Learning Relational Decomposition of Queries for Question Answering from Tables

Raphaël Mouravieff
Sorbonne Université,
CNRS, ISIR,
F-75005 Paris, France
`raphael.gervillie@isir.upmc.fr`

Benjamin Piwowarski
Sorbonne Université,
CNRS, ISIR,
F-75005 Paris, France
`benjamin.piwowarski@isir.upmc.fr`

Sylvain Lamprier
LERIA,
Université d'Angers,
France
`sylvain.lamprier@univ-angers.fr`

Abstract

Table Question-Answering involves both understanding the natural language query and grounding it in the context of the input table to extract relevant information. In this context, many methods have highlighted the benefits of intermediate pre-training using SQL queries. However, while most approaches aim at generating final answers directly from inputs, we claim that there is better to do with SQL queries during training. By learning to imitate a restricted subset of SQL-like algebraic operations, we demonstrate that their execution flow provides intermediate supervision steps that allow for increased generalization and structural reasoning compared to classical approaches. Our method, bridges the gap between semantic parsing and direct answering methods, offering valuable insights into which types of operations should be predicted by a generative architecture and which should be executed by an external algorithm. Our code can be found at `https://github.com/RaphaelMouravieff/Partial-Exec`.

1 Introduction

The field of Table Question Answering (QA), which encompasses complex content manipulation tasks like projection, sorting, grouping, and aggregation, presents considerable challenges for Natural Language Processing (NLP). Its complexity and growing relevance across diverse sectors, from business to academic research, have attracted widespread attention. This domain has evolved quickly with the rise of Pretrained Language Models (PLMs), but this field remains challenging for current models (Jin et al., 2022).

Former studies focused on Semantic Parsing (SP) techniques tailored for well-structured and clean table data, as highlighted in (Shi et al., 2020). However, real-world scenarios often involve heterogeneous resources, for example combining both numerical and textual content in some cells, like in WikiTableQuestions (Pasupat and Liang, 2015). Among the proposed solutions, (Liu et al., 2021) tried to *generate directly* the answer and therefore bypass the generation of logical forms. Despite this advantage, these methods exhibit limitations, particularly when executing numerical operations (e.g., computing a mean, counting). To cope with this, a natural solution is to propose hybrids that stand as intermediary solutions between semantic parsing and direct generation. For instance, (Herzig et al., 2020; Zhou et al., 2022b) have combined basic table selection methods (e.g., selecting rows and columns, or cells) before computing aggregations or performing basic numerical operations. However, they often fail to address intricate queries necessitating the synthesis of diverse table views and interactions because of the limited expressivity of their underlying algebra.

In this work, we propose to study the continuum between semantic parsing-based and direct generative methods, to leverage the strengths of both. Going beyond previous works, we propose a novel framework that facilitates reasoning over heterogeneous table resources. This framework relies on the definition of an algebra over tables inspired by relational algebra. Based on this algebra, each question in natural language and its corresponding table can be translated into a computational graph. By varying a cut-off criterion that specifies which part of the graph should be computed directly by the model (i.e. direct generation) and which part should be computed outside of it (i.e. semantic parsing), we can study different trade-offs and their effect in terms of effectiveness.

Beyond providing a stronger interpretation of the user query in the context of the table compared to semantic parsing-based (SP) methods, our framework addresses the common execution challenges associated with SP methods, which require clean tables to allow full SQL execution. Our approach predicts operators with associated "clean" operands from the input, thanks to the generative ability of the Transformer architectures.

To train our model, we leverage a pre-training procedure (Pruksachatkun et al., 2020; Geva et al., 2020; Yu et al., 2020) that helps neural architectures manipulate tabular data before dealing with complex Table QA tasks. This is done by first learning to generate from SQL queries rather than from natural language. On top of that, our main contribution focuses on the fine-tuning of such models when dealing with natural language questions, through the development of a flexible framework that allows partial execution of corresponding SQL queries, enabling effective handling of real-world tables while still obtaining good performance on numerical reasoning questions (Figure 1). Our research addresses the following key questions :

1. Is there an intermediate supervision for table QA that bridges the gap between semantic parsing and direct answer generation, thereby enhancing generalization and structural reasoning capabilities (Table 2)?

2. What types of algebraic operations should be predicted by a generative architecture, and which should be preferably executed by an external algorithm (Tables 3a and4a) ?

3. Does partial execution of SQL graphs help to improve robustness to perturbations in the input tables (Table 3b and 4b) ?

Through our experiments, we demonstrate that, for some intermediate cut-off levels, our approach generalizes better and is more robust compared to direct answer methods, which are usually limited in their structural reasoning capacities. This framework provides insights into the trade-off between different levels of supervision, paving the way for future research and innovation in the field of Table QA.

