NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES

Data Science and Information Technologies

SPECIALIZATION

Big Data and Artificial Intelligence

MASTER'S THESIS

Verbalising Query Results to Text

Michail S. Xydas

ATHENS

JUNE 2023

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Επιστήμη Δεδομένων και Τεχνολογίες Πληροφορίας

ΕΙΔΙΚΕΥΣΗ

Μεγάλα Δεδομένα και Τεχνητή Νοημοσύνη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Λεξικοποίηση αποτελεσμάτων επερωτήσεων

Μιχαήλ Σ. Ξύδας

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2023

Master's Thesis

Verbalising Query Results to Text

Michail S. Xydas

S.N.: DS1200015

SUPERVISORS:

Georgia Koutrika, Research Director, Athena Research Center

EXAMINATION COMMITTEE:

Georgia Koutrika, Research Director, Athena Research Center

Nassos Katsamanis, Principal Researcher, Athena Research Center

Kurt Stockinger, Professor, Zurich University of Applied Sciences

JUNE 2023

Διπλωματική Εργασία

Λεξικοποίηση αποτελεσμάτων επερωτήσεων

Μιχαήλ Σ. Ξύδας

Α.Μ.: DS1200015

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Γεωργία Κούτρικα, Διευθύντρια Έρευνας, Ερευνητικό Κέντρο Αθηνά

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Γεωργία Κούτρικα, Διευθύντρια Έρευνας, Ερευνητικό Κέντρο Αθηνά

Αθανάσιος Κατσαμάνης, Κύριος Ερευνητής, Ερευνητικό Κέντρο Αθηνά

Κιουρτ Στόκινγκερ, Καθηγητής, Πανεπιστήμιο Εφαρμοσμένων Επιστημών Ζυρίχης

ΙΟΥΝΙΟΣ 2023

ABSTRACT

Database democratization focuses on making databases accessible to non-expert users that are not familiar with database query languages like SQL. In this direction, a lot of effort has already been put into two problems: Text-to-SQL, which focuses on translating a natural language query to SQL, and SQL-to-Text, which is the inverse problem. However, work has lagged behind in explaining query results in natural language. We first define the Query Results-to-Text problem as: *given the results of a query, produce a natural language verbalisation describing these results*. Then we attempt solving Query Results-to-Text by defining a model, namely QR2T. We propose pretraining QR2T using real-world table datasets focusing on table understanding. We use a preprocessing step that transforms the query so that the query results, which are the input of our model, include additional information, which QR2T can utilize leading to a more informative verbalisation. Finally, we create two Query Results-to-Text benchmarks, which are the first datasets that contain query result verbalisations for both fine-tuning and evaluation.

SUBJECT AREA: Database Systems

KEYWORDS: machine learning, deep learning, model pretraining

ΠΕΡΙΛΗΨΗ

Η δημοκρατικοποίηση των βάσεων δεδομένων επικεντρώνεται στο να γίνουν οι βάσεις δεδομένων διαθέσιμες σε χρήστες που δεν έχουν γνώσεις ή επαφή με γλώσσες επερωτήσεων. Προς αυτή την κατεύθυνση έχει δοθεί ήδη πολύ προσπάθεια στην επίλυση 2 προβλημάτων: Text-to-SQL που επικεντρώνεται στην μετάφραση φυσικής γλώσσας σε SQL και SQL-to-Text που επιλύει το αντίστροφο πρόβλημα. Όμως, ένα πρόβλημα για το οποίο δεν έχει προταθεί λύση ακόμα είναι η εξήγηση των αποτελεσμάτων των επερωτήσεων σε φυσική γλώσσα. Πρώτα ορίζουμε το πρόβλημα Query Results-to-Text ως: *δωσμένων των αποτελεσμάτων ενός επερωτήματος, παράγουμε φυσική γλώσσα που περιγράφει τα αποτελέσματα*. Προτείνουμε την χρήση προ-εκπαίδευσης του μοντέλου χρησιμοποιώντας δεδομένα πινάκων από τον πραγματικό κόσμο με σκοπό την κατανόηση των πινάκων. Δεύτερον, προτείνουμε ένα βήμα προεπεξεργασίας του επερωτήματος ώστε τα αποτελέσματα, που είναι η είσοδος του μοντέλου μας, να περιέχουν επιπρόσθετα χαρακτηριστικά και οδηγούν σε πιο πλούσιες λεξικοποιήσεις. Τέλος δημιουργούμε δύο Query Results-to-Text Benchmarks, τα οποίο είναι τα πρώτα σύνολα δεδομένων που περιέχουν αποτελέσματα επερωτήσεων για τελική εκπαίδευση αλλά και για αξιολόγηση του μοντέλου.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Συστήματα Βάσεων Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: μηχανική μάθηση, βαθυά μάθηση, προ-εκπαίδευση μοντέλου

ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to Dr. Georgia Koutrika, Research Director at Athena Research Center, for the excellent guidance and persistent support. I would also like to thank my family and friends for the love and support they have provided me throughout this work.

This work was supported by computational time granted from the National Infrastructures for Research and Technology S.A. (GRNET S.A.) in the National HPC facility - ARIS - under project ID mxydas-d2t. The computational time was used for the training of all models presented in this thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1. INTRODUCTION

Databases are easy to use by people with knowledge of a database query language. However, often, the person knowing the query language (usually a programmer) is different from the person able to interpret the results, who may not be a computer scientist. This leads to time-consuming iterative communication between the query language expert and the domain expert. Natural language interfaces can provide to the latter an easier way to access data leading to accelerated analysis and research.

A NL interface comprises 3 main components as shown in Figure 1. A *Text-to-SQL* component takes as input a user question in natural language and produces a SQL query to be executed over the database [34,38,45]. A *SQL-to-Text* component describes a SQL query in natural language so that the user can use its natural language output to confirm the correctness of the query being executed [1,42]. Finally, a *Query Results-to-Text* component takes the query results in a structured format, usually a table, and returns their verbalisation. While research has mostly focused on Text-to-SQL, followed by SQL-to-Text, work has lagged behind for the query results verbalisation.

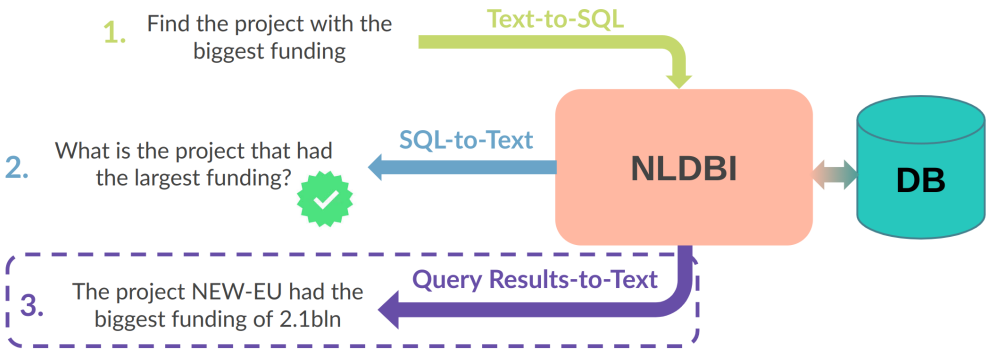

Figure 1: Components of a Natural Language Database Interface

In this work, we focus on the Query Results-to-Text problem. Query result verbalization can help the user get a quick understanding of the results, typically returned in tabular form. Furthermore, a system answer in NL may be sufficient in cases where the user is interested in a response, rather than looking into the tabular data.

Query Results-to-Text has two main challenges. First, we need to create a ”universal” system, which is able to verbalise results from databases of many domains. For example, the same system should be able to produce verbalisations of results for an astrophysics database and for a sports database without requiring any manual work.

Second, the results of a query might not include the necessary information required for a valid verbalisation. For example, in Figure 2 we can see that two columns of the results had the same name since this was a result of joining two tables, *movie, director*.

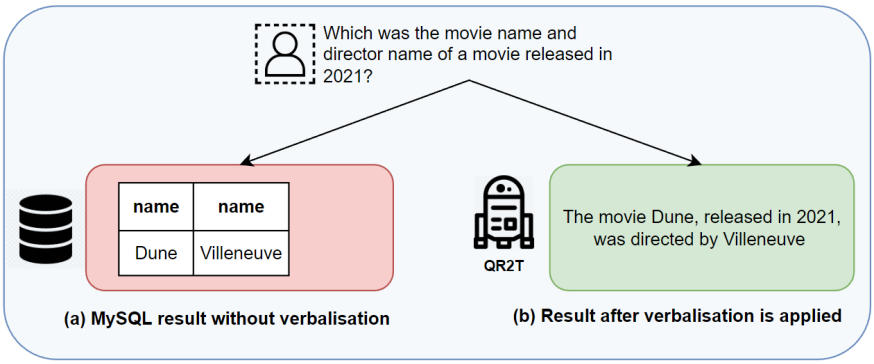

Figure 2: A user asks a question in natural language. On the left, we have the result that the SQL database would return. On the right, we have the result verbalized.

Existing approaches to the Query Results-to-Text problem use templates, manually created by humans, to perform the verbalisation [3,6]. However, manually creating templates may be a burden when adapting the solution to a new database. However, these solutions do not address the first challenge of creating a system applicable to many databases of different domains.

Solution Overview.

We solve the challenge of database generalization by using a pretrained language model, namely T5 [31]. To the best of our knowledge, this is the first approach to the Query Results-to-Text problem using neural networks. T5 is able to understand text, and consequently, the contents of the query results making it a suitable fit for our task. However, introducing T5 into our solution comes with its own challenges. T5 was created to solve text-to-text problems such as text summarization [28], but in our case, the input is a query results table along with any extra information like the query in natural language.

We define a module that will serialize the input into a continuous string which is appropriate for T5. However, a continuous string lacks structure, which a query result table should have. For example, the model will have to implicitly discover the connection between a column (i.e. *age*) and a following value (i.e. *24*). In order to explicitly teach the model these connections, we first pretrain it with a set of tasks. For example, we mix the column names but keep the values in the same order and then task the model with retrieving the correct match between columns and values.

Another challenge introduced by T5 is its unfamiliarity with SQL since it has been pre-trained on understanding and producing text. We define a set of rules that will transform said SQL notation to text, for example, ”AVG(age)” will be transformed to ”average age”.

For finetuning and evaluation, we face additional challenges. Table-to-Text datasets used for training did not suit the more complex Query Results-to-Text task since we require a query and underlying database as part of our input. For this reason, we created two benchmark datasets. The first one, QR2T Benchmark ($QR2T_{bench}$) has 1,400 query results, manually verbalised by human database experts. The queries and databases are selected from the Spider [46] dataset which is widely used in Text-to-SQL. This benchmark can be used for both training and evaluating our model. While being relatively small, the benchmark covers a variety of SQL queries including aggregates, JOINs, GROUP BYs applied on a variety of databases. The second benchmark, CORDIS, contains 222 queries on a popular production database, namely Cordis [1], and gives us insights for the real-world performance of our solution. Due to our problem being unexplored so far, these are the first such datasets proposed.

The performance of our solutions is evaluated through quantitative and qualitative analysis of the results. We report automatic metrics BLEU [26], PARENT [8], and Bertscore [47] between the generated text and the annotated verbalisations. Finally, we also perform human evaluation aiming to pinpoint the exact cases that QR2T fails.

