

Titre: Comparaison et étude de la généralisation de modèles neuronaux
pré-entraînés pour la génération de requêtes SPARQL

Auteur: Samuel Reyd
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Reyd, S. (2023). Comparaison et étude de la généralisation de modèles
neuronaux pré-entraînés pour la génération de requêtes SPARQL [Mémoire de
maîtrise, Polytechnique Montréal]. PolyPublie.
Citation: <https://publications.polymtl.ca/54902/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/54902/>
PolyPublie URL:

**Directeurs de
recherche:** Amal Zouaq
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Comparaison et étude de la généralisation de modèles neuronaux pré-entraînés
pour la génération de requêtes SPARQL**

SAMUEL REYD

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Août 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

**Comparaison et étude de la généralisation de modèles neuronaux pré-entraînés
pour la génération de requêtes SPARQL**

présenté par **Samuel REYD**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Quentin CAPPART, président

Amal ZOUAQ, membre et directrice de recherche

Gilles PESANT, membre

DÉDICACE

Machines take me by surprise with great frequency.

— Alan Turing

REMERCIEMENTS

Je remercie Prof. Amal Zouaq pour avoir dirigé ma maîtrise, et m'avoir conseillé tout au long du projet. Je remercie tous les membres du laboratoire LAMA-WeST et notamment Rose Hirigoyen qui a travaillé sur le début de ce projet et m'a aidé à en comprendre une grande partie ainsi que Karou Diallo qui prendra ma suite sur ce même projet et m'a accompagné sur certaines expériences de ce mémoire.

Je remercie également Henriette Ducrot qui m'a écouté et soutenu pendant toute la durée de ce master. Merci à Vi Retault pour m'avoir aidé pour toutes les manipulations sur les différents serveurs. Merci à Clélia Mérel pour l'accompagnement quotidien. Merci à tous les amis de Montréal et notamment Pênio et Nero.

Je remercie également ma famille et particulièrement ma mère et mon beau-père pour m'avoir soutenu pendant toutes ces années.

RÉSUMÉ

Les bases de connaissances sont une méthode prometteuse pour stocker et organiser des données. Cependant, il est difficile d'accéder à de telles bases de données pour les utilisateurs qui ne sont pas des experts dans le domaine, car les requêtes doivent être exprimées dans le langage de requête SPARQL, qui est difficile à maîtriser pour les personnes en dehors du domaine du Web sémantique. Cette difficulté peut être surmontée par la conception de modèles efficaces de traduction de questions en langage naturel vers des requêtes SPARQL.

Les approches modernes pour résoudre cette tâche impliquent des architectures de traduction neuronale qui considèrent SPARQL comme une autre langue naturelle. La tâche a connu des progrès récents qui ont été encouragés par le développement d'architectures neuronales de plus en plus puissantes ainsi que par la conception de mécanismes spécifiques qui s'attaquent aux principales difficultés de la tâche.

Ce mémoire propose une comparaison complète des approches récentes de traduction neuronale sur la base de l'annotation de questions, des ensembles de données, des architectures de modèles et de l'utilisation du mécanisme de copie. Il présente la première tentative de combiner la force du mécanisme de copie et l'utilisation de modèles pré-entraînés. Il aborde également la question de la généralisation de ces approches en définissant les difficultés clés liées aux jeux de données de référence et en concevant un algorithme pour re-diviser les ensembles de données afin d'évaluer la façon dont les modèles traitent ces difficultés. Notamment, nous évaluons la capacité des modèles à gérer les questions contenant des ressources (URI) inconnues et les questions ayant une structure non vue en entraînement.

Nos résultats montrent que l'annotation adéquate des questions et l'usage de modèles pré-entraînés avec ou sans le mécanisme de copie permettent d'obtenir des performances de pointe sur chacun des trois jeux de données de références utilisés. De plus, nos résultats sur de nouvelles séparations de ces jeux de données montrent que les modèles pré-entraînés ainsi que le mécanisme de copie permettent de gérer les URIs inconnues. Cependant, bien que certains modèles soient capables de générer des requêtes avec les bonnes réponses sur un jeu de données simple, nos résultats montrent que nos modèles ont beaucoup de difficultés à généraliser à des structures inconnues.

ABSTRACT

Knowledge bases are a promising way of storing and organizing data. However, accessing such knowledge bases for non-expert users is not trivial since queries must be expressed in the SPARQL language, which is hard to master for people outside the field of the Semantic Web. One way to deal with this difficulty would be to design efficient models to automatically translate Natural Language questions into SPARQL queries.

Modern approaches to solving this task involve Neural Machine Translations architectures that consider SPARQL as another language. Recent progress has been driven by the development of powerful neural architectures as well as the design of specific mechanisms that address the main difficulties of the task.

This Master’s thesis offers a comprehensive comparison of recent approaches on neural machine translation, by evaluating various question annotation, datasets, model architectures, and the usage of the copy mechanism. It also features the first approach to combine the strength of the copy mechanism and pre-trained models. It also studies the issue of generalization of these approaches by defining key difficulties that models might encounter in real word settings and designing an algorithm to re-split datasets in order to evaluate how models deal with these difficulties. We notably evaluate how these models manage questions that contain unknown resources (URI) and questions’ structures unseen in training.

Our results show that the right question annotation and the use of pre-trained models with or without the copy mechanism enable new state-of-the-art results for each of our three datasets. Moreover, our results on the new splits of these datasets show that pre-trained models and the copy mechanism handle unknown URIs to some extent. However, even though some models are able to generate queries that yield correct answers, our results show that pretrained models have significant difficulties to generalize to unknown question’s structures.

TABLE DES MATIÈRES

DÉDICACE	iii
REMERCIEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vi
TABLE DES MATIÈRES	vii
LISTE DES TABLEAUX	x
LISTE DES FIGURES	xi
LISTE DES SIGLES ET ABRÉVIATIONS	xii
CHAPITRE 1 INTRODUCTION	1
1.1 Définitions et concepts de base	2
1.1.1 Web sémantique	2
1.1.2 Requêtes	5
1.1.3 Traduction automatique de questions en requêtes SPARQL	6
1.2 Éléments de la problématique	8
1.2.1 Lien entre le langage naturel et la base de connaissances	8
1.2.2 Concepts hors vocabulaire	9
1.2.3 Adaptabilité du mécanisme de copie	10
1.2.4 Comparabilité des approches de référence	10
1.2.5 Généralisation	10
1.3 Objectifs de recherche	11
1.4 Plan du mémoire	12
CHAPITRE 2 REVUE DE LITTÉRATURE	13
2.1 Architectures neuronales pour la traduction automatique de requêtes SPARQL	13
2.1.1 Architecture encodeur-décodeur	13
2.1.2 Architectures non pré-entraînées	15
2.1.3 Architectures pré-entraînées	16
2.1.4 Architecture de copie	19

2.2	Métriques d'évaluation de la tâche	21
2.2.1	Score BLEU	21
2.2.2	Précision des réponses	22
2.2.3	Score F1 des réponses	23
2.3	Traduction automatique pour SPARQL	23
2.3.1	Approche classique	23
2.3.2	Transfert d'apprentissage	25
2.3.3	Mécanisme de copie	26
2.3.4	Comparabilité	27
2.4	Généralisation pour la génération de requêtes SPARQL	28
2.4.1	La généralisation compositionnelle	28
2.4.2	Généralisation pour la génération de requêtes SPARQL	29
2.4.3	Lien avec les difficultés de l'approche par traduction automatique neuronale	31
CHAPITRE 3	GÉNÉRATION DE REQUÊTES SPARQL	32
3.1	Jeux de données de référence	32
3.1.1	Patrons de génération	32
3.1.2	Questions reformulées	32
3.1.3	Les jeux de données	33
3.2	Jeux de données finaux	34
3.2.1	Vocabulaire	34
3.2.2	Annotation des questions	35
3.2.3	Méthodologie d'annotation	37
3.3	Méthodologie	39
3.3.1	Configurations des expériences	39
3.3.2	Sélection des hyper-paramètres	42
3.3.3	Entraînement et validation	43
3.3.4	Génération sur l'ensemble de test	44
3.3.5	Évaluation	45
3.4	Résultats	47
3.4.1	Modèles non pré-entraînés	47
3.4.2	Modèles pré-entraînés	48
3.4.3	Meilleurs résultats	50
3.5	Discussion	51
3.5.1	Le cas de LC-QuAD 2.0	51

3.5.2	Difficulté de l’annotation tag-final pour le mécanisme de copie	53
3.5.3	Limitations du mécanisme de copie	54
3.5.4	Autres modèles	54
3.5.5	Limites de l’approche	55
CHAPITRE 4	GÉNÉRALISATION	56
4.1	Propriété de généralisation	56
4.2	Séparation des jeux de données	57
4.2.1	Groupes de données	58
4.3	Statistiques des nouveaux jeux de données	68
4.4	Généralisation des modèles non pré-entraînés	69
4.4.1	Séparation avec patrons globaux inconnus	69
4.4.2	Séparation avec URIs inconnues	70
4.5	Généralisation des modèles pré-entraînés	71
4.5.1	Séparation avec patrons globaux inconnus	71
4.5.2	Séparation avec URIs inconnues	72
4.6	Discussion	72
4.6.1	Faibles performances des modèles non pré-entraînés sur les questions avec tag-final sur la séparation des URI inconnues	72
4.6.2	Baisse du score BLEU pour la séparation avec patrons inconnus	73
4.6.3	Difficulté de LC-QuAD 2.0	74
CHAPITRE 5	CONCLUSION	76
5.1	Synthèse des travaux	76
5.2	Limites de la solution proposée	77
5.2.1	Optimisation des hyper paramètres	77
5.2.2	Mécanisme de copie	78
5.2.3	Méthode d’annotation	78
5.2.4	Méthode de séparation avec URIs inconnues	79
5.2.5	Distributions similaires des ensembles de validation et d’évaluation	79
5.3	Améliorations futures	79
5.3.1	Mécanisme de copie	79
5.3.2	Méthodologie d’évaluation des capacités de généralisation	80
5.3.3	Résumé du travail réalisé	81
RÉFÉRENCES		82

LISTE DES TABLEAUX

Tableau 3.1	Taille des jeux de données	34
Tableau 3.2	Nombre de chaque type de patrons de génération	35
Tableau 3.3	Taille des vocabulaires	36
Tableau 3.4	Statistiques supplémentaires sur le vocabulaire de langue naturelles, des questions avec tag-interne, sur les ressources, les classes et propriétés ainsi que les littéraux de la BC	36
Tableau 3.5	Hyper paramètres des modèles	42
Tableau 3.6	Nombre d'époques d'entraînement	43
Tableau 3.7	Taille des batchs	44
Tableau 3.8	Statistiques des réponses dans nos jeux de données	46
Tableau 3.9	Résultats des modèles non pré-entraînés sur les jeux de données originaux	49
Tableau 3.10	Résultats des modèles pré-entraînés sur les jeux de données originaux	50
Tableau 3.11	Précision des requêtes masquées pour un patron de question commun au patron de question "what is <mask> <mask> of <mask> that is <mask> is <mask> ?" en fonction de la fréquence des patrons de requêtes associés dans le jeu d'entraînement	53
Tableau 4.1	Nombre d'URIs, de patrons globaux, et de paires de questions-requêtes dans les jeux de données avec les nouvelles séparations	69
Tableau 4.2	Résultats des modèles non pré-entraînés sur la séparation avec patrons inconnus	70
Tableau 4.3	Résultats des modèles non pré-entraînés sur la séparation avec des URIs inconnues	71
Tableau 4.4	Résultats des modèles pré-entraînés sur la séparation avec patrons inconnus	71
Tableau 4.5	Résultats des modèles pré-entraînés sur la séparation avec URIs inconnues	72
Tableau 4.6	Exemple des sorties de BART pour un patron du jeu de test de LC-QuAD 1.0	75

LISTE DES FIGURES

Figure 1.1	Une fraction de la page Wikipédia de Stanley Kubrick organisée comme dans le Web classique	2
Figure 1.2	Une fraction de la page Wikipédia de Stanley Kubrick incluant les informations de DBPedia	3
Figure 1.3	Un exemple d'ensemble de triplets RDF	4
Figure 1.4	Exemple d'une partie d'un graphe de connaissances	4
Figure 1.5	Exemple d'une question associé à une requête SPARQL et la réponse à la requête après exécution	5
Figure 2.1	L'architecture encodeur-décodeur.	15
Figure 2.2	Illustration du mécanisme de copie	22
Figure 3.1	Exemple d'un patron de génération et d'une donnée générée par ce patron	33
Figure 3.2	Exemple d'une requête et des trois versions de la question correspondante	37
Figure 3.3	Exemple d'expression régulière dérivée d'un patron global.	38
Figure 3.4	Exemple de segmentation d'une requête par le segmenteur de BART.	43
Figure 3.5	Les meilleurs scores F1 pour chaque jeu de données	52
Figure 4.1	Exemple de graphe utilisé pour la création des groupes pour la séparation avec URIs inconnues	60
Figure 4.2	Proportion d'entrées du jeu de données avec une URI rare et nombre de composantes connexes en fonction de la fréquence minimale de rareté	62
Figure 4.3	Distribution des groupes pour la séparation avec patrons inconnus . .	65
Figure 4.4	Distribution des groupes pour la séparation avec URIs inconnues . . .	65

LISTE DES SIGLES ET ABRÉVIATIONS

BC	Base de Connaissances
TALN	Traitement Automatique de la Langue Naturelle
RDF	Resource Description Framework
URI	Universal Resource Identifier
SPARQL	SPARQL Protocol and RDF Query Language
HV	Hors Vocabulaire

CHAPITRE 1 INTRODUCTION

Le web sémantique est une extension du web qui vise notamment à faciliter un traitement automatisé de son contenu. Cependant, son utilisation par le public reste limitée. La recherche d'information dans les bases de données qui le composent, appelées bases de connaissances (BC), est difficile pour des humains. Le langage de requêtes associé aux BC (SPARQL) n'est pas facilement utilisable par des gens qui ne sont pas spécialistes du sujet. Ce langage nécessite de bien connaître le schéma de la base de connaissances, ce qui n'est pas le cas du grand public. L'une des manières de mieux inclure le public serait de pouvoir obtenir les informations de ces bases de connaissances à l'aide de questions en langage naturel. La tâche à réaliser devient donc de transformer automatiquement les questions en requêtes SPARQL. Les approches actuelles utilisent le paradigme de la traduction automatique à l'aide d'apprentissage profond [1]. Elles considèrent la requête SPARQL comme une question dans une autre langue naturelle, sans considérer dans leur approche le fait que SPARQL est en fait un langage formel. Les progrès récents dans les architectures neuronales du traitement automatique de la langue naturelle (TALN) [2, 3], puis des modèles pré-entraînés [4–6], ont permis à ces approches d'atteindre de très bonnes performances [7, 8]. Cependant, ces méthodes sont peu comparables, car elles utilisent notamment des jeux de données et des métriques différents. D'autre part, elles proposent des solutions qui pourraient être complémentaires. Enfin, les progrès récents des performances mesurées sur les jeux de données classiques de la tâche posent la question de la difficulté de ces jeux de données et notamment de leur potentiel à évaluer les capacités de généralisation des modèles. Ce mémoire s'intéressera donc à la tâche de traduction de questions en anglais vers des requêtes SPARQL à l'aide de modèles de traduction automatique ainsi qu'à l'analyse de leurs capacités de généralisation.

Les contributions de ce mémoire sont 1) la mise en place d'une comparaison systématique des méthodologies inspirées de la littérature [7, 8], 2) l'introduction de l'usage du mécanisme de copie avec les modèles pré-entraînés, démontrant son impact positif pour un modèle (BART [5]) et négatif pour un autre (T5 [6]), 3) de nouvelles performances de références pour trois jeux de données classiques de la tâche, 4) la mise en évidence de bonnes capacités de généralisation aux URIs inconnues pour les modèles dotés du mécanisme de copie et / ou d'un pré-entraînement utilisant des questions annotées, 5) la mise en évidence de limites des capacités de généralisation à des données générées par des patrons inconnus de tous les modèles.

Les expériences de ce mémoire peuvent être reproduites en utilisant le code disponible sur le

GitHub du laboratoire¹.

Les sections suivantes introduisent les concepts de base de ce mémoire, à savoir les concepts importants du Web sémantique et de la traduction automatique neuronale. Puis les éléments de problématique sont énoncés. Enfin, le plan du mémoire est décrit.

1.1 Définitions et concepts de base

1.1.1 Web sémantique

Idée générale

Le web sémantique est une vision du Web dans laquelle les données sont représentées comme des concepts liés par des relations. Dans le Web classique, les pages contiennent du contenu brut comme du texte ou des images. Les liens entre les pages sont tous de même nature et ne sont compréhensibles que par des humains capables d'interpréter ce contenu brut. Dans le web sémantique, les briques élémentaires sont les Universal Resource Identifiers (URIs). Ce sont des concepts liés entre eux par des relations explicites. La figure 1.1 montre le genre de donnée brute incluant des liens arbitraires avec le format du Web classique. La figure 1.2 montre la sous-section de la même page Wikipédia qui contient en partie les mêmes informations structurées selon le mode du Web sémantique.

Stanley Kubrick (prononcé en anglais : /ˈstænli ˈkubɹɪk/^{N 1}) est un réalisateur, photographe, scénariste et producteur de cinéma américain, né le 26 juillet 1928 à Manhattan (New York^{1, N 2}) et mort le 7 mars 1999 dans son manoir de Childwickbury, entre St Albans et Harpenden (Hertfordshire, au nord de Londres)².

Après des débuts dans la photographie, Kubrick, autodidacte, est également son propre directeur de la photographie, producteur, scénariste et monteur. Ses treize longs métrages en quarante-six ans de carrière l'imposent comme l'un des cinéastes majeurs du xx^e siècle. Quatre de ses films sont classés dans le Top 100 de l'American Film Institute (*AFI's 100 Years...100 Movies*).

FIGURE 1.1 Une fraction de la page Wikipédia de Stanley Kubrick organisée comme dans le Web classique

1. https://github.com/Lama-West/SPARQL_Query_Generation_PLM_General_ISWC23

Nom de naissance	Stanley Kubrick
Naissance	26 juillet 1928 Manhattan (État de New York, États-Unis)
Nationalité	 Américaine
Décès	7 mars 1999 (à 70 ans) St Albans (Angleterre, Royaume-Uni)
Profession	Photographe Réalisateur Scénariste Producteur
Films notables	<i>voir filmographie</i>

FIGURE 1.2 Une fraction de la page Wikipédia de Stanley Kubrick incluant les informations de DBPedia

Le Web sémantique représente donc les connaissances sous une forme structurée. On appelle ces bases de données des bases de connaissances. Il existe certaines bases de connaissances à grande échelle, comme Wikidata² ou DBPedia³ qui sont construites en partie par des experts et en partie par extraction automatique sur plusieurs sites du Web classique.

Les bases de connaissances sont organisées suivant une syntaxe commune appelée Resource Framework Description (RDF)⁴. Cette syntaxe permet d'exprimer une base de connaissances sous la forme d'un graphe orienté. On parlera alors indifféremment dans ce mémoire de base de connaissances ou de graphe de connaissances.

Les nœuds et les arêtes de ce graphe sont des éléments appartenant à deux ensembles. Premièrement, l'ensemble des URI qui représentent les concepts nommés de la base de connaissances. Deuxièmement, les littéraux qui sont des données brutes comme des chiffres ou du texte. La base de connaissances peut alors être représentée comme un ensemble de triplets. Un triplet est composé d'un sujet, d'une propriété et d'un objet. Dans le graphe orienté, le sujet est l'extrémité initiale de l'arête, la relation est l'arête elle-même et l'objet est l'extrémité finale de l'arête. Des règles syntaxiques permettent de définir comment organiser les éléments de la BC. Par exemple, un sujet ou une propriété sont forcément une URI et un objet peut être une URI ou un littéral. Un exemple d'une partie de la base de connaissances DBPedia est illustré sous forme de triplets RDF dans la figure 1.3 et de graphe dans la Figure 1.4. Ici, les URIs sont composées d'un préfixe qui indique l'espace de nommage sous forme d'abréviation et de l'identifiant au sein de cet espace

2. <https://www.wikidata.org/>

3. <https://www.dbpedia.org/>

4. <https://www.w3.org/TR/rdf12-concepts/>

de nommage. Par exemple, l'URI notée ici `dbr:Stanley_Kubrick` a pour forme complète `<https://dbpedia.org/page/Stanley_Kubrick>`. La partie `https://dbpedia.org/page/` permet d'indiquer où se situe cette URI et la partie `Stanley_Kubrick` permet d'identifier cette URI au sein de son espace de nommage. Cela permet notamment d'utiliser les mêmes identifiants pour définir des concepts similaires dans différents lieux du Web Sémantique. Un document RDF ou une requête SPARQL commence généralement par un en-tête qui permet de définir les préfixes. Ce genre d'en-tête n'est pas utilisé dans ce mémoire, où les préfixes usuels sont utilisés par défaut dans les URIs.

```
dbr:Stanley_Kubrick rdf:type dbo:Person ;
                        dbo:birthDate "07-26-1926" .
dbr:The_Shining_(film) dbo:director dbr:Stanley_Kubrick ;
                        rdf:type dbo:Film .
dbr:2001:_A_Space_Odyssey_(film) dbo:director dbr:Stanley_Kubrick .
                        rdf:type dbo:Film .
```

FIGURE 1.3 Un exemple d'ensemble de triplets RDF

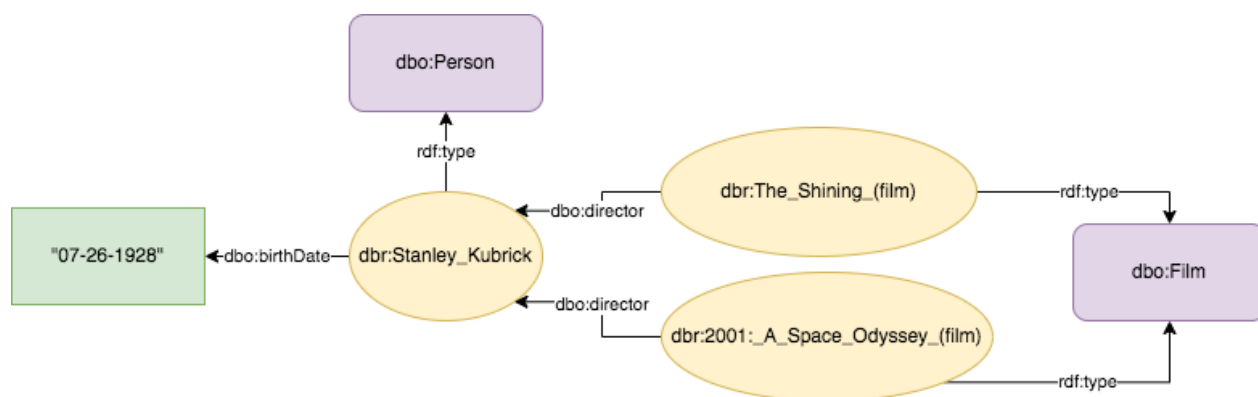


FIGURE 1.4 Exemple d'une partie d'un graphe de connaissances

Les bases de connaissances permettent également l'organisation des données de manière hiérarchique [9]. Les URIs peuvent en effet être des entités ou des classes. Les entités sont des concepts élémentaires du monde réel et les classes sont des concepts généraux. Ainsi, les classes peuvent s'imbriquer pour créer un arbre hiérarchique de concepts et les entités appartiennent finalement à une ou plusieurs classes. De plus, le web sémantique offre également la possibilité de construire des règles logiques qui régissent les classes et leurs éléments.

L'ensemble de la hiérarchie des classes et des règles sémantiques qui s'y ajoutent est appelé ontologie. La syntaxe utilisée pour créer ces ontologies est appelé OWL ⁵.

1.1.2 Requêtes

Le langage de requêtes associé aux bases de connaissances du Web sémantique est appelé SPARQL. Ce langage est basé sur des triplets dont certains éléments ont été remplacés par des variables. Cela permet d'extraire les triplets de la BC dans lesquels des URIs peuvent s'unifier avec les variables de la requête. Par exemple, la requête associée à la question "Combien de films a réalisé Stanley Kubrick" contiendra le triplet avec variable `?x` `dbo:director` `dbr:Stanley_Kubrick`. Ainsi, cela permettra de retourner notamment des triplets tels que :

```
dbr:The_Shining_(film) dbo:director dbr:Stanley_Kubrick .
```

ou

```
dbr:2001:_A_Space_Odyssey_(film) dbo:director dbr:Stanley_Kubrick .
```

On voit alors que la variable `?x` peut prendre les valeurs `dbr:The_Shining_(film)` et `dbr:2001:_A_Space_Odyssey_(film)`.

Les requêtes SPARQL contiennent ensuite une partie structurelle qui permet de filtrer ou de faire des opérations sur les résultats obtenus. Par exemple, pour la question "combien de films a réalisé Stanley Kubrick", on veut compter le nombre d'URIs distinctes qui peuvent remplacer `?x`. On obtient alors la structure suivante :

```
select distinct count ( ?x ) where { schema }
```

La Figure 1.5 montre la question et la requête qui sont liées à cet exemple que l'on peut trouver dans l'un des jeux de données utilisés dans ce mémoire.

Question :
How many movies are there whose director is Stanley Kubrick ?
Requête :
<code>select distinct count (?uri) where { ?uri dbo:director dbr:Stanley_Kubrick . ?uri rdf:type dbo:Film }</code>
Réponse : 16

FIGURE 1.5 Exemple d'une question associé à une requête SPARQL et la réponse à la requête après exécution

5. <https://www.w3.org/TR/2012/REC-owl2-overview-20121211/>

1.1.3 Traduction automatique de questions en requêtes SPARQL

Une des approches actuelles pour obtenir des informations d'une BC consiste en l'utilisation du TALN pour traduire la question en une requête SPARQL comme s'il s'agissait d'une autre langue naturel. Le domaine de la traduction automatique utilise des notions de bases du TALN par apprentissage profond. On utilise des fonctions paramétriques appelées modèles qui associent à une entrée donnée une sortie prédite. Généralement, les paramètres de ces fonctions sont initialisés aléatoirement et sont modifiés lors de l'entraînement de façon à minimiser l'erreur du modèle sur un jeu de données (appelé jeu d'entraînement) dont la sortie attendue du modèle est connue. Pour la tâche de traduction automatique, l'entrée est un texte dans une langue (ici la question en anglais) et la sortie est un texte au sens similaire dans une autre langue (ici la requête SPARQL). Le modèle doit donc interpréter la phrase d'entrée puis générer une phrase de sortie en intégrant le sens de l'entrée.

Encodage d'une question. La première étape de la tâche de traduction automatique est l'encodage de la phrase d'entrée, ici la question. Cette étape est réalisée par un bloc du modèle appelé encodeur. Elle consiste en la transformation d'une suite de mots en une suite de vecteurs. Chaque vecteur correspond à un des mots de la phrase d'entrée et ses poids doivent permettre d'interpréter le sens du mot correspondant relativement au reste du texte. Par ailleurs, ce "sens" est défini par rapport à la tâche. On considère ici que les vecteurs obtenus ont bien le sens attendu s'ils permettent d'obtenir la bonne traduction à la fin du procédé.

La méthodologie utilisée pour faire cette transformation est généralement la suivante. On définit un vocabulaire d'entrée du modèle qui sont tous les mots que le modèle est susceptible de rencontrer. En général, on utilise l'ensemble des mots qui sont présents dans le jeu d'entraînement du modèle. Pour chaque mot, on associe un vecteur appelé plongement initial. Lorsque le modèle reçoit une phrase d'entrée, il initialise l'ensemble des vecteurs de plongements initiaux. Ensuite, il effectue une suite d'opérations paramétriques qui transforment chaque vecteur en fonction du reste des vecteurs. Les poids des plongements initiaux et des fonctions paramétriques qui les transforment sont les poids de l'encodeur.

Génération de la requête. Pour générer la requête, on utilise un autre bloc du modèle appelé décodeur. Ce bloc transforme une suite de mots en une autre suite de mots. Pour chaque mot d'entrée, le mot correspondant dans la sortie doit être le mot suivant dans le texte à prédire. Pour obtenir ce résultat, le décodeur passe par deux étapes. Dans un premier temps, il transforme les mots d'entrée en vecteurs qui comportent le sens du mot relativement

à l'entrée et aux mots précédents, puis il utilise ces vecteurs pour générer les mots en sortie.