2 Related Work

2.1 Table Question Answering Architectures

Table question answering is a very active field with many recent developments. This ranges from specifically designed transformer architectures, with sparse (Eisenschlos et al., 2021) or biased (Golchin and Surdeanu, 2023) attention matrices that capture table structures, or specialized table embeddings as in TUTA (Wang et al., 2021) and GENTAP (Shi et al., 2022), to large language models (LLMs) that leverage in-context learning to deal with table structures (Chen, 2022; Cheng et al., 2022; Wang et al., 2024). While our study, orthogonal to these directions, could be applied in the context of any family of architectures including LLMs (e.g., fine-tuned using low-rank adaptation (Hu et al.; Dettmers et al.)), we chose to build on compact architectures based on reasonably-sized pre-trained language models (PLMs) such as BERT or BART, as considered in popular recent works TAPEX (Liu et al., 2021) or OmniTab (Jiang et al., 2022). Beyond scalability, such architectures, which do not require specific prompt design that could bias conclusions, offer easier comparison opportunities.[1] Finally, we believe that our developed approach, which consists of predicting and using external programs as tools when generating answers (e.g., in the vein of Toolformer (Schick et al.)), are still fully valuable in the context of LLMs, providing increased inference speed and stability. Our work is a step towards showing how tools can be used with structured data like tables, which can be transferred to LLMs in future works.

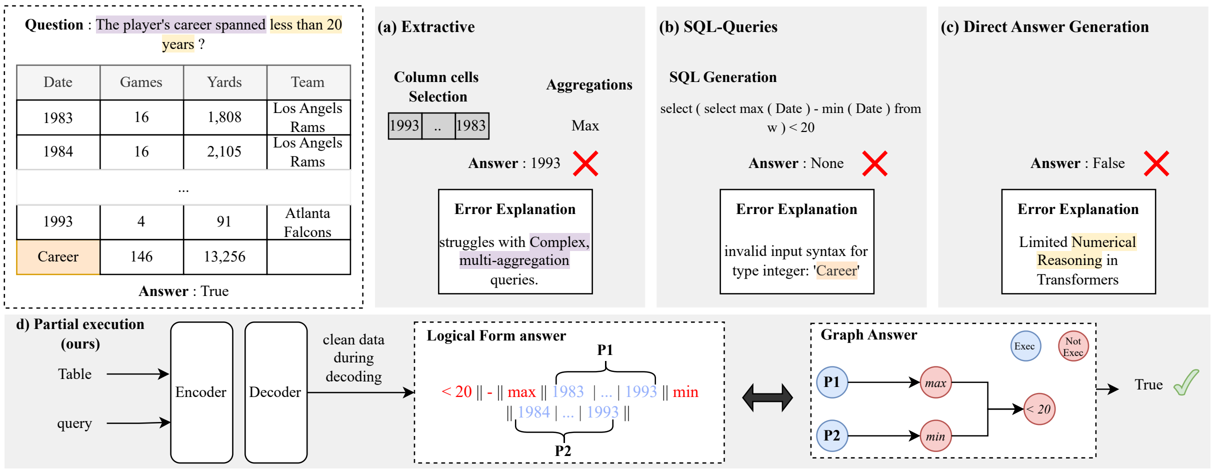
In the following, we focus on differences between table question answering approaches regarding their output strategies, which is more strongly related to the study of this paper.

2.2 Output Strategies in Table QA

Table Question-Answering models can be distinguished on their answer generation, which is either a formula operating on the table (semantic parsing) or a direct answer (direct generation), or a hybrid of both.

Semantic Parsing Semantic parsing aims to transform natural language into executable queries, primarily SQL. Sketch-based models decompose SQL query construction by breaking down and classifying query components, enhancing structured SQL generation (Jin et al., 2022). Generation-based methods like RAT-SQL (Wang et al., 2019) directly produce SQL queries using an encoder-decoder architecture that considers both the question and the table context for generation. Under weak supervision, (Min et al., 2019) optimize the probability of the correct answers over a set of possible latent representations, facilitating the model's

[1]We also note that it has recently been shown in a broader context that LLMs are usually contaminated by evaluation benchmarks (Golchin and Surdeanu, 2023), which could alter the results of our study.



Question : The player's career spanned less than 20 years ?
Date
Games
Yards
Team
1983
16
1,808
Los Angels Rams
1984
16
2,105
Los Angels Rams
...
1993
4
91
Atlanta Falcons
Career
146
13,256
Answer : True
(a) Extractive
Column cells Selection
Aggregations
1993 .. 1983
Max
Answer : 1993
Error Explanation
struggles with Complex, multi-aggregation queries.
(b) SQL-Queries
SQL Generation
select (select max (Date) - min (Date) from w) < 20
Answer : None
Error Explanation
invalid input syntax for type integer: 'Career'
(c) Direct Answer Generation
Answer : False
Error Explanation
Limited Numerical Reasoning in Transformers
d) Partial execution (ours)
Table
query
Encoder
Decoder
clean data during decoding
Logical Form answer
P1
< 20 || - || max || 1983 | ... | 1993 || min || 1984 | ... | 1993 ||
P2
Graph Answer
Exec
Not Exec
P1
P2
max
min
< 20
True

Figure 1: Overview of the different approaches for Table QA and their limits (a-c), along with our proposition (d)

ability to infer correct responses without explicit answer mappings. Another stategy is to use reinforcement learning where the execution result is used as rewards to train models (Zhong et al., 2017). Despite SQL's effectiveness in QA over tables (Shi et al., 2020), its limitations with non-database tables and question translation are a major drawback. Our approach seeks to transcend these bounds by introducing a logical form independent of the table during execution.