The Query Results-to-Text problem resembles the highly explored Table-to-Text problem [14, 20], in that, both take input tables and generate descriptions of the table contents. However, they are quite different. Table-to-Text focuses on verbalizing tables, such Wikipedia tables, that do not require external information in order to be understood by a human or a system. However, in our case, tables contain the results of a query executed over an underlying database, and that gives rise to several challenges particular to the problem such the aforementioned *result ambiguities* and *SQL notation*, which make Table-to-Text solutions not applicable.

We note that our system is able to verbalise a single row of query results. Examining techniques to verbalise many rows collectively is considered future work.

Contributions. Summarizing, our contributions are:

1. We propose the first neural approach to the Query Results to Text problem that is

[1] https://cordis.europa.eu/

built on T5.

2. We adapt T5 to the Query Results-to-Text problem so as it takes as input query results and produces text.

3. We pretrain T5 so as it can understand the structure of the table from the serialized table.

4. We propose a query preprocessing algorithm that injects schema information (i.e. adding table names next to column names on the result table) and transforms names of aggregate results (i.e. *AVG(age)* to *average age*).

5. We create two benchmarks for Query Results-to-Text, one that expands the existing Spider [46] dataset and one on the Cordis database having 1,400 and 222 queries respectively.

6. We conduct experiments reporting both automatic metrics and human evaluation results.

2. RELATED WORK

Our work falls at the intersection of two deeply explored research fields, Natural Language Database Interfaces (NLDBIs), and Data-to-Text.

NLDBIs focus on bridging the gap between a person that has no query language knowledge and databases by providing a user experience based on natural language only. There are 3 components as shown in Figure 1. Text-to-SQL translates a query from natural language to SQL, SQL-to-Text does the reverse so that the user confirms the correctness of the created query, and QR2T takes the query results and verbalises them.

Data-to-Text is an umbrella term that covers models that take as input data in a structured format and transform it into text. Its main subfields are Table-to-Text, where the input is a table, and Graph-to-Text, where the input is a graph, and Query Results-to-Text, where the input is the result of a query.

2.0.1 Text-to-SQL

Text-to-SQL approaches take as input a question in natural language and generate the corresponding SQL query. For this purpose, they have to understand natural language as well as the contents of the database based on the given schema. The generated queries must cover a variety of query types including JOINs, GROUP BYs, etc.

Dataset	Question	Table/Database	Annotation
WikiSQL	How many CFL teams are from York College?	a table with 5 columns e.g. player, position, ...	SELECT COUNT CFL.team FROM CFL.draft WHERE college= 'YORK'
Spider	Find the first and last names of the students who are living in the dorms that have a TV Lounge as an amenity.	database with 5 tables e.g.student, dorm_amenity, ...	SELECT T1.FNAME, T1.LNAME FROM STUDENT AS T1 JOIN LIVES IN AS T2 ON T1.STUID=T2.STUID WHERE T2.DORMID IN (SELECT T3.DORMID FROM HAS AMENITY AS T3 JOIN DORM_AMENITY AS T4 ON T3.AMENID=T4.AMENID WHERE T4.AMENITY NAME= 'TV LOUNGE')

Figure 3: Examples of the most used Text-to-SQL, SQL-to-Text datasets.

First Text-to-SQL approaches did not use neural networks, instead, they focused on grammar-based techniques and intermediate representations.

NaLIR [19] is a system that gradually builds a desired query through iterative interaction with the user, dealing with ambiguities that exist in NL but are not allowed in a strict language like SQL. In human-human interactions, we solve these ambiguities with brief clar-

ifications. Inspired by these clarifications, NaLIR discovers these ambiguities, proposes default values based on the context, and interacts with the user to confirm or fix these values. This entails the need for an intermediate representation (IR) that both the user and the system can understand. For example, the DB might contain tables such as *cars, boats, airplanes* and the NL query might contain the word *speed*. The system will ask the user to specify the table to which *speed* refers.

With great advancements in A.I., Text-to-SQL solutions leveraging neural models and pre-trained language models (PLMs [7, 31]) have emerged achieving impressive results.

There are 2 main datasets used for training such models for Text-to-SQL. The first one is WikiSQL [48], a collection of 80K questions and SQL query pairs from 30K Wikipedia tables. The queries contain filtering and aggregations, but remain relatively simple without JOINs and nested queries. The second one is Spider [46], a newer and more complex dataset since many of its 10k question-query pairs require JOINs, GROUP BYs, and nested queries. It includes 200 databases with each database containing many tables. Its test set can be split into two parts, a seen one and an unseen one with the second one having databases that do not exist in the training set.

X-SQL [12] utilized the language understanding power of BERT [7] along with joined encoding between the natural language query and the schema. Intuitively, when SQL users are shown with a new schema and a question, they first read the question and focus on the columns that seem relative, ignoring the rest of the schema. X-SQL achieves that by encoding the columns, using the “global context” obtained by the question and the schema. To obtain the global context, X-SQL takes as input the serialized database and the question through a BERT encoder and keeps the representation of the [CLS] token as the global context. Then, at decoding, when the SQL query is generated, an attention mechanism is used over the global context along with the local context.

RAT-SQL [38] focuses on schema linking, i.e. mapping words from the NL query to entities from the DB schema, such as table and column names. It extends the self-attention mechanism [37] so that it takes into account relationships between the question tokens and the database schema. These relationships could be an exact match, partial match, or even value matching where a word of the question is found in the values of one of the columns of the database. Similarly to X-SQL, the initial representations of the query and the schema are obtained through a BERT model.

On the other hand, PICARD [34] suggests using encoder-decoder PLMs. It focuses on guiding the outputs of the model using rules like preventing the model to generate a column name that does not exist. PICARD is only applied at inference time without requiring any changes to how the model is trained. Also, the assumption that the model is an encoder-decoder black box permits the usage of powerful end-to-end models like T5 [31] and BART [18].

DBPal [39] is a data augmentation technique that creates synthetic data generated by template matching and paraphrasing. The SQL queries are generated through SQL templates, for example, “Select Attribute(s) From Table Where Filter”. Then, the values are randomly sampled using the schema information. Generating the NL questions is a bit trickier since language is characterized by variance and expressivity. For that reason, DBPal creates NL slots like “SelectPhrase”, “Attribute(s)”, “WherePhrase” etc. Then, it fills these phrase slots using manually crafted dictionaries. For example, a SelectPhrase can be replaced by phrases like ”what is” and ”show me”. By randomly sampling schema values and NL dictionaries, DBPal can generate an infinite number of question-SQL pairs. Second, it augments the NL questions in an attempt to capture the expressivity of language, by performing paraphrasing using the Paraphrase Database (PPDB). Finally, to accommodate for the possible missing NL content, DBPal randomly remove specific words, not necessary to answer the query. Both the aggressiveness of paraphrasing and word removal is controlled by hyper-parameters.

After the great performance leap due to the advances in deep learning, the next great improvement came by proposing pretraining tasks focusing on making the model understand the database schema and SQL queries. During pretraining the model solves tasks that will explicitly teach it aspects of the downstream task, eg. teaching schema linking for Text-to-SQL. We further analyze the power of pretraining in Section 2.0.4.

Focusing on Text-to-SQL, the model takes as input query-schema tuples, which differs from the expected BERT input. As a result, when we fine-tune the BERT model to the Text-to-SQL task, it must perform a big distribution shift that takes time and can be unstable or lead to worse performance. To alleviate this issue researchers have focused on creating pre-training tasks that can facilitate a smoother distribution shift.

GraPPa [45] proposes two pre-training tasks that utilize synthetic data. By observing the Spider dataset they create pairs of natural language-SQL query templates, which they

use to produce a large syntehtic dataset. The model is then pretrained on the synthetic dataset through two pretraining tasks. First, given a natural language query, the model is tasked to find which database tables and columns are needed. Second, given again a query in natural language, the model must predict the clauses the final SQL query should have *eg. "SELECT AND GROUP BY HAVING"*.

GAP [36] also proposes a set of pre-training tasks. The *column prediction* task has as input an utterance and a schema and the model must label which columns are used in the utterance and which are not, teaching it how to detect and match column mentions inside the NL query. The *column recovery* task will randomly replace a column name with a value of its cells and then they task the model to retrieve it, teaching it how to detect column usage based on values.

2.0.2 SQL-to-Text

SQL-to-Text takes as input a SQL query and explains it in natural language.

Since it is the reverse task of Text-to-SQL the main datasets used remain the same (i.e. WikiSQL and Spider).

CODE-NN [13] is an end-to-end approach that takes as input the SQL query and decodes it to NL. CODE-NN performs content selection (which part of the SQL query is important) and realization (generating the explanation in natural language) jointly using an attention-based architecture. The authors underline that the so-far architectures used a disjoint approach where content selection and realization models were independent of each other.

Xu et al. [43] suggest modeling the SQL query as a graph which allows the usage of a Graph2Seq model instead of a Seq2Seq model like CODE-NN above. The Graph2Seq model choice proves to be really effective, since the graph input better represents the structure of a SQL query instead of simply serializing it. Their main contribution is how one could represent an SQL query as a graph. They propose these two rules:

1. *SELECT clause*. They first create a node with the text attribute ”select”. Then they connect the column names that are selected with the ”select” node.
2. *WHERE clause*. They follow the same approach as the SELECT clause. However, they also add any logical operator like AND, OR, NOT as nodes which they then connect with the SELECT node.

Berant et al. [2] propose a more visual approach to explaining what a query is doing. They propose a context-free grammar which in turn creates a parse tree of the query. They then create an explanation of the query in natural language using hand-crafted templates but also highlight on the table which columns and cells are of the most importance for executing the query. Specifically, they highlight these 3 types of cells:

1. P_O, Cells returned in the query execution output (or used in calculating the aggregation eg. SUM).
2. P_E, Cells examined during the execution of the query.
3. P_C, Cells that were either aggregated or projected (full columns).

2.0.3 Data-to-Text

Data-to-Text approaches take structured data as input (i.e. a table or a graph) and transform it into natural language.

Data-to-Text can be distinguished into two main categories based on the type of input: Table-to-Text, and Graph-to-Text. Table-to-Text models take tables as input (eg. from Wikipedia) and have to solve challenges such as representing the table in a format that the model will understand or forcing the model to verbalise the whole table and not parts of it. Graph-to-Text models are given as input a graph and generate a natural language description of the graph. The model must simultaneously understand the components of a graph, nodes, and edges, and also manage to generate a valid verbalisation.

In Table 1, we can see a list of representative datasets used in both Table-to-Text and Graph-to-Text solutions. While there is a great number of datasets in the field of Data-to-Text none of them fits our QR2T use case as we show in Section 3.0.1.

Table 1: List of influential Data-to-Text datasets

Dataset	Size	Structure	Collection	Domain
WikiBio (2015)	728K	Info-boxes	Auto	First sentence of Wikipedia biographies
ROTOWIRE (2017)	5K	Table	Human	Basketball game summaries
ToTTo (2020)	100K	Table	Human	Description of Wikipedia tables
WebNLG (2020)	17K	Graph	Human	15 categories (building, person)
LDC (2020)	59K	Graph	Human	Graphs of ”who is doing what to whom”.

2.0.3.1 Table-to-Text

In Table-to-Text the input of the model is a table and the output is its verbalisation. We first explore some of the most influential datasets in Table-to-Text.

Wikibio [15] gathers 728,321 biographies from English Wikipedia. For each article, they provide the first paragraph and the infobox. The input infobox and the expected text output have a length of 26 and 53 words respectively. Its small input and output length, its big size, and the simplicity of the task make it a relatively easy dataset to solve, especially for current models.