Pour transformer les mots d'entrée en vecteurs, le décodeur fonctionne de manière assez similaire à l'encodeur. On définit d'abord un vocabulaire du décodeur de la même manière que le vocabulaire de l'encodeur. Le décodeur associe ensuite à chaque mot un plongement initial et applique des fonctions paramétriques qui utilisent les plongements des mots précédents ainsi que les vecteurs obtenus par l'encodeur. Parmi ces fonctions paramétriques, on retrouve généralement des mécanismes dits d'attention croisée. Ces mécanismes utilisent les vecteurs obtenus par l'encodeur et calculent des poids d'attention croisée pour obtenir une somme pondérée de ces vecteurs. Cette somme sera utilisée pour modifier la représentation du mot que le décodeur interprète. Les poids d'attention croisée peuvent également être interprétés comme l'importance de chaque mot de l'entrée pour la génération d'un mot de la sortie.

Une fois les vecteurs représentant les mots de l'entrée de l'encodeur obtenus, le modèle applique une fonction de classification à chacun. Le domaine de la classification est l'ensemble du vocabulaire du décodeur. Ainsi, en prenant le résultat de la classification, on obtient le mot suivant prédit par le décodeur pour chaque mot qui lui a été passé. L'ensemble de poids des plongements initiaux, des fonctions de transformations de ces plongements et de la fonction de classification sont les poids du décodeur.

Pour générer une requête, on commence par passer au décodeur un jeton spécial symbolisant le début de la phrase. On répète ensuite le processus itérativement en utilisant la sortie du décodeur à l'étape $i - 1$ comme entrée du décodeur à l'étape i . On répète cela jusqu'à ce que le modèle génère un jeton spécial de fin de phrase ou bien qu'il dépasse un nombre maximal d'itérations.

Traduction automatique. La tâche de traduction automatique est donc réalisée par des modèles composés d'un bloc encodeur et d'un bloc décodeur. Un jeu de donnée de cette tâche est composé de paires de phrases, ici des paires questions-requêtes. On obtient les poids du modèle en minimisant l'erreur qu'il commet en générant les requêtes associées aux questions du jeu d'entraînement. On doit également construire un vocabulaire d'entrée basé sur des mots en langue naturelle et un vocabulaire de sortie basé sur des mots du langage SPARQL et des éléments de la BC. Lorsqu'on génère une requête à partir d'une nouvelle question avec le modèle entraîné, si celle-ci contient des mots qui ne sont pas dans son vocabulaire, on les remplace par le mot spécial `<unk>`. L'ensemble des mots que le modèle peut reconnaître, c'est-à-dire son vocabulaire ainsi que les mots spéciaux tels que `<bos>` (début de phrase), `<eos>` (fin de phrase), `<unk>`, etc., sont appelés jetons.

L’essor des modèles pré-entraînés. Depuis quelques années, le domaine de la traduction automatique a été révolutionné par le concept de transfert d’apprentissage [4]. Au lieu d’initialiser les poids du modèle aléatoirement, on utilise des modèles dits pré-entraînés. Un modèle pré-entraîné est basé sur une architecture d’encodeur-décodeur classique (en général l’architecture Transformer [2]) et est dans un premier temps entraîné sur des tâches dites bidon sur un très gros volume de données. Il pourra ensuite être entraîné à nouveau sur des tâches entre autres de traduction avec des jeux d’entraînement particuliers. Le premier entraînement est appelé pré-entraînement et le second est appelé affinage.

Les tâches bidon de préapprentissage consistent à appliquer de petites modifications (suppression, masquage, mélange de mots, etc.) dans un texte et de demander au modèle de produire la phrase d’origine. Comme ce procédé permet de générer automatiquement des paires de texte à partir de n’importe quel texte, il peut être appliqué facilement sur de très gros volume de données.

Afin de pouvoir gérer de gros volumes de texte et de pouvoir s’adapter à de nouvelles tâches lors de l’affinage, les vocabulaires des modèles pré-entraînés sont construits différemment de ceux des modèles non pré-entraînés. Ce vocabulaire contient de nombreux mots fréquents dans les langues du corpus d’entraînement ainsi qu’une liste de sous mots et l’ensemble de caractères qui peuvent être rencontrés. Ainsi, même si le modèle rencontre un mot inconnu, il pourra l’interpréter comme une suite de sous mots ou de caractères.

Les modèles pré-entraînés ont permis d’immenses améliorations des performances sur les tâches de TALN en général, et dans la génération de requêtes SPARQL en particulier [8].

1.2 Éléments de la problématique

1.2.1 Lien entre le langage naturel et la base de connaissances

Dans ce mémoire, SPARQL est considéré comme une langue naturelle dans laquelle doit être traduite une question. La principale difficulté est le lien entre le langage naturel et la base de connaissances. En effet, l’apprentissage machine est basé sur l’optimisation des poids du modèle afin de minimiser l’erreur du modèle sur de nombreux exemples d’entraînement. Or, le nombre de mots du langage SPARQL présent dans les jeux d’entraînement est relativement faible et permet donc au modèle d’avoir beaucoup d’exemples pour chaque utilisation de chaque mot de SPARQL. Ainsi, la génération de la structure de la requête est une tâche assez simple pour la plupart des modèles. Cependant, le nombre d’URIs d’une BC est extrêmement important et les jeux d’entraînement contiennent très peu d’exemples d’utilisation de la plupart d’entre elles. Ainsi, la génération des URIs est la principale difficulté des modèles.

Pour surmonter cette difficulté, il convient d’inclure dans l’entrée des informations sur les URIs qui sont implicites dans la question afin de faciliter leur génération, en identifiant notamment explicitement les URIs qui devront être utilisées. Ce procédé s’appelle l’annotation et peut être fait de plusieurs façons. Or, les approches d’annotation de questions n’ont jamais été comparées dans des contextes similaires, ce qui pose la question de leurs intérêt et impact sur la tâche de génération.

1.2.2 Concepts hors vocabulaire

Comme évoqué dans la section 1.1.3, le concept de base de la traduction automatique par apprentissage profond est l’encodage d’un texte par transformation des représentations vectorielles de ses mots, puis la prédiction du mot suivant lors de la génération parmi un vocabulaire fixé. Ces deux étapes nécessitent donc un vocabulaire fini et fixé (dans le premier cas pour associer chaque mot à sa représentation vectorielle). Cela pose un problème lorsque l’on demande au modèle de traduire un texte qui contient de nouveaux mots. En effet, ces vocabulaires sont généralement fixés en utilisant les mots présents dans l’ensemble des données d’entraînement. Cette limitation de l’approche par traduction automatique est particulièrement importante dans le contexte de la génération de requêtes SPARQL.

L’un des principaux problèmes liés à la génération de requêtes SPARQL est lié au fait qu’il s’agit d’un langage formel. Dans une langue naturelle, si un mot est inconnu, il peut être remplacé par un synonyme ou une périphrase et l’on perd très peu de sens au texte généré. Or, ici, on veut une requête capable de produire la bonne réponse. Si un mot est remplacé par un autre dans une requête SPARQL, il n’y a pratiquement aucune chance d’avoir une requête valide.

D’autre part, la nature de la tâche implique qu’il sera nécessaire et même fréquent de rencontrer des mots inconnus lors de l’usage du modèle. En effet, les bases de connaissances classiques sur lesquels les requêtes portent sont généralement à très grande échelle. Ainsi, aucun jeu de données ne permettra d’entraîner un modèle sur l’ensemble des concepts qu’elle contient. Ces concepts (représentés par des URIs dans la BC) risquent d’apparaître comme des mots inconnus dans la question. Le modèle ne pourra donc pas comprendre sur quoi porte la question. De plus, si l’URI qui doit être générée n’appartient pas au vocabulaire du modèle, il est impossible que ce dernier génère une requête correcte.

De manière plus générique, on observe en pratique que générer la mauvaise URI est la principale source d’erreur des modèles, y compris lorsqu’il s’agit d’URIs connues. Le nombre extrêmement important d’URIs jouant généralement des rôles similaires dans les requêtes implique une grande difficulté à convenablement inférer quelle URI inclure dans la requête.

Ainsi, la principale problématique de cette tâche consiste en la gestion des URIs. C'est-à-dire générer le bon ensemble d'URIs pour une requête donnée et les placer aux bons endroits dans la requête.

1.2.3 Adaptabilité du mécanisme de copie

L'approche de la plupart des études récentes sur la tâche de génération de requêtes SPARQL consiste en l'annotation des URIs attendues dans la question, ce qui facilite la génération de requêtes contenant les bonnes URIs. Ces méthodes ont permis de très grand progrès dans la qualité des générations des modèles de traduction automatique.

Deux classes de méthodes sont généralement employées pour effectuer ce transfert. D'une part, on peut utiliser des modèles pré-entraînés qui ont intégré la capacité de comprendre quels éléments du texte source intégrer dans le texte généré. D'autre part, on peut utiliser une fonction paramétrique particulière, un bloc supplémentaire ajouté au modèle, conçu spécifiquement pour choisir si certains éléments de la question doivent être copiés dans la requête. On appelle ce genre de fonction un mécanisme de copie.

Ces deux approches ont été utilisées indépendamment, mais il peut être pertinent de se demander si les utiliser ensemble permet d'obtenir de meilleures performances. Un des éléments de la problématique de ce mémoire est donc de vérifier si l'ajout d'un mécanisme de copie sur des modèles pré-entraînés permet l'amélioration de la génération de requêtes SPARQL.

1.2.4 Comparabilité des approches de référence

Par ailleurs, les études de référence utilisent souvent des jeux de données différents pour l'entraînement et l'évaluation de leurs modèles. Les métriques utilisées pour évaluer les performances sont aussi peu cohérentes d'une étude à l'autre. Certaines études comparent uniquement les requêtes prédites et attendues alors que d'autres comparent directement les réponses renvoyées par les requêtes. Cependant, même en utilisant les mêmes métriques, les méthodes pour les calculer ne sont pas toujours les mêmes. Les sous-sections précédentes mettent également en exergue le fait que les différentes méthodes d'annotation, ainsi que les deux méthodes de transfert des URIs de la question vers la requête, ne sont pas comparées.

1.2.5 Généralisation

Finalement, avec l'augmentation des performances des approches de référence, la question de la capacité des jeux de données à fournir un cadre d'apprentissage et d'évaluation efficace pour les modèles se pose. Notamment, la question de la généralisation à des données suivant

différentes distributions est une problématique majeure dans de nombreux domaines de l'apprentissage machine [10]. Dans le cadre des modèles dits de séquence à séquence (la classe de famille utilisée pour la traduction automatique comme introduit en Section 1.1.3), le problème de la généralisation compositionnelle est un enjeu majeur [11], c'est-à-dire, la question de savoir si un modèle est capable d'intégrer les règles de composition qui régissent l'agencement des mots des langues qu'ils interprètent et génèrent. Dans le cadre de la génération de requêtes SPARQL, cela peut s'assimiler à deux difficultés spécifiques de la tâche.

En effet, d'une part, comme évoqué précédemment, la présence d'URIs inconnues lors de l'usage du modèle est inévitable à cause de la taille des BC utilisées. Cependant, un modèle capable de comprendre comment les URIs sont générées en fonction de la position dans la phrase du concept relié et des mots qui l'entourent devrait réussir à gérer ce problème.

D'autre part, l'un des soucis majeurs des jeux de données actuels utilisés pour la tâche est le fait qu'ils sont générés automatiquement en suivant un nombre limité de structures. Cela pose la question de l'impact de l'usage de ces quelques structures. On se demande donc si les modèles sont capables d'intégrer le sens des questions et les règles pour en déduire une requête SPARQL équivalente, ou s'ils ont juste appris comment assimiler une structure de question à une structure de requête parmi celles rencontrées pendant l'entraînement.

Ce mémoire pose donc également la problématique de savoir comment les modèles sont capables de généraliser au-delà de leurs données d'entraînement et plus précisément comment ils se comportent face à des URIs inconnues ou face à des structures inconnues.

1.3 Objectifs de recherche

Étant données les problématiques soulevées dans la section précédente, ce mémoire s'intéresse à trois principaux objectifs de recherche.

On a soulevé la problématique du lien entre le langage naturel et la BC. Ce point est également directement lié à la problématique des concepts hors vocabulaire. Les méthodes classiques pour résoudre ces problématiques impliquent l'annotation de la question par les éléments de la BC et l'utilisation d'une architecture permettant le transfert de ces éléments vers la requête. On a vu que les deux méthodes principalement utilisées sont les modèles pré-entraînés [8] et le mécanisme de copie [7]. Bien que ces méthodes aient déjà été étudiées indépendamment, la problématique du manque de comparabilité les unes aux autres rend difficile l'analyse de l'impact de chaque méthode. Ce mémoire pose donc directement les questions de recherche suivantes.

1. Quel est l'impact de la méthodologie d'annotation des éléments de la base de connais-

sances sur les capacités de génération de requêtes SPARQL ?

2. Quel est l'impact du mécanisme de copie sur la capacité de génération de requêtes SPARQL par des modèles de traduction automatique ?
3. Quel est l'impact du pré-entraînement sur la génération de requêtes SPARQL ?

Les approches de pointe utilisent deux stratégies de transfert des éléments annotés dans la question vers la requête indépendamment. Cependant, l'une de ces méthodologies (l'usage de modèles pré-entraînés) repose sur les poids du modèle et l'autre (le mécanisme de copie) sur l'ajout d'un bloc supplémentaire à la suite du modèle de base. Ces deux méthodologies peuvent donc potentiellement être complémentaires. Ce qui soulève la question de recherche suivante.

4. L'association du mécanisme de copie et du pré-entraînement permet-elle de dépasser les performances obtenues par l'usage de ces approches indépendamment ?

Finalement, ce mémoire étudie également une problématique annexe soulevée par la hausse des performances de modèles de traduction automatique. Il s'agit de la capacité de ces modèles à généraliser ce qu'ils ont appris à des données possédant des caractéristiques différentes de celles vues en entraînement. Cela nous mène à la dernière question de recherche suivante.

5. Les modèles de traduction automatiques sont-ils capables de généraliser à des URIs ou à des structures de questions-requêtes inconnues ?

1.4 Plan du mémoire

Afin de répondre aux questions de recherche indiquées dans la section précédente, ce mémoire s'organisera en cinq chapitres. Le chapitre 1 introduit le sujet et les concepts de base ainsi que les éléments de problématiques et les questions de recherche associées à ce mémoire. Le chapitre 2 présente l'état de la littérature sur la tâche de génération de requêtes SPARQL par traduction automatique ainsi que l'étude de leur capacité de généralisation. Le chapitre 3 décrit l'approche de ce mémoire pour la génération de requêtes SPARQL. Il présente les jeux de données, les modèles et la méthodologie expérimentale utilisés pour les expériences de ce mémoire ainsi que leurs résultats. Le chapitre 4 présente l'étude des capacités de généralisation de ces modèles. On y expose les définitions de généralisation utilisées, puis les méthodes pour obtenir des nouvelles séparations des jeux de données et enfin les résultats des expériences sur ces nouvelles séparations. Le chapitre 5 conclut ce mémoire en proposant un résumé des travaux réalisés, puis en décrivant les limitations de l'approche utilisée et les améliorations futures envisagées.

CHAPITRE 2 REVUE DE LITTÉRATURE

Ce chapitre présente plus en détail les bases théoriques nécessaires pour expliquer les approches de la littérature. Dans un premier temps, on décrit les principales architectures neuronales utilisées pour la tâche. Dans un second temps, on présente les principales métriques utilisées pour évaluer les performances de ces modèles.

Ce chapitre présente ensuite l'état de la recherche dans le domaine de ce mémoire. Il passe en revue les techniques développées pour la traduction automatique neuronale de requêtes SPARQL avec les modèles traditionnels non pré-entraînés et les études plus récentes utilisant des modèles pré-entraînés ainsi que les études utilisant un mécanisme de copie. Enfin, les études portant sur la généralisation des systèmes de génération de requêtes SPARQL sont également décrites.

2.1 Architectures neuronales pour la traduction automatique de requêtes SPARQL

2.1.1 Architecture encodeur-décodeur

Concepts de base. Les architectures encodeurs décodeurs sont une approche pour résoudre les problèmes de séquence à séquence [12]. C'est-à-dire des problèmes dans lesquels l'entrée et la sortie sont des séquences de taille variables et dont l'ordre des mots a une importance. La traduction automatique est une tâche de séquence à séquence, car la question et la requête sont des séquences de jetons de tailles variables. Les modèles encodeurs décodeurs sont constitués de deux blocs. Chaque bloc est constitué de plusieurs couches qui ont pour but de transformer les représentations vectorielles de chaque mot d'une phrase donnée en intégrant le contexte de ce mot. Un schéma de l'architecture est représenté dans la Figure 2.1.

L'encodeur. Soit une séquence d'entrée $w_{0:n}$. L'encodeur utilise un vocabulaire prédéfini pour associer à chaque jeton w_i une représentation initiale $\mathbf{w}_i$. On obtient alors une matrice de plongement initiale de la séquence d'entrée $E_0 \in \mathbb{R}^{n \times H_{enc}}$ avec H_{enc} la taille des représentations internes de l'encodeur. L'encodeur transforme ensuite les représentations des mots en des représentations contextuelles prenant en compte tous les autres mots de la phrase. Pour cela, l'encodeur utilise une série de L_E opérations Enc_k telles que la modification de représentation d'un jeton w_i à la couche k s'écrit $\mathbf{w}_i^k = \text{Enc}_k(\mathbf{w}_i^{k-1} | \mathbf{w}_{0:n}^{k-1})$. On obtient donc en sortie des couches d'encodage une matrice E_{L_E} dans laquelle chaque jeton est représenté

par un vecteur qui intègre son contexte.

Le décodeur. Le décodeur va ensuite générer la séquence de sortie itérativement. Pour cela, on passe à chaque étape la phrase générée jusqu'à présent et le décodeur prédira le jeton suivant. On commence donc par uniquement passer le jeton de début de phrase $\langle \text{bos} \rangle$. À chaque étape i , le décodeur prend en entrée une séquence de mots déjà prédits $t_{0:i-1}$ qui sont, eux aussi, mis sous la forme d'une matrice $D_0 \in \mathbb{R}^{i \times H_{dec}}$, avec H_{dec} la taille des plongements du décodeur. Cette matrice est ensuite transformée en passant par les différentes couches du décodeur. De la même manière que dans l'encodeur, cette représentation subit L_D opérations notées Dec_k , pour $k \in \{1, \dots, L_D\}$. Chaque opération correspond à une couche du décodeur. On appelle donc L_D le nombre de couches du décodeur. Cependant, dans le cas du décodeur, ces opérations consistent en deux étapes. D'une part, une fonction $\text{Dec}_k^{target}(\mathbf{t}_i^{k-1} | \mathbf{t}_{0:i-1}^{k-1})$ intègre les représentations des autres jetons de la phrase générée jusqu'à présent. Il est important de noter que dans le cas du décodeur, cette fonction utilise uniquement les jetons antérieurs à celui auquel on applique la fonction. On utilise pour cela un masque qui remplace chaque jeton postérieur à celui auquel est appliquée la fonction par des vecteurs vides. Ce masque est appelé masque d'attention causale. D'autre part, on a une fonction $\text{Dec}_k^{input}(\mathbf{t}_i^{k-1} | \mathbf{w}_{0:n}^{L_E})$ qui intègre le sens de la phrase d'entrée. À la sortie de toutes les couches de décodage, on obtient une matrice D_{L_D} qui représente chaque jeton en intégrant le sens des autres jetons de la séquence de sortie ainsi que le sens de la phrase d'entrée. Cette matrice de dimension $i \times H_{Dec}$ est ensuite transformée par une opération linéaire en une autre matrice de taille $i \times T$ où T est la taille du vocabulaire de sortie. On applique ensuite une fonction softmax à cette matrice afin d'obtenir une distribution de probabilité sur le vocabulaire de sortie. On obtient alors pour chaque jeton la probabilité du jeton suivant. On peut alors reproduire l'opération à l'étape $i + 1$ en fixant le jeton $i + 1$ comme celui ayant la plus grande probabilité de suivre le jeton i . Les distributions de probabilité finale produite par le décodeur sont souvent appelé "logits".

Toutes les architectures utilisées dans ce mémoire utilisent la même fonction Dec^{input} appelée mécanisme d'attention croisée, mais diffèrent par la nature des fonctions Enc et Dec^{target} .

Mécanisme d'attention croisée. Toutes les architectures utilisées ici utilisent la même fonction Dec^{input} . Cette fonction consiste en une moyenne convexe de vecteurs calculés à partir des représentations de sortie de l'encodeur. Étant donné le jeton auquel la fonction est appliquée $\mathbf{t}_i^k$ et les sorties de l'encodeur $\mathbf{w}_j^{L_E}$ avec $j \in \{0, \dots, n\}$, on calcule les vecteurs dits de clef $k_i = K_{CA} \times \mathbf{t}_i^k$, de requête $q_j = Q_{CA} \times \mathbf{w}_j^{L_E}$ et de valeur $v_j = V_{CA} \times \mathbf{w}_j^{L_E}$. Où K_{CA} , Q_{CA} et V_{CA} sont les matrices apprises durant l'entraînement. On calcule alors les scores d'attention

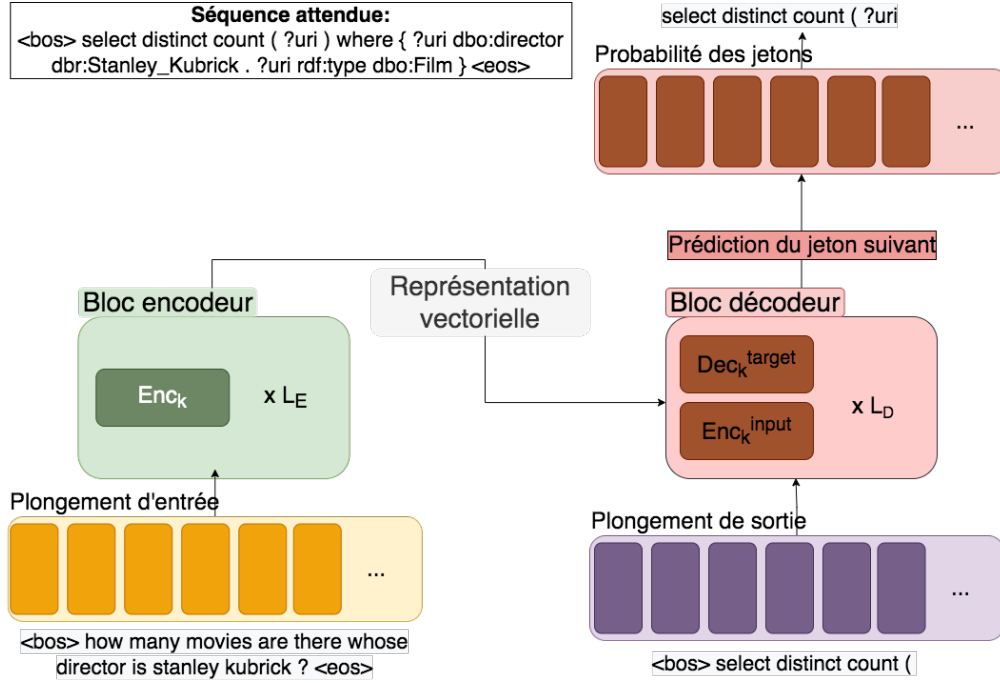


FIGURE 2.1 L'architecture encodeur-décodeur.

croisée entre $\mathbf{t}_i^k$ et chaque $\mathbf{w}_j^{L_E}$ par produit scalaire pondéré du vecteur de clef et du vecteur de requête : $S_{i,j} = \frac{q_j \cdot k_i}{\sqrt{d}}$ où d est la dimension des vecteurs q_j , k_i et v_j . Puis, on obtient le poids d'attention croisée par application d'une fonction softmax sur les scores d'attention croisée : $W_{i,j} = \text{softmax}(S_{i,j})$. On obtient alors la sortie de la fonction appliquée à $\mathbf{t}_i^k$ en faisant la somme des valeurs v_j pondérée par les poids d'attention croisée $W_{i,j}$: $\text{Dec}^{\text{input}}(\mathbf{t}_i^k | \mathbf{w}_{0:n}^{L_E}) = \sum_{j=0}^n W_{i,j} \times v_j$. On peut alors noter de façon matricielle $\text{Dec}^{\text{input}}(D_i | E_{L_E}) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$. Notamment, on obtient également une matrice $W = \text{softmax}(\frac{QK^T}{\sqrt{d}})$ des poids d'attention croisée, qui peut être interprétée tel que chaque poids $W_{i,j}$ représente l'impact du jeton w_j pour la génération du jeton t_i .

2.1.2 Architectures non pré-entraînées

Les modèles décrits ici sont les architectures utilisées dans les articles [7, 13] rapportant les meilleures performances avec des modèles de base dit non pré-entraînés, par opposition aux modèles pré-entraînés décrits dans la partie suivante. Ces deux architectures utilisent le mécanisme d'attention croisée comme fonction $\text{Dec}^{\text{input}}$, cependant elles se distinguent par différentes fonctions Enc et $\text{Dec}^{\text{target}}$.

Modèle séquence-à-séquence convolutif. Le modèle Séquence-à-Séquence Convolutif (ConvSeq2seq) [3] est un encodeur-décodeur qui utilise des opérations de convolutions pour intégrer le contexte d'un jeton dans sa représentation vectorielle transformée. Ainsi, les matrices apprises lors de l'entraînement de ce modèle sont des filtres de convolutions. Pour chaque couche k , la fonction f_k associée à Enc_k ou à $\text{Dec}_k^{\text{target}}$ peut être écrite ainsi : $f_k(x_i^k | x_{0:f}^{k-1}) = v([W_k \ x_{i-\frac{l}{2}:i+\frac{l}{2}}] + b_k) + x_i^k$. Ici, x_i^k est le jeton auquel la fonction est appliquée, $x_{0:f}^{k-1}$ est la représentation de la phrase totale en sortie de la couche précédente, conv est la fonction de convolution incluant une non-linéarité σ telle que pour deux vecteurs A et B on a $v([A \ B]) = A * \sigma(B)$, $b_k \in \mathbb{R}^{2H}$ (avec H la taille des vecteurs de représentation) est un vecteur de biais, l est la taille de la fenêtre de convolution et $W_k \in \mathbb{R}^{2H \times lH}$ est le filtre de convolution de la couche k .

Modèle Séquence-à-Séquence Transformer. Dans le cas du Transformer [2], les transformations de chaque couche sont obtenues par le mécanisme d'auto attention. Ce mécanisme est similaire à celui de l'attention croisée à la différence que les vecteurs de clef et de valeur sont calculés avec les représentations de la phrase d'où provient le jeton auquel la fonction est appliquée. Ainsi les fonctions Enc_k et $\text{Dec}_k^{\text{target}}$ peuvent être écrites sous la forme d'une fonction $f_k(x_i^k | x_{0:f}^{k-1}) = \sum_{j=0}^n \text{softmax}(\frac{q_j \cdot k_i}{\sqrt{d}}) \times v_j$ avec $k_i = K_{AA} \times x_i^{k-1}$, $q_j = Q_{AA} \times x_j^{k-1}$ et $v_j = V_{AA} \times x_j^{k-1}$ où K_{AA} , Q_{AA} et V_{AA} sont les matrices apprises durant l'entraînement associées au mécanisme d'auto attention.

2.1.3 Architectures pré-entraînées

Transfert d'apprentissage. L'une des premières approches de pré-entraînement est BERT [4]. Il s'agit d'un encodeur de l'architecture Transformer sur lequel différentes couches linéaires permettent de réaliser différentes tâches. Son pré-entraînement est basé sur deux tâches : le remplissage de masque et la prédiction de phrase suivante. Dans le premier cas, des phrases sont présentées au modèle dans lesquelles des jetons sont masqués. Le modèle utilise alors une couche linéaire afin de prédire pour chaque jeton d'entrée sa distribution de probabilité sur le vocabulaire. Ainsi, le modèle doit prédire quels mots se trouvent à la place du jeton masqué. La seconde tâche utilise une suite de deux phrases séparées par un jeton **<sep>**. Une couche linéaire est ajoutée au modèle pour une tâche de classification binaire consistant à prédire si la deuxième phrase peut apparaître à la suite de la première.