Direct Answer Generation In contrast to semantic parsing, direct answer generation produces final answers, bypassing the step of converting questions into formulas. This directly addresses the limitations of SQL-based systems, enabling the processing of various table formats. For instance, (Mueller et al., 2019) use a GNN-based encoder to encode the table structure and a decoder to output the answers conditioned on the graph and the query. An additional benefit of this method is its compatibility with advanced data augmentation techniques (Eisenschlos et al., 2020). This includes transformations from SQL to its result as in TAPEX (Liu et al., 2021), or from Excel formula to its execution as in FORTAP (Cheng et al., 2021). However, a notable challenge for transformers in this domain is handling numerical reasoning queries effectively (Zhou et al., 2022a).

Hybrid Methods Hybrid methods extract pertinent tokens from tables to create responses, typically employing an aggregator to associate with and route these tokens to a specifically designed executor. TAGOP (Zhu et al., 2021) uses sequence tagging for extracting relevant cells and a classifier for assembling them into coherent symbolic reasoning programs. TAPAS (Herzig et al., 2020) employs a classifier layer at the end of a BERT-like encoder for selecting content from tables and determining the aggregation operation to apply to it. These methods have good numerical abilities, but however, unlike other output strategies, they have limited expressiveness and struggle with complex multi-aggregation queries (Herzig et al., 2020). Our proposed supervision using intermediate logical form addresses this issue by enabling complex multi-aggregation representations.

3 Model

The goal of Table QA is to find the answer A given a natural language question q posed on a table T. We consider the setting where, at train time, queries in natural language are associated with corresponding SQL queries.

In this section, we describe 1) the manipulated data structures; 2) the algebraic operators that correspond to nodes of SQL execution graphs we consider; 3) our way to build intermediate supervision through partial execution of that graph; and 4) our way for linearizing the resulting graph to form the target of our transformer architecture.

3.1 Tabular algebra

In this section, we describe the algebra, inspired by the relational one (Codd), that we use to represent any operation on tables.

Structures Table Question Answering is the task of finding an answer A from a table $T \in \mathcal{T}$, where

Operation	Function Definition	Parameters	Description
Projection	$P : \mathcal{T} \to \mathcal{T}$	$J = \{c_i\}_{i \in 1 \ldots k}$	Extracts k columns from a table T, specified by their names $J \subseteq h_T$.
Comparison	$C : \mathcal{T} \cup \mathcal{G} \times \mathcal{T} \to \mathcal{B}$	$c \in \{>, <, ..\}$	Compares $T_1 \in \mathcal{T} \cup \mathcal{G}$ with $T_2 \in \mathcal{T}$ using c. T_2 either has the same number of rows as T_1 or only 1 that is broadcast to fit T_1.
Having	$H : \mathcal{G} \times B \to \mathcal{G}$	-	Selects from G where B is true, with $N^B_{rows} = N^G_{rows}$.
Group By	$GB : \mathcal{T} \to G$	$J = \{c_i\}_{i \in 1 \ldots k}$	Groups elements in T with equal values from columns in $J \subseteq h_T$.
Aggregation	$A : \mathcal{T} \cup \mathcal{G} \to \mathcal{T}$	$f \in \{sum, avg, ..\}$	Aggregates T using function f.
Operator	$OP : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$	$o \in \{+, -, *, ..\}$	Performs the term-wise operation o on two tables $T_1 \in \mathcal{T}$ and $T_2 \in \mathcal{T}$.
Order By	$OB : \mathcal{T} \to \mathcal{T}$	$d \in \{asc, desc\}$	Orders table T by criterion with direction d.
Limit	$L : \mathcal{T} \to \mathcal{T}$	$k \in \mathbb{N}$	Selects top k elements from T.
Selection	$S : \mathcal{T} \times B \to \mathcal{T}$	-	Selects from T where B is true, with $N^B_{rows} = N^T_{rows}$.

Table 1: Algebra to manipulate tabular data. See section 3.1 for notations.

$T = \left((x_{r,c})_{c=1 \ldots N^T_{col}} \right)_{r=1 \ldots N^T_{row}}$ is a matrix of values $x_{r,c}$, which can be numbers or strings. Differently from relational algebra, we view tabular data as a sequence of tuples which we suppose to be *ordered*. A table can have a header, which corresponds to a sequence of column names $c_1 \ldots c_{N^T_{col}}$. When no header is given, each c_i corresponds to the column index, and $h_T = \{c_i\}_{i=1}^{N^T_{col}}$ stands as the set of column names from T. Views on the original table, that correspond to results from algebraic operations, are also considered as table $T \in \mathcal{T}$.

