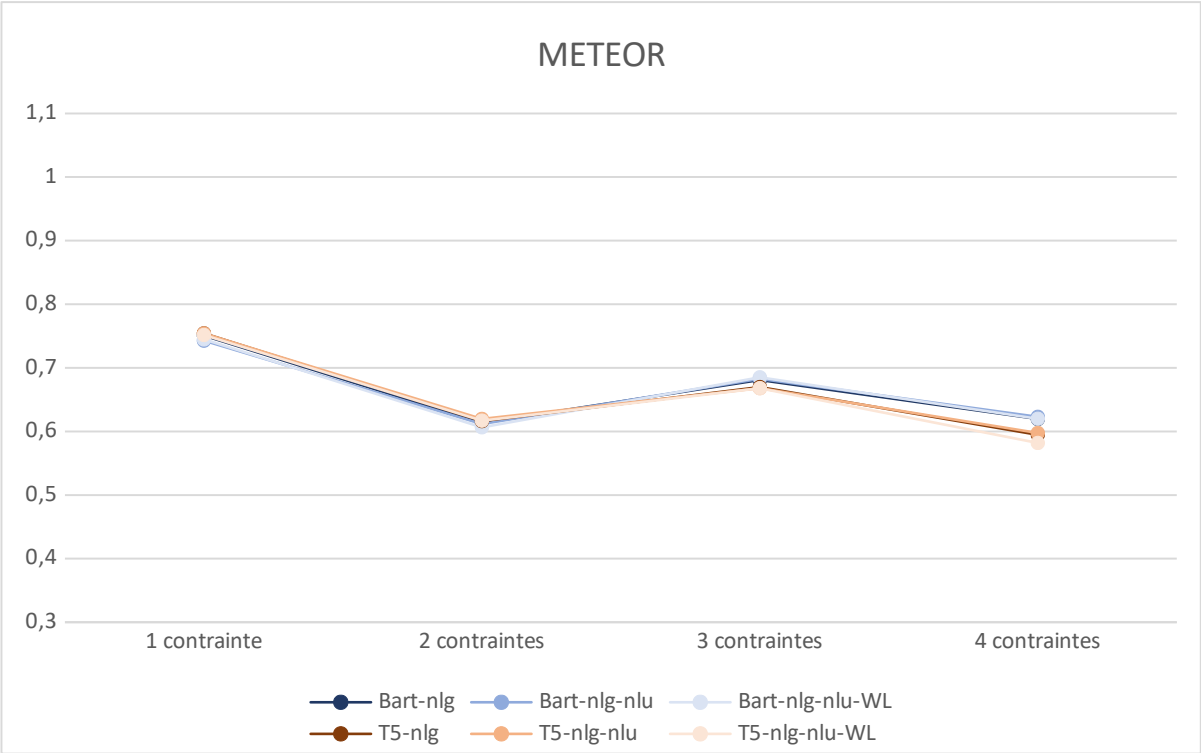

Figure 17: Scores METEOR par nombre de contraintes pour le NLG sur LCQuAD

Dans ce même souci de complétude et dans le temps de stage qu'il me restait, j'ai mené une évaluation filtrée sur le corpus LCQuAD. Ici le critère de filtrage était moins évident et restait à trouver. En effet, une requête SPARQL a une structure plus complexe qu'un simple ensemble de triplets. Plusieurs approches peuvent être adoptées, selon ce que l'on considère comme un facteur de complexité supplémentaire. Le nombre de variables sur lesquelles la requête porte (trouvables après le mot-clé `SELECT`), le nombre de contraintes dans la partie `WHERE` de la requête, ou encore le nombre et le type de tâches supplémentaires à effectuer dans le traitement de requête (mots-clé `FILTER`, `COUNT`, `LIMIT`, etc …) peuvent tous être considérés comme des facteurs de complexité, en ce qu'ils peuvent se refléter dans la traduction en question en langage naturel. Le problème étant complexe et afin d'avoir des résultats avant la fin de mon stage, j'ai choisi de filtrer selon le nombre de contraintes. Bien que semblables à des triplets, je les appelle contraintes car elles contraignent ce qui peut être considéré comme une réponse dans la base de données.

Comme on peut le voir sur la figure 16, ces contraintes sont au plus au nombre de 4 dans une requête. L'évolution de la performance est similaire entre tous les modèles. La courbe est cependant chaotique avec une recrudescence pour les requêtes à 3 contraintes, que je n'ai pas trouvé comment expliquer.