On the other hand, ROTOWIRE [40] is one of the most challenging datasets, consisting of (human-written) NBA basketball game summaries aligned with their corresponding statistics. There are 4853 distinct summaries, covering NBA games played between 1/1/2014 and 3/29/2017 with some games having multiple summaries. The summaries have been randomly split into training, validation, and test sets consisting of 3398, 727, and 728 summaries, respectively. The input statistics have on average 628 records and the output has an average length of 805 words. As a result, the model not only has to pick the most interesting insights from the statistics that a journalist would report but also generate a lengthy article that even current models would struggle with.

ToTTo [27] is an open-domain English table-to-text dataset with over 135,000 datapoints, which proposes a controlled generation task: given a Wikipedia table and a set of highlighted table cells, produce a one-sentence description. During the dataset creation process, tables from English Wikipedia are matched with (noisy) descriptions. Each table cell mentioned in the description is highlighted and the descriptions are iteratively cleaned and corrected to faithfully reflect the content of the highlighted cells. ToTTo is currently one of the most impactful datasets in Table-to-Text since it is big (135K), diverse (having tables from multiple Wikipedia domains such as politics and entertainment), and of high quality since the verbalisations were manually created and corrected.

We explore some of the key table-to-text solutions.

The Field-gating Seq2Seq architecture [20] proposes enhancing the LSTM architecture, adding a content selection gate that decides which fields are more important along with a content planner which decides the order that these fields will be verbalized. This explicit method of deciding which parts of the table to verbalize, forces the model to remain faithful

to the input table, without modifying existing values or adding new ones.

AlignNet [21] is a schema-agnostic model that aims to better generalize in real-world use cases, where the columns and values found in the train set may not necessarily be the same as in the test set. AlignNet first aligns the test schema (columns) of the answer with a known train schema, replacing unknown attributes with known ones, and then use a state-of-the-art table-to-text generator, to generate an NL description of the table. The task of the schema aligner is: given an unknown input schema and a set of k support schemas from the training set, find the support schema that better aligns with the unknown one. Alignment is calculated based on the shared columns and cell values of the unknown schema and the candidate support schema.

Puduppully et al. [29] suggest a two-stage approach. The first stage creates a content plan which decides the order in which the records of the table should be generated. Content planning is performed through a pointer network which at each time-step t decides which record should be generated based on the record table and the previously selected records. Specifically, they use an LSTM with its hidden state initialized with the average of the record embeddings. Then the probability of a record being selected is obtained through attention over the input records and the hidden state. They then employ an LSTM network that takes the encoded plan into account and generates its verbalisation.

As in Text-to-SQL and SQL-to-Text, the advancements of PLMs greatly impacted Table-to-Text too.

TableGPT [10] utilizes the power of GPT-2 [30], an autoregressive language model that writes text based on an input prompt (in our case the table). However, a simple application of GPT-2 will not be enough, since the input is now structured and not text that these language models expect. As a solution, they propose bringing the input closer to text and teaching the model about table structure. Specifically, they suggest these pretraining tasks:

1. *Table Structure Reconstruction*, where given the representations of the values of the table they task the model to retrieve the column they belong, directly teaching the model the correlation between columns and values.

2. *Content matching*, where they reward the model if the order that it verbalises table values, is the same as in the target verbalisation in the dataset.

Similarly, the authors that formed the ToTTo [27] dataset attempted the straightforward solution of using T5 [31] a text-to-text pre-trained language model, which we present later. Impressively, simple application of T5 manages to beat all the other intricate models mentioned above.

Table-to-Text methods were the initial inspiration for finding a solution to our QueryResults-to-Text problem. Our pipeline, QR2T, has T5 [31] as its main building block, which is also used by state-of-the-art table-to-text solutions [27]. However, there are important differences between the Table-to-Text task and Query Results-to-Text:

1. The user does not want a general verbalisation of the table but a specific answer to a query.
2. There is information in the database schema that we should utilise. For example, during verbalisation of JOIN queries, we must take into account the table that each column belongs to.
3. Databases might not be as *"PLM friendly"* as Wikipedia tables, which have easy-to-understand column names and avoid abbreviations.

2.0.3.2 Graph-to-Text

In Graph-to-Text the task is to verbalise the graph given as input.

The WebNLG [9] corpus comprises of sets of triplets describing facts (entities and relations between them) and the corresponding facts in the form of natural language text. The corpus contains sets with up to 7 triplets each along with one or more reference texts for each set. The test set is split into two parts: seen, containing inputs created for entities and relations belonging to DBpedia categories that were seen in the training data, and unseen, containing inputs extracted for entities and relations belonging to 5 unseen categories.

LDC[1] is a graph dataset describing "who is doing what to whom" in a sentence. Each sentence is paired with a graph that represents its whole-sentence meaning in a tree structure. The sentences are taken from journal posts, forum discussions, and blogs, and their semantic tree is manually created.

Zhu et al. [49] in a similar fashion to RAT-SQL, extend the attention mechanism to include

[1] https://catalog.ldc.upenn.edu/LDC2020T02.

information about the relationships between two nodes that are not necessarily directly connected (1-hop). They first find the shortest path between two nodes of the Abstract Meaning Representation (AMR) graph. Then they propose a series of methods for transforming the path to an embedding that can then be injected into the attention mechanism. Specifically, the two best-performing methods for doing path2vec were a CNN-based solution where we slide a window over the path and traditional self-attention over the nodes of the path. A similar approach is followed by Cai et al. [4] but they use the transformer architecture for generating the path and a bi-directional GRU for encoding the information of the path.

Ribeiro et al. [32] investigate and prove that pretrained models, while hard to modify, outperform complex and intricate architectures. This is further supported by Kale et al. [14] where they discover that simply applying a pretrained model ”out of the box” will not only achieve state-of-the-art in Graph-to-Text but in Table-to-Text also.

We note that while Graph-to-Text solutions could be applied in the general context of an NLDBI and database democratization, for example explaining the database schema in natural language, but for our Query Results-to-Text task are not as applicable as Table-to-Text solutions.

2.0.3.3 Query Results to Text

Finally, Query Results to Text, which is the task that we are also aiming to solve, has seen some advancements. Deutch et al. [3] attempt to give explanations on how the result of a query occurred. To provide these explanations they face two main challenges. First, they must define how to transform the explanation into natural language. Second, there might be a number of explanations for the same query result. For the first challenge, specific templates are defined under the limitation that input queries are conjunctive. For the second challenge, an algorithm factorizes the possible explanations and then selects the shortest one as the one that is more readable.

However, the above solution requires the design of templates that will be filled in order to produce the final verbalisation. Such a requirement will not allow us to address the challenge of covering different databases and types of queries (i.e. JOINs, GROUP BYs, nestings, ...), since the number of required templates would be huge.

Table 2: Different pretraining tasks for language modelling, table understanding, and text-to-SQL. The base model field notes if the starting checkpoint was randomly initialised (-) or used another pretrained model.

Model	Task	Task Goal	Base Model	Downstream Tasks
BERT [7]	masking	Language modelling.	-	sentiment classification, summarization, …
	next sentence prediction	Understanding the relationship between two sentences.		
T5 [31]	random spans	Language modelling.	-	question answering, summarization, …
GPT [30]	sentence completion	Language modelling.	-	question answering, summarization, translation
TaBERT [44]	masked column prediction	Encourages the model to recover the names and data types of masked columns.	BERT	table question answering
	cell value recovery	Ensures information of representative cell values in content snapshots is retained after additional layers of vertical self-attention.		
Grappa [45]	column/table appearance	Familiarization with SQL.	RoBERTa [23] (similar to BERT)	text-to-sql
	query type prediction	Familiarization with SQL.		
GAP [36]	column prediction	Encourages the model to capture the alignment signals between the utterance and schema.	BART [18] (similar to T5)	text-to-sql
	column recovery	Strengthens the model's ability to discover the connections between the cell values and the column names.		
SeaD [41]	erosion	The model is required to identify the schema entities that are truly related to the NL question.	-	text-to-sql
	shuffle	Captures the inner relation between different entities and therefore contributes to the schema linking performance.		

2.0.4 Pretraining

Pre-training is often used for a variety of problems such as in computer vision [16, 25] but especially in natural language processing with huge transformer models managing to successfully model language (words, phrases, texts) into vectors that can then be used regardless of the downstream task [7, 31].

Pretraining aims to make the model understand and solve a general task, named *pretraining task*, for example, language modeling. In language modeling [7, 31] the model learns to predict the probability of a word occurring in a sentence given the rest of the words. While this task itself might not have many real-world applications, it creates a model that is able to understand language i.e. syntax, grammar, and semantics. This model can then be finetuned on a specific task, named *downstream task*, such as text summarization, and it will have greater performance compared to a model being trained on the downstream task only.For example, in text-to-SQL, pretraining will explicitly teach the model basic tasks such as schema linking [36, 45] before solving the much more complex text-to-SQL.

Creating such a model tends to be computationally expensive. However, the same model can then be finetuned to different tasks such as text summarization, sentiment classification, machine translation, and table-to-text without requiring to pretrain the model from scratch.

Finally, it is not mandatory for a pretrained model to be directly finetuned on a downstream task, but instead it can be further pretrained. Returning to the text-to-SQL example, the model has to understand syntax and semantics of text before being pretrained for more task-specific tasks like schema linking. For this reason, the starting checkpoint for pretraining is an already pretrained language model like BERT [7] or T5 [31].

In Table 2, we present some of the most influential pretrained models in the fields of language modelling, table understanding, and text-to-SQL, along with their respective pretraining tasks. We notice that most of the tasks erode in some way the input of the model (i.e. masking and shuffling) and then task the model to retrieve back the original input.

3. PROBLEM FORMULATION

The problem of Query Results to Text can be seen as defining a function f that, given the query results r, can produce a verbalisation v, which describes the contents of these results in an informative and fluent way.

$$f(r) \rightarrow v$$

3.0.1 Challenges of Query Results-to-Text

The Query Results-to-Text problem hides several difficulties.

Query Semantics. The input to the problem is the result of a query; however, the query semantics may not be evident in the results. For example, in Figure 4, the result does not convey that the query focuses on movies released after 1997.

Result ambiguities. Furthermore, query results returned by a relational database system might not include important information that both the user and the model that performs the verbalisation would require. This is mainly caused by the way that JOIN results are returned since they only contain the column names and not the table names by default. Continuing our example in Figure 4, the results do not tell that the first column refers to movies and the second to directors.

SQL notation. SQL notation poses additional challenges. For example, how to understand result attributes such as *AVG(age)*? Furthermore, such translations may depend on the data. In a DB containing sport data, the `MAX(lap_time)` refers to the *"slowest lap time"*, while in a database containing products, the `MAX(price)` refers to the *"highest product price"*.

Result Schema. Query results are typically given as a table whose schema, i.e., attributes, vary depending on the query. Even for the same database, the number of possible different query result schemas can be extremely large. Note that, in contrast, in Data-to-Text, the input typically abides to a specific schema.

Domain generalization. Furthermore, different databases cover different domains (e.g., e-commerce, health sciences, astrophysics) meaning that our solution should also be able to cover different domains and database schemas. This creates additional complexity to

the problem.