À la suite de BERT [4] de nombreux modèles pré-entraînés ont été développés qui varient suivant leurs architectures, leurs tâches de pré-entraînement et leur corpus de pré-entraînement. Ce mémoire utilise deux modèles pré-entraînés qui suivent une architecture d'encodeur-

décodeur Transformer, à savoir BART [5] et T5 [6]. Ces deux architectures ont déjà été utilisées pour la génération de requêtes SPARQL [8] et ont démontré de très bonnes performances sur la tâche. Ce mémoire suit les hyper-paramètres de la précédente sur le sujet [8].

BART. BART [5] est un modèle pré-entraîné basé sur l’architecture Transformer. Sa structure et son pré-entraînement sont inspirés de BERT [4], l’un des premiers Transformers pré-entraînés. La taille des unités cachées (c’est-à-dire la dimension des vecteurs de représentation interne des jetons) et le nombre de couches superposées sont des hyper-paramètres. Le pré-entraînement repose sur le concept de dé-bruitage. Des phrases sont perturbées par plusieurs opérations de bruitage et le modèle doit reconstituer la phrase initiale. Les opérations sont les suivantes :

- Masquage de jeton : certains jetons de la phrase d’entrée sont remplacés par le jeton `<mask>`. Cette opération est directement inspirée de la méthodologie de pré-entraînement de BERT.
- Suppression de jeton : certains jetons sont supprimés et, contrairement au masquage, le modèle doit trouver les positions dans le texte ou ajouter des jetons.
- Remplissage de texte : un nombre variable (potentiellement nul) de jetons est remplacé par le jeton `<mask>`. Cela permet au modèle de savoir combien de jetons doivent être générés.
- Permutation de phrases : les phrases d’un document sont permutées. Cela permet au modèle de comprendre l’enchaînement des phrases.
- Rotation de document : un jeton est choisi au hasard comme début du document, puis les jetons suivants sont placés à la suite jusqu’au dernier jeton du document et finalement les jetons précédents sont ajoutés. Cela permet au modèle de déterminer quel est le début du document.

Ce modèle a ensuite été testé avec affinage sur plusieurs tâches de TALN telles que la traduction ou le résumé et montre de très bonnes performances sur des jeux de données classiques de ces tâches.

Sur la tâche de génération de requêtes SPARQL, ce modèle est supposément [8] moins bon que le suivant (T5), mais produit tout de même d’assez bonnes performances. En effet, son pré-entraînement basé sur la modification d’une phrase d’entrée permet naturellement le transfert d’élément de l’entrée vers la sortie. Cela permet de gérer les URIs inconnues lorsque celles-ci sont annotées dans la question.

T5. Le modèle T5 [6] (Text-to-Text Transfer Transformer) est un modèle pré-entraîné basé sur l'architecture Transformer. Contrairement à BART, le pré-entraînement de T5 inclut un mélange de tâches non supervisées et de tâches supervisées. Leur approche est basée sur le passage de toute tâche de TALN sous une forme de problème séquence à séquence.

La tâche non supervisée utilise le jeu de données C4 (Colossal Clean Crawled Corpus) fabriquée à partir de données textuelles du web par les heuristiques suivantes :

- Ils gardent uniquement les lignes qui finissent par un signe de ponctuation.
- Ils gardent uniquement les pages avec plus de cinq lignes et les lignes avec plus de trois mots.
- Ils retirent les phrases avec des mots inappropriés.
- Ils retirent toutes les pages qui contiennent le mot "JavaScript" ou "lorem ipsum".
- Ils retirent toutes les pages avec le signe "{".
- Ils ne gardent qu'une page lorsque plusieurs pages ont trois phrases en commun de suite.

La tâche non supervisée consiste alors en la reconstitution de phrases dans lesquelles 15% des jetons ont été masqués.

Les tâches supervisées sont celles de références de GLUE [14], c'est-à-dire :

- Jugement d'acceptabilité des phrases
- Analyse de sentiment
- Reformulation/similarité de phrases
- Inférence en langage naturel
- Résolution de coréférence
- Désambiguïsation du sens des mots
- Réponse aux questions

Les auteurs de T5 développent une méthodologie particulière de reformulation de l'entrée et de la sortie pour chaque tâche. Le résultat de ce transfert d'apprentissage bien plus riche est que T5 possède de bien plus grande connaissances de la langue naturelle et est bien plus efficace sur de très nombreuses tâches, y compris sans affinage.

Cependant, la particularité de ces tâches de pré-entraînement est qu'elles sont toutes complètement orientées sur du langage naturel et aucune ne contient de code. L'adaptation à la tâche de génération de requêtes SPARQL est donc un défi et nécessite un affinage.

Une autre spécificité du modèle T5 est qu'il contient dans son vocabulaire un certain nombre de jetons dits "sentinelles" qui sont utilisés pour indiquer au modèle des indications sur la tâche sur laquelle il est affiné. Ces jetons n'ont pas de sens propre avant l'affinage, mais permettent une souplesse sur l'usage du vocabulaire de T5 si de nouveaux jetons sont nécessaires.

2.1.4 Architecture de copie

La principale difficulté pour les modèles non pré-entraînés pour la tâche de génération de requêtes SPARQL est la génération des URIs correctes dans la requête. Afin de dépasser cette difficulté, les approches les plus performantes de l'état de l'art [7,8] annotent lesdites URIs dans la question et utilisent des modèles capables de transférer ces éléments de la question vers la requête. Dans le cas de [8], l'usage de modèles pré-entraînés permet ce transfert. Dans le cas de [7], l'approche utilise des modèles non pré-entraînés. Il faut donc inclure dans le modèle un mécanisme supplémentaire capable d'effectuer ce transfert. Des mécanismes de copie ont été introduits dans la littérature [15, 16] pour répondre à la problématique de transfert d'éléments du texte source vers le texte généré d'un encodeur décodeur. Ces mécanismes ne sont cependant pas adaptés à une question annotée et ne peuvent pas viser des éléments particuliers du texte source.

Ainsi, un mécanisme de copie spécifique a été développé pour répondre à cette problématique dans le cadre de la génération de requêtes SPARQL. Ce mécanisme de copie a été conçu au sein du laboratoire LAMA-WeST et a été présenté pour la première fois dans [7]. Une partie des expériences rapportées dans ce papier ont été réalisées dans le cadre de ce mémoire et sont les bases des expériences qui y sont décrites dans les sections suivantes. Ce mémoire utilise le même mécanisme que présenté dans [7] mais en l'adaptant à une nouvelle méthode d'annotation (tag-final) et en étendant son usage aux modèles pré-entraînés. La suite de cette section décrit le fonctionnement de ce mécanisme de copie.

Ce mécanisme de copie utilise trois vocabulaires différents. Premièrement, le vocabulaire de langue naturelle W qui est composé des mots des questions non annotés. Deuxièmement, le vocabulaire de la base de connaissances K qui est composé des URIs et des littéraux inclus dans les requêtes et annotés dans les questions. Troisièmement, le vocabulaire des mots clefs SPARQL S qui structurent les requêtes. Ce mécanisme est conçu de manière que le vocabulaire de l'encodeur soit limité à W , celui du décodeur à S et que les jetons de K soient les seuls pouvant être copiés. Dans le cas de question dont les annotations remplacent leur formulation en langue naturelle, cela permet une très large diminution du vocabulaire de l'encodeur. Dans tous les cas, cela permet une très large diminution du vocabulaire du

décodeur. Les détails sur l'échelle de cette diminution sont donnés dans le chapitre 3. De plus, les modifications apportées par cette version du mécanisme de copie permettent aux modèles de gérer indifféremment les jetons inconnus et connus de K lors de l'évaluation.

Le mécanisme de copie de ce mémoire utilise la même architecture neuronale que le mécanisme PGN [16]. Il calcule, à chaque étape de génération, une probabilité de copie pour le prochain jeton. Cette probabilité pondère le score de génération et le score de copie du prochain jeton. Pour chaque mot du vocabulaire cible, le score de génération est basé sur les logits du décodeur. Le score de copie est basé sur le poids d'attention croisée entre le prochain jeton prédit et chaque jeton de l'entrée. Contrairement à PGN [16], cette implémentation du mécanisme de copie est conçue pour copier uniquement des jetons d'un vocabulaire spécifique, ici le vocabulaire de la BC K . Lorsque la phrase d'entrée est fournie à l'encodeur, les mots de ce vocabulaire sont masqués. Ensuite, les scores de copie sont uniquement calculés pour ces jetons masqués. Le masquage des jetons de K est l'étape clef permettant la réduction de taille des vocabulaires de l'encodeur et du décodeur ainsi que la gestion des jetons inconnus de K .

Plus formellement, à l'étape de génération i , la probabilité de génération d'un jeton dans $S \cup K$ est calculée de la manière suivante :

$$P(t_i|w_{0:m}; t_{0:i-1}) = P_{copy}(t_i|\tilde{w}_{0:m}; t_{0:i-1}) \times P_C(t_i|w_{0:m}; t_{0:i-1}) + (1 - P_{copy}(t_i|\tilde{w}_{0:m}; t_{0:i-1})) \times P_G(t_i|\tilde{w}_{0:m}; t_{0:i-1}) \quad (2.1)$$

Où t_i est le i -ème jeton généré, $\tilde{w}_{0:m}$ sont les jetons donnés en entrée à l'encodeur où les jetons de K ont été masqués, $w_{0:m}$ sont les jetons d'entrée d'origine avant masquage, P_{copy} est la probabilité d'utiliser la copie au lieu de la génération, P_C est la distribution de probabilité sur les jetons qui peuvent être copiés, et P_G est la distribution de probabilité sur les jetons qui peuvent être générés.

On peut exprimer P_G , P_C et P_{copy} de la manière suivante :

$$P_G(t_i|\tilde{w}_{0:m}; t_{0:i-1}) = \begin{cases} \text{softmax}_S(\text{Dec}_i(t_i|\tilde{w}_{0:m}; t_{0:i-1})) & \forall t_i \in S \\ 0 & \forall t_i \notin S \end{cases} \quad (2.2)$$

$$P_C(t_i|w_{0:m}; t_{0:i-1}) = \begin{cases} \text{softmax}_{K \cap w_{0:m}}(W_{k,i}) & \forall t_i \in K \text{ t.q. } \exists k : w_k = t_i \\ 0 & \text{sinon} \end{cases} \quad (2.3)$$

$$P_{copy}(t_i|\tilde{w}_{0:m}; t_{0:i-1}) = \sigma(\text{Dec}_i(t_i|\tilde{w}_{0:m}; t_{0:i-1})) \times B \quad (2.4)$$

Où Dec_i sont les logits du décodeur à l'étape i , $W_{k,i}$ est le poids de l'attention croisée calculé par l'encodeur-décodeur entre le k -ième terme de l'entrée et le i -ième terme de la sortie, $w_i \in W \cup K$ est le jeton à la i -ième position dans la phrase d'entrée avant d'être masqué, $B \in \mathbb{R}^{|S| \times 1}$ est une matrice avec des poids appris lors de l'entraînement et σ est la fonction sigmoïde.

La figure 2.2 montre un résumé de la façon dont le bloc de copie est ajouté au-dessus du modèle de base. Avec ce mécanisme de copie, le vocabulaire de l'encodeur est W , le vocabulaire du décodeur est S , et seuls les jetons de K peuvent être copiés de la question vers la requête.

2.2 Métriques d'évaluation de la tâche

L'évaluation de la tâche de génération de requêtes SPARQL nécessite la comparaison entre deux requêtes. Cette tâche se situe à l'intersection entre la tâche de question-réponses sur une base de connaissances et la tâche de traduction automatique. On trouve dans la littérature des métriques liées à la capacité des modèles à générer des requêtes proches des requêtes attendues en termes des jetons qui la composent, telles que le score BLEU [17]. Mais, on trouve également des métriques qui comparent directement les réponses des requêtes générées aux réponses des requêtes attendues, telles que la précision des réponses et le score F1. Ce mémoire utilise ces trois métriques qui sont présentées plus en détails dans le reste de cette section.

2.2.1 Score BLEU

Le score BLEU (Bilingual Evaluation Understudy) est une métrique très utilisée dans le domaine de la traduction automatique. Elle compare les n -grams (suite de n jetons) présents dans un candidat de traduction avec ceux dans une référence. Elle peut être interprétée comme un score de précision corrigé pour tenir compte de la taille du candidat. L'avantage de cette métrique est qu'elle permet de distinguer deux requêtes fausses en indiquant laquelle est la plus proche de la référence. Elle est donc très utilisée dans les travaux ayant des scores relativement faibles et dont les scores basés sur les réponses seraient trop faibles pour permettre une évaluation efficace. Cette métrique est très adaptée à la traduction automatique entre deux langues naturelles, car elle coïncide avec la proximité sémantique entre deux textes. Cependant, pour la tâche de génération de requêtes SPARQL, elle est assez limitée. Comme SPARQL est un langage formel, changer certains mots peut rendre la requête non

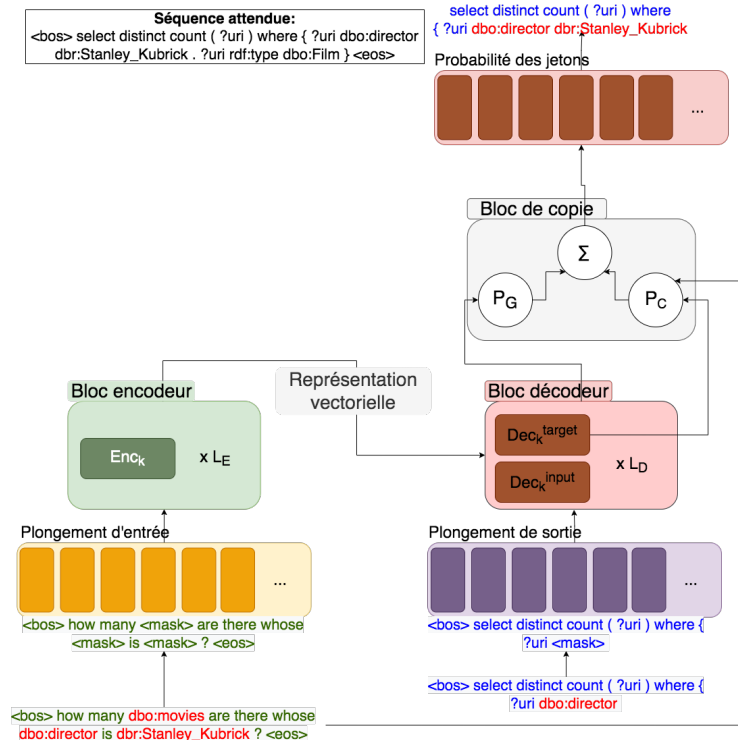


FIGURE 2.2 Illustration du mécanisme de copie

fonctionnelle tout en gardant un bon score BLEU. Ainsi, cette métrique permet de mettre en avant des facilités ou des difficultés d'un modèle à générer les bons jetons fréquemment, mais elle n'évalue pas la qualité des requêtes à être correctes.

2.2.2 Précision des réponses

Cette métrique est très utilisée dans le domaine des questions à des bases de connaissances. On peut également l'appeler métrique d'exactitude et se dit "accuracy" en anglais. Elle compte le nombre de réponses renvoyées par les requêtes candidates qui sont égales aux réponses renvoyées par les requêtes de référence. Elle est très peu permissive, car elle mesure exactement la réussite du modèle sur la tâche de produire une requête équivalente à la question posée. Elle est aussi très peu nuancée, car pour chaque entrée de l'ensemble de test, la moindre erreur peut conduire à une réponse différente. Il n'y a donc pas la possibilité pour une réponse d'être "proche" de la réponse attendue. Cette métrique peut également être appelée exactitude.

Pour une question donnée ayant un ensemble de réponse attendue et dont la requête proposée par le modèle a fourni un ensemble de réponses candidates, on a :

$$\text{précision} = \begin{cases} 1 & \text{si réponses de référence} = \text{réponses candidates} \\ 0 & \text{sinon} \end{cases}$$

On fait ensuite la moyenne de la précision calculée sur chaque entrée pour obtenir la métrique de précision du modèle sur un jeu de test.

2.2.3 Score F1 des réponses

Cette métrique permet d'apporter de la nuance à la métrique précédente. En effet, comme la plupart des réponses attendues sont des ensembles, on peut comparer si l'ensemble candidat est proche de l'ensemble de référence. Pour une question donnée, le score de précision (accuracy) prendra uniquement les valeurs entre 0 et 1 suivant si l'ensemble de réponse est correcte ou non. Le score F1 peut prendre des valeurs intermédiaires entre 0 et 1 suivant si les réponses attendues et prédites sont plus ou moins proche. Pour une question donnée, on calcule le score F1 défini ainsi :

$$\text{F1-score} = \frac{2 \times \text{précision} \times \text{recall}}{\text{précision} + \text{recall}} \quad (2.5)$$

$$\text{précision} = \frac{|\text{réponses de référence} \cap \text{réponses candidates}|}{|\text{réponses candidates}|} \quad (2.6)$$

$$\text{rappel} = \frac{|\text{réponses de référence} \cap \text{réponses candidates}|}{|\text{réponses de référence}|} \quad (2.7)$$

On calcule ensuite le score F1 sur l'ensemble de test par la moyenne des scores F1 de chaque entrée.

Le terme précision utilisé dans la définition du score F1 se traduit en anglais par le terme "precision" et renvoie à une notion différente de celle de "précision des réponses" définie dans la sous-section précédente.

2.3 Traduction automatique pour SPARQL

2.3.1 Approche classique

L'une des premières approches qui utilise des architectures de traduction automatique neuronale pour la génération de requêtes SPARQL est présentée par Luz et al. [18]. Ils utilisent

un modèle LSTM (Long Short Term Memory) [19] comme architecture des blocs encodeur et décodeur. Au lieu d’initialiser aléatoirement les plongements initiaux, ils utilisent des vecteurs appelés plongements lexicaux. Ces vecteurs sont obtenus en minimisant le produit scalaire des représentations de mots qui peuvent être trouvés dans des contextes similaires. Cela permet d’accélérer l’entraînement et même d’obtenir des meilleures performances que lorsque l’on utilise des plongements initiaux aléatoires. De plus, ils utilisent les poids d’attention croisée obtenus par le modèle pour analyser l’impact des mots en langue naturelle dans la génération des mots des requêtes. Ils obtiennent alors un alignement entre les mots du vocabulaire de langue naturelle et les mots de SPARQL ou les URIs de la BC. Ils évaluent leurs résultats en utilisant la précision des requêtes.

Une autre approche menée par Soru et al. [1] utilise une architecture appelée NSpM (Neural SPARQL Machines). Il s’agit d’une approche qui génère un jeu de données puis entraîne et évalue un modèle dessus. Ils produisent des structures de paires de questions-requêtes avec des espaces vides réservés. Ces espaces sont ensuite "remplis" par des concepts spécifiques de la base connaissance pour donner des paires de questions-requêtes utilisables. Les données produites sont consignées dans le jeu de données Monument. Ensuite, ces données sont fournies à un modèle d’encodeur-décodeur basé sur l’architecture Recurrent Neural Network (RNN) [20]. Ils évaluent les performances de leurs modèles à l’aide du score BLEU [17].

Ces deux approches utilisent des modèles de TALN neuronal qui ne sont plus utilisés aujourd’hui car de nouveaux modèles ont été ensuite développés et montrent régulièrement de bien meilleures performances. Yin et al. [13] utilisent des modèles plus à jour, les Transformers [2] et les modèles séquence-à-séquence convolutifs, ConvSeq2Seq [3]. Ils évaluent leurs performances sur les jeux de données Monument [1], LC-QuAD 1.0 [21] et DBNQA [22]. Ils évaluent les requêtes prédites avec le score BLEU [17], la perplexité et le score F1 entre les jetons de la requête attendue et ceux de la requête générée. Ces trois scores calculent de manière légèrement différente à quel point les jetons de la requête prédite et de la requête attendue sont similaires.

L’approche de Yin et al. [13] permet une nette amélioration des performances sur le jeu de données Monument [1] et utilise deux jeux de données plus complexes contenant nettement plus de données et couvrant des thèmes largement plus génériques au sein des bases de connaissances utilisées. Cependant, la qualité des prédictions étant toujours faible, peu de requêtes produisent des réponses justes et aucune métrique ne porte directement sur la proportion de réponses correctes. Hirigoyen et al. [7] reprennent les mêmes jeux de données et les mêmes architectures neuronales en introduisant, d’une part, une annotation des questions et, d’autre part, un mécanisme permettant le transfert de ces éléments annotés de la

question vers la requête. Cette étude a permis de mettre en avant l’impact de l’annotation des questions sur la génération des URIs. De plus, elle a montré que le mécanisme de copie utilisé permettait de gérer les URIs inconnues de l’ensemble de test. Cela a permis une forte amélioration des performances de références sur les trois jeux de données utilisés en termes de score BLEU. La précision des réponses y est également rapportée et fixe une référence de performances également par rapport à cette métrique. Cette étude a été faite dans le laboratoire LAMA-WeST et ce mémoire en est la continuation.

2.3.2 Transfert d’apprentissage

Suite au développement de l’architecture Transformer et à la suite du succès de BERT [4] dans de nombreux champs du TALN, le transfert d’apprentissage devient très populaire dans la traduction automatique. Plusieurs types de modèles pré-entraînés se popularisent : les encodeurs pré-entraînés tels que BERT [4], les décodeurs pré-entraînés tels que GPT2 [23] ou les modèles encodeurs décodeurs tels que BART [5].

Suite au développement de ces techniques, certaines approches utilisent des modèles pré-entraînés pour la génération de requêtes SPARQL. Huang et al. [24] utilisent un algorithme d’annotation pour trouver les bonnes URIs à générer. Ils remplacent les mots des questions qui correspondent à ces URIs par des jetons spéciaux. Ils affinent ensuite des modèles pré-entraînés (BERT [4], GPT2 [23] et BART [5]) pour générer des requêtes contenant les mêmes jetons spéciaux. Ces jetons spéciaux sont ensuite remplacés par l’URI correspondante dans une étape de traitement a posteriori. Ils génèrent ainsi plusieurs requêtes possibles à l’aide d’une recherche par faisceau. Cette méthode consiste à considérer plusieurs jetons possibles à chaque étape lors de la génération de la requête. Cela permet d’obtenir une liste de requêtes ordonnées selon la probabilité de les générer. Plus de détails sur cette méthode sont donnés dans la section 3.3.4. Ils récupèrent ensuite les réponses de ces requêtes et évaluent leurs modèles à l’aide de la métrique Hit@1. Cette métrique est équivalente à la précision des requêtes. Ils l’appellent Hit@1 pour suggérer le fait qu’ils utilisent uniquement la requête la plus probable de la liste obtenue pour calculer la métrique.

Une autre approche introduit une modification plus poussée de GPT2 [23]. Rony et al. [25] développent le modèle SGPT (SPARQL-GPT) qui intègre diverses informations sémantiques telles qu’un étiquetage morphosyntaxique et une annotation facultative des questions avec les éléments de la base de connaissances correspondant aux concepts qui s’y trouvent. Comme pour [24], la question contient des jetons spéciaux qui sont utilisés dans la génération de la requête et permettent le transfert d’éléments. Leur évaluation est conduite sur les jeux de données LC-QuAD 2.0 [26], VQuAnDa [27] et QALD-9 [28] en utilisant le score BLEU [17]

et le score F1 sur les jetons de la requête. Le score F1 sur les jetons de la requête mesure une similarité sémantique comparable au score BLEU en se basant sur le nombre de n-grams communs entre la requête prédite et la requête attendue. Le calcul fait pour obtenir le score F1 sur les jetons de la requête est le même que celui pour obtenir le score F1 sur les réponses, mais en considérant les n-grams de la requête attendue comme ensemble de référence et les n-grams de la requête générée comme l'ensemble candidat.

Cependant, GPT2 [23] est un modèle décodeur qui n'est pas forcément adapté à la traduction automatique. Debayan et al. [8] utilisent deux modèles encodeurs décodeurs pré-entraînés en incluant une annotation systématique a priori, tel que l'avait fait l'approche d'Hirigoyen et al. dans notre laboratoire [7]. Les modèles sont BART [5] et T5 [6]. Leur approche, comme celle de [7], permet une amélioration notable des performances qui leur permet de rapporter de bons scores de précision des réponses obtenues par les requêtes générées par les modèles. Ils testent leur approche sur les ensembles LC-QuAD 1.0 [21] et LC-QuAD 2.0 [26].

2.3.3 Mécanisme de copie

Le transfert d'éléments de la phrase source vers la phrase générée est un défi dans de nombreuses tâches de la génération de séquences en TALN. Dans le cas de la génération de requêtes SPARQL, la difficulté principale des modèles, spécifiquement ceux non pré-entraînés, est de générer convenablement les éléments de la base de connaissances, car ceux-ci sont souvent inconnus ou rares et leur sémantique risque d'être mal apprise par les modèles lors de l'entraînement. Si l'on suppose avoir un moyen de repérer ces éléments dans la question, on retrouve une problématique proche du problème de transfert classique.

Pour résoudre ce problème dans les tâches de génération de texte classique, CopyNet [15] propose d'utiliser un mécanisme de copie qui s'ajoute à une architecture encodeurs décodeurs classique en intégrant une couche dense supplémentaire capable de décider si un jeton doit être généré par le décodeur ou copié à partir de la séquence d'entrée. Une amélioration notable de ce mécanisme appelée Pointer Generator Network (PGN) [16] utilise une couche dense uniquement pour calculer une pondération entre une distribution de génération et une distribution de copie. La distribution de génération étant la distribution générée par le décodeur et la distribution de copie étant calculée par application d'un softmax sur les poids d'attention croisée entre le jeton à prédire et l'ensemble des jetons de la séquence d'entrée.

PGN [16] a été appliqué directement à la génération automatique de requêtes SPARQL [8]. Cette approche utilise une architecture LSTM [19] avec les plongements de jetons de BERT [4] concaténés à un vecteur dont toutes les valeurs sont fixées à 1.0 dans le cas de mots de langue naturelle ou à un plongement de graphe de connaissances (ici TransE [29]) pour les

URIs. Cependant, ils rapportent de beaucoup moins bons résultats que pour leurs autres expérimentations utilisant BART [5] et T5 [6].

Hirigoyen et al. [7] proposent une méthodologie particulière pour appliquer PGN [16] à la tâche de génération de requêtes SPARQL. Il s’agit de la méthode présentée dans la section précédente. La question est annotée en intégrant les URIs à la place de leur équivalent en langue naturelle dans la question. Lorsque la question est passée au modèle, les URIs sont masquées. Puis, lors du calcul des scores de copie, seuls les jetons masqués sont pris en compte. Les URIs sont ensuite remplacés dans la requête par le jeton correspondant dans la question d’après le mécanisme de copie décrit dans la section 2.1.4.

2.3.4 Comparabilité

L’une des principales limites des approches modernes de génération de requêtes SPARQL est leur manque de comparabilité. Les paramètres suivants ont une place importante dans la mesure du succès de chaque approche. Cependant, ils ne sont pas utilisés dans les mêmes conditions, ce qui rend impossible la comparaison entre les approches.

- **L’annotation des questions.** Toutes les approches performantes utilisent des questions annotées dans lesquelles sont indiquées les URIs qui doivent être présentes dans la requête. Certaines approches utilisent des annotations à la fin de la question [8] et d’autres intègrent les annotations à la place de leur équivalent en langue naturelle [7].
- **Les jeux de données utilisés.** On retrouve plusieurs jeux de données populaires dans la littérature. Notamment, les deux approches les plus performantes [7, 8] utilisent un jeu de donnée en commun. Cependant, celui-ci reste assez facile. L’une des études [8] utilise également un jeu de données plus récent et plus difficile alors que l’autre [7] en utilise un à bien plus grande échelle.
- **Les méthodologies d’évaluation.** Les méthodes classiques ont tendance à utiliser le score BLEU pour la subtilité d’évaluation qu’il permet pour des modèles peu performants. Les approches plus récentes utilisent des métriques de la tâche de question-réponse plus réalistes, telle que la proportion de réponses correctes générées. Cependant, l’usage de métriques basées sur les réponses nécessite l’exécution des requêtes sur un point d’accès (SPARQL end-point) d’une base de connaissances. Lorsque le point d’accès n’est pas adapté aux requêtes (par exemple si la base de connaissances a évolué depuis la création du jeu de données), les réponses attendues peuvent être vides et ne permettent pas d’évaluer efficacement la qualité des réponses générées par le modèle. Cette difficulté n’est pas toujours prise en compte dans la littérature, ce qui fausse les comparaisons entre les études.