9.4. Évaluation filtrée par nombre de contraintes pour le NLU sur LCQuAD

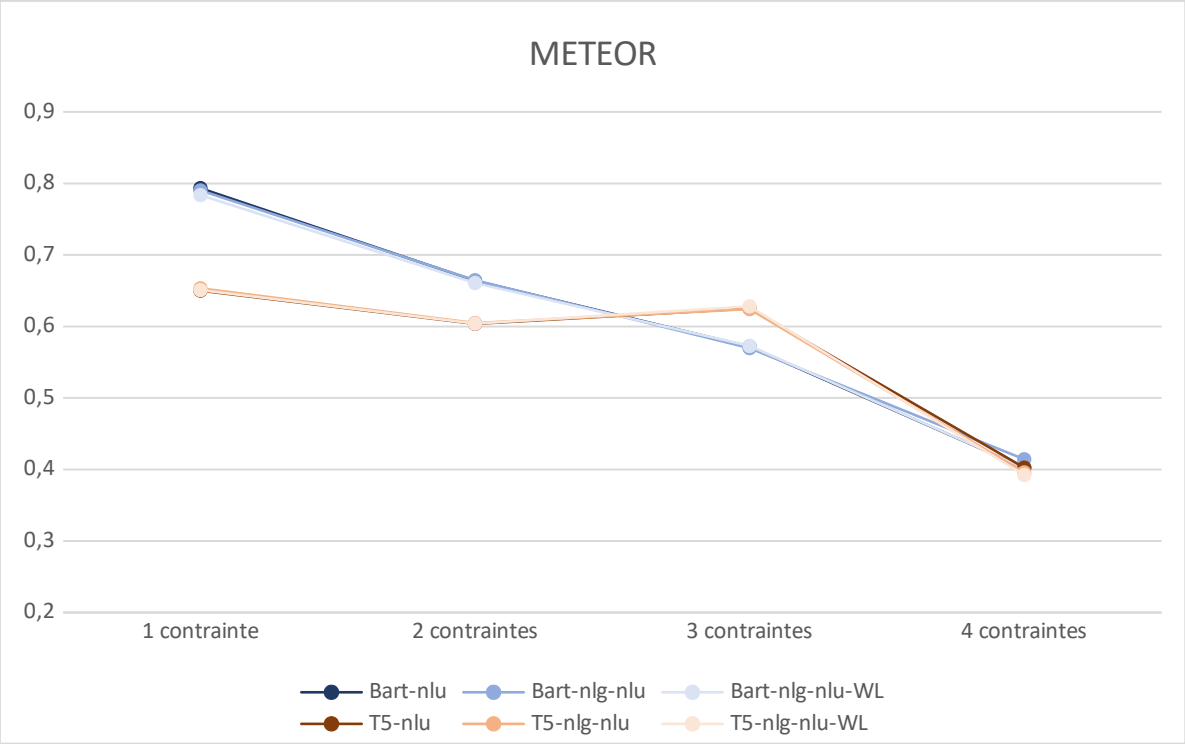

Figure 18: Scores METEOR par nombre de contraintes pour le NLU sur LCQuAD

Les scores du NLU sur LCQuAD sont présentés dans la figure 17. Encore un fois, ces scores sont à prendre avec précautions étant donné que les métriques de NLG ne sont pas adaptées à l'évaluation de la tâche de NLU sur des requêtes SPARQL.

En ce qui concerne les modèles de type T5, la courbe à un peu la même allure que pour le NLG, avec cette même recrudescence pour 3 contraintes mais cependant une chute de performance plus brutale pour quatre contraintes. La perte de performance globale entre 1 et 4 contraintes est de 0,25.

Du côté des modèles de type BART, la courbe a une allure plus proche de notre hypothèse, car plus régulière dans sa descente. Ces modèles semblent meilleurs sur les requêtes à une seule contrainte, à l'instar de ce que nous avions aperçu lors du NLU sur WebNLG. Cependant leurs performances diminuent rapidement pour atteindre le même niveau que celles des modèles du type T5 sur les requêtes à 4 contraintes. Leur perte de performance globale entre 1 et 4 contraintes est de 0,4.

Conclusion, perspectives et bilan

Au cours de mon stage, j'ai entraîné et évalué une série de modèles différents dans le but de déterminer si l'apprentissage de plusieurs tâches en même temps et sur des données diverses a un impact sur les performances. Ces modèles étaient des transformers de type BART ou T5. Chaque modèle a été entraîné sur un ou deux corpus (WebNLG et/ou LCQuAD) et pour effectuer une ou deux tâches (NLU et/ou NLG). La différence entre les deux corpus repose sur la représentation formelle du langage utilisée : triplets RDF pour WebNLG, requête SPARQL pour LCQuAD. Sur les 14 modèles résultants, 8 effectuent donc une seule tâche sur un seul corpus, 4 effectuent les deux tâches sur un seul corpus et 2 effectuent les deux tâches sur les deux corpus. Chaque modèle a ensuite été évalué pour chacune des combinaisons corpus-tâche pour lesquelles il a été entraîné.