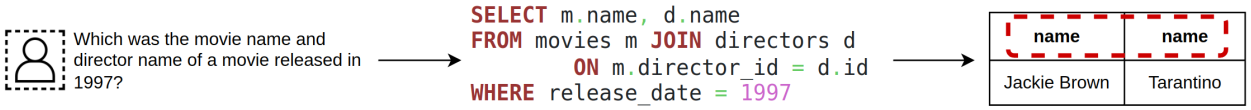

Figure 4: Result of a JOIN query on a movies database

Lack of problem-specific datasets. There is no dataset that targets our task of verbalising query results. A naive approach that we examine in our work is to just use a Table-to-Text dataset, in our case ToTTo [27], which we consider the most fitting Table-to-Text dataset for the Query Results-to-Text task. However, a complete solution for Query Results-to-Text task should take into account the queries in both natural and query language, and the DB schema in order to solve the result ambiguities mentioned above.

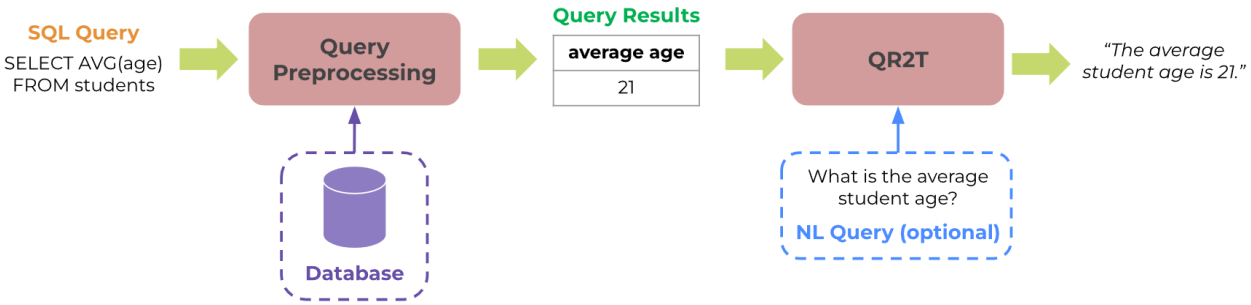

Figure 5: The pipeline we follow to create the verbalisation of a query result.

3.1 System outline

Our system consists of two pipelines.

The main pipeline, shown in Figure 5, aims at translating the query results of an input SQL query in natural language. Before executing the SQL query, it preprocesses it to inject extra information, in order to address the *result ambiguities* and *SQL notation* challenges mentioned previously in Section 3.0.1. The preprocessed query is executed. Then, the query results and the query in natural language are passed to the QR2T model which will produce the final verbalisation. We note that the existence of a natural language translation of the query is not a required parameter and our system will work without it, as we also show in the experiments in Section 5.0.5.

Our QR2T model has as its main building block T5 [31], a powerful PLM. T5 is an encoder-decoder transformer model pretrained on the C4 dataset, a huge amount of texts crawled

from the web. T5 has proven its worth in multiple tasks such as machine translation [5], text summarization [11], and question answering [33], regardless of the text domain, directly addressing the *domain generalisation* challenge.

However, employing T5 for our task is not straight-forward and comes with its own challenges. First, T5 expects as input text and not a table of query results, creating the need for an appropriate serialization of the input into text as shown in Figure 7 (see Section 4.0.1). Second, while T5 is a powerful text-to-text model, it has no notion of what a table is and the correlations that exist between:

1. *column-value*, for example, *director* is a *job*.
2. *column-column*, for example, the *job* is done by the person described in *name*.
3. *value-value*, for example, *Tarantino* is a *director*.

For humans, these correlations (Figure 6) are obvious since we are used to seeing and interpreting tables in our everyday life. However, the model will get as input a serialized version of the table as shown in Figure 7. The serialized version has special notation such as *<col1> title | text | Dune* which T5 will be seeing for the first time.

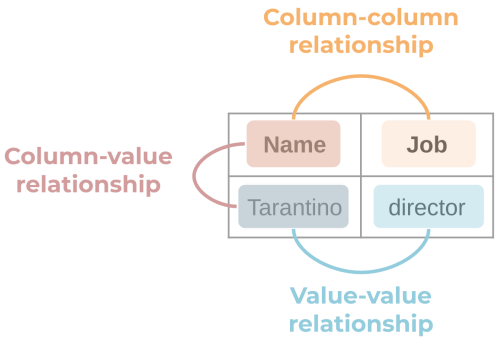

Figure 6: Correlations that exist in a table.

This challenge is addressed through the second pipeline (Figure 8) which will take as input the T5 model and pretrain it using tasks that will force it to understand the above correlations as shown in Figure 6. We explain in detail the idea behind each pretraining task in Section 4.0.2. Then, the model is finetuned first on ToTTo, a table-to-text dataset. While our end goal is creating a model that addresses Query Results-to-Text problem, we observe that first finetuning our model on a table-to-text dataset of high quality and big size, like ToTTo [27], could offer a smoother transition towards our Query Results-to-Text model. The final stage of the pipeline is finetuning the model on the QR2T Benchmark,

the dataset we created that specifically fits our Query Results-to-Text dataset. The output of the pipeline is the QR2T model which can then be used for inference.

In a nutshell, we start with T5 to achieve *domain generalization* in the NL side, we use pre-training tasks to help the model understand *query semantics* and *schema*, and we fine-tune with ToTTo to help *domain generalization* from the tabular data side. Finally, we further tune it on a dataset specifically designed for the problem. We note that each of these can be omitted from the pipeline, for example, we might avoid finetuning on ToTTo. We explore the impact of each stage and we present our results in Section 5.0.3.

4. METHODS

4.0.1 Input Encoding

T5 has the requirement that both the input and the output are texts. While we cover said requirement for the output, which is the query result verbalisation, our input is a table along with some supporting information such as table names and the query in natural language. We define a compact way of serializing our tables which fits the Query Results-to-Text task and generates a small number of tokens making it harder to reach our T5 limit of 512 tokens, as shown in Figure 7.

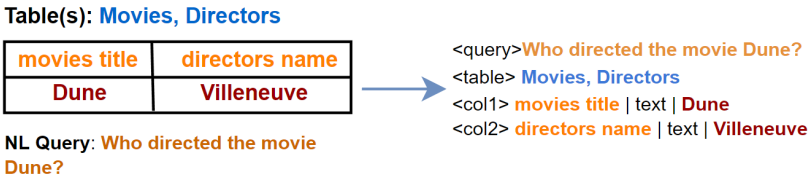

Figure 7: Example serialization of a query result to string.

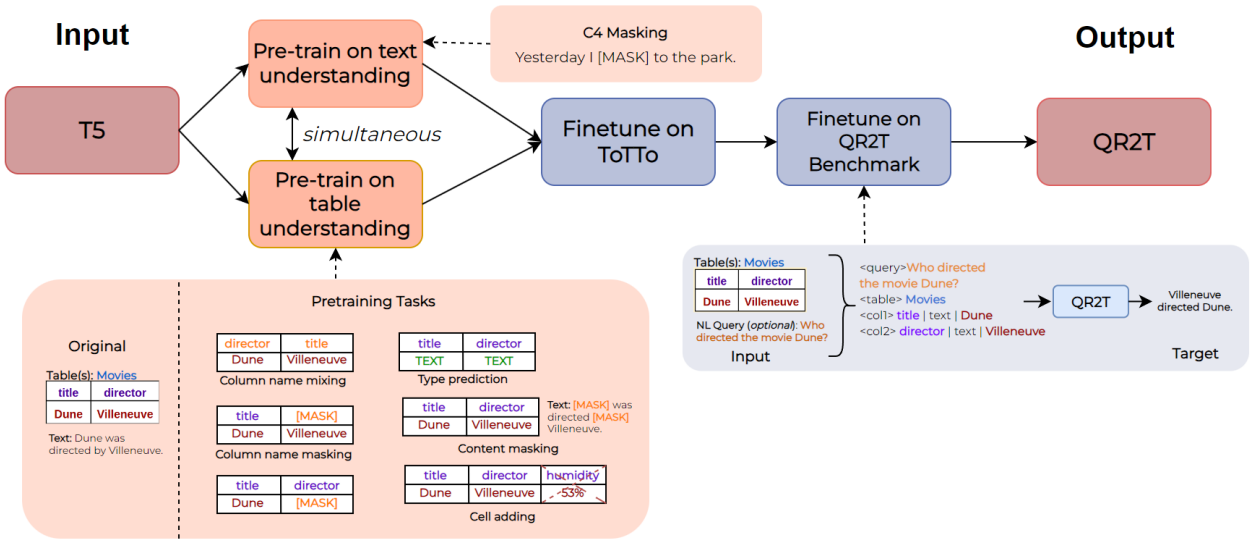

Figure 8: The pipeline we follow to create the QR2T model by pretraining, finetuning on ToTTo, and finetuning on our QR2T Benchmark.

We also experimented with the input format proposed by the creators of ToTTo [14], however, we quickly noticed an issue with its structure. An XML-like encoding ends up being too verbose and the model will have to first understand the different tags and their meaning.

Having formulated our problem into a text-to-text framework, we can then finetune T5 on our QR2T Benchmark as we would in any other downstream task like text summarization.

4.0.2 Pretraining T5 for Table Understanding

Simply finetuning T5 on our QR2T Benchmark does not address specific challenges. The model will not be able to understand the structure of a table, meaning the correlations that exist between column names and values.

To solve this challenge, we propose further pretraining T5 using tasks that focus on table understanding. Further pretraining already pretrained models with domain-specific tasks is a technique followed successfully in other cases too. For example, in TaBERT [44], the authors propose further pretraining BERT on a set of table understanding tasks. We note that while TaBERT is a model able to understand what a table is, which fits our goal, we cannot use it due to it being an encoder-only model and cannot generate text. However, some of our pretraining tasks are based on TaBERT and are adapted to fit T5.

In what follows, we present our pretraining tasks.

Column name mixing *adapted from SeaD* [41]
We mix the order of some columns (30%) while keeping their respective cell values in the same place. We then require the model to output all the column names, even the ones that were unchanged, in the correct order.
Goal: This task will force the model to understand the correlation between a cell and its column name.

Column name masking *adapted from SeaD* [41]
Similarly, instead of mixing the columns we replace them with a mask. Specifically, we replace the column names using the mask token that T5 expects which is *"<extra_id_i>"* where *i* is the *i-th* mask of the input. As an output, we expect only the masked column names.
Goal: Understand how a cell value and adjacent column names correlate with a column.

Column adding *adapted from SeaD* [41]
We add a column randomly picked from another table of the dataset and then expect the model to return only the original columns without the added one.
Goal: The model will have to understand the general content of the table to find the added column.

Column type prediction *adapted from TaBERT* [44]
We hide the cell values from the model and then ask for the types of each column based

on the column names. The allowed types that the model can choose are TEXT, DATE, and NUMERIC which have been extracted automatically.
Goal: If the model can understand the type of each column then it has a better chance of using it in a syntactically correct way.

Cell value masking *adapted from TaBERT* [44]
We randomly mask a cell value of the input table, and then the model has to retrieve it using the other column info, values, and the table name.
Goal: Forces the model to understand both the general context of the table and the relationship between a column name and its value.

Content masking.
Many of the tables have text preceding or following them. We first split these texts into sentences and then find the sentence that has the most overlap with one of the table rows. We then pass through the model both the masked sentence and the table with this row and the model has to fill the masks. When masking the sentence we purposefully increase the chance of choosing text spans that are also in the tables.
Goal: This is the task closest to our Query Results-to-Text task and forces the model to understand the relationship between a text passage and a table.