- **L’usage de mécanismes spécifiques de transfert d’éléments de la question vers la requête.** Les modèles pré-entraînés [8] et l’usage d’un mécanisme de copie [7] ont chacun permis indépendamment d’importantes augmentations des performances par rapport aux modèles non pré-entraînés classiques. Cependant, le mécanisme de copie n’a jamais été associé aux modèles pré-entraînés.

2.4 Généralisation pour la génération de requêtes SPARQL

Les performances de la génération de requêtes SPARQL ont beaucoup augmenté au fil des études récentes. Notamment, [7] ou [8] atteignent des scores presque parfaits sur LC-QuAD 1.0 [21]. Cela pose la question de la performance de ces modèles dans des contextes plus complexes. L’approche de traduction automatique neuronale est associée à plusieurs difficultés lorsque les modèles doivent généraliser leur apprentissage à des données différentes de celles vues lors de leur apprentissage.

2.4.1 La généralisation compositionnelle

Les modèles de traduction automatique neuronale (et plus généralement de séquence à séquence) ont connu de très gros succès dans les dernières années grâce à la quantité de données disponible et à la puissance de calcul allouée à l’entraînement et au pré-entraînement. Cependant, contrairement aux méthodes symboliques traditionnelles, ces modèles ne fonctionnent pas en appliquant des règles explicites qui permettent de déterminer le sens d’un texte grâce à une connaissance du fonctionnement de la langue. Or, d’après Baroni [30], ce fonctionnement est lié au concept de généralisation compositionnelle. Le fait de comprendre le sens d’un verbe comme "sauter" et l’association "sauter deux fois" devrait permettre à un modèle de comprendre la différence entre "courir" et "courir deux fois" même sans jamais avoir vu le mot "courir". Baroni [30] montre que les modèles de séquence à séquence sont capables d’intégrer certaines règles implicitement en exhibant de bonnes performances d’intuition grammaticale. Cependant, il analyse également des expériences sur le jeu de données SCAN [31] qui montrent que ces modèles n’arrivent pas à intégrer des règles de compositions explicites. Ce jeu de données comporte des paires de séquences composées de phrases en langue naturelle d’une part et de séquences d’instruction symbolique équivalente d’autre part. Par exemple, la phrase "sauter deux fois" est associée à la séquence symbolique "[SAUTER] [SAUTER]". Ce jeu de données permet de tester la capacité de généralisation compositionnelle des modèles de séquence à séquence, car le jeu de test contient des actions qui ne sont pas dans le jeu d’entraînement. Si le modèle a réussi à apprendre comment "deux fois" permet de composer l’action, alors il devrait être capable d’interpréter "courir deux fois" même sans avoir vu le

terme "courir" en entraînement. Cette étude conclut que, dans ce cas, les modèles classiques de séquence à séquence ont du mal à intégrer la généralisation compositionnelle.

Cependant, Lake [10] montre qu'en modifiant l'architecture des modèles de séquence à séquence et leur méthodologie d'entraînement, il est possible d'améliorer leur capacité de généralisation compositionnelle. En utilisant le paradigme du méta-apprentissage, il arrive à augmenter les performances de référence pour le jeu de données SCAN. Son modèle de méta-apprentissage est entraîné par une séquence de tâches, c'est-à-dire est entraîné séquentiellement sur plusieurs parties du jeu de donnée, qui nécessitent une généralisation compositionnelle les unes par rapport aux autres. Le modèle est également capable de faire appel à des exemples vu précédemment pour influencer la génération d'un nouvel exemple. Cette approche permet au modèle une abstraction sur les concepts d'une phrase en le mélangeant à des concepts similaires vus dans des exemples précédents. Cela conduit à l'utilisation implicite de règle de composition par le modèle.

2.4.2 Généralisation pour la génération de requêtes SPARQL

Keysers et al. [11] ont spécifié la notion de généralisation compositionnelle pour la tâche de génération de requêtes SPARQL. Ils définissent des atomes qui peuvent être par exemple la structure de la phrase "Quel est le [concept1] qui [lien1] [concept2]" ou bien les concepts qui se trouvent dans les espaces réservés. Puis, ils définissent des "compositions" comme l'arbre d'application des axiomes qui permet d'obtenir la séquence. Ils définissent ensuite un critère pour mesurer la capacité d'un jeu de données de génération de requêtes SPARQL à généraliser de manière compositionnelle. Ce jeu de données maximise la similarité des distributions des atomes dans son jeu de test et dans son jeu d'entraînement. De même, il maximise la différence entre les distributions des "compositions" dans le jeu d'entraînement et le jeu de test. Ils fabriquent un jeu de données de génération de requêtes SPARQL avec de telles propriétés et montrent que les modèles classiques tels que les RNN [20] et les Transformer [2] ont des performances inversement corrélées à la mesure de généralisation compositionnelle du jeu de données.

Gu et al. [32] utilisent une définition plus variée de la généralisation basée sur des composantes des paires questions-requêtes qu'ils appellent "formes logiques". Par exemple, Jiang et al. [33], qui reprennent les mêmes définitions, utilisent comme "formes logiques" les classes et propriétés de la BC, les comparateurs SPARQL (par exemple $\leq$, $=$, etc.) et le mot clef **COUNT** du vocabulaire SPARQL. Basé sur ces "formes logiques", chaque donnée de l'ensemble de test d_0 est associée à un des trois niveaux de généralisation suivant. 1) le niveau dit Indépendant et Identiquement Distribué (IID) : il y a une donnée dans l'ensemble d'entraînement qui a

le même ensemble de "formes logiques" que d_0 . 2) le niveau dit compositionnel : toutes les "formes logiques" présentes dans d_0 sont présentes dans une donnée du jeu d'entraînement, mais aucune données d'entraînement n'a exactement le même ensemble de "formes logiques" que d_0 . 3) le niveau dit Zero-shot : il y a au moins une "forme logique" de d_0 qui n'est dans aucune donnée d'entraînement. Ils considèrent qu'un jeu de données capable de mesurer convenablement la capacité de généralisation d'un modèle doit avoir un jeu de test dont les données sont réparties dans ces trois catégories de manière équilibrée.

Jiang et al. [33] utilisent les mêmes définitions de généralisation pour évaluer un grand ensemble de jeux de données du domaine. Ils montrent que la plupart des jeux de données populaires ont de très mauvaises propriétés de généralisation. Ils développent un algorithme pour diviser les jeux de données en de nouveaux jeux de test et d'entraînement afin d'en augmenter les propriétés de généralisation. Ils testent ensuite des modèles de traduction automatique neuronale sur les nouveaux jeux obtenus et montrent qu'ils sont mauvais pour les données qui sont associées aux niveaux compositionnels et zéro-shot. Cette approche est intéressante, car elle se base sur la séparation de jeux de données existants pour évaluer les capacités de généralisation des modèles. Ce mémoire reprend cette logique et traite de deux limitations de [33]. Premièrement, les définitions des distributions des jeux d'entraînement et de test nécessaires pour évaluer les capacités de généralisation des modèles sont améliorées. Au lieu de parler de "formes logiques" dont la nature est arbitraire, on préférera utiliser les URIs du graphe de connaissances, qui sont les éléments qui risquent d'être inconnus lors de l'usage du modèle. De plus, [33] vise une distribution uniforme d'entrée ayant les trois niveaux de généralisation et ce mémoire vise à avoir 100% d'entrées de test avec un niveau zero-shot. Deuxièmement, ce mémoire propose un algorithme de séparation qui possède des garanties théoriques sur les distributions obtenues.

La gestion des concepts inconnus a également été étudiée dans le cadre du mécanisme de copie [7]. Des données de test ont été générées de manière à ne contenir que des URIs inconnues pour les modèles. Les performances sur ces ensemble sont très solides, ce qui montre qu'à la manière du méta-apprentissage proposé par [10], le mécanisme de copie pourrait permettre l'utilisation d'un certain niveau d'abstraction afin de généraliser l'apprentissage à des concepts inconnus. Cependant, ces jeux de test sont générés à petite échelle et peuvent difficilement constituer une référence pour l'évaluation de la capacité de généralisation des modèles. Ce mémoire vise également à analyser la capacité du mécanisme de copie à gérer des URIs inconnues. Contrairement à [7], ce mémoire utilise de nouvelles séparations de jeux de données populaires plutôt que de construire de nouveaux jeux de test.

2.4.3 Lien avec les difficultés de l’approche par traduction automatique neuronale

Comme vu en introduction, la tâche de génération de requêtes SPARQL soulève notamment comme difficultés la gestion des URIs inconnues. De plus, le fait que les ensembles de données utilisés soient générés automatiquement implique que les questions et les requêtes suivent un ensemble de structures fixes et en nombre limité. Cela peut impliquer que les modèles se basent sur ces structures pour générer la requête plutôt que sur le sens de la question. Ainsi, l’évaluation de la capacité des modèles à gérer des structures différentes de celles vues en entraînement est une grande difficulté de la tâche.

Ces deux difficultés sont très liées aux concepts de généralisation tels que définis précédemment. Une approche utilisant un ensemble de règles qui se base sur la structure de la question pour générer la requête semble pouvoir traiter facilement des URIs inconnues. Ainsi, comme les modèles neuronaux ne savent, en général, pas apprendre ce genre de règles, la gestion des URIs inconnues et des structures de questions-requêtes inconnues peuvent leur poser des difficultés.

Bien que certaines approches [7, 11] proposent la construction de jeux de données pour évaluer ces capacités, elles sont peu systématiques. La comparaison avec un jeu de données classique est compliquée et peu d’études utilisent ces jeux de données spéciaux. Une autre approche consiste à diviser un jeu de données existant en un nouveau jeu d’entraînement et jeu de test ayant les bonnes propriétés de généralisation. Cette approche a déjà été employée [33] mais l’algorithme en question ne permet pas un contrôle précis des proportions entre le jeu d’entraînement et le jeu de test. Dans ce mémoire, nous tenterons de pallier cette limitation.

CHAPITRE 3 GÉNÉRATION DE REQUÊTES SPARQL

Ce chapitre présente la méthodologie et les expériences faites dans ce mémoire afin de comparer, analyser et étendre les approches de la littérature. La comparaison systématique de ces méthodes et leurs associations inédites a permis de poser de nouvelles références dans les résultats de la tâche pour trois jeux de données différents. Dans un premier temps, ce chapitre décrit le format des données de la tâche ainsi que les spécificités des jeux de données de référence utilisés. Dans un second temps, on décrit les étapes nécessaires pour obtenir les jeux de données finaux utilisés dans ce mémoire. On explique ensuite la méthodologie utilisée pour la réalisation des expériences. Enfin, on présente les résultats de ces expériences.

Ce chapitre a mené à la rédaction d'un article actuellement en cours de relecture par une revue qui est pour l'instant disponible sur arXiv [34].

3.1 Jeux de données de référence

3.1.1 Patrons de génération

L'une des particularités de la tâche de génération automatique de requêtes SPARQL est la difficulté d'obtenir des données réelles en grande quantité. Les jeux de données classiques pour cette tâche sont donc généralement générés automatiquement à l'aide de patrons prédéfinis contenant des espaces réservés correspondant aux éléments de la base de connaissances dans la question et dans la requête. Les paires questions-requêtes sont ensuite obtenues en remplissant les espaces réservés avec des concepts du graphe de connaissances. La requête est complétée avec des URIs ou des littéraux du graphe de connaissances. La question est complétée par des formulations en anglais de ces concepts. Par exemple, pour le concept de film, l'URI sera `dbo:Film` et peut être complété par le mot "film" ou encore le mot "movie". La Figure 3.1 montre un patron ainsi qu'une paire question-requête générée par ce patron. Dans la suite de ce mémoire, on réfère à ces paires de structures contenant des espaces réservés correspondants en tant que "patrons", "patrons globaux" ou "patrons de génération". On utilisera également les termes "patron de question" ou "patron de requête" pour référer à uniquement la partie question ou la partie requête du patron global.

3.1.2 Questions reformulées

Certains jeux de données essaient de compenser la génération automatique des données en demandant à des humains de reformuler les questions de manière plus naturelle. Par exemple,

Patron de la question :
what is the <1> of <2> ?
Patron de la requête :
select distinct ?uri where {<2> <1> ?uri}
Exemple de question :
what is the allegiance of john kotelawala ?
Exemple de requête :
select distinct ?uri where {dbr:John_Kotelawala dbp:allegiance ?uri}

FIGURE 3.1 Exemple d'un patron de génération et d'une donnée générée par ce patron

la question "How many movies are there whose director is Stanley Kubrick?" est reformulée en "How many movies did Stanley Kubrick direct?".

3.1.3 Les jeux de données

Ce mémoire utilise trois jeux de données classiques du domaine, ayant chacun leurs particularités. Les tailles des jeux de données peuvent être trouvées dans la table 3.1 et le nombre de patrons dans la table 3.2. La table 3.1 rapporte la taille des jeux de test utilisés dans leur ensemble et la taille des jeux de test conservés pour l'évaluation du fait que les réponses sont non vides avec le point d'accès utilisé. Pour plus de détails, voir la section 3.3.5.

LC-QuAD 1.0. Le jeu de données LC-QuAD 1.0 [21] est un ensemble de taille réduite considéré comme facile sur lequel de très bonnes performances ont été rapportées à la fois par l'utilisation du mécanisme de copie et par des modèles pré-entraînés [7,8]. Il contient des questions reformulées. Il est basé sur la version 2016 de DBPedia.

LC-QuAD 2.0. Le jeu de données LC-QuAD 2.0 [26] est un ensemble de taille moyenne et est le plus difficile des jeux de données utilisés. Il contient des opérateurs SPARQL supplémentaires tels que **FILTER**, ce qui implique qu'il inclut également des littéraux. Comme expliqué dans l'introduction, les littéraux sont les données brutes des graphes de connaissances tels que les textes, les chiffres ou les dates. C'est le seul jeu de données utilisé dans ce mémoire qui contient des littéraux et il contient plus de mots clefs SPARQL que tous les autres jeux de données utilisés malgré sa taille moyenne. Par ailleurs, il contient notablement beaucoup d'URIs inconnues dans son jeu de test. Il est basé sur la version 2018 de DBPedia.

DBNQA. Le jeu de données DBNQA [22] est un ensemble de très grande taille. Il permet de démontrer la capacité de mise à l’échelle des modèles utilisés. Cependant, ce mémoire utilise uniquement un sous-ensemble de ce jeu de données. En effet, comme il sera expliqué plus tard, les méthodes d’annotation des questions utilisent les patrons de génération. Or, ce jeu de données ne fournit pas d’information sur les patrons utilisés pour chaque paire question-requête. Ce mémoire utilise donc uniquement les paires dont il a été possible d’extraire les patrons suivant la méthodologie décrite dans la section 3.2.3. Ce jeu de données est basé sur la version 2018 de Wikidata.

3.2 Jeux de données finaux

La version finale des jeux de données utilisés dans ce mémoire reprend les requêtes de jeux de données bruts ainsi que trois versions de la question, une brute et deux annotées. Cette section présente les vocabulaires qui permettent la description rigoureuse des annotations utilisées. On présentera ensuite lesdites annotations et la méthodologie sélectionnée pour les obtenir. Pour chaque donnée d’un jeu, on note $w_{0:n}$ les jetons de la séquence d’entrée et $\hat{t}_{0:m}$ les jetons de la requête correspondante.

3.2.1 Vocabulaire

On définit trois vocabulaires d’intérêt.

- Le vocabulaire de langue naturelle noté W . Il s’agit des mots en anglais qui sont présents dans les questions brutes.
- Le vocabulaire de la base de connaissances noté K qui est composé des URIs et des littéraux présents dans la requête qui appartiennent au graphe de connaissances.
- Le vocabulaire SPARQL noté S qui est le reste des jetons de la requête. Il contient les mots clefs tels que **COUNT** ou **SELECT**, les signes spéciaux tels que les parenthèses et les accolades ainsi que les variables telles que `?uri` ou `?x`.

La table 3.3 présente les tailles des trois vocabulaires introduits ici. L’abréviation HV signifie

	Total	Entrainement	Validation	Test	Test non vide
LC-QuAD 1.0	5000	4000	500	500	500
LC-QuAD 2.0	30225	21761	2418	6046	4849
DBNQA	382794	306236	38279	38279	30780

TABLEAU 3.1 Taille des jeux de données

	Global	Question	Requête
LC-QuAD 1.0	35	23	34
LC-QuAD 2.0	34	27	33
DBNQA	512	507	157

TABLEAU 3.2 Nombre de chaque type de patrons de génération

hors vocabulaire. La table 3.4 présente cinq vocabulaires supplémentaires. Ces vocabulaires sont des sous-ensembles de ceux de la table précédente. On y trouve les sous-catégories d’éléments de la BC, à savoir les ressources, les propriétés et classes, et les littéraux de la BC. On trouve ensuite la taille du vocabulaire de langue naturelle dans le cas des questions dans lesquelles les concepts de la BC sont remplacés par les URIs correspondantes. Cette version est appelée tag-interne et est décrite plus en détail dans la sous-section suivante. Ces tables rapportent également le nombre d’éléments inconnus dans l’ensemble de test pour chaque vocabulaire. On peut voir que W et K contiennent beaucoup d’éléments inconnus dans le jeu de test de chaque jeu de données, contrairement à S . Plus spécifiquement, ce sont surtout des ressources qui sont inconnues pour LC-QuAD 1.0 [21] et DBNQA [22], contrairement à LC-QuAD 2.0 [26] qui a des éléments inconnus dans chaque sous-vocabulaire de K . De plus, on peut voir que le vocabulaire de langue naturelle diminue énormément lorsque l’on utilise des questions avec tag-interne. Par ailleurs, ce vocabulaire de langue naturelle réduit contient très peu de mots inconnus dans l’ensemble de test de chaque jeu de données. Cela montre bien que les mots inconnus dans les questions de l’ensemble de test correspondent aux éléments de la BC.

3.2.2 Annotation des questions

En plus de la question brute non annotée, on utilise deux versions annotées de la question. Un exemple des différentes versions d’une question est présenté à la figure 3.2. Avec l’annotation, les questions sont des séquences de jetons telles que $w_{0:n} \subset W \cup (K \cap \hat{t}_{0:m})$.

Tag-interne. Dans cette version, on remplace les formulations des concepts en langue naturelle par les éléments de la base de connaissances qu’ils représentent. Cette annotation s’obtient par la tâche d’étiquetage. Cette tâche consiste à trouver dans un texte en langage naturel les mentions d’éléments symboliques tel que des URIs ainsi que leurs positions. Cette tâche est très étudiée dans le cadre des ressources. Leur description en langage naturel est souvent proche de leurs labels ou limitée dans la variété de formulations qu’elle peut prendre. Cependant, cette tâche est particulièrement difficile dans le cas des relations ou propriétés

	Total	Entrainement	Val.	Test	Jeton HV
Vocabulaire LN, noté W					
LC-QuAD 1.0	6141	5481	1631	1644	368
LC-QuAD 2.0	25975	21863	5602	10124	5751
DBNQA	95974	89128	32267	32440	3836
Vocabulaire SPARQL, noté S					
LC-QuAD 1.0	13	13	13	13	0
LC-QuAD 2.0	45	45	45	45	0
DBNQA	33	33	33	33	0
Vocabulaire BC, noté K					
LC-QuAD 1.0	4751	4150	1068	1065	309
LC-QuAD 2.0	31173	24964	4595	9992	6750
DBNQA	157673	142986	36845	37130	7998

TABLEAU 3.3 Taille des vocabulaires

	Total	Entrainement	Val.	Test	Jetons HV
Vocabulaire LN des questions avec tag interne					
LC-QuAD 1.0	298	274	119	115	7
LC-QuAD 2.0	31	31	31	31	0
DBNQA	364	364	364	364	0
Ressources de la BC					
LC-QuAD 1.0	3968	3385	615	631	318
LC-QuAD 2.0	26495	21272	3831	8134	5804
DBNQA	157037	142361	36268	36541	7983
Propriétés et classes de la BC					
LC-QuAD 1.0	783	765	453	434	0
LC-QuAD 2.0	4523	3792	1147	1858	920
DBNQA	635	624	577	589	15
Littéraux de la BC					
LC-QuAD 1.0	0	0	0	0	0
LC-QuAD 2.0	3783	2885	435	941	773
DBNQA	0	0	0	0	0

TABLEAU 3.4 Statistiques supplémentaires sur le vocabulaire de langue naturelles, des questions avec tag-interne, sur les ressources, les classes et propriétés ainsi que les littéraux de la BC

qui peuvent être exprimées de nombreuses façons, voire de façon ambiguë ou sous-entendue. Cependant, cette version contient de gros avantages. Comme les jetons de W qui représentent les éléments de K sont remplacés, cela limite très grandement la taille de W . Tous les jetons restants de W sont alors très fréquents.

Tag-final. Dans cette version, les éléments du graphe de connaissances sont rapportés à la fin de question et l’on ajoute juste à côté le label de l’élément dans le graphe de connaissances. On sépare chaque paire élément-label par le jeton `<sep>`. Cette annotation peut être obtenue par la tâche de liaison (linking). Cette tâche consiste à trouver les éléments d’un graphe de connaissances qui sont mentionnés dans un texte, sans forcément trouver leur position. Cette tâche est plus simple que celle de l’étiquetage, notamment pour les relations. Cependant, cette méthode d’annotation offre moins d’avantages que celle du tag-interne. En effet, cela fournit des séquences plus longues et la taille de W n’est pas diminuée, mais au contraire potentiellement légèrement augmentée. En effet, les labels récupérés dans le graphe de connaissances ne sont pas forcément les mêmes que les formulations utilisées par les jeux de données dans la question.

3.2.3 Méthodologie d’annotation

Les tâches d’étiquetage et de liaison, bien que très étudiées, ne sont pas encore parfaitement résolues. Toutefois, nous ne nous attaquons pas directement à la résolution de ces tâches. Dans ce mémoire, on utilise des questions parfaitement annotées. Pour cela, on utilise le fait que les paires questions-requêtes sont générées automatiquement par remplissage d’espaces réservés tel qu’expliqué ci-dessous.

Exemple de requête :
<code>select distinct ?uri where {dbr:John_Kotelawala dbp:allegiance ?uri}</code>
La question sans annotation :
<code>what is the allegiance of john kotelawala ?</code>
La question avec tag interne :
<code>what is the dbp:allegiance of dbr:John_Kotelawala ?</code>
La question avec tag final :
<code>what is the allegiance of john kotelawala ? <sep> dbr:John_Kotelawala john kotelawala <sep> dbp:allegiance allegiance</code>

FIGURE 3.2 Exemple d’une requête et des trois versions de la question correspondante

Identification des patrons globaux

Les jeux de données utilisés ne fournissent pas explicitement les identifiants des patrons de génération utilisés pour chaque entrée. Il faut donc commencer par identifier le patron global de chaque paire question-requête. Une fois les patrons identifiés, des expressions régulières en sont dérivées afin de tester si ces patrons permettent effectivement de reconnaître les données. La Figure 3.3 montre un exemple d’expression régulière dérivée d’un patron global.

LC-QuAD 1.0 et LC-QuAD 2.0. Ces deux jeux de données fournissent le patron de requête uniquement. Il a donc fallu trouver les patrons de question associés et ainsi identifier des patrons globaux. Cette opération a été faite à la main en regardant plusieurs exemples de données associées à un patron de requête. Comme ces jeux de données contiennent seulement une trentaine de patrons globaux, cette opération est assez simple à réaliser.

DBNQA. Ce jeu de donnée ne donne aucune indication sur aucun patron utilisé. Il a donc fallu utiliser plusieurs méthodes successives pour identifier autant de patrons que possible et ainsi être capable d’annoter autant de paires de questions-requêtes que possible.

1. Les patrons des autres jeux de données ont été utilisés pour identifier autant de données que possible.
2. Le jeu de données a été parcouru pour trouver les questions dans lesquels les concepts en langue naturelle sont identiques à la suite des mots qui composent l’URI. Cela a permis d’identifier automatiquement de nombreux patrons globaux.
3. On a construit un grand nombre de patrons à la main en regardant des exemples de données non reconnues par les patrons identifiés jusque là.

Ces trois méthodes ont permis d’identifier les patrons d’environ la moitié du jeu de données

Patron de la question :
what is the <1> of <2> ?
Patron de la requête :
select distinct ?uri where {<2> <1> ?uri}
Expression régulière pour la question :
what is the (.*?) of (.*?) ?
Expression régulière pour la requête :
select distinct \?uri where \{(.*) (.*) \?uri\}

FIGURE 3.3 Exemple d’expression régulière dérivée d’un patron global.

brut DBNQA [22]. On utilise donc un sous-ensemble du jeu original tout en gardant cependant une taille très importante.

Utilisation des patrons globaux pour l’annotation

Les éléments du graphe de connaissances sont insérés dans la question avec un marqueur de tag. Par exemple, l’URI `dbr:Stanley_Kubrick` sera inséré dans la question sous la forme `<<dbr:Stanley_Kubrick>>`.

Tag-final. On utilise l’expression régulière générée à partir du patron pour récupérer tout ce qui est présent dans les espaces réservés de la requête. S’il s’agit d’un littéral, on ajoute à la fin de la phrase le littéral avec le marqueur de tag, puis le littéral brut. S’il s’agit d’une URI, on ajoute l’URI avec son marqueur de tag suivi de son label dans le graphe de connaissances.

Tag-interne. On utilise la correspondance des espaces réservés pour remplacer le contenu de l’espace réservé dans la question par le contenu du même espace réservé dans la requête. Les éléments insérés dans la question ont un marqueur de tag.

3.3 Méthodologie

3.3.1 Configurations des expériences

Comparabilité des approches de référence. Comme indiqué dans le chapitre 2, les deux approches ayant rapporté des résultats significatifs sur la tâche sont [7] et [8]. Ces deux études utilisent des jeux de données différents, des méthodes d’annotation différentes et des architectures différentes. L’objectif des expériences présentées dans ce mémoire est de comparer et d’étendre ces méthodologies. On présentera donc les expériences alliant les quatre architectures neuronales (ConvSeq2Seq, Transformer, BART, T5), les trois versions d’annotation (sans annotation, tag-interne et tag-final), les trois jeux de données (LC-QuAD 1.0, LC-QuAD 2.0 et DBNQA) ainsi que l’usage ou non de la copie. Comme la copie ne peut pas se faire sans annotation, on présente donc 20 expériences différentes pour chaque jeu de données. Sur les 60 expériences au total, 14 sont des reproductions de la littérature et 46 sont des résultats originaux.

Pré-traitement et segmentation. Comme expliqué dans le chapitre 2, les modèles non pré-entraînés nécessitent un vocabulaire fixe. Lorsque l’on initialise un encodeur-décodeur pour une tâche, on prend donc l’ensemble mots présents dans le corpus d’entraînement. Dans

le cas de ce mémoire, on effectue donc une partie de pré-traitement pour transformer les phrases en suites de jetons en minuscule et séparés par des espaces. Cela implique notamment des modifications de la ponctuation de la question et de la requête (par exemple, des points d'interrogations peuvent être collés au dernier mot de la question ou des accolades peuvent être collés à des URIs dans la requête). Il faut également uniformiser les termes SPARQL pour lesquels la casse n'influe pas sur le sens.

On transforme ensuite la question et la requête en suites d'indices dans ce vocabulaire avant de les passer aux modèles. Dans le cas de questions annotées, tous les termes compris dans le marquage de tag (c'est-à-dire `<<...>>`) sont considérés comme un seul jeton.

La segmentation des données est différente pour les modèles pré-entraînés, car ils possèdent déjà un vocabulaire et un segmenteur pré-entraîné. Ces segmenteurs sont capables de diviser les mots en sous-mots et peuvent donc gérer n'importe quel texte sans avoir à intégrer de jeton inconnu `<unk>`.

Cependant, afin de faciliter la comparaison avec les modèles non pré-entraînés, on utilise les mêmes données pré-traitées avec l'ajout d'espace avant les signes de ponctuation et la mise en minuscule. Cela implique également une étape de post-traitement, car les segmenteurs de ces modèles ne décodent pas forcément les mots avec les bonnes positions d'espaces.