Classically, tables only include atomic values. To cope with set aggregations (i.e., involving a group-by operation), we also manipulate group-by tables $G \in \mathcal{G}$, where $G = \left((g_{r,c})_{c=1 \ldots N^G_{col}} \right)_{r=1 \ldots N^G_{row}}$, with each component $g_{r,c}$ corresponding to a set of values. We also note columns boolean matrices as $B = (b_{r,1}))_{r=1 \ldots N^B_{row}}$, with $b_{r,1} \in \{0, 1\}$.

Operators Table 1 describes the different operators that we use to manipulate tables T or group-by tables G, whose behavior can be conditioned on parameters (e.g. “order by” can be ascending or descending). These operators follow roughly standard relational algebra operators and cover a broad range of SQL queries. A notable difference with classical relational algebra, which was dictated by the fact we want to further decompose operations for analysis purposes, is the fact that the selection operation simply corresponds to a filter given a column of boolean values produced by a separated comparison operator and that the order of tuples is used for comparisons (e.g. >, <) and operations (e.g. +, -).

Translating from SQL to our algebra is straightforward. We rely on the SQLGlot library[2] to obtain a parse tree from any SQL query. This parse tree is then translated into a computational graph. Each node n of this graph is denoted as $\phi\left(x_n, [n_1, \ldots, n_K]\right)$ where x_n is either a table in $\mathcal{T}$, a group-by table in $\mathcal{G}$ or an operator in $\mathcal{O}$ (an operator is both the operation, e.g. "limit", and its parameters, e.g. k). In the case of operators, $n_1, \ldots, n_K$ correspond to the arguments of the operators, i.e. other nodes in the computation graph corresponding to its operands, and $x_n(.)$ the application of the operator on the corresponding list of child nodes. By abuse of notation, in the following we note $n = \phi\left(x_n, [n_1, \ldots, n_K]\right) \in \mathcal{X}$, with $\mathcal{X}$ a given set, to denote $x_n \in \mathcal{X}$.

3.2 Partial Execution of the computational graph

Now that we have defined the data and the algebra, we can present how this can be leveraged to produce various representations. For this, we rely on a graph transduction function v operating recursively on any node n of the graph. That is, given a set of operators $\mathcal{O}^*$ we allow to be executed, $v(n) = \phi(x_n(v(n_1), \ldots, v(n_K)))$ if $x_n \in \mathcal{O}^* \wedge \forall i \in 1 \ldots K, v(n_i) \in \mathcal{T} \cup \mathcal{G}$, and $v(n) = n$ otherwise. In other words, we execute from any leaf to the root of the computational tree every allowed operation in $\mathcal{O}^*$ until execution is blocked (because x_n not in $\mathcal{O}^*$ or one of its depen-

[2]https://github.com/tobymao/sqlglot

dencies cannot be executed).

The computation graph can hence be partially executed through this transformation v, allowing for flexible handling of SQL operations, by applying v on the root node.

3.3 Linearizing the Graph

As the computational graph must be generated sequentially, we need to define how to transform it into a sequence of tokens, i.e. how to *linearize* it. To do so, we use a linearization function that we denote l, which takes a node $n = \phi(x_n, n_1, \dots, n_K)$ in input and returns a sequence of tokens.

In the case of tables (i.e., when $x_n \in \mathcal{T} \cup \mathcal{G}$), we use a simple markup where we separate rows with the symbol "`|`" and columns with a comma "`,`". In the case of operators, i.e. $x_n \in \mathcal{O}$, the linearization l corresponds to the name of the operator followed by its parameters. For instance, the sequence `LIMIT 1` corresponds to the limit operator with 1 as its parameter.

For operator nodes, we define various linearization schemes depending on the order (pre- or post-order) and the usage of aliases to avoid duplicating the same information (the graph is a directed acyclic graph, but there can be different paths between two nodes since results might be re-used).

Pre-order vs post-order We can either use a pre-order linearization scheme where the operator appears before its operands: $l_{pre}(n) = l(x_n) \oplus \oplus_i(|| \oplus l_{pre}(n_i))$ or a post-order one: $l_{post}(n) = \oplus_i(|| \oplus l_{post}(n_i)) \oplus || \oplus l(x_n)$. In both cases, '$||$'denotes a separator token and $\oplus$ concatenation.

Using aliases In the above linearizations, re-used results will be linearized several times. This happens frequently with queries with some aggregation. The problem is that this can result in longer sequences, which in turn might be harder to generate. To tame this problem, we associate each node with a given alias the first time it is linearized (e.g. `N13`) and use this reference instead of its linearization in subsequent occurrences (see appendix A.1.1 for details).

Finally, tables are linearized either before or after the operators. After some preliminary experiments, we chose this to make the grammar of the sequence more regular for a transformer (not mixing operators and content).