C4 masking.
This is the original pretraining task of T5 on the C4 dataset where we mask text spans.
Goal: This task acts as a regularizer so that our model does not overfit on the above tasks and forgets all about text understanding and natural language generation.

Concerning pretraining datasets, we used two different sources of tables, each one with its advantages and disadvantages. The first approach focuses on quantity over quality since we are using tables crawled from the web from a table corpus called WDC [17]. WDC contains 233 million tables extracted from the Common Crawl [1], a *crawl of all web pages found in the internet*. The second approach focuses on quality over quantity by using ToTTo [27] which is a carefully and manually made Table-to-Text dataset. Depending on the dataset we use, some of the pertaining tasks change accordingly as described below.

WDC contains a subset of the HTML tables on the Web that contain relational data which can be useful for various applications. It has a great diversity of topics covered since it is not produced by a specific domain (eg. Wikipedia). We apply some cleaning based on

[1] https://commoncrawl.org/

the following rules:

- table is in English
- table is not empty
- table has a header (column names)
- table is not huge (< 50 columns)
- table has a title
- table does not have numbers in the header (eg. 1, 2, ...)

While being big and diverse, WDC has two main pain points that impact the performance of the model after pre-training on our downstream task, Query Results-to-Text. First, the quality of the data is low with many tables being parsed incorrectly or having non-expressive column names and values such as abbreviations or single letters. Second, the size of WDC is way too big for the computational power that we had available for our experiments forcing us to use a smaller subset of it (15%).

For these reasons, we also examine pretraining on ToTTo too, which is manually annotated and hence clean and of high quality. Also, the smaller size makes training less computationally expensive so that we utilize the whole dataset and not just a part of it. Note that we will refer to the tasks on ToTTo as pretraining tasks while we will refer to the tasks on WDC as pretraining tasks to avoid confusion. Semantically and implementation-wise nothing changes.

Table 3: The table shows which tasks were used for each dataset during pretraining.

	WDC	ToTTo
Column name mixing	X	X
Column name masking	X	X
Column adding		X
Column type prediction	X	X
Cell value masking	X	X
Content masking	X	

In Table 3, we can see which table understanding tasks were used for each dataset.

4.0.3 Preprocessing

While the results of an SQL query are usually returned in a table, this table often does not contain the necessary information that would aid the model to generate better verbalisations. Especially since the verbalisation is performed by T5, which is a model pretrained in NL generation, the column names and contents, ideally, should be as close as possible to natural language. For example, the column name *"AVG(age)"* is less probable to be understood by T5 in comparison with *"average age"*. A second problem with verbalising the results table is the lack of necessary information. Some SQL engines (e.g. MySQL), will not return the table names of returned columns leading to confusion as seen in Figure 2. Preprocessing attempts to alleviate these two issues by transforming the query before being executed.

Algorithm 1 Query Preprocessing

Input: query q, database db

validate_query(q, db)
Extractions
$sel\_cols \leftarrow extract\_sel\_cols(q)$
$where\_cols \leftarrow extract\_where\_cols(q, db)$
$group\_by\_cols \leftarrow extract\_group\_by\_cols(q)$
$tables \leftarrow extract\_from\_tables(q)$
Injections
$injected\_cols \leftarrow (where\_cols \cup group\_by\_cols) - sel\_cols$
$q \leftarrow add\_injected\_cols(q, injected\_cols)$
if $q\ is\ join\ query$ **then**
    $q \leftarrow apply\_join\_aliases(q, tables)$
end if
if $q\ is\ aggregate\ query$ **then**
    $q \leftarrow verbalise\_aggregates(q)$
end if
$q \leftarrow add\_limit\_1(q)$
return q

Algorithm 1 provides an overview of the preprocessing steps applied on the query before being executed. As input, it receives the user query and database information like the

schema. In line 1 we check if the original query is valid saving preprocessing time since the final query would also be incorrect. Then, lines 3-5 extract column names found in the SELECT, FROM, WHERE, and GROUP BY clauses. While extracting from the WHERE clause we are careful to not extract any primary/foreign keys that are used in joining two tables since. Usually, these keys do not have any semantic meaning and we do not want them included in our final verbalisation. For example, if the WHERE clause is *"movie.dir_id = dir.id AND dir.age > 40"* only *"age"* would get extracted without *"dir_id"* and *"movie.dir_id"* since they are just used for joining the two tables. Finally, we also extract the tables that the query includes in the FROM clause.

Lines 8-13 perform the query injections. Specifically, line 8 calculates which columns should get injected in the SELECT clause. As stated above these include non-primary/foreign key columns that are found in the WHERE and GROUP BY clauses (Figure 9 (b)). The assumption is that these columns would provide enough database info so that the model will have a better chance of understanding the contents of the final query results.

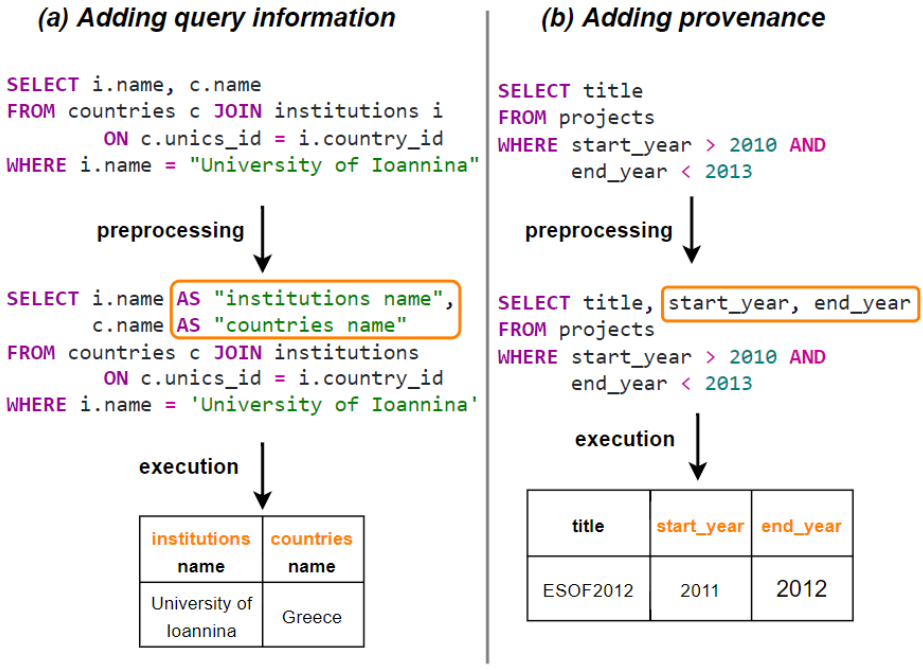

Figure 9: Examples of adding JOIN aliases (a) and injecting where clause columns (b) so that the final table will be more informative for the final task of verbalising it.

JOINs are an interesting case since in SQL the result table will not include the table name but only the names of the columns. As seen in Figure 2, the model will have a hard time understanding that the first *name* refers to ”institutions” and the second *name* to ”countries”. We aid the model by adding the table name in front of the column name. This preprocessing step executes only if the query contains JOINs (Figure 9 (a)).

Also, aggregates require some preprocessing so as to make them more easily understood by the model that is pretrained on natural language. Specifically, we apply a set of mapping rules shown in Table 4. Note that the COUNT aggregate is a special case since the user does not want the count of a column but the count of result rows. So the user will likely pick star (*COUNT(*)*). In this case, instead of using the column name, we use the name of the table.

Table 4: Aggregate verbalisation rules applied on column *year* of a table named *projects*.

Aggregate	Verbalisation
SUM(year)	sum of year
MAX(year)	maximum year
MIN(year)	minimum year
AVG(year)	average year
COUNT(*)	**count of projects**

Finally, since we are verbalising a single row of the results, LIMIT 1 is added in line 14. The final preprocessed query is executed on the database and the returned results can then be the input of the model that will perform the verbalisation. In Figure 10 we provide examples that showcase the importance of these preprocessing steps.

We note that the preprocessing algorithm is extensible and new rules could be added to cover other query cases such as arithmetic operations (+, -) in SELECT.

5. EXPERIMENTS

We first describe all the datasets used for pre-training, fine-tuning, and evaluation, then we provide some details about our experiment setup and finally provide results (both automatic and human evaluations) and ablation studies.

5.0.1 Datasets

Solving a problem with neural networks requires a sufficient amount of data for training and evaluation. Our pipeline uses two categories of datasets. First, WDC and C4, which are huge crawled unlabelled datasets, used for pretraining. Second, ToTTo, QR2T Benchmark, and Cordis, which are golden datasets annotated by humans, used for pretraining tasks, fine-tuning, and evaluation.

C4 and Web Table Corpora (WDC) have both been used in the past for pretraining (T5 [31], TaBERT [44]). C4 is a colossal, cleaned version of Common Crawl's web crawl corpus. It was based on Common Crawl dataset which consists of all the websites on the internet. In our case, C4 has a regulatory effect on our pretraining process and we use a small part of it (360,000 sentences).

WDC contains a subset of the HTML tables on the Web that contain relational data. It has a great diversity of topics since it is not produced from a specific domain (eg. Wikipedia). We apply relatively straightforward filtering so as to avoid non-expressive tables such as the ones that have numbers for column names only. Since this is not a human-annotated or supervised dataset it will be noisy and of low quality.

For pretraining tasks, fine-tuning, and evaluation three datasets are being used, ToTTo [27], the QR2T Benchmark created by our team, and Cordis also created by our team.

In ToTTo, given a Wikipedia table and a set of highlighted cells, the model should generate a single-sentence description. It is currently the largest Table-to-Text dataset annotated by humans with over 120,000 training examples. It covers a big variety of domains such as sports, countries, and politics. While our Query Results-to-Text task is not the same as solving Table-to-Text, we can see some similarities between them since they are both trying to verbalise a table. This motivates us to examine how knowledge from the Table-to-Text task can transfer to our task.

A problem that becomes quickly apparent is the lack of a dataset specific to our Query Results-to-Text task, making both training and evaluation hard. We decided to create our own and to our knowledge the first, dataset which contains verbalisations of query results, named QR2T Benchmark. We obtain the queries from a subset of the Spider dataset [46] which contains natural language queries with their SQL translation and the databases needed to execute these queries. In other words, QR2T Benchmark is an extension of the Spider dataset. While we are not verbalising the whole Spider dataset, we are careful so that there is enough representation of different query types such as simple SELECT FROM WHERE without any joins, JOINs, aggregates, and GROUP BYs. In Table 5 we can see the number of queries per category.

Table 5: Number of queries per category in the QR2T Benchmark dataset created by us.

Category	Number of Queries
SELECTs	580
Aggregates	404
JOINs	290
JOINs + Aggregates	137
GROUP BYs	144

The annotators were shown the query in natural language, the table names needed for executing the query, and the results of the preprocessed query. Then, they were tasked to verbalise every part of the query results table. Note that as the last step of query processing, we are applying a *LIMIT 1* so the annotators will have to verbalise only one row of the results. For the training portion of the dataset, we have a single annotation while for evaluation we have three annotations per query result. By having multiple annotations we alleviate issues that occur from n-gram based automatic metrics such as BLEU. The training and evaluation portions of the QR2T Benchmark consist of 1,244 and 311 datapoints respectively.