Pour préserver le sens des mots de langue naturelle et introduire un sens particulier aux jetons des requêtes SPARQL, on suit les recommandations de l'étude ayant déjà utilisé ces modèles [8]. On utilise donc des jetons spéciaux pour représenter les mots du vocabulaire SPARQL. Pour T5, ces mots sont remplacés par les jetons sentinelles compris dans le vocabulaire de base du modèle pour ce genre d'utilisations. Pour BART, on introduit de nouveaux jetons spéciaux dans son vocabulaire. De plus, pour limiter les erreurs fréquentes de génération d'URIs provoquées par la segmentation en sous-mots, on ajoute également à ces jetons spéciaux les préfixes des URIs. Les littéraux et le reste des URIs ne sont pas modifiés. L'introduction de ces jetons spéciaux, et notamment pour le préfixe des URIs, nécessite également une étape de post-traitement pour récupérer des requêtes formatées correctement.

Une fois les séquences d'entrée et de sortie passées dans le segmenteur, on ajoute également un remplissage final afin de pouvoir mettre chaque lot sous la forme d'un tenseur. En effet, en pratique, au lieu de passer les entrées une par une au modèle, on groupe les données en "lots" pour permettre au modèle de traiter plusieurs entrées en même temps. Le segmenteur transforme chaque phrase en une séquence de jetons qui seront ensuite remplacés par leur indice dans le vocabulaire d'entrée ou de sortie. On obtient alors un vecteur de taille correspondant au nombre de jetons dans la phrase. Un lot de séquences peut alors être composé de vecteurs de tailles différentes. Pour éviter cela, on ajoute à la fin de chaque phrase un certain nombre

de jetons de remplissage <pad>. Ainsi, toutes les séquences font la même taille et peuvent être alignées en lots. De plus, même si la requête prévue ne comporte pas le même nombre de mots que la requête attendue, elles ont toutes le même nombre de jetons en ajoutant le remplissage.

Adaptation du mécanisme de copie aux modèles pré-entraînés. Comme indiqué précédemment, la principale limitation des modèles non pré-entraînés concerne les mots hors vocabulaire. Cette limitation est particulièrement critique dans notre cas, où les mots inconnus font souvent référence à des URIs provenant de la BC. Ces URIs inconnues conduisent presque certainement à des requêtes qui ne peuvent pas avoir de réponses correctes car les URIs pertinentes ne se trouvent pas dans le vocabulaire du décodeur. Même lorsque les URIs sont déjà rencontrées lors de l'entraînement, elles y sont souvent rares et générer la bonne URI reste une tâche difficile pour les modèles. C'est pourquoi le mécanisme de copie présenté dans le chapitre 2 est un ajout particulièrement notable sur les modèles non pré-entraînés.

Pour les modèles pré-entraînés, ces limitations sont déjà prises en compte car le transfert de l'entrée à la sortie fait partie de leur comportement naturel. Cependant, empiriquement, la génération d'URIs correctes reste toujours leur principale source d'erreur. Dans le cas de Wikidata (utilisé par LC-QuAD 2.0), les URIs sont une séquence de chiffres, les modèles ont donc du mal à comprendre la séquence exacte. Dans le cas de DBpedia (utilisé pour LC-QuAD 1.0 et DBNQA), les URIs sont composées de mots que le modèle peut reconnaître et donc générer plus facilement, mais il existe une grande variation dans les normes de nommage des URI. Les modèles génèrent souvent des erreurs de casse ou ajoutent/suppriment des sous-tirets (underscores), ce qui conduit à une mauvaise génération d'URI. Par conséquent, considérer les URIs comme un seul jeton peut aider à la correction des requêtes générées, car les modèles ne peuvent plus commettre d'"erreurs d'orthographe" liées aux URIs.

Le mécanisme de copie tel que décrit dans la section 2 permet de conserver les URIs en un seul jeton. Ce mémoire étend donc le mécanisme de copie développée dans [7] aux modèles pré-entraînés. De plus, ce mémoire est la première étude à utiliser le jeu de données LC-QuAD 2.0 avec une méthodologie de copie similaire à celle de [7]. Il est donc le premier à intégrer les littéraux parmi les mots masqués pouvant être copiés.

L'adaptation aux modèles pré-entraînés se fait quasiment automatiquement du fait que le mécanisme de copie utilise uniquement les sorties du décodeur et la phrase d'entrée avant d'être masquée.

La difficulté réside dans le masquage des URIs dans la question et la requête. Pour cela, on commence par ajouter si nécessaire les jetons supplémentaires au vocabulaire du segmenteur

pré-entraîné des modèles (voir la section 3.3.1). Puis, on initialise de nouveaux plongements initiaux pour compléter le vocabulaire du modèle si nécessaire. Enfin, on ajoute le vocabulaire de la BC au segmenteur pré-entraîné sans modifier le modèle. Le masquage peut alors se faire sur les jetons dont l’indice dans le vocabulaire est supérieur à celui du vocabulaire de base plus les jetons ajoutés initialement. Lors de l’évaluation, on peut ajouter autant de jetons que nécessaire au segmenteur, car ils seront masqués une fois passés à la partie encodeur-décodeur. La figure 3.4 montre quatre versions de segmentation avec le segmenteur de BART suivant les modifications que l’on apporte au vocabulaire. Dans cette figure, les jetons sont séparés par des espaces. Le symbole $\mathring{G}$ est un symbole utilisé par le segmenteur de BART pour indiquer le début d’un mot. Dans cet exemple, les jetons annotés seront ensuite masqués et peuvent donc ne pas appartenir au vocabulaire du modèle, ils ne posent donc pas de problème s’ils sont inconnus lors de l’évaluation.

3.3.2 Sélection des hyper-paramètres

Ce mémoire ne se concentre pas sur l’optimisation des hyper-paramètres, mais sur la comparaison entre les approches. Par conséquent, les hyper-paramètres sont basés sur les suggestions des approches de pointe [7, 8, 13] et sur les limites des ressources matérielles.

Pour les architectures Transformer et ConvSeq2Seq, les hyper-paramètres sont inspirés de [13]. Pour les modèles BART et T5, les hyper-paramètres suivent les recommandations de [8]. Le Tableau 3.5 résume les hyper-paramètres principaux de chaque architecture. Le taux d’apprentissage est rapporté sous l’abréviation TA. Seul T5-small a été utilisé, car [8] rapporte de meilleures performances avec cette version sur LC-QuAD 1.0 et des performances similaires entre T5-small et T5-large sur LC-QuAD 2.0. BART-base est utilisé pour comparer les résultats avec ceux de [8].

Les Tableaux 3.6 et 3.7 rapportent les nombres d’époques et de tailles de lots utilisés. Certains ont été adaptés pour s’adapter à la RAM des machines utilisées. Notamment, la taille des lots et le nombre d’époques ont été considérablement réduits pour DBNQA. Il a cependant été observé pendant l’expérimentation que la taille de lots a un impact important sur les

Modèle	Couches	Unités cachées	Optimiseur	TA	Dropout
ConSeq2Seq	15	512	SGD	0.5	0.2
Transformers	6	1024	Adam	0.0005	0.3
BART-base	6	768	Adam	0.000015	0.1
T5-small	6	512	Adam	0.0015	0.1

TABLEAU 3.5 Hyper paramètres des modèles

Requête brute :
<code>select distinct count (?uri) where { ?uri dbo:director dbr:Stanley_Kubrick . }</code>
Requête segmentée sans jetons spéciaux sans considérer l'annotation :
<code>select Ġdistinct Ġcount Ġ(Ġ? uri Ġ) Ġwhere Ġ{ Ġ? uri Ġd bo : director Ġd br : Stan ley _ K ub rick Ġ. Ġ}</code>
Requête segmentée avec jetons spéciaux sans considérer l'annotation :
<code><select> <distinct> <count> <(<?uri> <)> <where> <{> <?uri> <dbo:> director <dbr:> Stan ley _ K ub rick <.> <}></code>
Requête segmentée sans jetons spéciaux en considérant l'annotation :
<code>select Ġdistinct Ġcount Ġ(Ġ? uri Ġ) Ġwhere Ġ{ Ġ? uri «dbo:director» «dbr:Stanley_Kubrick» . Ġ}</code>
Requête segmentée avec jetons spéciaux en considérant l'annotation :
<code><select> <distinct> <count> <(<?uri> <)> <where> <{> <?uri> «dbo:director» «dbr:Stanley_Kubrick» <.> <}></code>

FIGURE 3.4 Exemple de segmentation d'une requête par le segmenteur de BART.

performances. Notamment, pour le jeu de donnée DBNQA et l'architecture Transformer, baisser la taille de lot entraîne de fortes pertes de performances. Dans ce cas particulier, la taille de lot a été poussée aussi haute que possible étant donné les limites du matériel disponible.

3.3.3 Entraînement et validation

Pour chaque exécution, le modèle est entraîné pendant le nombre d'époques indiqué et le modèle avec la meilleure perte de validation est conservé pour les tests. La fonction de perte utilisée est l'entropie croisée. Le processus entraînement-validation-évaluation est répété trois fois avec des graines aléatoires différentes. Les résultats rapportés sont les moyennes des trois résultats.

L'entraînement et la validation sont effectuées avec l'utilisation de la méthode dite d'ensei-

	LC-QuAD 1.0	LC-QuAD 2.0	DBNQA
Transformers	500	150	50
ConvSeq2Seq	500	150	50
BART-base	200	50	20
T5-small	200	50	20

TABLEAU 3.6 Nombre d'époques d'entraînement

	LC-QuAD 1.0	LC-QuAD 2.0	DBNQA
Transformers	32	16	32
ConvSeq2Seq	32	16	5
BART-base	16	8	5
T5-small	16	8	5

TABLEAU 3.7 Taille des batchs

gnement forcé (teacher forcing). Il s’agit d’une technique de génération utilisable lorsque la requête attendue est connue. On donne la séquence attendue entière au décodeur lors de la phase de génération avec le masque d’attention causale qui permet que chaque mot soit uniquement influencé par les mots précédents. Ainsi, la sortie sera le jeton suivant prédit pour chaque étape i étant donnée la séquence de jetons correcte jusqu’à l’étape i . Cette méthode permet d’augmenter fortement la vitesse d’entraînement, car elle nécessite un seul passage dans le décodeur pour chaque donnée. Il n’y a donc pas de génération de requêtes à proprement parlé durant l’entraînement ni la validation.

On peut également préciser que l’échantillonnage des lots a été effectué différemment selon les expériences. Pour les modèles pré-entraînés sans copie, on tire aléatoirement les lots de manière indépendante (ce qui implique que certaines données peuvent être vues plusieurs fois) alors que pour les autres expériences, on partitionne aléatoirement les données d’entraînement pour obtenir les lots (chaque donnée est vue une fois par époque). La première version est inspirée du code de [8] et la seconde du code de [7]. Le choix d’utiliser deux méthodes différentes a été fait au vu des résultats supérieurs de la première méthode (tirage aléatoire des lots) dans le cas de modèles pré-entraînés sans copie.

3.3.4 Génération sur l’ensemble de test

Durant l’évaluation, les requêtes attendues sont supposées inconnues et la méthode d’apprentissage forcé ne peut pas être utilisée. L’objectif de la génération est de produire la séquence de jetons dont le produit des probabilités générées par le décodeur est maximal. Ceci est un problème combinatoire qui ne peut pas être résolu en temps raisonnable. Étant donnés des tailles maximales de séquence de 150 jetons et des vocabulaires de sortie de taille caractéristique d’environ $10e^4$, on doit évaluer les probabilités d’environ 10000^{150} séquences. Il faut donc une heuristique pour générer les séquences les plus probables en utilisant un nombre raisonnable de passages dans le décodeur.

L’une des méthodes de génération les plus répandues est appelée recherche par faisceau. On conserve à chaque étape de génération i un total de m séquences et leur probabilité. À l’étape

$i + 1$ on génère les n jetons les plus probables pour chacune des m séquences. On obtient alors $m \times n$ séquences et on conserve les m plus probables.

Dans ce mémoire, on utilise une méthode plus simple. Le mécanisme de copie étant implémenté sans l'usage d'aucune librairie, les implémentations classiques de recherche par faisceaux n'ont pas pu être utilisées. On utilise donc une recherche gloutonne. C'est-à-dire que l'on conserve uniquement une seule séquence à générer qui est constituée du jeton le plus probable à chaque étape de génération.

Suite à une erreur de vectorisation, on a observé que les requêtes contenant l'expression SPARQL `limite 25` ont été vectorisées sous la forme `limit 2 5`. Cela implique que les modèles sont entraînés à prédire `limit 2 5`. Cela représente environ 7% des données de LC-QuAD 2.0 et n'est pas présent dans les deux autres jeux de données. Étant donné que cela semble être la seule erreur de ce type, au lieu de refaire toutes les expériences en corrigeant le problème de vectorisation, on a simplement appliqué une étape de post-traitement qui remplace toutes les occurrences de `limit 2 5` en `limit 25`. Cette opération est effectuée directement sur les requêtes générées sur l'ensemble de test.

3.3.5 Évaluation

Ce mémoire utilise les trois métriques d'évaluation présentées dans la section 2, le score BLEU, la précision des réponses et le score F1 sur les réponses.

Pour obtenir les réponses à partir des requêtes candidates ou de référence, on utilise des points d'accès SPARQL aux graphes de connaissances correspondant aux jeux de données. Cependant, certains jeux de données sont assez anciens et ne sont pas à jour avec les versions actuelles des bases de connaissances. Ainsi, certaines requêtes de référence produisent des réponses vides lorsqu'elles sont envoyées au point d'accès. Cela pose un problème, car des requêtes syntaxiquement justes, mais avec les mauvaises URIs, renverront aussi des réponses vides. Ainsi, on donnerait de bonnes valeurs de métriques de réponses à des requêtes incorrectes. Par exemple, imaginons qu'un jeu de données ancien contienne la question "What are the movies whose director is Stanley Kubrick?". Lors de la construction de la base de connaissances, la requête correspondante est `select distinct ?uri where {?uri dbo:director dbr:Staley_Kubrick.}`. Cette requête, au moment de la publication du jeu de données, a pour réponse attendue un ensemble de 16 URIs qui représentent les films de Stanley Kubrick. Maintenant, imaginons que les gestionnaires de la BC sur laquelle doit se faire la requête décident que le concept de réalisateur est trop générique. Ils décident alors de changer les triplets à propos des réalisateurs de films en utilisant l'URI `dbo:film_maker` à la place de `dbo:director`. La requête dans la base de données fournira alors comme réponse l'ensemble

vide. Si un modèle prédit la requête cible, la réponse obtenue sera aussi l'ensemble vide et les métriques basées sur des réponses indiqueront 1. Cependant, si un modèle prédit la requête `select distinct ?uri where {?dbo:director dbo:director ?uri.}`, la réponse sera également l'ensemble vide et le modèle aura également un score de 1 sur des métriques basées sur les réponses. Pourtant, la requête prédite par le modèle est visiblement fausse. Ainsi, toutes les requêtes de référence avec des réponses vides ne permettent pas d'évaluer un modèle avec des métriques basées sur les réponses.

On a donc utilisé le point d'accès le plus proche possible de la version de la base de connaissances utilisée pour générer les jeux de données et conservé uniquement les entrées dont la requête fournit une réponse non vide pour le jeu de test. La table 3.8 résume les points d'accès utilisés, la proportion de réponses non vides et la proportion de requêtes produisant une erreur pour chaque jeu de données. Le point d'accès DBPedia local 2016 a été mis en place sur le réseau local du laboratoire LAMA-WeST à partir d'une sauvegarde de DBPedia de 2016¹. Le point d'accès Wikidata public est disponible sur le web² à partir de la version de Wikidata courante mise à jour régulièrement.

Par ailleurs, on a observé que certains modèles produisaient des requêtes contenant uniquement des variables, par exemple `select distinct ?uri where { ?uri ?x ?y }`. Ce genre de requête renvoie l'intégralité des éléments du graphe de connaissance et prend donc un temps d'exécution énorme sans pour autant renvoyer d'erreur. Afin d'éviter ce genre de problèmes qui perturbe grandement l'exécution des requêtes prédites, on a ajouté l'expression `limit 50` à la requête SPARQL, pour le calcul des métriques. Cette opération n'a été effectuée que lorsque la requête prédite est différente de la requête attendue. Étant donné le très

Jeu de données	Point d'accès	Ensemble	Réponses non vides (%)	Erreurs (%)
LC-QuAD 1.0	DBPedia local 2016	entraînement	100	0.00
		validation	100	0.00
		test	100	0.00
LC-QuAD 2.0	Wikidata public	entraînement	81.02	0.15
		validation	80.19	0.12
		test	82.20	0.18
DBNQA	DBPedia local 2016	entraînement	80.28	0.16
		validation	80.33	0.15
		test	80.41	0.19

TABLEAU 3.8 Statistiques des réponses dans nos jeux de données

1. <http://downloads.dbpedia.org/wiki-archive/downloads-2016-10.html>

2. <https://query.wikidata.org/>

faible écart entre les scores de précision et de F1 qu'on a observé dans nos résultats, il semble que très peu de requêtes renvoient des résultats corrects sans être la requête attendue. Cela suggère que l'ajout du `limit 50` sur les requêtes prédites différentes des requêtes attendues a très peu d'impact sur les résultats.

3.4 Résultats

3.4.1 Modèles non pré-entraînés

Les résultats des expériences sur les modèles non pré-entraînés sont rapportés dans la table 3.9.

Résultats de reproduction. Pour les modèles sans et avec le mécanisme de copie, LC-QuAD 1.0 et DBNQA ont déjà été testés sans annotation [7, 13] et avec des questions contenant des tag-internes [7].

Les résultats de ce mémoire améliorent légèrement ceux rapportés par [7, 13] pour LC-QuAD 1.0 avec les modèles Transformer et ConvSeq2Seq avec ou sans le mécanisme de copie. Les résultats de ce mémoire sont également similaires pour DBNQA avec ConvSeq2Seq avec ou sans le mécanisme de copie.

Cependant, les résultats des Transformers sans la fonction de copie pour le jeu de données DBNQA sont inférieurs à ceux rapportés par [7, 13]. Sans annotation de question, les résultats de ce mémoire démontrent une baisse d'environ 7 points BLEU par rapport à [13], qui a utilisé une taille de batch de 128 et environ 4 points par rapport à [7], qui a utilisé une taille de batch de 64. Avec des questions contenant des tag-internes, les résultats de ce mémoire perdent 14 points BLEU par rapport à [7]. Cette baisse de performance est due à la réduction de la taille des lots, en raison d'une limitation de la puissance de calcul disponible. En ce qui concerne la précision et le score F1, on ne peut pas comparer les résultats de ce mémoire à ceux de [7], car ils n'ont pas exclu les réponses vides.

Impact de l'annotation. Pour presque tous les résultats, on peut voir que l'annotation des questions améliore les performances. Les performances ne diminuent que légèrement pour Transformer sur DBNQA et pour ConvSeq2Seq sur LC-QuAD 2.0 avec des questions avec tag-internes par rapport à aucune annotation (brut). À l'exception de ces deux contre-exemples parmi les 18 expériences sur des modèles non pré-entraînés sans mécanisme de copie, on constate une amélioration systématique grâce à l'annotation.

Avec le mécanisme de copie, on remarque que les Transformers atteignent de bonnes per-

performances avec des questions avec tag-internes et échouent complètement avec des questions avec tag-final. Les modèles ConvSe2Seq atteignent de bonnes performances avec les deux paramètres.

Impact du mécanisme de copie. Le mécanisme de copie a presque toujours un impact positif conséquent sur les performances. L’impact de la copie est toujours plus significatif avec des questions avec tag-internes qu’avec des questions avec tag-final. Enfin, on peut remarquer que l’ensemble de données DBNQA est celui qui bénéficie le moins du mécanisme de copie. Cela est probablement dû à la quantité importante de données qui permet aux modèles non pré-entraînés d’apprendre déjà suffisamment bien sans le mécanisme de copie. Cependant, il y a toujours une amélioration significative des performances par rapport aux modèles sans copie.

3.4.2 Modèles pré-entraînés

Les résultats des expériences sur les modèles pré-entraînés sont rapportés dans la table 3.10.

Résultats de reproduction. Pour les modèles BART et T5, LC-QuAD 1.0 et LC-QuAD 2.0 ont déjà été testés [8] avec tag-final. Pour les questions avec tag-final sans le mécanisme de copie, les résultats rapportés dans ce mémoire sont meilleurs ou proches des résultats précédemment rapportés [8]. On observe une nette amélioration pour BART, mais une baisse de 2 points pour T5 sur LC-QuAD 2.0. Cela peut être dû au fait que ce mémoire inclut les littéraux à la fin des questions, contrairement à [8]. Une autre raison de cet écart de performance pourrait être que l’ensemble de test n’est pas exactement le même, le point d’accès à Wikidata étant différent (ce mémoire utilise le point d’accès actuel de Wikidata plutôt qu’une version de sauvegarde d’octobre 2021 [8]). Enfin, la stratégie de génération est différente. Ce mémoire évalue la génération du modèle de manière gloutonne, tandis que [8] conserve les 10 premières requêtes générées par recherche par faisceau, puis exécute chacune d’entre elles sur le point d’accès, et n’évalue que la première à retourner une réponse non vide.

Comparaison BART et T5. Sans l’usage du mécanisme de copie, on constate que T5 surpasse de manière constante BART, et que les questions avec tag-interne impliquent toujours des performances similaires ou meilleures que les questions avec tag-final.

		Transformer						ConvSeq2Seq					
		Sans copie			Avec copie			Sans copie			Avec copie		
		bleu	acc.	F1	bleu	acc.	F1	bleu	acc.	F1	bleu	acc.	F1
LC 1	Brut	69	33	37	/	/	/	69	23	27	/	/	/
	Tag-interne	82	41	44	98	95	95	78	28	32	98	95	95
	Tag-final	74	38	41	49	1.9	1.9	75	25	28	91	77	78
LC 2.0	Brut	56	0.9	1.1	/	/	/	75	9.9	11	/	/	/
	Tag-interne	59	1.5	1.6	82	69	69	76	9.0	10	88	69	70
	Tag-final	65	1.4	1.7	57	2.2	2.2	77	14	15	91	66	66
DBNQA	Brut	62	1.7	2.2	/	/	/	89	65	65	/	/	/
	Tag-interne	53	0.6	0.8	95	89	89	91	67	67	95	89	90
	Tag-final	60	1.7	2.2	60	7.0	7.2	92	76	76	96	88	88

TABLEAU 3.9 Résultats des modèles non pré-entraînés sur les jeux de données originaux

Impact de l’annotation. Dans l’ensemble, sans le mécanisme de copie, on peut remarquer que même si les annotations ont aidé à améliorer les performances pour les modèles non pré-entraînés, l’impact est beaucoup plus important pour les modèles pré-entraînés qui atteignent souvent une performance presque parfaite avec les questions annotées.

Notamment, on constate que les modèles non pré-entraînés gagnent en moyenne 2,30 points de F1 en passant d’aucune annotation (brut) à des questions annotées, tandis que les modèles pré-entraînés gagnent en moyenne 54.58 points de F1.

Impact du mécanisme de copie. On peut remarquer que BART bénéficie beaucoup plus du mécanisme de copie que T5. Sauf pour LC-QuAD 2.0 avec des questions avec tag-interne, le mécanisme de copie permet toujours une amélioration des performances pour BART. Le cas de LC-QuAD 2.0 avec des questions avec tag-interne et le mécanisme de copie montre une difficulté spécifique qui est discutée dans la section 3.5. De manière opposée, le mécanisme de copie ne permet une amélioration des performances pour T5 qu’avec le jeu de données LC-QuAD 1.0. Cependant, il diminue les résultats pour les questions avec tag-interne des ensembles de données LC-QuAD 2.0 et DBNQA.

		BART						T5					
		Sans copie			Avec copie			Sans copie			Avec copie		
		bleu	acc.	F1	bleu	acc.	F1	bleu	acc.	F1	bleu	acc.	F1
LC 1.0	Brut	77	38	41	/	/	/	72	24	26	/	/	/
	Tag-interne	96	85	85	99	96	96	98	95	95	99	97	98
	Tag-final	95	83	83	96	93	94	98	94	94	98	96	96
LC 2	Brut	68	0.5	0.6	/	/	/	74	0.7	0.9	/	/	/
	Tag-interne	94	80	80	89	72	72	98	94	94	85	68	68
	Tag-final	90	67	67	96	88	88	96	90	90	92	77	78
DBNQA	Brut	87	61	61	/	/	/	88	60	60	/	/	/
	Tag-interne	94	79	79	94	90	90	97	93	94	76	54	56
	Tag-final	94	82	83	94	91	91	97	94	94	24	16	17

TABLEAU 3.10 Résultats des modèles pré-entraînés sur les jeux de données originaux

3.4.3 Meilleurs résultats

Pour chaque jeu de données, ce mémoire rapporte des résultats dépassants ceux de la littérature. La figure 3.5 illustre graphiquement les meilleurs résultats en fonction des modèles, de l’annotation des données et de l’utilisation du mécanisme de copie. Les meilleurs résultats en termes de score F1 et de précision de réponse sont rapportés ici, puis les scores de précision sont comparés avec les approches précédentes les plus performantes [7,8]. On note cependant que bien que les deux études [7,8] utilisent un score de précision de réponse, il existe des différences méthodologiques. D’une part, [7] ne rejette pas les réponses vides, ce qui augmente les performances. D’autre part, [8] utilise la première réponse non vide parmi plusieurs requêtes générées par la recherche en faisceaux, ce qui peut également améliorer les performances par rapport à l’approche de ce mémoire, qui génère une seule requête. Les éléments suivants résument les meilleurs résultats pour chaque ensemble de données :

- **LC-QuAD 1.0** : les résultats de ce mémoire montrent jusqu’à 97,13% de score F1 et 97,07% de précision avec BART avec le mécanisme de copie et des questions avec tag-interne. Ces performances dépassent [8] de 5 points et [7] de 2 points.
- **LC-QuAD 2.0** : les résultats de ce mémoire montent jusqu’à 94,15% de score F1 et 94,02% de précision avec T5 sans le mécanisme de copie avec des questions avec tag-interne. Cela surpasse [8] de 2 points.
- **DBNQA** : les résultats de ce mémoire montent jusqu’à 93.55% et 93.67% (qui représentent un écart négligeable) de score F1 et respectivement 93.48% et 93.65% de précision avec T5 sans le mécanisme de copie avec des questions avec tag-interne et des questions avec tag-final. Cela surpasse [7] de 7 points.

La Figure 3.5 illustre les huit meilleurs résultats pour chaque jeu de données. On peut observer que LC-QuAD 1.0 peut être qualifié de facile car les meilleures performances sont très hautes et les huit résultats sont tous au-dessus de 90% de score F1. On voit ensuite que LC-QuAD 2.0 est beaucoup plus difficile, car, bien que le meilleur résultat soit très bon, on voit que les résultats suivants tombent en dessous de 90% et même en dessous de 75% à partir du 6ème meilleur résultat. Le jeu de donnée DBNQA semble de difficulté intermédiaire comme les deux meilleurs résultats restent au-dessus de 90% et aucun des huit meilleurs résultats ne se trouve en dessous de 80%. De plus, on peut voir que le mécanisme de copie semble très présent parmi les meilleurs résultats pour LC-QuAD 1.0 et DBNQA mais pas pour LC-QuAD 2.0. Enfin, on peut constater l’omniprésence de T5 parmi les modèles les plus performants.

3.5 Discussion

3.5.1 Le cas de LC-QuAD 2.0

La plupart des résultats sur LC-QuAD 2.0 sont faibles (comparés aux résultats sur d’autres jeux de données), en particulier avec des modèles non pré-entraînés. Cela est probablement dû au grand nombre de mots inconnus dans l’ensemble de test et à la présence de littéraux. De plus, LC-QuAD 2.0 inclut des structures de requêtes beaucoup plus variées, comme le suggère une grande taille de vocabulaire SPARQL rapportée dans le tableau 3.3.