4 Experiments

4.1 Dataset and Evaluation Metrics

In our experiments, we used the **WikiTableQuestions** (WTQ) dataset (Pasupat and Liang, 2015), which fulfills all predefined criteria for our study: it is characterized by its provision of complex numerical reasoning questions, tables with missing information, mixed cell types (e.g. text and numbers), and availability of SQL supervision. The SQL annotations supplied by **SQuALL** (Shi et al., 2020) enable the coverage of approximately 80% of the questions from WTQ in the training and validation sets only. We also experimented with the **WikiSQL** (Zhong et al., 2017) dataset. As with WikiTableQuestions, WikiSQL provides the necessary supervision for training our model. However, unlike WikiTableQuestions, the questions are much simpler (no order-by, group-by, or arithmetic operators), and the tables are already clean.

Results are reported using the Denotation Accuracy (DA) metric as our primary evaluation criterion. DA checks if the execution of the predicted answer is equal to the target answer. When the answer is a list of results, DA disregards the order (i.e., set equality). We decomposed this metric into two categories: the **Strict Denotation Accuracy (SDA)**, which is the traditional one used, and the **Flexible Denotation Accuracy (FDA)**, which compares results after removing units (years, \$, kg, etc.). The choice to employ both SDA and FDA stems from our dependence on external tools' APIs for execution. As a result, our execution outcomes are unit-less, and using SDA would hide the improvements brought by our model – note that we could extend our method to generate an arbitrary sentence containing the result in future works.

4.2 Inputs and outputs

The query encoding is straightforward but table encoding presents a challenge due to its inherent structure. We follow TAPEX and OmniTAB (Liu et al., 2021; Jiang et al., 2022), and represent the transformed table as T^* = [HEAD], c_1, ..., c_N, [ROW], 1, r_1, [ROW], 2, r_2, ..., r_M. The tokens `[HEAD]` and `[ROW]` delimit the table's header and row sections, respectively, with subsequent numbers indicating row indices. Additionally, we use a vertical bar `|` to delineate headers or cells in separate columns. We then concatenate the query with the linearized table as the input to the encoder.

Outputs in our model correspond to linearized computational graphs. We considered 42 experimental conditions. First, we use one of the following seven sets of operators as $\mathcal{O}^*$: **(P)** Only projection operators; **(+C)** P with comparison operators; **(+S)** +C with selection operators; **(+GB+H)** +S with group-by and having; **(+A)** +GB+H with aggregations; **(+OP)** +A with operators; **(Full)** with all operators, i.e. as TAPEX (Liu et al., 2021). Second, we used six possible linearizations: pre-order, post-order, and pre/post-order-alias-start/end. Examples of different linearizations, with different partial executions, are given in the Appendix A.1, tables 6, 7, 8 and 9.

4.3 Training pipeline

Our training methodology employs a sequence-to-sequence (seq-2-seq) framework, utilizing BART as the fondational architecture (Lewis et al., 2019). We initialize our parameters using the TAPEX (Liu et al., 2021) checkpoint, as preliminary experiments have demonstrated improved performance. Subsequently, we fine-tune the model using natural language questions, replacing the SQL queries used in the pre-training. Our additional hyperparameters correspond solely to the choice of operators in $\mathcal{O}^*$, as discussed below.

4.4 Overall performance

In this section, we compare our model with the state-of-the-art ones, on the test split of the WTQ Dataset. Results are shown in Table 2, distinguishing between those employing fine-tuning techniques from BART-like architectures, and those considering in-context learning of LLMs, using specific prompting strategies. We report SDA as well as FDA for the model for which we reproduced the results. We report in Table 2 the results of the best-performing set of operators we experimented, namely $\mathcal{O}^* = \{P, C, S\}$, as well as our ensemble model that leverages various granularities $\mathcal{O}^*$.

The semantic parsing baseline SQuALL does not perform well, especially if tables are not manually cleaned up (dropping from 54.3% to 27.2% for FDA), while other methods do not require this costly cleaning step. Our models showcase notable achievements, with our best one (selected on the validation set) reaching an FDA of 61.4%. This is comparable to OmniTab which relies on sophisticated data augmentation techniques. Our performance can even increase to 66.3% when leveraging ensemble methods (see Section 4.7). We also show later that besides obtaining state-of-the-art results

Table 2: Comparison of Model Performance on WikiTableQuestions test data

Model	**SDA**	**FDA**
Fine-Tuned Models		
TABERT (Yin et al., 2020)	52.3	-
MATE (Eisenschlos et al., 2021)	51.5	-
TableFormer (Yang et al., 2022)	52.6	-
GRAPPA (Yu et al., 2020)	52.7	-
DoT (Krichene et al., 2021)	54.0	-
REASTAP (Zhao et al., 2022)	58.6	-
TaCube (Zhou et al., 2022a)	60.8	-
TAPAS (Herzig et al., 2020)	48.8	50.2
TAPEX (Liu et al., 2021)	55.5	57.9
OmniTab (Jiang et al., 2022)	61.8	62.1
Prompt-based LLMs		
ChatGPT (Cheng et al., 2022)	43.3	-
Codex (Ye et al., 2023)	47.6	-
StructGPT (Cheng et al., 2022)	48.4	-
Codex-COT (Chen, 2022)	48.8	-
Binder (Cheng et al., 2022)	64.6	-
LEVER (Cheng et al., 2022)	65.8	-
DATER (Cheng et al., 2022)	65.9	-
Chain-of-Table (Wang et al., 2024)	67.3	-
Semantic parsing with cleaned tables		
SQuALL (Shi et al., 2020)	50.4	54.3
Semantic parsing		
SQuALL (Shi et al., 2020)	23.2	27.2
Our models		
+P+C+S	59.0	61.4
Ensemble	63.3	66.3

(for similarly sized architectures), our models are also more robust (Table 3b).