QR2T Benchmark is used both for training and evaluation and spans a big number of databases. However, we should also evaluate QR2T on (a) a real-world database actually used in production and (b) on a database that has great dissimilarity with the databases found in the QR2T Benchmark. For example, QR2T Benchmark databases on average only have 4 tables which is a small number compared to real-world production databases. For that reason, we also evaluate our model on the CORDIS database.

CORDIS[1] is a real database used by the European Commission to store information about EU-funded programs such as projects, participants, institutions, etc. We have chosen CORDIS as our real-world database not only because it is actually used in production, but because we also had access to 222 curated queries that could have been asked by real users. We consider this a key point in robust evaluation. When we first tried to create our own queries we observed that our lack of familiarity with the database led to simple and non-informative queries. The curated queries cover a range of patterns such as JOINs, aggregates, and GROUP BYs. Similarly, as in the QR2T Benchmark, the query results were verbalised by a set of annotators familiar with both CORDIS and SQL.

Table 6: Overview of the datasets used.

Dataset	Size	Usage	Train/Test Split
$C4_{subset}$	360,000	pretraining	-
WDC_{subset}	1,440,000	pretraining	-
ToTTo	145,000	finetuning, evaluation	80%/20%
QR2T Bench.	1,555	finetuning, evaluation	75%/25%
CORDIS	222	finetuning, evaluation	50%/50%

5.0.2 Setup

In all of our experiments, we use both the T5 model and tokenizer provided by huggingface [2]. We specifically perform all the following experiments using *t5-small*. While in other tasks the size of T5 directly correlates with the performance, in our case we did not observe any improvements by using a bigger variation (*t5-base*). This is further supported by the findings of Mehta et al. [24] who also used *t5-small* in their Data-to-Text task. Concerning hyperparameters, we are using the AdaFactor optimizer with a constant learning rate of 0.001 and a batch size of 24. In pretraining tasks we train only for one epoch as proposed in the pretraining of T5 [31], while for finetuning we are training for up to 7 epochs keeping as our best checkpoint the one with the highest validation BLEU score.

[1] https://data.europa.eu/data/datasets/cordisref-data?locale=en

[2] https://huggingface.co/docs/transformers/model_doc/t5.

5.0.3 Results

In Section 4.0.2 we have defined a pipeline that takes as input the original T5 model and through some steps (pretraining tasks, finetuning on ToTTo) creates the final QR2T model. However, there are a lot of ”moving” parts in this pipeline, which need careful evaluation of their impact on the final performance. From a technical perspective, this is effortless since each part of the pipeline is independent and can simply be removed without requiring changes on the other components.

The goals of our experiments are:

- To provide a baseline for the QR2T task.
- To examine the impact of transfer learning using the ToTTo dataset.
- To compare the performance boost from pretraining tasks.

We use automatic evaluation metrics that have been used on similar tasks, such as Table-to-Text. All of our metrics take as input two texts: the reference and the generated verbalisation, and return a number from 0 to 1.

BLEU [26], calculates the overlap of n-grams of different sizes between the reference and the predicted sentences. While it is the most widely used metric for comparing texts, it fails to take into account the semantic similarity of the two sentences.

PARENT [8], is more of a task-specific metric since it takes into account that our goal is to verbalise a table. At its core, it uses the BLEU formula for calculating n-gram similarity. However, it penalizes the model less if our reference text contains information not included in the input table or if our predicted text includes information that is in the table but not in the reference text. Both of these modifications perfectly fit our Query Results-to-Text task making us believe that PARENT correlates more with our task than BLEU.

Bertscore [47], instead of using n-gram comparisons like BLEU and PARENT, compares embeddings generated by a BERT [7] model. This way we are comparing the semantic meaning of the reference and the predicted sentences. For example, the two sentences ”The lake is beautiful”, ”The lake is magnificent”, will have a low BLEU score but a high Bertscore, since BERT will generate similar embeddings for the words ”beautiful” and ”magnificent”.

Table 7: Results of the different model pipelines on 3 datasets, QR2T Benchmark, Cordis, and ToTTo. Each row represents a different pipeline that outputs a model that takes input tables or query results and outputs a verbalization. The letters *P* and *F* express pretraining and finetuning respectively

	Evaluation Dataset								
	$QR2T_{Bench}$			Cordis			ToTTo		
Training Pipeline	**BLEU**	**PARENT**	**Bertscore**	**BLEU**	**PARENT**	**Bertscore**	**BLEU**	**PARENT**	**Bertscore**
$ToTTo(F)$	0.298	0.271	0.925	0.315	0.372	0.929	0.410	0.478	0.940
$WDC(P) - ToTTo(F)$	0.280	0.356	0.924	0.297	0.356	0.924	**0.416**	**0.554**	0.941
$ToTTo(P) - ToTTo(F)$	0.288	0.269	0.924	0.323	0.376	0.934	0.415	0.476	**0.948**
$QR2T_{Bench}(F)$	0.580	0.547	0.955	0.323	0.357	0.929	0.167	0.2	0.889
$ToTTo(F) - QR2T_{Bench}(F)$	0.627	0.594	0.957	**0.372**	0.406	0.935	0.38	0.459	0.935
$WDC(P) - ToTTo(F) - QR2T_{Bench}(F)$	0.632	0.402	0.96	0.332	**0.45**	0.927	0.381	0.551	0.935
$ToTTo(P) - QR2T_{Bench}(F)$	0.610	0.573	0.960	0.315	0.358	0.929	0.322	0.395	0.925
$ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$	**0.634**	**0.597**	**0.962**	0.371	0.421	**0.935**	0.381	0.455	0.935

In Table 7 we can see the results of all the pipelines on all of our evaluation datasets. For example, the pipeline $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$ can be described as:

1. Pretraining (P) T5 on the ToTTo dataset with the pretraining tasks define in Section 4.0.2
2. Finetuning (F) of the model of step 1 on the **ToTTo** training set
3. Finetuning (F) of the model of step 2 on the **QR2T** training set

The first pretraining step aims to familiarize T5 with what a table is. Next, we finetune the model on ToTTo. While ToTTo does not match our task of verbalising query results, its task of Table-to-Text combined with its quality and size has a positive impact on our final model. Next, we finetune the model on the QR2T Benchmark. While the QR2T Benchmark is much smaller than ToTTo, it was specifically created for the task of Query Results-to-Text. The approach of starting with bigger datasets that are not fully specific to our task and transitioning to smaller datasets that fit exactly our task is often used in low-resource cases such as machine translation of rarely used languages [22]. Then every model (or every row in Table 7) is evaluated on the evaluation sets of ToTTo, QR2T Benchmark, and Cordis.

In the first three rows we examine the performance of pipelines that are not using our QR2T dataset. The metrics on the QR2T dataset are low, which is expected since the model has never seen query results as input, instead it has only seen Wikipedia tables.

This also proves that a Table-to-Text model will not be able to perform well on the Query Results-to-Text task. On the other hand, these models perform adequately on the ToTTo evaluation set with the $WDC(P) - ToTTo(F)$ having slightly better performance. We consider this as an indication that pretraining on the WDC dataset helps the model have a better understanding of what a table is.

On the final 5 rows of the table, we can see pipeline variations that all end with a finetuning step on our QR2T Benchmark. We can see that having a dataset that is task-specific, even if it is low-resource, has a great impact on our scores. This is expected since the tables from ToTTo have important differences from QR2T Benchmark. Specifically, the latter also has the query in natural language and the table contents (i.e. the query results) are from databases instead of Wikipedia tables.

In addition, the $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$ pipeline seems to be the best performing on our Query Results-to-Text task. This confirms that the model through the pretraining tasks defined in Section 4.0.2, learns structural information such as the relationship between a column name and its cell value or the correlations between columns.

The best-performing model for Query Results-to-Text is $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$. This is a positive result that further supports the usage of pertaining the model using pretraining tasks on a high quality Table-to-Text dataset, in our case ToTTo.

NL Query	Result Table	QR2T(F)	ToTTo(F)-QR2T(F)	Totto (P)-ToTTo(F)-QR2T(F)
Which papers have the substring "Database" in their titles?	**papers** **title** Making Standard ML a Practical Database Programming Language	The paper Making Standard ML a Practical Database Programming Language has the substring Database in their titles	The paper's title is Making Standard ML a Practical Database Programming Language.	The paper titled Making Standard ML a Practical Database Programming Language has the substring "Database" in their titles
What is the average, maximum, and minimum for the number of hours spent training?	**Player** **average HS** | **maximum HS** | **minimum HS** 1150 | 1600 | 300	The average HS is 1150, maximum HS is 1600 and minimum for the number of hours spent training	The average hours spent training are 1150, 1600 and 300.	The average HS is 1150, the maximum is 1600 and the minimum is 300.
What are the names of captains that have either the rank Midshipman or Lieutenant?	**captain** **name** | **rank** Captain Sir Henry Langford | Midshipman	Captain Sir Henry Langford has a rank Midshipman or Lieutenant.	Captain Sir Henry Langford has the rank Midshipman.	Captain Sir Henry Langford is ranked as Midshipman.
What is the origin and destination for all flights whose price is higher than 300?	**flight** **price** | **origin** | **destination** 780.99 | Los Angeles | Tokyo	The origin is Los Angeles and the destination is Tokyo.	The flight origins in Los Angeles and the destination Tokyo have a price of 780.99	The flight from Los Angeles to Tokyo has a price of 780.99

Figure 10: Inference examples of different of the same query results using three different models.

In Figure 10 we have picked some example inferences that showcase both some impressive results and shortcomings of the QR2T models created.

The first example shows that the $ToTTo(F) - QR2T_{Bench}(F)$ model fails to take into ac-

count the NL query which requires the paper with *"Database"* in its title. This pattern appears consistently on the $ToTTo(F) - QR2T_{Bench}(F)$ model suggesting that training only on ToTTo before QR2T will make the model focus specifically on the table contents, ignoring additional metadata like the table title.

The second example tasks the model with verbalising 3 columns with similar content (minimum, maximum, average). The models without the pretraining tasks fail to take into account all the columns correctly. However, the $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$ model generates a complete verbalisation without being confused by the number or the content of the columns.

Table 8: Performance of the best pipeline $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$ per query type.

	BLEU	PARENT	Bertscore
SELECTs	0.6	0.57	0.96
Aggregates	0.71	0.66	0.98
JOINs	0.6	0.6	0.95
JOINs + Aggregates	0.63	0.5	0.95
GROUP BYs	0.57	0.53	0.96

In Table 8 we present the results of the pipeline that performed the best $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$ for each query type. We observe that the model best verbalises aggregate queries. Intuitively this makes sense since aggregate queries have a relatively uniform way of being verbalised (i.e. "The average *X* is *Y*") and almost all the time have a single column returned, which again makes the task easier. On the other hand, JOINs are more challenging since our model must understand the underlying relationship between the tables. Similarly, our model struggles with GROUP BYs since GROUP BYs have the goal of calculating "a value per group based on a condition". While for an SQL expert this is apparent for QR2T it is not and has a hard time understanding it through training.

5.0.4 Human Evaluation

Automated metrics have two main issues that should be addressed in order to have a complete overview of the performance and the shortcomings of the model. The first problem is that the metric does not always correlate with the correctness of the verbalisations. After a manual inspection on the results of the best-performing model ($ToTTo(P) - ToTTo(F) -$

$QR2T_{Bench}(F)$) out of the 45 inferences that got a BLEU score of 0, only 13 were actually wrong. The second problem is that these metrics are not interpretable meaning that a low score does not explain which type of error we had.