Les résultats des évaluations nous font apparaître plusieurs tendances :

1) La plus notable et celle qui répond à l'interrogation principale de mon sujet de stage est que le fait d'ajouter des tâches et des corpus ne semble pas affecter négativement les performances des modèles, pas de manière significative en tout cas. Cela est plutôt encourageant pour la suite dans la perspective de créer un modèle capable de gérer plusieurs étapes de la chaîne de traitement d'un système de questions-réponses.

2) En ce qui concerne la comparaison des performances des deux types de modèles entre eux, les modèles de type de type T5 semblent meilleurs au moins sur le corpus WebNLG, c'est-à-dire dans la manipulation de phrase affirmative et de triplets RDF, et en particulier pour la tâche de NLU. Ils sont également à peu près aussi bons que les modèles de l'état de l'art (les 3 meilleurs candidats du challenge). Du côté du corpus LCQuAD, il est plus difficile de se prononcer en raison de l'absence de métrique correcte pour le NLU et du temps plus restreint que j'ai pu consacrer à l'étude de ce corpus lors de mon stage. Ceci étant dit, les premières observations semblent nous faire pencher en faveur des modèles de type BART, en tout cas pour le NLG où les métriques utilisées sont adaptées.

3) Enfin, au niveau des évaluations filtrées que j'ai pu menées, de nombreuses questions restent en suspens vis-à-vis de l'allure des courbes ainsi qu'au critère de filtrage dans le cas de LCQuAD. Cependant, il semblerait que les modèles T5 ont des performances plus robustes à l'augmentation de la complexité que les modèles BART.

Poursuite prévue par un sujet de thèse

Le travail mené dans le cadre de mon sujet de stage va se poursuivre dans le cadre d'une thèse intitulée « Deep Learning pour le Traitement Conjoint du Langage Naturel et des Connaissances ». Le doctorant devra non seulement approfondir les travaux sur les tâches de NLU et de NLG mais également trouver des solutions afin de permettre une meilleure généralisation des connaissances du modèle à d'autres tâches manipulant langage naturel et représentation formelle. De plus, il aura également pour objectif de rendre ces modèles capables de fonctionner dans des contextes énonciatifs plus complexes que de simples phrases affirmatives ou questions, (cf. les deux corpus que j'ai présentés), comme des discussions entières entre l'humain et la machine, où il est nécessaire de garder mémoire d'un historique de la conversation.

Bilan personnel de mon expérience de stage

Enfin, en terme de bilan personnel quant à cette expérience de stage, je dois dire qu'elle a été très positive pour moi. En plus de compléter mes connaissances théoriques dans certains domaines du Traitement Automatique des Langues (TAL) que je n'avais pas ou peu abordés en cours, ce stage m'a surtout permis de les appliquer de manière concrète et de gagner en savoir-faire et compétences techniques. J'ai également eu l'occasion de me familiariser avec le fonctionnement de la recherche en général (lectures d'articles, méthode scientifique, procédés d'entraînement et d'évaluation de modèle de TAL), ainsi qu'avec le fonctionnement d'une entreprise (réunions hebdomadaires, conduite de projets, présentation de diaporamas, etc.). Je trouve que j'ai particulièrement bien été accueilli et encadré durant toute la durée de ma présence au sein de l'équipe. Du matériel de qualité et un environnement de travail sain ont été mis à ma disposition. Je pouvais facilement interagir avec les autres membres de l'équipe, doctorants et alternants. Enfin, je me suis senti parfaitement encadré par mes tuteurs, avec un juste équilibre pédagogique entre libertés dans mon travail et lignes à suivre.

Bibliographie

- Zaib, Munazza, et al. « Conversational Question Answering: A Survey ». arXiv:2106.00874 [cs], juin 2021, http://arxiv.org/abs/2106.00874.

- Kapanipathi, Pavan, et al. « Leveraging Abstract Meaning Representation for Knowledge Base Question Answering ». arXiv:2012.01707 [cs], juin 2021, http://arxiv.org/abs/2012.01707.

- Garbacea, Cristina, et Qiaozhu Mei. « Neural Language Generation: Formulation, Methods, and Evaluation ». arXiv:2007.15780 [cs], juillet 2020, http://arxiv.org/abs/2007.15780.

- Guo, Qipeng, et al. « $\mathcalP^2$: A Plan-and-Pretrain Approach for Knowledge Graph-to-Text Generation ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 100–106, https://aclanthology.org/2020.webnlg-1.10.

- Moussallem, Diego, et al. « A General Benchmarking Framework for Text Generation ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 27–33, https://aclanthology.org/2020.webnlg-1.3.

- Guo, Qipeng, et al. « CycleGT: Unsupervised Graph-to-Text and Text-to-Graph Generation via Cycle Training ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 77–88, https://aclanthology.org/2020.webnlg-1.8.