Notably, prompting approaches based on LLMs, including the chain-of-thought method (Wei et al., 2022), demonstrate superior performance without necessitating model adaptation. It is important to note that (1) these approaches rely on much more parameters (GPT-3.5 has 175 billion parameters (Brown et al., 2020)), (2) our training approach could also be used to fine-tune LLMs, and (3) this paper contributes to the ongoing discussion in the community about leveraging external tools vs. relying solely on model capabilities. Recent studies, which highlight difficulties encountered by GPT-4 for counting rows of tables, are in line with this argument (Sui et al., 2024).

4.5 Sensitivity over questions types

In table 3a and 4a, we show the performance for different query types, distinguished by whether they contain operators such as Projection, Comparison, Selection, Group By, Order By, Aggregation, Operator, and Limit. It is important to note that queries containing a group-by are limited, so results reported in this column should be interpreted cautiously.

Among existing models, Omnitab slightly outperforms our model in some scenarios, showing the importance of its data augmentation techniques compared to Tapex. Tapas does perform worse on these query types, which shows the limits of its aggregation methodology based on column/row selection.

Among our models, PCS exhibits the best overall performance (as on the test set), thanks to its robust handling of query types. However, surprisingly, it performs worse on group-by queries compared to models that include GB in $\mathcal{O}^*$. We suppose that this might be due to the variance due to the limited number of queries of that type. Finally, our model exhibits a pattern where simpler operators (projection, comparison, selection) are better handled when generated directly, while others (order by, aggregation, operators) benefit from being executed externally.

Finally, Table 5 presents the performance of models based on the complexity of the query, as measured by the number of operators in the original computational graph. OmniTab and our models (especially +GB+H) demonstrate resilience with relatively stable performance across various operation ranges. Tapex and Tapas, however, show a decline in performance as complexity increases, with Tapas notably struggling in the 8+ operation category, highlighting the limit of extractive methods.

4.6 Comparing linearization methods

In figure 2, we show the impact of linearization on the performance of the models. We can first observe that differences between our model variants decrease as most of the computational graph is executed, which was expected. Contrary to our expectations, however, using aliases has a negative impact, especially when they are more frequently used (+P to +P+C+S), which shows that having too many aliases is problematic when generating a relational formula. When using aliases, putting the tables after the operators did somehow improve the results. We think that these results might change with better training procedures (e.g., data augmentation with perturbations): we observed that models using aliases were more robust, but their overall performance was nevertheless below that of non-aliases ones. Finally, we observe that there is a granularity level (+P+C+S) that achieves the best performance, corresponding to cases where only basic table selection is performed; moreover, this level is less prone to overfitting as discussed in Section 4.8.

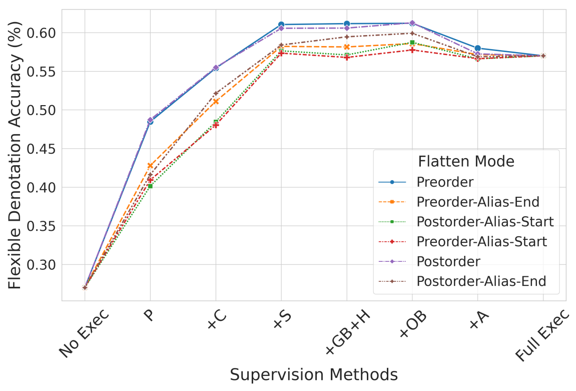


Figure 2: Evolution of FDA (test set) for different model variants.

4.7 Ensembling

Figure 3 illustrates the results that we obtained using different ensembling combinations. The ensemble prediction is given by a majority vote. In case of ties, we use the validation FDA to weight the votes. We experimented with two ensembling settings: going from semantic parsing models to full execution, or in the opposite direction, i.e. from full execution to semantic parsing.

First, performance improves whatever the ensembling method. This improvement can be explained with the analysis presented in Table **??**, where we analyzed the performance depending on the operators composing the computational graph. While certain models excel in specific types of operations, others may show superiority in different areas. Such diversity among the models is important for ensembling.

4.8 Sensitivity over table column cells perturbations

The Transformers architecture can easily overfit, especially in the case of a dataset like WTQ. To measure the importance of overfitting, we use the

Table 3: Performance (FDA) of models on the validation set of WikiTableQuestions, grouping results per type of query, for the models based on pre-order linearization (no alias). The column ALL reports FDA averaged over validation queries. The best results are in bold.