For this reason, we perform a manual evaluation on the inferences of 4 model pipelines, $QR2T_{Bench}(F)$, $ToTTo(F)-QR2T_{Bench}(F)$, $WDC(P)-ToTTo(F)-QR2T_{Bench}(F)$, $ToTTo(P)-ToTTo(F)-QR2T_{Bench}(F)$. The evaluation focuses on two axes correctness and fluency. Note that a verbalisation can be both incorrect and fluent and vice versa. Specifically, the evaluator picks a choice of ”Correct”, ”Incorrect” for correctness and ”Not fluent”, ”Adequate”, ”Perfect” for fluency. The evaluations are performed by 7 researchers familiar with both databases and the NLP domain.

If the evaluator has deemed the verbalisation incorrect we also ask them to categorize the error as erroneous (1) if the model used part of the query results in a wrong syntactical or semantic way, omission (2) if the model has ignored one of the columns of the results table or hallucination (3) if the model has generated a value or column that was not included in the input. These statistics have an important role in discovering weaknesses of our current approach so that we can have more specific targets for future improvements.

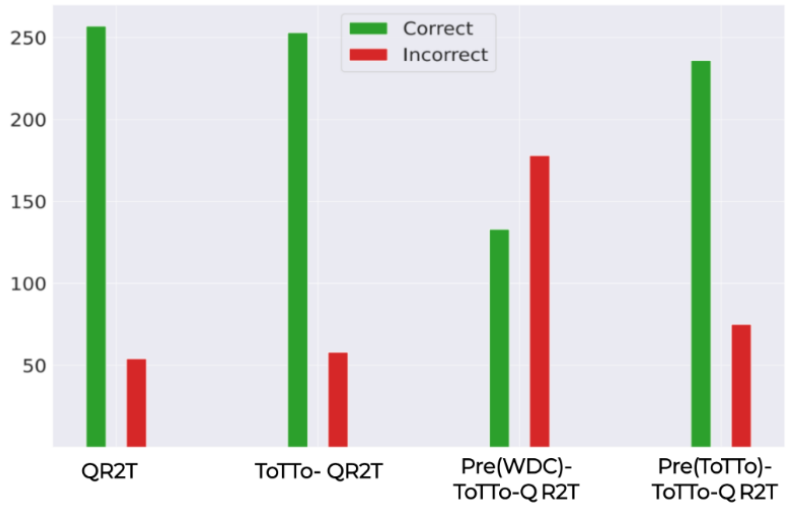

Figure 11: Human evaluation of the correctness of verbalisations on the QR2T dataset.

In Figure 11 we can see the correct (green) and incorrect (red) verbalisations. Interestingly, the $QR2T_{Bench}$ and $ToTTo-QR2T_{Bench}$ models have slightly more correct verbalisations than $ToTTo(P)-ToTTo(F)-QR2T_{Bench}(F)$ and all of them are marginally better than $WDC(P)-ToTTo(F)-QR2T_{Bench}(F)$. We suspect that the low quality of WDC which is used in the pre-training tasks has a negative impact on T5 which forgets to understand and generate semantically meaningful text.

Figure 12: Human evaluation of the fluency of verbalisations on the QR2T dataset.

We observe similar results in fluency in Figure 12. All the models except $WDC(P) - ToTTo(F) - QR2T_{Bench}(F)$ achieve great fluency with $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$ being slightly better. Again we notice the negative effect of the WDC dataset and the pre-training tasks.

Through human evaluation, we observe the main problem of automatic metrics (e.g. BLEU, PARENT). While our model does not achieve perfect automatic metrics, the human evaluators found that many verbalisations that got bad scores, were actually correct. Specifically, out of the 45 verbalisations that got a BLEU score of 0, only 13 of them were actually wrong. Also, out of the 100 worst-performing verbalisations, only 33 of them were actually incorrect. The opposite, having a good BLEU score and being deemed incorrect by the evaluator, is much rarer.

The results from the human evaluation seem really promising and we consider that QR2T could be integrated into a natural database interface. So far, the QR2T model has been integrated into the ODD [3] system of the INODE [4] project, providing natural language answers to queries that return a single row, Figure 13.

Finally, we are showing in Table 9 which were the main errors that lead to a verbalisation being incorrect. For all models, the main category of errors is ”Erroneous”, which is defined as failing to use the contents of the result tables in a semantically and syntactically correct way. For example, in Figure 10 in the 4th example the model $ToTTo(F) - QR2T_{Bench}(F)$

[3] https://www.inode-project.eu/odd3/

[4] https://www.inode-project.eu/.

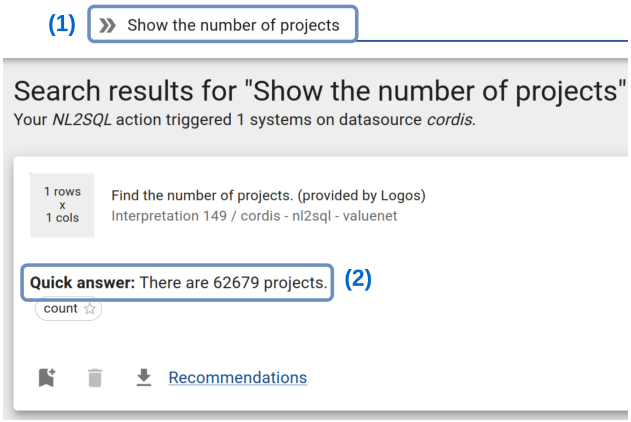

Figure 13: Integration of QR2T on the Open Data Dialog system. First, the user asks a natural language query (1) and then since the query results had a single row, we return its verbalisation (2).

Table 9: Analysis of the errors found in the human evaluation. Note that one verbalisation might have multiple errors.

	Erroneous	Omissions	Hallucin.
QR2T	38	19	4
ToTTo-QR2T	52	16	3
WDC(P)-ToTTo-QR2T	209	48	31
ToTTo(P)-ToTTo-QR2T	52	46	9

outputs ”the flight *have* a price” instead of ”the flight *has* a price”.

The second error category is omission. We can see that all the models have to deal with omissions but especially the $WDC(P) - ToTTo(F) - QR2T_{Bench}(F)$ and $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$. We find this an important observation that motivates future research to focus on forcing the model to verbalise the whole table of the query results.

Finally, we can see that hallucinations are rare in all the models except the $WDC(P) - ToTTo(F) - QR2T_{Bench}(F)$. Again the main suspect behind this is the low quality of WDC which is used in our pre-training. WDC is a huge dataset that is hard to analyze and considering that we are using a sample of it due to computational constraints. As a result, the sample might not be representative enough of many domains.

The main takeaways we obtain from our evaluation are:

- A powerful Table-to-Text model [14] will not perform well on the Query Results-to-Text problem.
- Pretraining on a clean dataset, in our case, ToTTo, will offer significant improvements

compared to pretraining on a much bigger but noisy dataset, like WDC.

- Pretraining on ToTTo offered significant improvements in all automated metrics on the QR2T Benchmark dataset.
- In contrast, human evaluation showed that pretraining did not improve performance. Specifically, pretraining increases the number of omissions compared to only fine-tuning, suggesting future research could focus on making the model to verbalise the whole table, not just parts of it.

5.0.5 Ablation Studies

Our ablation studies focus on examining more specific parts of the pipeline and their impact on our results. In Section 4.0.2, we proposed a set of pretraining tasks on the ToTTo dataset that will help the model understand what a table is. While there is a theoretical motivation behind each one of the tasks we want to examine their separate impact discovering the relative importance of each pretraining task.

To achieve this we perform two tests named ”all-but-one” and ”only-one” similar to BLEURT [35]. In ”all-but-one” we train T5 on all the pretraining tasks except one while in ”only-one” we train using just one of the pretraining tasks and remove all the others.

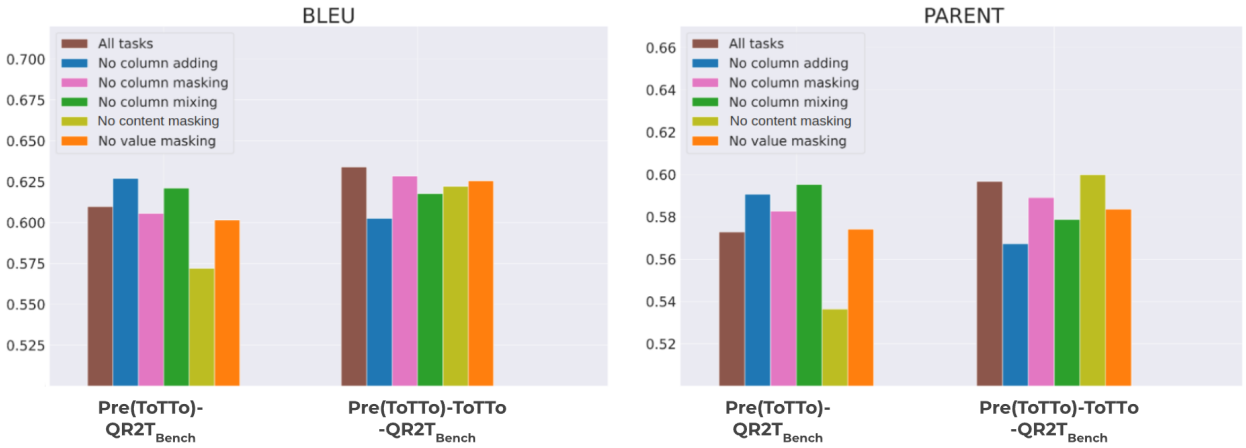

Figure 14: Impact of using all pretraining tasks except one.

In Figure 14, we can see that in the case of $ToTTo(P) - QR2T_{Bench}(F)$, we have the worst performance if we remove the ToTTo original task (Table-to-Text) while the performance of our model increases if we remove column adding or column mixing. On the other hand, the $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$ (our best model) has the best performance when all the tasks are included. Interestingly, the most effective task seems to be column

adding which had a negative effect in the case of $ToTTo(P) - QR2T_{Bench}(F)$. We also note that both metrics BLEU and PARENT correlate and lead us to the same conclusions.

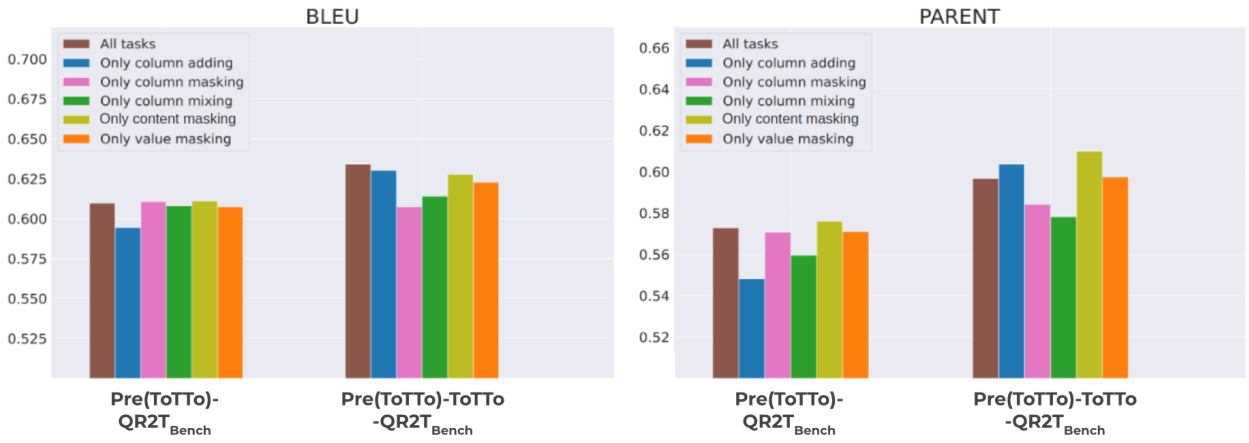

Figure 15: Impact of using a single pretraining task.