Un autre défi est que certains patrons de LC-QuAD 2.0 partagent un patron de langage naturel. Cela signifie que pour le même patron de question, on peut avoir différents patrons de requêtes. Étant donné que le mécanisme de copie est basé sur le masquage du vocabulaire de la base de connaissances, deux phrases qui partagent le même patron de question seront considérées comme exactement la même phrase par le bloc encodeur décodeur et seul le bloc de copie sera capable de "voir" les mots qui les différencient. Par conséquent, le modèle pourrait toujours associer des questions d’un patron de question donné au même patron de requête. Les questions qui partagent un patron de question sont générées par le modèle en suivant le patron de requête le plus fréquent associé à ce patron de question. Le tableau 3.11 montre un exemple de ce cas. Ce tableau rapporte la fréquence d’apparition dans le jeu d’entraînement et la précision des requêtes masquées dans le jeu de test pour le patron de question "what is <mask> <mask> of <mask> that is <mask> is <mask>?". La précision des requêtes masquées correspond à la proportion d’égalité parfaite entre les requêtes masquées attendues et les requêtes masquées prédites pour un patron de requête donné. Une requête masquée est une requête dans laquelle les jetons du vocabulaire de la BC sont remplacés par le jeton <mask>. Les statistiques rapportées dans la table 3.11 sont mesurées sur une

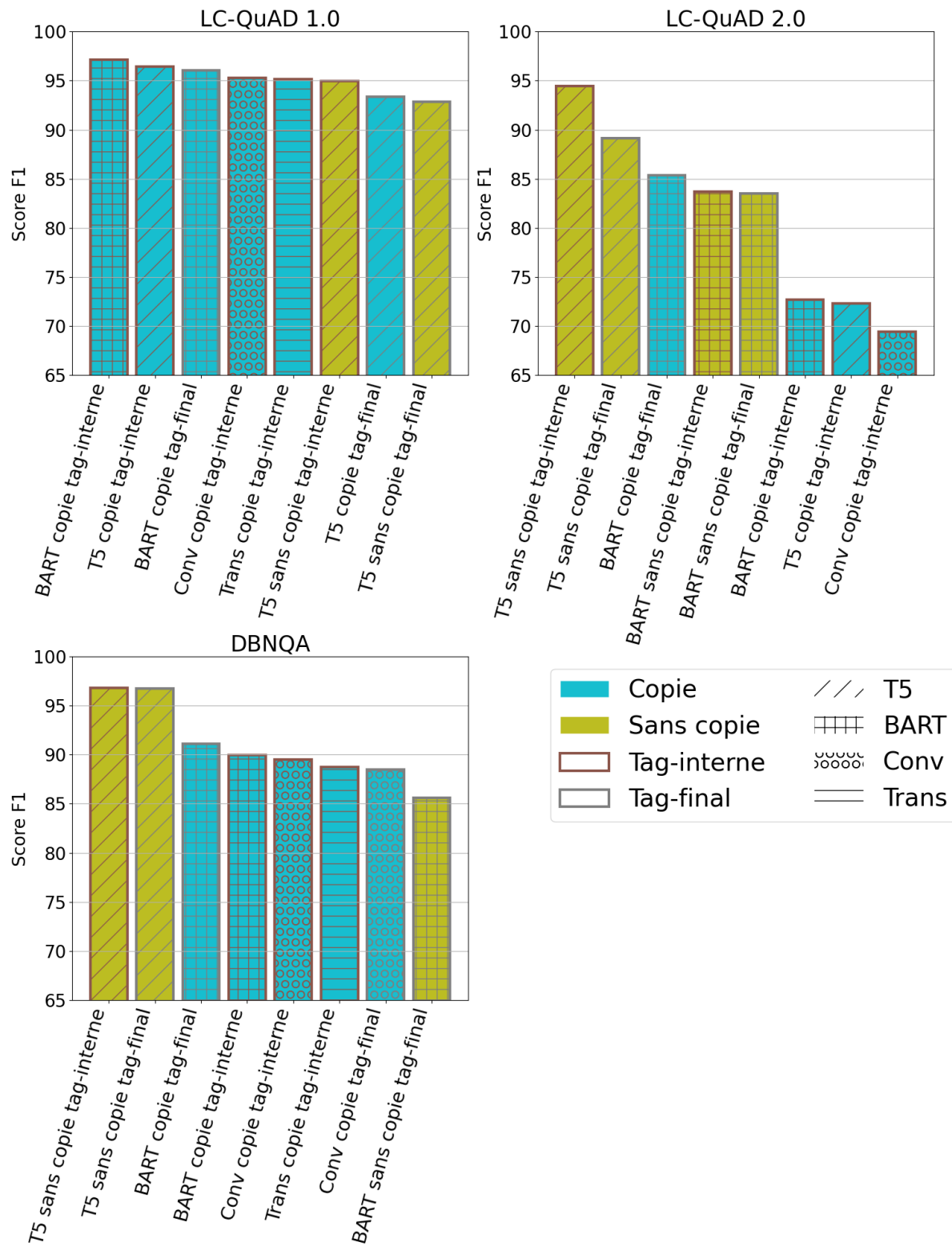


FIGURE 3.5 Les meilleurs scores F1 pour chaque jeu de données

exécution de ConvSeq2Seq et le mécanisme de copie avec des questions avec tag-interne sur LC-QuAD 2.0. On observe effectivement que lorsque le patron de la requête attendue est celui qui était le plus fréquemment associé au patron de question dans le jeu d'entraînement, alors la précision masquée est de 1.0. C'est-à-dire que le modèle prédit toujours la bonne structure. Cependant, lorsque le patron de la requête attendu est moins fréquent dans le jeu d'entraînement, alors la précision masquée vaut 0.0, ce qui signifie que le modèle prédit toujours la mauvaise structure.

Ce fait implique que toutes les requêtes attendues qui ne correspondent pas à ce patron de requête ne seront pas correctement générées. On constate que tous les modèles de copie avec des questions avec tag-interne ont des performances très similaires, autour de 70% de précision. Cette limite de performance ne se produit pas pour les questions avec tag-final. Dans le cas des questions avec tag-final, la question complète est transmise aux modèles. Bien que les éléments de la base de connaissances soient masqués, la sémantique de la question peut être identifiée par sa formulation en langue naturelle. On peut voir par exemple que BART parvient à dépasser la limite de 70% de précision avec le mécanisme de copie sur les questions avec tag-final.

Structure de requête	Fréquence	Précision des requêtes masquées
select ?obj where { <mask> <mask> ?s. ?s <mask> ?obj. ?s <mask> ?x filter (contains(year(?x) , <mask>))}	315	1.0
select ?obj where { <mask> <mask> ?s. ?s <mask> ?obj. ?s <mask> <mask>}	231	0.0
select ?obj where { <mask> <mask> ?s. ?s <mask> ?obj. ?s <mask> ?x filter (contains(?x, <mask>))}	42	0.0

TABLEAU 3.11 Précision des requêtes masquées pour un patron de question commun au patron de question "what is <mask> <mask> of <mask> that is <mask> is <mask> ?" en fonction de la fréquence des patrons de requêtes associés dans le jeu d'entraînement

3.5.2 Difficulté de l'annotation tag-final pour le mécanisme de copie

On peut également noter que de nombreux modèles ont du mal à traiter les questions avec tag-final lorsqu'ils utilisent le mécanisme de copie, cela se produit principalement avec certains modèles basés sur les Transformers, notamment le Transformer non pré-entraîné et T5. Cela est probablement dû au fait que la mise en place des éléments de la BC dans la requête est conditionnée par l'emplacement adéquat de l'URI correspondante dans la question. Dans le cadre des questions avec tag-interne, les têtes d'attention croisée apprennent facilement à

associer le bon endroit dans la question au bon endroit dans la requête. Cependant, avec les questions avec tag-final, le modèle doit associer l'élément de la BC à son libellé, puis associer ce libellé à sa formulation correspondante dans la question d'origine. Ces étapes supplémentaires pourraient être à l'origine des difficultés observées avec certains modèles de Transformers. On peut supposer que l'opération de convolution est plus adaptée pour traiter ces étapes, peut-être en raison de la proximité entre l'URI et son libellé à la fin des questions annotées. De plus, BART semble également surmonter ces difficultés, on peut suggérer que ses tâches de pré-entraînement basées sur le débruitage ont conduit à de meilleures associations de contexte à court terme.

3.5.3 Limitations du mécanisme de copie

La principale limitation de ce mécanisme est qu'il repose principalement sur la structure des paires questions-requêtes et sur la correspondance de position des URI entre les questions et les requêtes. Il est possible que ces modèles se concentrent uniquement sur la tâche de mise en correspondance des patrons de questions et de requêtes et de remplissage des espaces réservés. Cela peut poser des problèmes lorsque le modèle est exposé à une paire question-requête qui est générée par un modèle de question qui n'est associé à aucune ou à trop peu d'entrées d'entraînement. Dans certains cas, le mécanisme de copie réussit à améliorer les performances des modèles, mais il ne semble pas permettre de résoudre systématiquement le problème de patrons inconnus.

Une autre limitation de ces architectures améliorées est qu'elles sont aveugles à la sémantique des URIs. L'encodeur décodeur ne voit pas ces jetons, car ils sont masqués et le mécanisme de copie n'utilise que les positions pour copier les jetons. Cela peut poser des problèmes lorsque la requête attendue ne dépend pas exclusivement de la structure de la question.

3.5.4 Autres modèles

Ce mémoire se concentre sur les techniques de TALN basées sur des encodeurs décodeurs qui sont entièrement disponibles pour l'entraînement et les tests. De plus, seuls BART et T5 sont utilisés afin de comparer les résultats à ceux des approches de référence. Cependant, récemment, plusieurs gros modèles de langage tels que Codex [35] et InstructGPT [36] ont montré des performances impressionnantes dans la génération de texte et de code. Ces modèles n'ont pas été étudiés, car ils sont basés sur des API.

3.5.5 Limites de l’approche

Ce chapitre rapporte de très hautes performances sur chaque jeu de données, notamment grâce aux méthodologies d’annotation, aux modèles pré-entraînés et au mécanisme de copie. Cependant, les performances ne sont pas encore parfaites, voire nécessitent encore beaucoup d’amélioration dans le cas de LC-QuAD 2.0. D’une part, on a vu que la difficulté principale des modèles non pré-entraînés sans le mécanisme de copie est la gestion des URIs inconnues. Bien que l’augmentation des performances totales lorsque l’on utilise de meilleurs modèles semble suggérer que cette limitation est dépassée, on n’évalue pas directement si les modèles plus performants savent effectivement bien gérer les URIs inconnues. D’autre part, l’une des limitations relevées dans cette section du mécanisme de copie est sa tendance à se reposer sur la structure de la paire de questions-requêtes. Cela pose la question des performances que peuvent avoir les modèles avec copie s’ils sont confrontés à des structures questions-requêtes inconnues. De même, on peut se demander si les modèles pré-entraînés performant toujours aussi bien dans ce cas. Pour évaluer directement les capacités des modèles à gérer les URIs inconnues et les structures de questions-requêtes inconnues, le chapitre suivante propose une manière de séparer les jeux de données de façon à avoir ce genre d’entrées dans le jeu de test. Les modèles seront alors entraînés et évalués sur ces nouvelles séparations des jeux de données afin d’étudier leurs capacités de généralisation.

CHAPITRE 4 GÉNÉRALISATION

Ce chapitre vise à étudier les capacités de généralisation des modèles présentés dans le chapitre 3. On commence par décrire les propriétés nécessaires d'un jeu de données pour évaluer les capacités de généralisation d'un modèle. Puis, on présente un algorithme pour transformer les jeux de données populaires afin qu'ils aient les propriétés en question. Enfin, on entraîne et évalue les modèles sur ces nouveaux jeux de données pour analyser leurs capacités de généralisation.

Ce chapitre a mené à la rédaction d'un article (Reyd and Zouaq, 2023) [37] qui a été accepté à la conférence ISWC 2023.

4.1 Propriété de généralisation

Pour évaluer les capacités de généralisation des modèles, on définit les propriétés de généralisation des ensembles de données et de leur répartition en fonction de certaines caractéristiques des paires de questions-requêtes. Les caractéristiques des entrées sont des éléments spécifiques, tels que le patron global ou les URIs présents dans l'entrée. Les propriétés de généralisation évaluent la distribution de ces caractéristiques par rapport à la séparation de l'ensemble de données en jeu de test et jeu d'entraînement.

Plus formellement, soit D un ensemble de données, soit D_1 un sous-ensemble de référence de cet ensemble de données (généralement l'ensemble de test), et $\overline{D}_1$ le reste de l'ensemble de données (généralement l'ensemble d'entraînement). Pour tout $d \in D$, soit $c(d)$ l'ensemble des caractéristiques des données (ici, soit les URIs, soit le patron global d'une paire de questions-requêtes). On définit une propriété de généralisation gp comme vraie si chaque entrée d_1 de D_1 contient au moins une caractéristique qui n'apparaît pas dans $\overline{D}_1$, c'est-à-dire

$$gp(D, D_1) \equiv \forall d_1 \in D_1; \exists c_1 \in c(d_1) : c_1 \notin \cup_{d \in \overline{D}_1} (c(d)) \quad (4.1)$$

On définit ensuite deux propriétés des ensembles de données qui sont utiles pour évaluer les capacités de généralisation des modèles de traduction automatique neuronale.

Ces propriétés utilisent l'ensemble de test du jeu de données comme ensemble de référence. Il s'agit donc de propriétés portant sur un jeu de données et sur sa séparation. Ici, le terme séparation est utilisé comme traduction du terme anglais "split" qui caractérise le fait qu'on ait associé à un jeu de donnée une assignation de chaque entrée soit à l'ensemble d'entraînement,

soit à l'ensemble de test.

Propriété d'URIs inconnues. On évalue d'abord le problème des URIs inconnues. En effet, générer la bonne URI est nécessaire pour produire des requêtes SPARQL correctes. La capacité d'un modèle à généraliser à des URIs inconnues est très importante, car les bases de connaissances réelles sont extrêmement massives et aucun jeu de données de traduction de requêtes SPARQL ne peut les couvrir exhaustivement. Même si les jeux de données essaient de couvrir un maximum de sujets différents, il n'est pas possible que l'un d'entre eux contienne suffisamment de mentions de chaque URIs pour qu'un modèle neuronal apprenne une représentation significative de chacune. On définit donc une première propriété de généralisation :

Propriété 1 : Un jeu de donnée doit contenir dans son jeu de test uniquement des paires questions-requêtes ayant au moins une URI qui n'apparaît pas dans le jeu d'entraînement.

Propriété de patrons inconnus. Une autre difficulté particulière de la tâche est liée à la façon dont sont générées les paires questions-requêtes des jeux de données utilisés. Ces données sont générées automatiquement à l'aide d'un ensemble limité de structures. Cela signifie que toutes les paires suivent des structures très précises. Ainsi, il n'est pas clair si les modèles réussissent à effectivement traduire les questions en requêtes à partir de leur sens ou uniquement en reconnaissant leurs structures. Cela pose la question de si ces modèles sont capables de traiter des paires questions-requêtes qui ne suivent pas ces structures. On définit donc la deuxième propriété de généralisation :

Propriété 2 : Un jeu de donnée doit contenir dans son jeu de test uniquement des paires questions-requêtes ayant été générées par un patron global qui n'a généré aucune donnée du jeu d'entraînement.

4.2 Séparation des jeux de données

Afin d'obtenir des jeux de données avec ces propriétés de généralisation, on utilise les jeux de données classiques de la littérature pour lesquels on redéfinit la séparation entre l'ensemble de test et d'entraînement. Cette section décrit la procédure pour la création de cette nouvelle séparation.

4.2.1 Groupes de données

Pour obtenir ces séparations, on commence par partitionner le jeu de données en sous-ensembles que l'on appelle groupes. Ces groupes sont définis par le fait que tout sous-ensemble de groupes doit être une façon de séparer le jeu de données de manière à avoir une propriété de généralisation donnée.

Plus formellement, étant donné un ensemble de données D , et une propriété de généralisation gp (comme définie dans l'équation 4.1), on appelle groupe une partition de cet ensemble $\{g_1, \dots, g_k\}$ telle que :

$$\forall \mathcal{G} \subset \{g_1, \dots, g_k\} : gp(D, \bigcup \mathcal{G}) \quad (4.2)$$

On définit ensuite la procédure pour construire ces groupes pour chacune des deux propriétés de généralisation définies plus tôt.

Groupes pour la séparation avec patrons inconnus

Dans le cas de patrons globaux de paires questions-requêtes inconnus, les groupes sont composés de paires de questions-requêtes qui partagent le même patron global. Ces groupes peuvent ensuite être assignés à l'ensemble d'entraînement ou à l'ensemble de test. Cela garantit qu'il n'y a pas de patron commun entre eux.

Groupes pour la séparation avec URIs inconnues

La construction des groupes est plus complexe dans le cas de la séparation avec URIs inconnues. L'objectif est de construire des groupes qui contiennent des URIs qui n'apparaissent pas dans les autres groupes. On procède alors suivant les étapes suivantes.

- On définit une fréquence minimale f_{min} pour considérer une URI rare. C'est-à-dire que si une URI est présente dans plus de f_{min} paires de questions-requêtes, on la considère comme commune et sinon on la considère rare. f_{min} est un hyper-paramètre de cette procédure que l'on détermine expérimentalement.
- On fabrique un graphe dont les nœuds sont les entrées du jeu de données (c'est-à-dire les paires de questions-requêtes) et dans lesquels deux nœuds sont reliés si les deux paires de questions-requêtes ont une URI rare en commun.
- Les paires de questions-requêtes qui ne possèdent pas d'URI rare sont réservées pour

l'ensemble d'entraînement et ne sont pas concernées par le reste de la procédure. On considère donc maintenant uniquement un sous ensemble $\tilde{D}$ du jeu de données D .

- On calcule les composantes connexes du graphe basé sur $\tilde{D}$. Ces composantes connexes constituent les groupes pour la séparation pour les URIs inconnues. L'algorithme utilisé pour obtenir les composantes connexes est un parcours en largeur implémenté par la librairie python `networkxx`¹. Les groupes sont alors les paires de questions-requêtes qui peuvent être reliées dans le graphe.

La figure 4.1 montre un exemple d'un graphe de la sorte. Pour plus de lisibilité, les paires de questions-requêtes sont uniquement représentées par leur question. Les concepts correspondant aux URIs présentes dans les paires sont mis en couleurs dans les questions. Dans cet exemple, on prend $f_{min} = 3$. Ainsi, les URIs correspondant aux concepts "réalisateur", "film" et "date de naissance" sont communes et les URIs correspondant aux concepts "Stanley Kubrick", "Barack Obama", "Quentin Tarantino" et "Donald Trump" sont rares. On ne considère donc pas la paire dont la question est "Quels films ont plusieurs réalisateurs" qui ne contient que des URIs communes. Cette entrée est réservée pour l'ensemble d'entraînement et n'est pas concernée par la procédure de séparation, comme décrit précédemment. On relie ensuite les paires qui partagent des URIs rares et on crée les composantes connexes du graphe. On obtient ici trois groupes différents.

Ainsi, si on prend n'importe quel ensemble de ces groupes, toutes les paires de questions-requêtes de ces groupes présenteront au moins une URI qui n'apparaît pas dans le reste des groupes. En effet, chaque entrée d'un groupe possède au moins une URI rare et cette URI est forcément inconnue des autres groupes. Si ce n'était pas le cas, il y aurait une entrée d'un autre groupe qui aurait la même URI et cela aurait entraîné un lien dans le graphe que l'on a défini plus haut. Or cela est impossible, car les groupes sont des composantes connexes de ce même graphe. Ce lien n'existe donc pas, donc cette URI rare est forcément inconnue des autres groupes. On sait donc que toutes les entrées possèdent au moins une URI rare et que toutes les URIs rares d'un ensemble de groupes sont inconnues du reste des groupes. On peut donc en conclure que toutes les entrées d'un ensemble de groupes possèdent au moins une URI inconnue du reste de groupes.

Fréquence minimale pour considérer une URI rare

La fréquence minimale pour considérer qu'une URI est rare est un paramètre expérimental qui a été décidé en suivant les courbes rapportées dans la Figure 4.2 et en essayant d'obtenir

1. Documentation du module

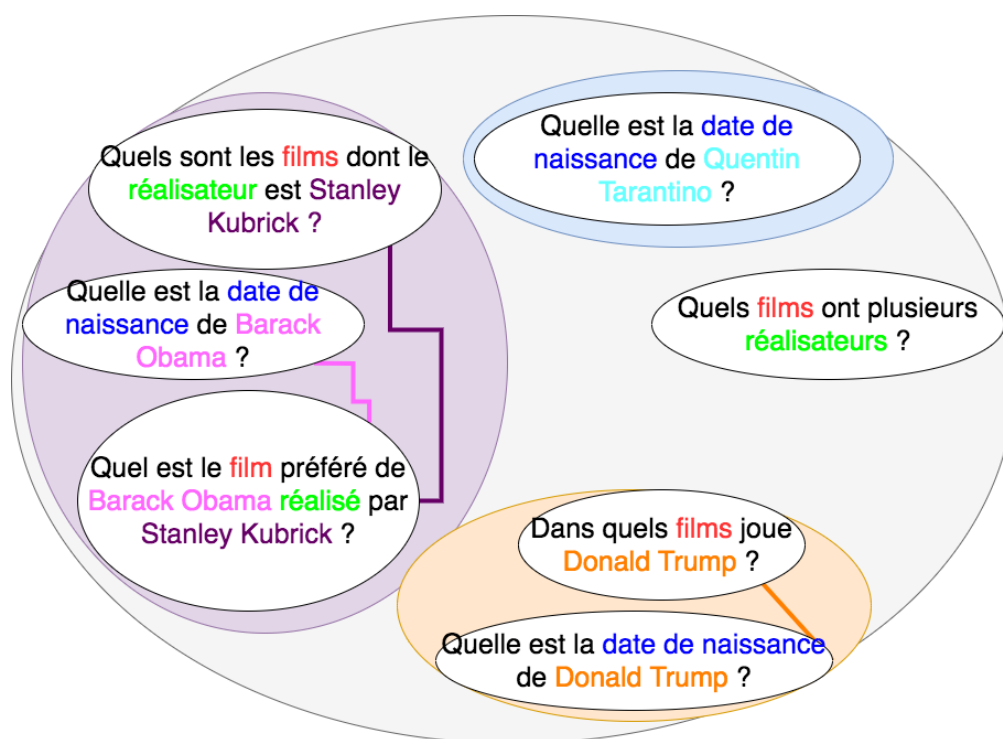


FIGURE 4.1 Exemple de graphe utilisé pour la création des groupes pour la séparation avec URIs inconnues

les deux éléments suivants. D'une part, on cherche à avoir un nombre important de groupes à répartir entre le jeu d'entraînement et le jeu de test. Il n'y a pas de référence a priori d'un "nombre important" de groupe. Cependant, on peut remarquer que pour le cas de la séparation avec patrons inconnus, on obtient une trentaine de groupes pour nos deux jeux de données. Or, on verra ensuite que ce nombre de groupes permet d'utiliser efficacement notre algorithme de séparation. On peut donc arbitrairement considérer qu'il faut au moins une trentaine de groupes pour faire fonctionner l'algorithme. Pour une approche plus rigoureuse de l'analyse des performances de la procédure de séparation en fonction du nombre de groupes, se référer à l'annexe ??.

On veut donc un "nombre important" de composantes connexes, ce qui implique une faible connectivité du graphe et donc une fréquence minimale faible. D'autre part, on cherche à avoir peu d'entrées réservées au jeu d'entraînement et n'appartenant à aucun groupe, c'est-à-dire peu d'entrées n'ayant aucune URI inconnue. Cela implique une fréquence minimale élevée.

On observe sur la Figure 4.2 que le nombre de composantes connexes est généralement très grand et que la proportion du jeu de données ayant une URI rare est très rapidement grande

lorsqu'on augmente la fréquence minimale. En pratique, on choisit une fréquence minimale de 5.

Nombre d'entrées voulu dans le jeu de test

Une fois ces groupes définis, on peut les associer au jeu de test ou d'entraînement sans plus aucune contrainte liée aux propriétés de généralisation. Cependant, on cherche à respecter des proportions particulières pour cette séparation. Ce mémoire vise toujours une séparation à 80% dans le jeu d'entraînement et 20% dans le jeu de test. On note ici que dans cette partie, on utilise le terme "jeu de test" en opposition au terme "jeu d'entraînement". Le premier contient les données qui serviront à optimiser les poids du modèle et le second contient les données qui serviront à évaluer le modèle. Dans le cadre d'une procédure d'entraînement d'un modèle, le jeu de test défini ici est ensuite divisé en un jeu de validation et un jeu également dit "de test". Dans le contexte de l'entraînement d'un modèle, le jeu de validation sert à calculer la perte du modèle au cours de l'entraînement afin de vérifier à quel moment arrêter le processus. Dans ce contexte, le jeu de test sert à l'évaluation finale du modèle après le processus complet d'entraînement du modèle. On utilise donc un jeu de validation ayant la même distribution que le jeu de test. On obtient donc une séparation finale pour l'entraînement et l'évaluation du modèle avec les proportions 80-10-10, où les jeux de validation et de test (dans le contexte de l'entraînement du modèle) contiennent tous les deux des données qui ont des caractéristiques non vues dans le jeu d'entraînement. Dans le reste du chapitre, on parlera à nouveau de jeu de test comme opposition au jeu d'entraînement.

Pour répartir les groupes dans le jeu de test ou d'entraînement, on définit une taille désirée du jeu d'entraînement notée N . Dans le cas de la séparation avec patrons inconnus, l'expression de N est immédiate : $N = 0.8 \times |D|$, où D est le jeu de données en question.

Dans le cas de la séparation avec URIs inconnues, on a déjà assigné les entrées sans URI rare à l'ensemble de test, donc on doit diminuer le nombre d'entrées que l'on veut encore assigner à cet ensemble. On obtient : $N = 0.8 \times |D| - (|D| - |\tilde{D}|) = |\tilde{D}| - 0.2 \times |D|$.

Procédure de répartition des groupes

On cherche à répartir les groupes dans le jeu d'entraînement *train* ou dans le jeu de test *test* de manière que la taille de *train* soit proche de sa taille désirée N .

Pour cela, on initialise l'ensemble d'entraînement *train* et l'ensemble de test *test* comme étant des ensembles vides. On itère ensuite sur chaque groupe g . Si la taille de *train* est supérieure à N , on assigne le groupe à *test*. À l'inverse, si la taille de *test* est supérieure à $|D| - N$,

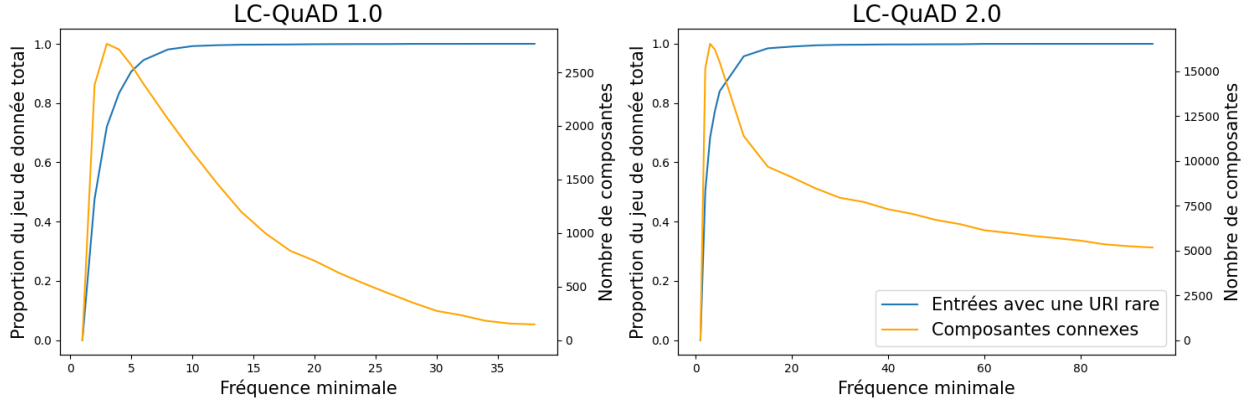


FIGURE 4.2 Proportion d’entrées du jeu de données avec une URI rare et nombre de composantes connexes en fonction de la fréquence minimale de rareté

on assigne le groupe à *train*. Enfin, si aucune de ces conditions n’est vraie, on calcule une probabilité $p = \frac{N - |train|}{|D| - |train| - |test|}$ et on assigne le groupe à *train* avec une probabilité p ou à *test* avec une probabilité $1 - p$. L’algorithme 1 reprend cette procédure en pseudo-code.

Étant donnée la conception des groupes, on a la garantie que le jeu de donnée muni de cette séparation possède les propriétés de généralisation souhaitées. On définit une métrique $\delta = \frac{|N - |train||}{|D|}$ que l’on souhaite minimiser. Dans cette procédure, une fois que l’ensemble d’entraînement ou de test a atteint ou dépassé sa taille souhaitée, aucun autre groupe ne peut lui être affecté. Ainsi, il n’est pas possible que la taille finale de l’ensemble d’entraînement ou de test soit supérieure à sa taille souhaitée plus la taille du dernier groupe qui lui a été affecté, c’est-à-dire $\delta < |g_f|/|D|$ où g_f est le dernier groupe à avoir été affecté à l’ensemble d’entraînement.