Model	**Projection (ALL)**	**Comparison**	**Selection**	**Group By**	**Order By**	**Aggregation**	**Operator**	**Limit**	σ
#	500	367	363	30	151	206	75	153	
Tapas	52.6	51.8	52.3	16.7	53.0	43.7	30.7	52.3	13.5
Tapex	55.2	55.9	56.5	50.0	60.9	38.8	44.0	60.8	7.9
Omnitab	**58.8**	**59.7**	**59.8**	**56.7**	61.6	47.1	45.3	60.8	6.4
P	44.6	40.9	41.3	40.0	49.7	43.7	28.0	49.0	6.8
+C	51.6	50.1	50.7	23.3	48.3	50.0	38.7	47.7	9.7
+S	58.6	58.0	58.4	40.0	58.3	**52.4**	**52.0**	57.5	6.4
+GB+H	57.8	57.8	58.4	23.3	57.0	49.5	49.3	56.2	11.8
+OB	57.6	57.5	57.8	53.3	58.9	51.5	50.7	58.2	3.3
+A	58.0	57.8	58.4	**56.7**	**62.2**	47.1	49.3	**61.4**	5.4
+OP	56.6	57.8	58.4	50.0	60.3	46.1	42.7	60.1	6.8

(a) Using validation data – the row # contains the number of matching queries (see Section 4.5)

Model	**Projection (ALL)**	**Comparison**	**Selection**	**Group By**	**Order By**	**Aggregation**	**Operator**	**Limit**	σ
Tapas	42.6	41.7	42.2	16.7	38.4	37.9	18.7	37.9	10.6
Tapex	43.4	43.0	43.5	43.3	44.4	35.4	29.3	44.4	5.5
Omnitab	45.4	44.7	44.9	36.7	42.4	39.3	30.7	42.5	5.1
P	43.2	39.8	40.2	36.7	45.0	44.2	28.0	44.4	5.7
+C	49.0	46.3	46.8	23.3	45.7	48.5	38.7	45.1	8.5
+S	**53.6**	**51.0**	**51.2**	40.0	**49.0**	**51.9**	**50.7**	**48.4**	4.2
+GB+H	51.6	49.6	50.1	23.3	45.0	49.5	48.0	44.4	9.2
+OB	50.6	50.7	51.2	40.0	43.7	48.1	46.7	43.1	4.1
+A	47.2	46.0	46.6	**50.0**	45.7	41.8	40.0	45.1	3.1
+OP	47.8	47.7	48.2	**50.0**	45.7	43.2	30.7	45.8	6.1

(b) Using validation data with *random permutations* of each column (see Section 4.8)

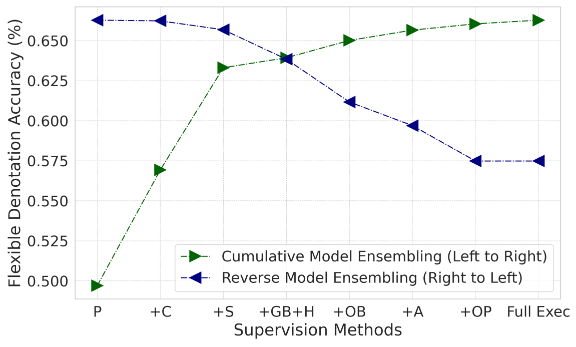


Figure 3: Evolution of FDA (test set) depending on the number of model variations in the ensemble. We either add models to the ensemble starting from the left (green) or the right (blue).

validation set (since the test set has no associated SQL queries) and perform random perturbations, i.e. we permute rows within each column. To avoid problems related to the maximum length of the input, we ensure these perturbations only affect the parts present in the input of the transformer – all models would have been affected, and this would have reduced the sensibility of our measures.

Results are shown in Table 3b using a pre-order (no alias) linearization (our best linearization method). We observe that perturbation strongly affects even the best-performing approaches on WTQ, as OmniTAB performance lowers from 58.8 to 45.4 (-13.4), Tapex from 55.2 to 43.4 (-11.8), and Tapas from 52.6 to 42.6 (-10.0). Our models are much less impacted. For instance, our best-performing approach (PCS) decreases its performance from 58.6 to 53.6 (-5.0), and surpasses the best baseline, Omnitab, by a large margin (53.6 vs 45.4), showing that data augmentation is less effective in preventing overfitting than generating formulas combining content and relational operators.

Among our models, we note that the lesser the amount of executed parts in the computation graph, the lower the decrease. As some models were initially more performant than others, we can note that the "P+C+S" model is the most effective one, with an average FDA of 53.6. Finally, we can see that the impact on some operators (e.g. group by, limit, comparisons) is even higher for models where most

Table 4: Performance (FDA) of models on the validation set of WikiSQL, grouping results per type of query, for the models based on pre-order linearization (no alias). The column ALL reports FDA averaged over validation queries. Best results are in bold.