In Figure 15, we examine training only on a single pretraining task instead of using all of them. In $ToTTo(P) - QR2T_{Bench}(F)$ we can see that we achieve similar performance with all tasks even when compared with using all the tasks. This suggests that the choice of the pretraining task might have a small role in performance. However, in $ToTTo(P) - ToTTo(F) - QR2T_{Bench}(F)$ we can see that just using column mixing has a great performance impact when compared with using all the tasks or column adding.

The second ablation study focuses on studying the effect of not including in the model input the natural language query. As shown in Figure 16 when we train on $QR2T_{Bench}$ we include in the input the query that the user asked in natural language. Since our model aims to be a part of a natural language database interface, this is an assumption that will hold most of the time. However, in case we do not have access to the NL query we want to investigate the expected performance drop.

As expected all models are negatively affected by a similar margin when they have no natural query in their input.

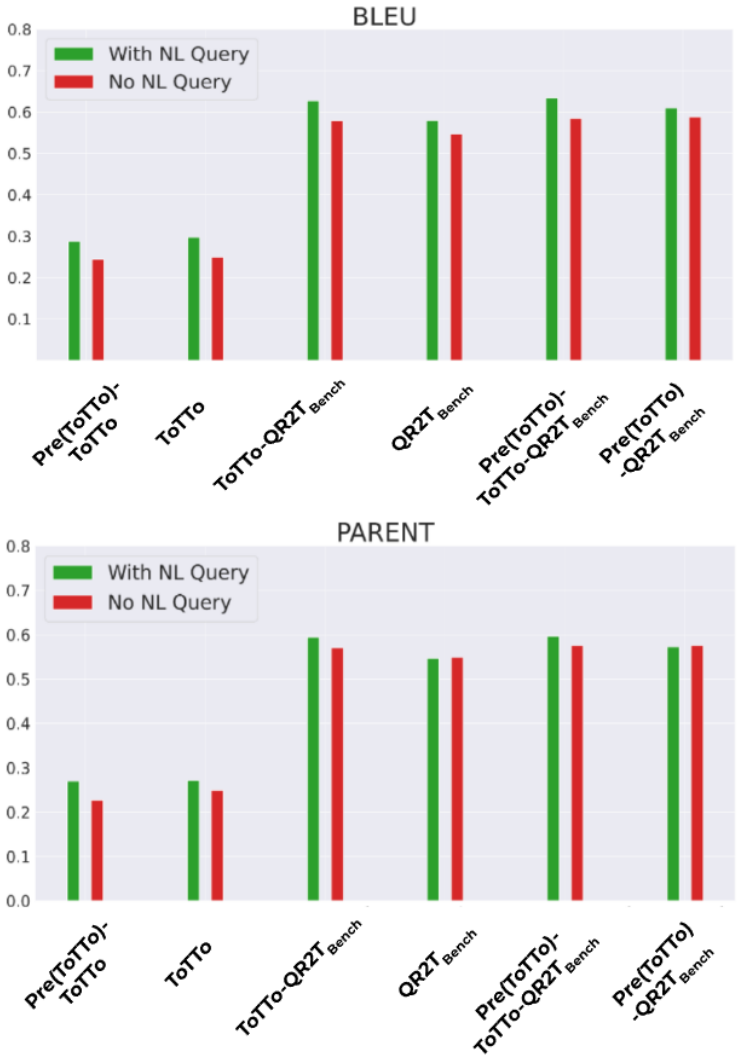

Figure 16: Impact of not having the NL query in the input of the model.

6. CONCLUSIONS

In this study, we presented Query Results-to-Text as a part of a natural language database interface. We formulate the problem and propose a pipeline solution that gets as input powerful PLMs that understand text and then guides them to also understand what table structure is through pre-training and pretraining tasks, and finally fine-tuning on our novel datasets, QR2T Benchmark and Cordis. We also propose a query preprocessing algorithm that leads to more informative and PLM ”friendly” query results that in turn enable the model to return more accurate and fluent verbalisations.

We evaluate our above pipeline and discover that the straightforward idea of directly fine-tuning T5 on our dataset will lead to adequate results which we then slightly improve through pre-training and pretraining tasks. Through human evaluation, we discover that QR2T performs adequately suggesting that it could be integrated into an NLDBI.

Future research could investigate Query Results-to-Text from different perspectives.

Improving the model itself by providing solutions to verbalise more than a single row through summarization and insight generation techniques instead of verbalising each row iteratively. In addition, future research could focus on guiding the verbalisation generation at inference time through a set of rules in order to force the model to remain faithful to the input query results table or the database schema, which is a similar approach to PICARD [34] but for the Text-to-SQL problem. Finally, proposing new metrics that are more representative of real performance with a possible solution combining both the table-aware PARENT [8] and the semantics-aware BERTscore [47].

ABBREVIATIONS - ACRONYMS

C4	Colossal Clean Crawled Corpus
DB	Database
CORDIS	Community Research and Development Information Service
e.g.	from the Latin phrase exempli gratia, meaning “for example.”
GPU	Graphical Processing Unit
i.e.	from the Latin phrase ”id est”, meaning “that is”
NL2SQL	Natural Language to SQL
OOV	Out Of Vocabulary
QR2T	Query Results to Text
SQL2NL	SQL to Natural Language
T5	Text-to-Text Transfer Transformer
WDC	Web Data Commons

REFERENCES

[1] Jonathan Berant, Daniel Deutch, Amir Globerson, Tova Milo, and Tomer Wolfson. Explaining queries over web tables to non-experts, 2018.

[2] Jonathan Berant, Daniel Deutch, Amir Globerson, Tova Milo, and Tomer Wolfson. Explaining queries over web tables to non-experts. *CoRR*, abs/1808.04614, 2018.

[3] Jonathan Berant, Daniel Deutch, Amir Globerson, Tova Milo, and Tomer Wolfson. Explaining queries over web tables to non-experts. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1570–1573. IEEE, 2019.

[4] Deng Cai and Wai Lam. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7464–7471, 2020.

[5] Zewen Chi, Li Dong, Shuming Ma, Shaohan Huang, Saksham Singhal, Xian-Ling Mao, Heyan Huang, Xia Song, and Furu Wei. mT6: Multilingual pretrained text-to-text transformer with translation pairs. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1671–1683, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[6] Daniel Deutch, Nave Frost, and Amir Gilad. Natural language explanations for query results. *ACM SIGMOD Record*, 47(1):42–49, 2018.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[8] Bhuwan Dhingra, Manaal Faruqui, Ankur P. Parikh, Ming-Wei Chang, Dipanjan Das, and William W. Cohen. Handling divergent reference texts when evaluating table-to-text generation. *CoRR*, abs/1906.01081, 2019.

[9] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. Creating training corpora for nlg micro-planning. In *55th annual meeting of the Association for Computational Linguistics (ACL)*, 2017.

[10] Heng Gong, Yawei Sun, Xiaocheng Feng, Bing Qin, Wei Bi, Xiaojiang Liu, and Ting Liu. TableGPT: Few-shot table-to-text generation with table structure reconstruction and content matching. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1978–1988, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.

[11] Mandy Guo, Joshua Ainslie, David C. Uthus, Santiago Ontañón, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. Longt5: Efficient text-to-text transformer for long sequences. *CoRR*, abs/2112.07916, 2021.

[12] Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. X-sql: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*, 2019.

[13] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, 2016.

[14] Mihir Kale. Text-to-text pre-training for data-to-text tasks. *CoRR*, abs/2005.10433, 2020.

[15] Rémi Lebret, David Grangier, and Michael Auli. Neural text generation from structured data with application to the biography domain. *arXiv preprint arXiv:1603.07771*, 2016.

[16] Hsin-Ying Lee, Jia-Bin Huang, Maneesh Singh, and Ming-Hsuan Yang. Unsupervised representation learning by sorting sequences. In *Proceedings of the IEEE international conference on computer vision*, pages 667–676, 2017.

[17] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, page 75–76, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee.

[18] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[19] Fei Li and Hosagrahar V Jagadish. Nalir: an interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 709–712, 2014.

[20] Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. Table-to-text generation by structure-aware seq2seq learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[21] Tianyu Liu, Wei Wei, and William Yang Wang. Table-to-text natural language generation with unseen schemas. *arXiv preprint arXiv:1911.03601*, 2019.

[22] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *CoRR*, abs/2001.08210, 2020.

[23] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[24] Sanket Vaibhav Mehta, Jinfeng Rao, Yi Tay, Mihir Kale, Ankur Parikh, Hongtao Zhong, and Emma Strubell. Improving compositional generalization with self-training for data-to-text generation. *arXiv preprint arXiv:2110.08467*, 2021.

[25] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI*, pages 69–84. Springer, 2016.

[26] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[27] Ankur P. Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. Totto: A controlled table-to-text generation dataset. *CoRR*, abs/2004.14373, 2020.

[28] Ratish Puduppully, Li Dong, and Mirella Lapata. Data-to-text generation with content selection and planning. *CoRR*, abs/1809.00582, 2018.

[29] Ratish Puduppully, Li Dong, and Mirella Lapata. Data-to-text generation with content selection and planning. *CoRR*, abs/1809.00582, 2018.

[30] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[32] Leonardo F. R. Ribeiro, Martin Schmitt, Hinrich Schütze, and Iryna Gurevych. Investigating pretrained language models for graph-to-text generation. *CoRR*, abs/2007.08426, 2020.

[33] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? *CoRR*, abs/2002.08910, 2020.

[34] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: parsing incrementally for constrained auto-regressive decoding from language models. *CoRR*, abs/2109.05093, 2021.

[35] Thibault Sellam, Dipanjan Das, and Ankur P Parikh. Bleurt: Learning robust metrics for text generation. *arXiv preprint arXiv:2004.04696*, 2020.

[36] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cícero Nogueira dos Santos, and Bing Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training. *CoRR*, abs/2012.10309, 2020.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[38] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. *CoRR*, abs/1911.04942, 2019.

[39] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. Dbpal: A fully pluggable nl2sql training pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2347–2361, 2020.

[40] Sam Wiseman, Stuart M Shieber, and Alexander M Rush. Challenges in data-to-document generation. *arXiv preprint arXiv:1707.08052*, 2017.

[41] Kuan Xu, Yongbo Wang, Yongliang Wang, Zujie Wen, and Yang Dong. Sead: End-to-end text-to-sql generation with schema-aware denoising. *arXiv preprint arXiv:2105.07911*, 2021.

[42] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. SQL-to-text generation with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 931–936, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[43] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. SQL-to-text generation with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 931–936, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[44] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. *CoRR*, abs/2005.08314, 2020.

[45] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. Grappa: Grammar-augmented pre-training for table semantic parsing, 2020.

[46] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[47] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

[48] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

[49] Jie Zhu, Junhui Li, Muhua Zhu, Longhua Qian, Min Zhang, and Guodong Zhou. Modeling graph structure in transformer for better amr-to-text generation. *CoRR*, abs/1909.00136, 2019.