Or, en pratique, la taille des groupes décroît très vite. Cela a pour conséquence que la taille du plus grand groupe peut être assez grande et donc que la borne sur δ peut être assez mauvaise si le dernier groupe est le plus grand. Cependant, toujours étant donnée la décroissance rapide de la taille des groupes, on peut montrer que la probabilité de sélectionner un petit groupe est très élevée. La sous-section 4.2.1 démontre formellement que selon les distributions observées dans les données, la probabilité d’avoir $\delta < 5\%$ est supérieure à 70%. Il est possible d’avoir une grande valeur de δ mais cela est très peu probable (d’autant peu probable que l’on a beaucoup de groupes ou que la distribution est déséquilibrée). On exécute donc la procédure plusieurs fois et on garde l’exécution avec le plus petit δ .

On peut remarquer que cette procédure est en fait une heuristique stochastique d’un problème de la somme de sous-ensembles. Ce mémoire ne vise pas à proposer une bonne solution à ce problème de manière générale, mais à obtenir une séparation de jeux de données suivant les

Algorithm 1 Procédure d'assignation des groupes

```

1: Entrée : un ensemble de groupes  $G$ , la taille désirée du jeu d'entraînement  $N$ 
2: Sortie : les jeux d'entraînement et de test
3:  $train \leftarrow set()$ 
4:  $test \leftarrow set()$ 
5: for all  $group \in G$  do
6:   if  $|train| > N$  then
7:      $assign\_train \leftarrow 0$ 
8:   else if  $|test| > |D| - N$  then
9:      $assign\_train \leftarrow 1$ 
10:  else
11:     $p\_train \leftarrow \frac{N - |train|}{|D| - |train| - |test|}$ 
12:     $assign\_train \leftarrow \text{Bernoulli}(p\_train)$ 
13:  end if
14:  if  $assign\_train = 1$  then
15:     $train \leftarrow train \cup group$ 
16:  else
17:     $test \leftarrow test \cup group$ 
18:  end if
19: end for
20: Renvoi :  $train, test$ 

```

propriétés de généralisations définies plus tôt et les proportions souhaitées. Cette procédure est simple et permet d'obtenir le résultat désiré, mais une étude plus poussée du problème général pourrait permettre un meilleur algorithme.

Preuve de la probabilité de majoration de δ

On a vu que la taille des groupes utilisés pour obtenir les nouvelles séparations des jeux de données décroît très rapidement. De plus, on a vu que l'on peut majorer la distance relative entre la taille voulue du jeu d'entraînement et sa taille obtenue par l'algorithme de séparation par la taille du dernier groupe associé au jeu d'entraînement. C'est-à-dire que l'on définit une métrique $\delta = \frac{|N - |train||}{|D|}$, avec N la taille souhaitée du jeu d'entraînement, $|train|$ la taille effective du jeu d'entraînement après la procédure de séparation et D le jeu de données utilisé. On a également montré que $\delta \leq \frac{|g_f|}{|D|}$, où g_f est le dernier groupe associé au jeu d'entraînement. Or, on peut supposer que ce dernier groupe est choisi uniformément parmi les groupes. Ainsi, étant donné la décroissance très rapide de la taille des groupes, on peut supposer que si le dernier groupe choisi dans l'ensemble d'entraînement est l'un des plus grands groupes, on aura une grande différence entre les tailles voulue et obtenue de l'ensemble d'entraînement. Cependant, cette décroissance très rapide permet aussi de supposer que l'on

a une bonne probabilité de finir par un groupe de faible taille et donc d'avoir une bonne taille de l'ensemble d'entraînement. Cette annexe tente de modéliser les distributions de la taille des groupes pour les jeux de données LC-QuAD 1.0 et LC-QuAD 2.0, puis de calculer en pratique et en théorie les probabilités d'avoir $\delta \leq \epsilon$ pour ϵ arbitrairement petit.

Dans un premier temps, on essaie de modéliser la distribution de la taille des groupes dans les jeux de données. La figure 4.3 montre que l'on peut assez bien approcher la distribution des groupes pour la séparation avec patrons inconnus par une progression géométrique. On modélise alors la suite réelle des tailles relatives $x_i = |g_i|/|D|$ des groupes g_i par la suite $\tilde{x}_i = x_0 \times \frac{1}{\lambda^i}$ avec $\lambda \in]1, \infty[$ le paramètre de la progression. Pour obtenir la valeur de λ , on calcule la moyenne de la différence entre les tailles des groupes dans le domaine logarithmique. La figure 4.3 rapporte la valeur estimée de λ ainsi que le MSE de l'estimation, c'est-à-dire $MSE = \frac{1}{k} \sum_{i=0}^{k-1} (x_i - \tilde{x}_i)^2$. Dans le cas des groupes pour la séparation avec URIs inconnues, on observe que le nombre de groupes est largement plus grand et que les valeurs des x_i sont inférieures aux valeurs estimées. Dans ce cas, on observe également que même le plus grand groupe représente 0.6% des données dans le cas de LC-QuAD 1.0 et 0.6% dans le cas de LC-QuAD 2.0. Pour les groupes pour la séparation avec URIs inconnues, on peut donc affirmer que $\mathbb{P}(\delta \leq 0.01) = 1$. Dans le cas des groupes formés sur les patrons, on va utiliser la modélisation obtenue pour calculer une borne inférieure à $\mathbb{P}(\delta \leq \epsilon)$ avec ϵ un réel strictement positif arbitrairement petit.

On pose donc une suite de k nombres $\{x_i\}_{i \in \{1, \dots, k-1\}}$ qui représentent les tailles relatives de groupes. On suppose, de plus, sans perte de généralité, que les x_i sont triés par ordre décroissant. Enfin, on suppose que ces groupes suivent une progression géométrique de paramètre $\lambda \in]1, \infty[$. On peut alors affirmer les propriétés suivantes :

$$\forall i \in \{1, \dots, k-1\} : x_i \in]0, 1[\quad (4.3)$$

$$\sum_{i=0}^{k-1} x_i = 1 \quad (4.4)$$

$$\forall i \in \{1, \dots, k-1\} \forall j \in \{1, \dots, k-1\}, i \leq j \implies x_i \geq x_j \quad (4.5)$$

$$\forall i \in \{1, \dots, k-1\} : x_i = \frac{1}{\lambda^i} g_1 \quad (4.6)$$

On pose alors un réel strictement positif arbitrairement petit ϵ et on cherche alors à démontrer l'inégalité suivante.

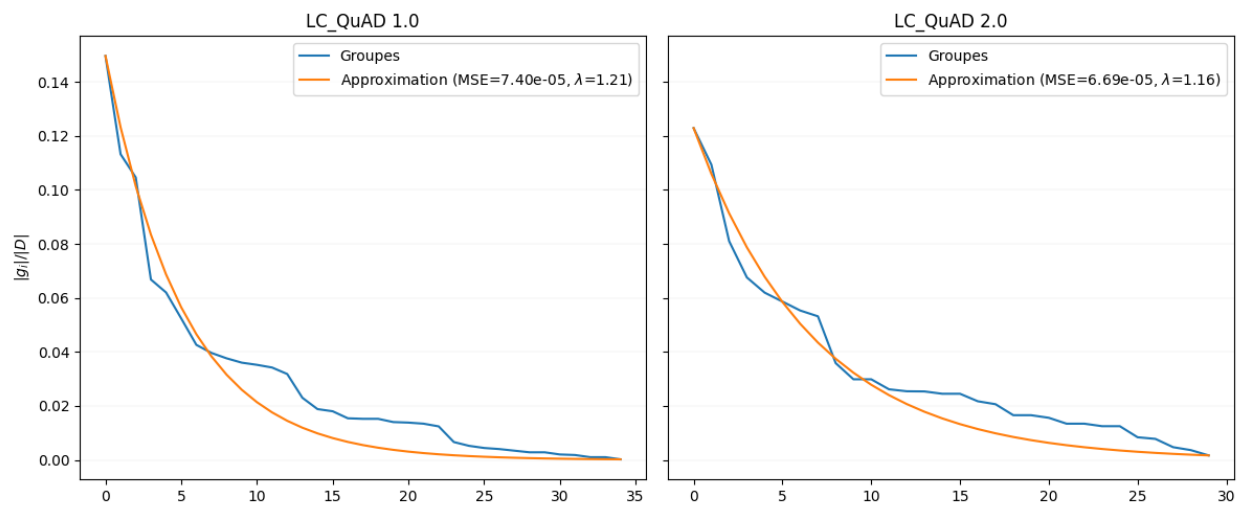


FIGURE 4.3 Distribution des groupes pour la séparation avec patrons inconnus

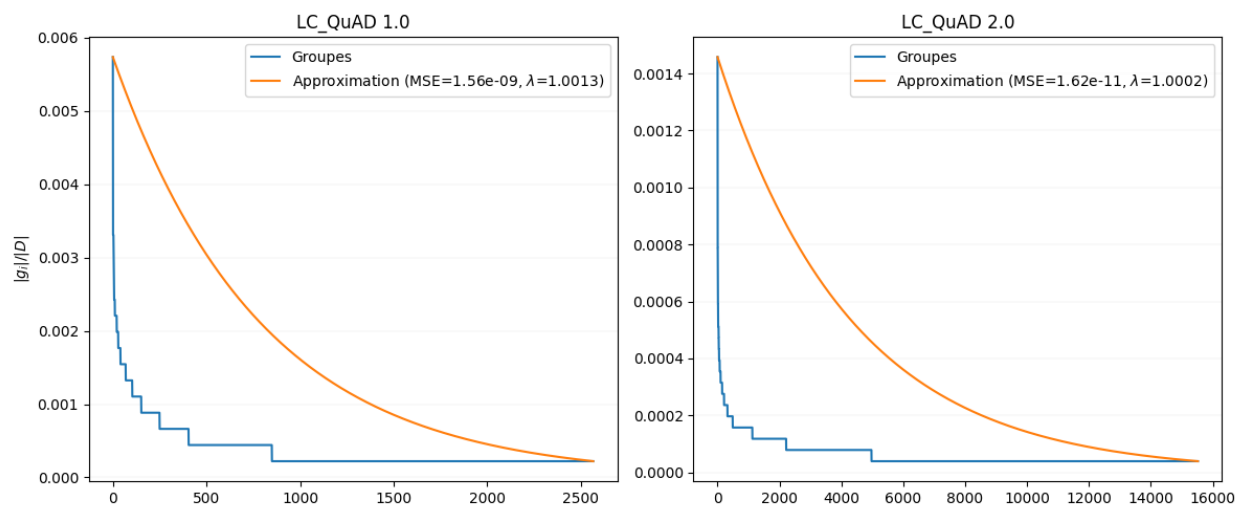


FIGURE 4.4 Distribution des groupes pour la séparation avec URIs inconnues

$$\mathbb{P}(\delta \leq \epsilon) \geq 1 - \frac{1}{k} \left\lceil \frac{1}{\ln(\lambda)} \ln \left(\frac{1}{\epsilon} \times \frac{\lambda - 1}{1 - \frac{1}{\lambda^k}} \right) \right\rceil$$

On définit ensuite une variable aléatoire de loi uniforme sur les indices des groupes :

$$I \sim \mathcal{U}(\{1, \dots, k-1\}) \quad (4.7)$$

On peut alors exprimer la probabilité de la majoration de x_I par ϵ :

$$\mathbb{P}(x_I \leq \epsilon) = \sum_{i=0}^{k-1} \mathbb{P}(x_I \leq \epsilon | I = i) \mathbb{P}(I = i) \quad (4.8)$$

$$= \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{P}(x_I \leq \epsilon | I = i) \quad (4.9)$$

$$= \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{P}(x_i \leq \epsilon) \quad (4.10)$$

$$= \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{1}_i(x_i \leq \epsilon) \quad (4.11)$$

$$= \frac{1}{k} (k - i_\epsilon) \quad (4.12)$$

$$= 1 - \frac{i_\epsilon}{k} \quad (4.13)$$

Où i_ϵ est le premier i tel que $x_i \leq \epsilon$.

On peut alors exprimer i_ϵ :

$$x_i \leq \epsilon \iff \frac{1}{\lambda^{i_\epsilon}} x_1 \leq \epsilon \quad (4.14)$$

$$\iff \lambda^{i_\epsilon} \geq \frac{x_1}{\epsilon} \quad (4.15)$$

$$\iff i_\epsilon \geq \log_\lambda \frac{x_1}{\epsilon} \quad (4.16)$$

Ainsi, on obtient :

$$i_\epsilon = \lceil \log_\lambda \frac{x_1}{\epsilon} \rceil \quad (4.17)$$

Or, comme $\sum_i x_i = 1$, on a $\sum_i \frac{1}{\lambda^i} x_1 = 1$ et on peut exprimer x_1 ainsi :

$$x_1 = \frac{1}{\sum_i \frac{1}{\lambda^i}} \quad (4.18)$$

Or

$$\sum_{i=0}^{k-1} \frac{1}{\lambda^i} = \frac{1}{\lambda} \times \frac{1 - \frac{1}{\lambda}^k}{1 - \frac{1}{\lambda}} = \frac{1 - \frac{1}{\lambda}^k}{\lambda - 1} \quad (4.19)$$

Donc

$$x_1 = \frac{\lambda - 1}{1 - \frac{1}{\lambda}^k} \quad (4.20)$$

Donc, on obtient

$$\mathbb{P}(x_I \leq \epsilon) = 1 - \frac{1}{k} \left\lceil \frac{1}{\ln(\lambda)} \ln \left(\frac{1}{\epsilon} \times \frac{\lambda - 1}{1 - \frac{1}{\lambda}^k} \right) \right\rceil \quad (4.21)$$

Or, on a vu que

$$\delta < x_I \quad (4.22)$$

Donc, on obtient bien :

$$\mathbb{P}(\delta \leq \epsilon) \geq 1 - \frac{1}{k} \left\lceil \frac{1}{\ln(\lambda)} \ln \left(\frac{1}{\epsilon} \times \frac{\lambda - 1}{1 - \frac{1}{\lambda}^k} \right) \right\rceil$$

On peut alors estimer une borne supérieure pour la probabilité de majoration de δ . Dans le cas de la séparation avec URIs inconnues, on a déjà vu que l'on a $\mathbb{P}(\delta \leq 0.01) = 1$ donc également $\mathbb{P}(\delta \leq 0.05) = 1$. Dans le cas de la séparation avec patrons inconnus, on obtient $\mathbb{P}(\delta \leq 0.05) = 83\%$ avec λ estimé sur les données de LC-QuAD 1.0 et $\mathbb{P}(\delta \leq 0.05) = 77\%$ avec λ estimé sur LC-QuAD 2.0. En utilisant les tailles des groupes utilisés en pratique, on obtient $\mathbb{P}(\delta \leq 0.05) = 83\%$ pour LC-QuAD 1.0 et $\mathbb{P}(\delta \leq 0.05) = 73\%$ pour LC-QuAD 2.0. On a bien une forte probabilité d'avoir une valeur faible de δ , ce qui assure que l'application de la procédure de séparation un certain nombre de fois donnera une taille du jeu d'entraînement proche de celle voulue.

4.3 Statistiques des nouveaux jeux de données

Pour l'étude de la généralisation, ce mémoire utilise uniquement les jeux de données LC-QuAD 1.0 et LC-QuAD 2.0. Ce choix a été fait après l'obtention des résultats sur les séparations originales. Ces résultats montrent que LC-QuAD 1.0 semble être le jeu de données le plus "simple" parmi les trois jeux considérés dans ce mémoire et que LC-QuAD 2.0 semble être le plus "difficile". Utiliser ces deux jeux de données devrait donc bien donner une idée des bornes inférieures et supérieures des capacités de généralisation des modèles. D'autre part, la taille extrêmement importante de DBNQA rend les expérimentations beaucoup plus longues sur ce jeu de données, ce qui a également motivé le choix de ne pas le considérer pour l'étude de la généralisation des modèles.

Une fois les nouvelles séparations obtenues, on observe expérimentalement que les propriétés de généralisation recherchées sont bien vérifiées. Les jeux de test de la séparation avec URIs inconnues sont bien constitués entièrement d'entrées qui possèdent au moins une URI inconnue. Les jeux de test de la séparation avec patrons inconnus sont bien constitués entièrement d'entrées générées par des patrons qui n'ont généré aucune donnée du jeu d'entraînement complémentaire.

La Table 4.1 présente le nombre d'URIs, de patrons globaux et de paires de questions-requêtes pour les ensembles d'entraînement/validation/test obtenus pour la séparation avec URIs inconnues et la séparation avec patrons inconnus. On observe qu'il y a moins d'URIs inconnues dans l'ensemble de test de la séparation avec URIs inconnues que dans l'ensemble de test de la séparation originale. Cela s'explique par le fait que la séparation originale n'utilisait pas une séparation de 80-10-10 et avait donc environ deux fois plus de données dans son ensemble de test. Cependant, la nouvelle division garantit que 100% des données de l'ensemble de test comportent au moins une URI inconnue, ce qui n'est pas le cas dans la division originale.

	Total	Entr.	Val.	Test	HV	Entr.	Val.	Test	HV
		Séparation avec patrons inconnus				Séparation avec URIs inconnues			
		URIs							
LC 1.0	4751	4183	925	912	282	3872	909	948	535
LC 2.0	31018	27986	4403	4429	1619	25121	4738	4708	3437
		Patrons globaux							
LC 1.0	35	31	4	4	4	35	32	30	0
LC 2.0	30	24	6	6	6	30	30	30	0
		Données							
LC 1.0	5000	4000	500	500	500	4000	500	500	500
LC 2.0	30225	24178	3023	3024	3024	24180	3022	3023	3023

TABLEAU 4.1 Nombre d’URIs, de patrons globaux, et de paires de questions-requêtes dans les jeux de données avec les nouvelles séparations

4.4 Généralisation des modèles non pré-entraînés

Cette section présente les résultats des entraînements et des évaluations de modèles sur les jeux de données originaux et les nouvelles séparations. Les tableaux de résultats rapportent les métriques d’évaluation en pourcentage arrondi à deux chiffres significatifs afin de pouvoir plus facilement comparer les résultats au sein d’un même tableau.

4.4.1 Séparation avec patrons globaux inconnus

Les résultats des expériences sur les modèles non pré-entraînés pour la séparation avec patrons inconnus sont rapportés dans la Table 4.2.

On observe que les modèles sans le mécanisme de copie subissent systématiquement une forte baisse de performance par rapport à la séparation originale. Pour LC-QuAD 1.0, on constate généralement une augmentation constante de la performance avec les questions annotées, mais la précision des réponses reste inférieure à 60%. Pour LC-QuAD 2.0 avec Transformer, les résultats sont encore plus bas que dans la séparation originale et les réponses sont presque toujours fausses. Pour ConvSeq2Seq, le modèle a de meilleures performances, mais elles restent faibles.

Cependant, on observe un impact notable du mécanisme de copie sur les résultats des modèles non pré-entraînés. Bien qu’ils restent inférieurs à ceux de la séparation originale, ils sont bien meilleurs avec le mécanisme de copie. De plus, le score BLEU est notablement plus bas qu’avec la division originale, alors que la précision des réponses diminue moins. Cela suggère que la structure des requêtes prédites ne correspond pas toujours à celle des requêtes attendues tout

en fournissant des réponses correctes. Des explications sur cette situation sont données dans la section 4.6.

Dans l'ensemble, les modèles non pré-entraînés semblent avoir du mal avec les patrons globaux qui n'ont pas été vus pendant l'entraînement.

4.4.2 Séparation avec URIs inconnues

Les résultats des expériences sur les modèles non pré-entraînés pour la séparation avec URIs inconnues sont rapportés dans la Table 4.3.

Ces résultats montrent clairement l'impact du mécanisme de copie sur la gestion des URIs inconnues et mettent également en évidence que cette séparation crée une difficulté réelle pour les modèles non pré-entraînés par rapport à l'ensemble de données d'origine.

Sans le mécanisme de copie, on constate qu'aucun modèle n'atteint 15% de précision sur LC-QuAD 1.0 ni 5% sur LC-QuAD 2.0. Même si les questions annotées conduisent à des performances légèrement plus élevées, l'impact de l'annotation est beaucoup moins perceptible que pour d'autres paramètres.

Au contraire, avec le mécanisme de copie, on remarque clairement que la performance (selon toutes les mesures) en utilisant des questions avec tag-interne est beaucoup plus élevée que sans le mécanisme de copie. On atteint les résultats de la séparation originale. Cela suggère que les URIs inconnues sont bien gérées, en particulier en utilisant les questions avec tag-interne. En effet, l'introduction d'URIs inconnues dans chaque donnée de test sans perte de score F1 (comme vu en comparant le Tableau 3.9 et le Tableau 4.3) suggère que les erreurs commises pourraient ne pas être causées par des URIs inconnues. Cependant, utiliser des questions avec tag-final n'améliore pas les performances des modèles.

		Transformer						ConvSeq2Seq					
		Sans copie			Avec copie			Sans copie			Avec copie		
		Bleu	Acc	F1	Bleu	Acc	F1	Bleu	Acc	F1	Bleu	Acc	F1
LC 1	Brut	37	6.9	8.0	/	/	/	43	17	19	/	/	/
	Tag-interne	36	6.3	6.4	50	24	24	46	19	22	50	41	41
	Tag-final	42	11	12	37	2.8	2.8	50	18	20	60	55	55
LC 2.0	Brut	43	0.2	0.3	/	/	/	52	3.9	4.5	/	/	/
	Tag-interne	39	0.1	0.2	64	31	34	52	1.3	1.8	63	33	34
	Tag-final	40	0.2	0.2	36	0.3	0.3	54	2.5	2.8	57	12	13

TABLEAU 4.2 Résultats des modèles non pré-entraînés sur la séparation avec patrons inconnus

		Transformer						Conv					
		Sans copie			Avec copie			Sans copie			Avec copie		
		bleu	acc.	F1	bleu	acc.	F1	bleu	acc.	F1	bleu	acc.	F1
LC 1	Brut	59	7.0	10	/	/	/	64	6.9	10	/	/	/
	Tag-interne	70	8.0	12	98	95	95	71	8.4	12	98	96	96
	Tag-final	67	6.1	8.6	46	0.2	0.2	70	6.9	9.6	86	60	62
LC 2	Brut	60	0.3	0.5	/	/	/	74	1.7	2.7	/	/	/
	Tag-interne	63	0.4	0.5	86	72	73	74	1.0	2.0	90	72	73
	Tag-final	61	0.4	0.6	54	1.8	1.8	73	1.9	2.9	87	53	53

TABLEAU 4.3 Résultats des modèles non pré-entraînés sur la séparation avec des URIs inconnues

4.5 Généralisation des modèles pré-entraînés

4.5.1 Séparation avec patrons globaux inconnus

Les résultats des expériences sur les modèles pré-entraînés pour la séparation avec patrons inconnus sont rapportés dans la Table 4.4.

Contrairement aux modèles non pré-entraînés, les résultats originaux des modèles pré-entraînés sont assez hétérogènes. Les résultats sur les questions brutes sont faibles pour T5 et BART (même s'ils sont meilleurs que n'importe quelle configuration de modèles non pré-entraînés sans le mécanisme de copie). BART sans le mécanisme de copie est la seule architecture pré-entraînée qui a des performances nettement inférieures à ConSeq2Seq avec le mécanisme de copie. Dans le cas de LC-QuAD 1.0 dans sa séparation originale, à la fois BART avec le mécanisme de copie et T5 avec et sans le mécanisme de copie ont de bonnes performances. Dans le cas de LC-QuAD 2.0, aucun modèle ne dépasse 50% de précision.

		Bart						T5					
		Sans copie			Avec copie			Sans copie			Avec copie		
		Bleu	Acc	F1	Bleu	Acc	F1	Bleu	Acc	F1	Bleu	Acc	F1
LC 1	Brut	62	41	42	/	/	/	65	52	53	/	/	/
	Tag-interne	79	87	88	63	99	99	80	92	92	57	50	50
	Tag-final	80	90	90	61	42	43	75	78	79	66	51	51
LC 2	Brut	50	0.4	0.4	/	/	/	59	1.1	1.2	/	/	/
	Tag-interne	70	47	47	65	34	34	69	48	48	61	14	14
	Tag-final	52	21	22	62	12	12	68	47	47	62	7.9	8.2

TABLEAU 4.4 Résultats des modèles pré-entraînés sur la séparation avec patrons inconnus

4.5.2 Séparation avec URIs inconnues

Les résultats des expériences sur les modèles pré-entraînés pour la séparation avec URIs inconnues sont rapportés dans la Table 4.5.

Pour LC-QuAD 1.0 les résultats restent similaires à ceux de la séparation originale. Cela suggère qu’avec des données simples, les modèles pré-entraînés sont capables de gérer les URI inconnues. On peut également constater que T5 avec le mécanisme de copie augmente légèrement ses performances. Dans le cas de LC-QuAD 2.0, on observe parfois une baisse des performances par rapport à la séparation originale. Cela suggère que la complexité des données ou la présence de littéraux pourraient avoir un impact sur la gestion des URIs inconnues pour les modèles pré-entraînés.

4.6 Discussion

4.6.1 Faibles performances des modèles non pré-entraînés sur les questions avec tag-final sur la séparation des URI inconnues

On observe que le mécanisme de copie permet aux modèles non pré-entraînés d’atteindre les mêmes performances sur la séparation des URI inconnues que sur la séparation d’origine pour les questions avec tag-interne, mais pas pour les questions avec tag-final. Cela peut s’expliquer par le fait que le mécanisme de copie utilise le libellé de l’URI, à la fin de la question, pour la faire correspondre à sa position dans la question et à sa position dans la requête. Mais, avec cette séparation, il y a de nombreuses URIs inconnues, et donc une très forte probabilité que le libellé indiqué à côté de l’URI soit également un jeton inconnu. Bien que le mécanisme de copie soit capable de surmonter les URIs inconnues, il a besoin d’une référence pour faire correspondre la position de l’URI donnée à la position de sa mention en

		Bart						T5					
		Sans copie			Avec copie			Sans copie			Avec copie		
		Bleu	Acc	F1	Bleu	Acc	F1	Bleu	Acc	F1	Bleu	Acc	F1
LC 1	Brut	77	38	41	/	/	/	72	24	26	/	/	/
	Tag-interne	96	85	85	99	96	96	98	95	95	99	97	98
	Tag-final	95	83	83	96	93	94	98	94	94	98	96	96
LC 2	Brut	68	0.5	0.6	/	/	/	74	0.7	0.9	/	/	/
	Tag-interne	87	72	72	87	62	62	98	94	94	85	68	68
	Tag-final	86	71	71	92	78	78	97	93	93	92	77	78

TABLEAU 4.5 Résultats des modèles pré-entraînés sur la séparation avec URIs inconnues

langage naturel dans la question, ce qui est impossible lorsque le libellé est hors vocabulaire.

4.6.2 Baisse du score BLEU pour la séparation avec patrons inconnus

On observe que dans la plupart des cas, la marge entre le score BLEU et les scores basés sur les réponses est réduite, voire inversée, pour les séparations avec patrons inconnus. Ce comportement suggère que les modèles prédisent des requêtes qui ne correspondent pas à celles attendues, bien qu'elles génèrent les réponses attendues.

Un exemple significatif de ce comportement est le cas de BART avec le mécanisme de copie sur LC-QuAD 1.0 avec des questions avec tag-interne. On rapporte pour ce cas 99% de score F1 et seulement 63% de score BLEU. En observant les requêtes générées, on peut voir qu'elles suivent toujours la structure d'un patron de requête de l'ensemble d'entraînement. Cela indique que le modèle a intégré la tâche d'association d'un patron de question à un patron de requête avec le remplissage d'espaces réservés. Lorsqu'il reçoit une question de l'ensemble de test, il essaie d'associer la structure inconnue à une structure connue de l'ensemble d'entraînement, puis génère la requête correspondante.

La table 4.6 montre un exemple des structures générées par le modèle sur le patron le plus commun de l'ensemble de test de LC-QuAD 1.0.

Ce patron étant :

Question : "what is the <1> of <2>?"