- Agarwal, Oshin, et al. « Machine Translation Aided Bilingual Data-to-Text Generation and Semantic Parsing ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 125–130, https://aclanthology.org/2020.webnlg-1.13.

- Castro Ferreira, Thiago, et al. « The 2020 Bilingual, Bi-Directional WebNLG+ Shared Task: Overview and Evaluation Results (WebNLG+ 2020) ». Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, 2020, p. 55–76, https://aclanthology.org/2020.webnlg-1.7.

- Novikova, Jekaterina, et al. « The E2E Dataset: New Challenges For End-to-End Generation ». arXiv:1706.09254 [cs], juillet 2017, http://arxiv.org/abs/1706.09254.

- Tseng, Bo-Hsiang, et al. « A Generative Model for Joint Natural Language Understanding and Generation ». arXiv:2006.07499 [cs], juin 2020, http://arxiv.org/abs/2006.07499.

- Pyatkin, Valentina, et al. « Asking It All: Generating Contextualized Questions for any Semantic Role ». arXiv:2109.04832 [cs], septembre 2021, http://arxiv.org/abs/2109.04832.

- Montella, Sebastien, et al. « Denoising Pre-Training and Data Augmentation Strategies for Enhanced RDF Verbalization with Transformers ». arXiv:2012.00571 [cs], décembre 2020, http://arxiv.org/abs/2012.00571.

- Tafjord, Oyvind, et Peter Clark. « General-Purpose Question-Answering with Macaw ». arXiv:2109.02593 [cs], septembre 2021, http://arxiv.org/abs/2109.02593.

- Zhu, Chenguang, et al. « Multi-task Learning for Natural Language Generation in Task-Oriented Dialogue ». Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, 2019, p. 1261–1266, https://doi.org/10.18653/v1/D19-1123.

- Pramanik, Soumajit, et al. « UNIQORN: Unified Question Answering over RDF Knowledge Graphs and Natural Language Text ». arXiv:2108.08614 [cs], septembre 2021, http://arxiv.org/abs/2108.08614.

- Kacupaj, Endri, et al. « VOGUE: Answer Verbalization Through Multi-Task Learning ». Machine Learning and Knowledge Discovery in Databases. Research Track, édité par Nuria Oliver et al., Springer International Publishing, 2021, p. 563-79, https://doi.org/10.1007/978-3-030-86523-8_34.

Sigles et abréviations utilisés

NLG	Natural Language Generation
NLU	Natural Language Understanding
RDF	Ressource Description Framework
SPARQL	SPARQL Protocol And RDF Query Language
BART	Bidirectional Auto-Regressive Transformer
T5	Text-To-Text Transfer Transformer
LCQuAD	Largescale Complex Question Answering Dataset
BLEU	BiLingual Evaluation Understudy
METEOR	Metric for Evaluation of Translation with Explicit ORdering
TER	Translation Error Rate
CHRF	CHaRacter F-score
BERT	Bidirectional Encoder Representations from Transformers
BLEURT	BiLingual Evaluation Understudy with Representations from Transformers

Table des matières

Table des matières

MOTS-CLÉS : Compréhension, Génération, Graphes de connaissances, Triplets RDF, requête SPARQL

RÉSUMÉ

Dans le cadre d'un projet plus vaste visant à améliorer les modèles neuronaux actuels de conversion entre langage naturel et langage formel, mon travail a consisté à rechercher comment rendre des modèles de type Transformer (BART et T5) capables d'effectuer cette conversion dans les deux sens. Le travail a été mené d'abord sur des énoncés affirmatifs issus du corpus WebNLG, puis interrogatifs, issus du corpus LCQuAD, et enfin sur ces deux types d'énoncés mélangés. Les résultats actuels démontrent que les modèles multi-tâches entraînés sur des données de types différents mélangées ne perdent pas en performance. Bien que certaines questions restent en suspens et que le travail doit être approfondi dans le cadre d'une thèse, ces premiers résultats sont encourageants et pourraient se traduire à terme par une amélioration des services de questions-réponses proposés par Orange.

KEYWORDS : NLU, NLG, Knowledge graphs, RDF triples, SPARQL queries

ABSTRACT

As part of a larger project aimed at improving the current neural models of conversion between natural language and formal language, my work consisted in reasearching how to make Transformers models (BART and T5) capable of performing this conversion in both directions. The work was carried out first on affirmative statements, taken from the WebNLG dataset, then on interrogative statements, taken from the LCQuAD dataset, and finally on a mix of these two types. The current results demonstrate that multi-task models trained on mixed data of different types do not lose performance. Although some questions remain unanswered and the work must be deepened in the context of a thesis, these initial results are encouraging and could ultimately lead to an improvement in the question-answer services offered by Orange.