Requête : `select distinct ?uri where { <2> <1> ?uri }`

Par exemple, pour la question de test :

"what is the dbp:hubs of dbr:Cascade_Airways?"

le modèle a prédit :

```
select distinct ?uri where { dbr:Cascade_Airways dbp:hubs ?uri .
                             dbr:Cascade_Airways dbp:hubs ?uri }
```

au lieu de :

```
select distinct ?uri where { dbr:Cascade_Airways dbp:hubs ?uri }
```

Le tableau 4.6 montre les différentes structures générées pour ce patron et les patrons de l'ensemble d'entraînement qui ont la même structure de requête. On observe que pour chaque requête générée par le modèle, la structure de la requête ne correspond pas à la structure du patron de requête attendu. Au contraire, ces structures de requêtes correspondent aux structures de patrons de requêtes de données de l'ensemble d'entraînement. On peut alors voir comment le modèle identifie des éléments de la question qui lui est soumise à des éléments des questions de son ensemble d'entraînement.

4.6.3 Difficulté de LC-QuAD 2.0

Les résultats montrent que pour la séparation avec patrons inconnus pour LC-QuAD 2.0, aucun modèle ne réussit à obtenir de performances au-dessus de 44% de score F1. Dans ce mémoire, il s'agit du seul cas dans lequel tous les modèles ont des performances si basses. Ce mémoire montre donc que l'usage soit de modèles pré-entraînés, soit du mécanisme de copie (ou encore de la combinaison des deux) avec la bonne méthode d'annotation permet de résoudre la plupart des difficultés de la génération automatique de requêtes SPARQL. Pour trois jeux de données classiques dans leur séparation originale, pour deux jeux de données avec une séparation avec URIs inconnues et pour un jeu de données avec une séparation avec des patrons inconnus, ce mémoire présente des modèles capables de dépasser les 90% de score F1. Ainsi, seulement pour LC-QuAD 2.0, obtenir de telles performances reste un challenge ouvert.

Structures générées à partir de questions dont le patron est : "what is the <1> of <2>?"	Patrons globaux dont le patron de requête suit la même structure
<pre>select distinct ?uri where { <mask> <mask> ?uri . <mask> <mask> ?uri }</pre> (298 occurrences)	<pre>- what is the <1> of the <2> and <3>? - select distinct ?uri where { <2> <1> ?uri. <3> <1> ?uri }</pre>
<pre>select distinct ?uri where { <mask> <mask> ?uri . <mask> <mask> ?uri . }</pre> (77 occurrences)	<pre>- who is the <1> of the <2> and <3> of the <4>? - select distinct ?uri where { <2> <1> ?uri. <4> <3> ?uri }</pre>
<pre>select distinct ?uri where { ?uri <mask> <mask> . }</pre> (1 occurrence)	<pre>- what is the <1> of the <2> and <3>? - select distinct ?uri where { <2> <1> ?uri. <3> <1> ?uri }</pre>
	<pre>- what are the <0> whose <1> is <2>? - select distinct ?uri where {?uri <1> <2> }</pre>

TABLEAU 4.6 Exemple des sorties de BART pour un patron du jeu de test de LC-QuAD 1.0

CHAPITRE 5 CONCLUSION

5.1 Synthèse des travaux

Ce mémoire a présenté une étude de la génération de requêtes SPARQL par traduction automatique avec apprentissage profond. Ses principales contributions sont l'amélioration de la comparabilité des approches de pointe, l'analyse poussée du mécanisme de copie en fonction du type d'architecture neuronale et du type d'annotation des questions, la mise en place de nouvelles performances de références pour la tâche sur trois jeux de données classiques, et l'étude des capacités de généralisation des modèles de référence.

En constatant le manque de comparabilité entre les approches de pointe de la tâche, ce mémoire a reproduit les méthodologies de la littérature et a présenté une série d'expériences exhaustives afin d'associer trois jeux de données, deux méthodes d'annotation ainsi que l'absence d'annotation, deux types d'architectures non pré-entraînées, deux modèles pré-entraînés et la présence ou non du mécanisme de copie. Les résultats montrent que dans la grande majorité des cas, les expériences de ce mémoire reproduisent les performances des études précédentes.

L'extension de ces expériences a également permis une analyse plus poussée de l'impact des différents paramètres ou modules (annotation, modèle, usage de la copie). De plus, ce mémoire propose l'expérimentation originale de l'association du mécanisme de copie et de modèles pré-entraînés. Cela a permis une analyse en profondeur du mécanisme de copie, à la fois sur ses forces et sur ses limitations.

Ainsi, les résultats rapportés dans ce mémoire posent de nouvelles références pour les performances démontrées sur chacun des trois jeux de données utilisés. Dans le cas du jeu de données le plus "simple" il s'agit effectivement de l'association entre le mécanisme de copie et un modèle pré-entraîné. Dans les deux autres cas, il s'agit d'un autre modèle pré-entraîné sans l'usage de la copie, mais à l'aide d'une méthode d'annotation qui n'avait pas été utilisée avec ce modèle.

Ce mémoire s'est ensuite intéressé aux capacités de généralisation des modèles de traduction automatique. Des propriétés des jeux de données et de leur séparation en jeu d'entraînement et jeu de test ont été définies. Cela a permis de vérifier si ces jeux de données sont capables de mesurer les capacités de généralisation des modèles entraînés et évalués sur ces séparations. Des procédures ont été décrites pour obtenir de nouvelles séparations des jeux de données classiques pour obtenir ces propriétés. Enfin, les modèles ont été entraînés et évalués sur ces

nouvelles séparations.

Les expériences menées sur des séparations des jeux de données classiques comprenant des URIs inconnues lors de l'évaluation ont permis de démontrer que les approches présentées dans ce mémoire (modèles pré-entraînés et/ou mécanisme de copie avec annotation des questions) permettent de gérer les URIs inconnues. Les modèles ne démontrent aucune perte de performances par rapport à la séparation initiale. Cette réussite est d'autant plus notable que les modèles classiques (architecture encodeur décodeur sans copie ni pré-entraînement) démontrent de très lourdes pertes de performances entre la séparation originale et celle dans laquelle se trouvent des URIs inconnues.

Enfin, les expériences sur les séparations des jeux de données classiques dans lesquels les questions-requêtes sont générées par des patrons différents lors de l'évaluation et lors de l'entraînement ont permis de montrer la difficulté que cette généralisation représente pour les modèles. Les modèles classiques n'arrivent pas du tout à gérer les structures inconnues. Sur LC-QuAD 2.0, tous les modèles présentent de grosses pertes de performances par rapport à la séparation initiale et le mécanisme de copie permet notamment aux modèles non pré-entraînés de presque rattraper les modèles pré-entraînés. Pour LC-QuAD 1.0, même si en général les modèles montrent des baisses de performances par rapport à la séparation initiale, T5 réussi à conserver de très bonnes performances sans copie et BART réussi à obtenir des performances presque parfaites avec la copie et des questions avec tag-interne. Cependant, même les requêtes générant la bonne réponses sont systématiquement différentes des requêtes attendues. De plus, pour LC-QuAD 2.0, aucun modèle, y compris les modèles pré-entraînés avec ou sans copie, n'est capable de dépasser 50% de score F1.

En résumé, ce mémoire reproduit et étend les approches de pointes de la littérature, améliore les performances de référence pour chaque jeu de données, analyse l'impact de l'annotation des questions, de l'usage de la copie et du pré-entraînement sur la tâche ainsi que les capacités de généralisation de ces approches.

5.2 Limites de la solution proposée

5.2.1 Optimisation des hyper paramètres

L'un des objectifs de ce mémoire est d'améliorer la comparabilité entre les approches de pointes de la tâche de génération de requêtes SPARQL. La plupart des hyper-paramètres utilisés sont donc inspirés des études utilisées. Cependant, étant donné le changement d'échelle du nombre d'expériences nécessaire dans ce mémoire par rapport à celles de références ainsi que les limitations matérielles que cela entraîne, certains paramètres tels que le nombre d'époques

ou encore la taille de lot ont dû être modifiés. Ces modifications ont montré l’impact de ce genre d’hyper-paramètre sur les performances finales des modèles. Ainsi, cela semble signifier qu’il est possible d’encore améliorer les performances de la tâche avec une recherche plus exhaustive d’hyper-paramètres. Notamment, les hyper-paramètres optimaux sont probablement différents entre les expériences.

5.2.2 Mécanisme de copie

Le mécanisme de copie décrit dans ce mémoire est inspiré de l’étude [7] dans laquelle il est appliqué à des modèles non pré-entraînés avec une annotation tag-interne. Dans ce cas précis, il montre de très grandes forces théoriques telles que la réduction de vocabulaire pour l’encodeur et le décodeur ou la capacité d’utiliser les positions des URIs dans la question pour directement les associer à une position dans la requête. Cependant, ce mémoire montre certaines limites de la méthodologie associée au mécanisme de copie.

D’une part, pour l’annotation tag-final, on observe que certains modèles échouent totalement à la génération de requêtes correctes avec le mécanisme de copie. Cela se produit quand l’emplacement des URIs dans la question est trop difficile à associer à un emplacement dans la requête.

D’autre part, le masquage des URIs implique que le modèle ne peut pas utiliser la sémantique des URIs comme indication pour la génération de la requête. Cela pose un problème dans des cas comme celui vu avec LC-QuAD 2.0 avec tag-interne, où certaines structures de questions peuvent mener à plusieurs structures de requêtes. De plus, dans le cas du tag final, les labels d’URIs inconnues risquent d’être inconnus également et de mener à une impossibilité pour le mécanisme de copie de comprendre où placer ladite URI.

5.2.3 Méthode d’annotation

L’annotation des éléments de la BC est une étape indispensable pour obtenir des systèmes de génération de requêtes SPARQL efficaces. Cependant, les tâches de liaison à une BC et d’étiquetage sont toutes les deux difficiles et loin d’être résolues. Ce mémoire ne confronte pas directement cette difficulté en utilisant les patrons de génération pour générer les questions annotées. C’est une limitation de l’approche qui ne lui permet pas d’être utilisée avec des données dont les patrons ne sont pas accessibles ou connus.

5.2.4 Méthode de séparation avec URIs inconnues

La méthodologie pour produire les séparations des jeux de données pour avoir les propriétés de généralisation souhaitées est adaptée aux jeux de données utilisés. Cependant, pour le cas de la séparation avec des URIs inconnues, la génération des groupes sur lesquels se base la séparation nécessite de construire un graphe composé d'une grande majorité des données et d'en trouver les composantes connexes. Ceci est un processus quadratique en la taille du nombre de nœuds du graphe. Ainsi, cette approche risque de devenir beaucoup plus longue pour des jeux de données plus grands. ‘

5.2.5 Distributions similaires des ensembles de validation et d'évaluation

Lors de l'évaluation des capacités de généralisation des modèles, on a séparé les jeux de données en 80-20% de données d'entraînement et de test ayant les propriétés de généralisation définies dans la section 4.1. On divise ensuite uniformément l'ensemble de "test" en un ensemble de validation utilisé pour trouver le meilleur modèle lors de l'entraînement et un ensemble d'évaluation utilisé pour calculer les performances du modèle. Utiliser la même distribution pour l'ensemble de validation et d'évaluation permet d'obtenir les modèles les plus susceptibles de généraliser étant donné la distribution de l'ensemble d'entraînement. Cela permet de limiter le sur-apprentissage sur la distribution d'entraînement, mais peut biaiser les modèles vers de meilleures performances. En effet, cela n'est pas réaliste en pratique, car la distribution de test n'est généralement pas utilisée dans l'ensemble de validation. Ce choix peut donc être vu comme une limitation et il pourrait être intéressant de voir l'impact d'utiliser la distribution d'entraînement pour l'ensemble de validation plutôt que la distribution de test.

5.3 Améliorations futures

5.3.1 Mécanisme de copie

Architecture du mécanisme

Comme indiqué dans la section précédente, la principale limitation du mécanisme de copie réside dans le masquage de URIs. Bien que cela permette une grande diminution du vocabulaire de l'encodeur-décodeur pour les modèles non pré-entraînés et de garder les URIs en un seul jeton pour les modèles pré-entraînés, cela implique que le mécanisme de copie peut uniquement se reposer sur la position de l'URI et pas du tout sur sa sémantique. Pour dépasser cela, il faut réussir à ne pas masquer les URIs tout en n'ayant pas de difficultés à apprendre

les plongements de nombreux jetons. Une méthode pour cela pourrait être d'utiliser des plongements de graphe de connaissances pour représenter les URIs. Ainsi, le modèle n'aurait pas à entraîner de représentations et donc n'aurait pas de problème de mots inconnus ni de division en plusieurs jetons. La difficulté est alors d'avoir des plongements de graphes adaptés aux dimensions, notamment des modèles pré-entraînés. De plus, cela représente une taille de vocabulaire extrêmement importante et soulève alors des difficultés d'implémentation. En effet, pour charger le modèle complet, il faudrait une RAM extrêmement importante. Il serait possible d'implémenter un code qui permettrait de charger en RAM uniquement une partie du modèle suivant le vocabulaire nécessaire. Ce genre d'implémentation est peu commune et risque de nécessiter une grande charge de travail.

Transfert d'apprentissage des URIs

Une première méthode pour utiliser les plongements de graphe de connaissances serait d'utiliser des méthodes classiques telles que TransE [29] qui se base uniquement sur la structure du graphe. Mais il existe également des méthodes [38–41] qui permettent d'aligner les plongements de URIs sur les représentations de modèles de langue. Ainsi, on augmente potentiellement grandement le temps de convergence du modèle lors de l'affinage.

5.3.2 Méthodologie d'évaluation des capacités de généralisation

Séparation avec URIs inconnues

Comme expliqué dans la section précédente, la séparation avec URIs inconnues demande de trouver les composantes connexes d'un graphe dont les nœuds sont la majorité des entrées d'un jeu de données. Cela permet d'avoir strictement 100% des données du jeu de test qui ont au moins une URI inconnue. Cependant, utiliser d'autres algorithmes de graphe, tel que des algorithmes de groupement ou de recherche de communautés, permettrait de maximiser le nombre d'entrées de test ayant des URIs inconnue avec une meilleure complexité.

Autres paramètres de séparation

Ce mémoire s'est concentré sur les URIs inconnues et les patrons de génération inconnus. En effet, cela semblait être les critères permettant de mesurer le plus directement possible les capacités de modèles de généraliser par rapport à deux limitations majeures des jeux de données classiques de traduction anglais-SPARQL. Cependant, il est possible d'imaginer d'autres critères de généralisation intéressants tels que la taille de la question ou le nombre de triplets de la requête par exemple. Enfin, une piste pourrait être de trouver des critères de

"difficulté" des entrées du jeu de données et d'évaluer la capacité des modèles à généraliser à des entrées plus difficiles lors de l'évaluation que lors de l'entraînement.

Une autre expérience qui n'a pas été faite dans ce mémoire qui relève de la capacité de génération serait de tester les modèles entraînés sur des questions générées automatiquement à l'aide des questions reformulées qui sont fournies avec LC-QuAD 1.0 et LC-QuAD 2.0.

5.3.3 Résumé du travail réalisé

Ce mémoire propose une étude de la tâche de génération de requête SPARQL. Il compare et dépasse les méthodes les plus performantes de la tâche, à savoir l'usage d'encodeurs décodeurs pré-entraînés ou du mécanisme de copie avec l'annotation des éléments de la BC dans la question. Il propose pour la première fois d'utiliser le mécanisme de copie avec les modèles pré-entraînés. De nouvelles performances de références sont définies sur trois jeux de données populaires. On propose également une analyse des forces et des faiblesses du mécanisme de copie et des différentes méthodes d'annotations. Enfin, ce mémoire propose une analyse des difficultés de la tâche liées à la notion de généralisation. Ce mémoire propose une méthode pour obtenir des séparations des jeux de données populaires qui permettent d'évaluer les capacités de généralisation des modèles. On montre ainsi que les approches les plus performantes permettent de gérer les URIs inconnues et les patrons inconnus pour LC-QuAD 1.0. La séparation avec des patrons inconnus pour LC-QuAD 2.0 reste cependant un cas difficile pour lequel aucun modèle n'obtient de performances satisfaisantes. Dans tous les autres cas, ce mémoire propose une nouvelle performance de référence supérieure à 90% de score F1.

RÉFÉRENCES

- [1] T. Soru, E. Marx, D. Moussallem, G. Publio, A. Valdestilhas, D. Esteves et C. B. Neto, “SPARQL as a foreign language,” dans *Proceedings of the Posters and Demos Track of the 13th International Conference on Semantic Systems - SEMANTiCS2017 co-located with the 13th International Conference on Semantic Systems (SEMANTiCS 2017), Amsterdam, The Netherlands, September 11-14, 2017*, ser. CEUR Workshop Proceedings, J. D. Fernández et S. Hellmann, édit., vol. 2044. CEUR-WS.org, 2017. [En ligne]. Disponible : <http://ceur-ws.org/Vol-2044/paper14/>
- [2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser et I. Polosukhin, “Attention is all you need,” dans *Advances in Neural Information Processing Systems 30 : Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan et R. Garnett, édit. Curran Associates, Inc., 2017, p. 5998–6008. [En ligne]. Disponible : <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
- [3] J. Gehring, M. Auli, D. Grangier, D. Yarats et Y. N. Dauphin, “Convolutional sequence to sequence learning,” dans *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup et Y. W. Teh, édit., vol. 70. PMLR, 2017, p. 1243–1252. [En ligne]. Disponible : <http://proceedings.mlr.press/v70/gehring17a.html>
- [4] J. Devlin, M.-W. Chang, K. Lee et K. Toutanova, “BERT : Pre-training of deep bidirectional transformers for language understanding,” dans *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota : Association for Computational Linguistics, juin 2019, p. 4171–4186. [En ligne]. Disponible : <https://aclanthology.org/N19-1423>
- [5] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov et L. Zettlemoyer, “BART : denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” dans *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, D. Jurafsky, J. Chai, N. Schluter et J. R. Tetreault, édit. Association for Computational Linguistics, 2020, p. 7871–7880. [En ligne]. Disponible : <https://doi.org/10.18653/v1/2020.acl-main.703>

- [6] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li et P. J. Liu, “Exploring the limits of transfer learning with a unified text-to-text transformer,” *J. Mach. Learn. Res.*, vol. 21, p. 140 :1–140 :67, 2020. [En ligne]. Disponible : <http://jmlr.org/papers/v21/20-074.html>
- [7] R. Hirigoyen, A. Zouaq et S. Reyd, “A copy mechanism for handling knowledge base elements in SPARQL neural machine translation,” dans *Findings of the Association for Computational Linguistics : ACL-IJCNLP 2022*. Online only : Association for Computational Linguistics, nov. 2022, p. 226–236. [En ligne]. Disponible : <https://aclanthology.org/2022.findings-acl.22>
- [8] D. Banerjee, P. A. Nair, J. N. Kaur, R. Usbeck et C. Biemann, “Modern baselines for SPARQL semantic parsing,” dans *SIGIR ’22 : The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, E. Amigó, P. Castells, J. Gonzalo, B. Carterette, J. S. Culpepper et G. Kazai, édit. ACM, 2022, p. 2260–2265. [En ligne]. Disponible : <https://doi.org/10.1145/3477495.3531841>
- [9] Y. Qian, J. Liang et C. Dang, “Knowledge structure, knowledge granulation and knowledge distance in a knowledge base,” *International Journal of Approximate Reasoning*, vol. 50, n^o. 1, p. 174–188, 2009.
- [10] B. M. Lake, “Compositional generalization through meta sequence-to-sequence learning,” dans *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019. [En ligne]. Disponible : <https://proceedings.neurips.cc/paper/2019/hash/f4d0e2e7fc057a58f7ca4a391f01940a-Abstract.html>
- [11] D. Keysers, N. Schärli, N. Scales, H. Buisman, D. Furrer, S. Kashubin, N. Momchev, D. Sinopalnikov, L. Stafiniak, T. Tihon, D. Tsarkov, X. Wang, M. van Zee et O. Bousquet, “Measuring Compositional Generalization : A Comprehensive Method on Realistic Data,” juin 2020, arXiv :1912.09713 [cs, stat]. [En ligne]. Disponible : <http://arxiv.org/abs/1912.09713>
- [12] I. Sutskever, O. Vinyals et Q. V. Le, “Sequence to sequence learning with neural networks,” dans *Advances in Neural Information Processing Systems 27 : Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence et K. Q. Weinberger, édit. Curran Associates, Inc., 2014, p. 3104–3112. [En ligne]. Disponible : <https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html>

- [13] X. Yin, D. Gromann et S. Rudolph, “Neural machine translating from natural language to SPARQL,” *Future Gener. Comput. Syst.*, vol. 117, p. 510–519, 2021. [En ligne]. Disponible : <https://doi.org/10.1016/j.future.2020.12.013>
- [14] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy et S. Bowman, “GLUE : A multi-task benchmark and analysis platform for natural language understanding,” dans *Proceedings of the 2018 EMNLP Workshop BlackboxNLP : Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium : Association for Computational Linguistics, nov. 2018, p. 353–355. [En ligne]. Disponible : <https://aclanthology.org/W18-5446>
- [15] J. Gu, Z. Lu, H. Li et V. O. K. Li, “Incorporating copying mechanism in sequence-to-sequence learning,” dans *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1 : Long Papers*. The Association for Computer Linguistics, 2016, p. 1631–1640. [En ligne]. Disponible : <https://doi.org/10.18653/v1/p16-1154>
- [16] A. See, P. J. Liu et C. D. Manning, “Get to the point : Summarization with pointer-generator networks,” dans *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1 : Long Papers*, R. Barzilay et M. Kan, édit. Association for Computational Linguistics, 2017, p. 1073–1083. [En ligne]. Disponible : <https://doi.org/10.18653/v1/P17-1099>
- [17] K. Papineni, S. Roukos, T. Ward et W. Zhu, “Bleu : a method for automatic evaluation of machine translation,” dans *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, p. 311–318. [En ligne]. Disponible : <https://aclanthology.org/P02-1040/>
- [18] F. F. Luz et M. Finger, “Semantic parsing natural language into SPARQL : improving target language representation with neural attention,” *CoRR*, vol. abs/1803.04329, 2018. [En ligne]. Disponible : <http://arxiv.org/abs/1803.04329>
- [19] S. Hochreiter et J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, p. 1735–80, 12 1997.
- [20] D. E. Rumelhart, G. E. Hinton et R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, n°. 6088, p. 533–536, oct. 1986. [En ligne]. Disponible : <https://doi.org/10.1038/323533a0>
- [21] P. Trivedi, G. Maheshwari, M. Dubey et J. Lehmann, “Lc-quad : A corpus for complex question answering over knowledge graphs,” dans *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II*, ser. Lecture Notes in Computer Science, C. d’Amato,

- M. Fernández, V. A. M. Tamma, F. Lécué, P. Cudré-Mauroux, J. F. Sequeda, C. Lange et J. Heflin, édit., vol. 10588. Springer, 2017, p. 210–218. [En ligne]. Disponible : https://doi.org/10.1007/978-3-319-68204-4_22
- [22] A.-K. Hartmann, E. Marx et T. Soru, “Generating a large dataset for neural question answering over the dbpedia knowledge base,” dans *Workshop on Linked Data Management, co-located with the W3C WEBBR*, vol. 2018, 2018.
- [23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, n^o. 8, p. 9, 2019.
- [24] X. Huang, J.-J. Kim et B. Zou, “Unseen entity handling in complex question answering over knowledge base via language generation,” dans *Findings of the Association for Computational Linguistics : EMNLP 2021*. Punta Cana, Dominican Republic : Association for Computational Linguistics, nov. 2021, p. 547–557. [En ligne]. Disponible : <https://aclanthology.org/2021.findings-emnlp.50>
- [25] M. R. A. H. Rony, U. Kumar, R. Teucher, L. Kovriguina et J. Lehmann, “SGPT : A generative approach for SPARQL query generation from natural language questions,” *IEEE Access*, vol. 10, p. 70 712–70 723, 2022. [En ligne]. Disponible : <https://doi.org/10.1109/ACCESS.2022.3188714>
- [26] M. Dubey, D. Banerjee, A. Abdelkawi et J. Lehmann, “Lc-quad 2.0 : A large dataset for complex question answering over wikidata and dbpedia,” dans *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. F. Cruz, A. Hogan, J. Song, M. Lefrançois et F. Gandon, édit., vol. 11779. Springer, 2019, p. 69–78. [En ligne]. Disponible : https://doi.org/10.1007/978-3-030-30796-7_5
- [27] E. Kacupaj, H. Zafar, J. Lehmann et M. Maleshkova, “Vquanda : Verbalization question answering dataset,” dans *European Semantic Web Conference*. Springer, 2020, p. 531–547.
- [28] A. Perevalov, D. Diefenbach, R. Usbeck et A. Both, “Qald-9-plus : A multilingual dataset for question answering over dbpedia and wikidata translated by native speakers,” dans *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*. IEEE, 2022, p. 229–234.
- [29] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston et O. Yakhnenko, “Translating embeddings for modeling multi-relational data,” dans *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani et K. Weinberger,

- édit., vol. 26. Curran Associates, Inc., 2013. [En ligne]. Disponible : https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf
- [30] M. Baroni, “Linguistic generalization and compositionality in modern artificial neural networks,” *Philosophical Transactions of the Royal Society B : Biological Sciences*, vol. 375, n°. 1791, p. 20190307, déc. 2019, publisher : Royal Society. [En ligne]. Disponible : <https://royalsocietypublishing.org/doi/full/10.1098/rstb.2019.0307>
- [31] B. Lake et M. Baroni, “Generalization without systematicity : On the compositional skills of sequence-to-sequence recurrent networks,” dans *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy et A. Krause, édit., vol. 80. PMLR, 10–15 Jul 2018, p. 2873–2882. [En ligne]. Disponible : <https://proceedings.mlr.press/v80/lake18a.html>
- [32] Y. Gu, S. Kase, M. Vanni, B. Sadler, P. Liang, X. Yan et Y. Su, “Beyond i.i.d. : Three levels of generalization for question answering on knowledge bases,” dans *Proceedings of the Web Conference 2021*, ser. WWW ’21. New York, NY, USA : Association for Computing Machinery, 2021, p. 3477–3488. [En ligne]. Disponible : <https://doi.org/10.1145/3442381.3449992>
- [33] L. Jiang et R. Usbeck, “Knowledge graph question answering datasets and their generalizability : Are they enough for future research?” dans *SIGIR ’22 : The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, E. Amigó, P. Castells, J. Gonzalo, B. Carterette, J. S. Culpepper et G. Kazai, édit. ACM, 2022, p. 3209–3218. [En ligne]. Disponible : <https://doi.org/10.1145/3477495.3531751>
- [34] S. Reyd, A. Zouaq et P. A. K. K. Diallo, “A comprehensive evaluation of the copy mechanism for natural language to sparql query generation,” *arXiv preprint arXiv :2304.07772*, 2023.
- [35] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever et W. Zaremba, “Evaluating large language models trained on code,” *CoRR*, vol. abs/2107.03374, 2021. [En ligne]. Disponible : <https://arxiv.org/abs/2107.03374>

- [36] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, “Training language models to follow instructions with human feedback,” *Advances in Neural Information Processing Systems*, vol. 35, p. 27 730–27 744, 2022.
- [37] S. Reyd et A. Zouaq, “Assessing the generalization capabilities of neural machine translation models for sparql query generation,” dans *The 22nd International Semantic Web Conference (ISWC) (accepté)*, 2023.
- [38] R. Xie, Z. Liu, J. Jia, H. Luan et M. Sun, “Representation Learning of Knowledge Graphs with Entity Descriptions,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, n°. 1, mars 2016, number : 1. [En ligne]. Disponible : <https://ojs.aaai.org/index.php/AAAI/article/view/10329>
- [39] I. Yamada, H. Shindo, H. Takeda et Y. Takefuji, “Joint Learning of the Embedding of Words and Entities for Named Entity Disambiguation,” *arXiv :1601.01343 [cs]*, juin 2016, arXiv : 1601.01343. [En ligne]. Disponible : <http://arxiv.org/abs/1601.01343>
- [40] K. Cheng, X. Li, Y. E. Xu, X. L. Dong et Y. Sun, “Pge : Robust product graph embedding learning for error detection,” *arXiv preprint arXiv :2202.09747*, 2022.
- [41] H. Xiao, M. Huang, L. Meng et X. Zhu, “SSP : Semantic Space Projection for Knowledge Graph Embedding with Text Descriptions,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, n°. 1, févr. 2017. [En ligne]. Disponible : <https://ojs.aaai.org/index.php/AAAI/article/view/10952>