Model	Projection (ALL)	Comparison	Selection	Aggregation
#	8419	8355	8355	2404
Tapex	86.8	86.8	86.8	81.4
P	76.4	76.2	76.2	73.4
+C	88.2	88.2	88.2	82.4
+S	**89.4**	**89.4**	**89.4**	**83.0**
+GB+H	89.4	89.4	89.4	83.0
+OB	89.4	89.4	89.4	83.0
+A	89.2	89.3	89.3	82.2
+OP	89.2	89.3	89.3	82.2

(a) Using validation data – the row # contains the number of matching queries (see Section 4.5)

Model	Projection (ALL)	Comparison	Selection	Aggregation
Tapex	77.3	77.3	77.3	76.4
P	76.1	76.0	76.0	73.3
+C	**86.7**	**86.6**	**86.6**	**80.0**
+S	79.5	79.4	79.4	77.6
+GB+H	79.5	79.4	79.4	77.6
+OB	79.5	79.4	79.4	77.6
+A	79.3	79.4	79.4	76.7
+OP	79.3	79.4	79.4	76.7

(b) Using validation data with *random permutations* of each column (see Section 4.8)

or all of the computational graph is executed. On WikiSQL (Table 4b), we still observe some improvements over methods as Tapex, especially after perturbing the validation tables, showcasing an improved robustness even in such simpler settings.

5 Conclusion

We explored the realm between semantic parsing and direct output generation for table QA, showing that PLMs can leverage an appropriate level of granularity where basic table manipulations (cleaning, selection) can be handled by the transformer itself while higher-level operations (e.g. aggregation, arithmetic) are better handled by dedicated tools. We showed that a model, appropriately trained, achieves high performance compared to state-of-the-art, and that, more importantly, most PLM baselines are prone to overfitting (by using a simple permutation of table cells), while our method is much less affected and outperforms the best baseline, OmniTab, by a wide margin. Future works will include more sophisticated training procedures, a sparse attention mechanism to cope with long tables, and more in-depth error analysis.

Table 5: Performance (FDA) with respect to the number of operators

Model	1-4	4-8	8+
Tapex	65.5	44.3	55.2
Tapas	66.5	49.0	32.4
Omnitab	65.0	**54.2**	55.2
+P	53.2	42.2	32.4
+C	61.1	46.4	42.9
+S	67.0	53.7	51.4
+GB+H	**67.5**	49.5	54.3
+OB	63.6	52.6	55.2
+A	65.0	53.1	**57.1**
+OP	63.1	50.0	56.2

6 Limitations & Risks

Our models have not been trained with data augmentation, which would help make them more robust – even if other models could benefit from it (e.g., Tapex or Tapas), we hypothesize that it would have an even bigger impact on our model. The best baseline, OmniTab, was already trained with augmented data. Experimenting with more datasets would also have strengthened our results. However, as for all works on Table QA, WikiTableQuestions is still a resource of reference.

We did not compare our results thoroughly with LLMs but did report the results from the original papers. However, the gap between the best-performing LLMs and our model is not that high, showing the potential benefit of using partially executed formulas. Future works could include the fine-tuning of LLMs with our proposed supervision.

Our methodology requires SQL annotations. In the case of datasets without any SQL supervision, but with questions in natural language, a simple approach would be to rely on a weak/distant supervision method, i.e., using semantic parsing models' predictions as SQL annotations.

Risks involved in this research are similar to those arising from any NLP research, as an automatic understanding of data can be used maliciously, e.g., leaking confidential information from tables. However, this work focuses on an exploratory study of learning abilities, which is dedicated to the scientific community only.

7 Acknowledgments

This work was partly funded by the ANR-21-CE23-0007 ACDC project. Experiments were performed using HPC resources from GENCI-

IDRIS (Grant AD011014032R1, AD011014110 and A0151014638).

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Wenhu Chen. 2022. Large language models are few (1)-shot table reasoners. *arXiv preprint arXiv:2210.06710*.

Zhoujun Cheng, Haoyu Dong, Ran Jia, Pengfei Wu, Shi Han, Fan Cheng, and Dongmei Zhang. 2021. Fortap: Using formulas for numerical-reasoning-aware table pretraining. *arXiv preprint arXiv:2109.07323*.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*.

E F Codd. A Relational Model of Data for Large Shared Data Banks. 13(6).

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs.

Julian Martin Eisenschlos, Maharshi Gor, Thomas Müller, and William W Cohen. 2021. Mate: multi-view attention for table transformer efficiency. *arXiv preprint arXiv:2109.04312*.

Julian Martin Eisenschlos, Syrine Krichene, and Thomas Müller. 2020. Understanding tables with intermediate pre-training. *arXiv preprint arXiv:2010.00571*.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. *arXiv preprint arXiv:2004.04487*.

Shahriar Golchin and Mihai Surdeanu. 2023. Time travel in llms: Tracing data contamination in large language models. *arXiv preprint arXiv:2308.08493*.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. Omnitab: Pre-training with natural and synthetic data for few-shot table-based question answering. *arXiv preprint arXiv:2207.03637*.

Nengzheng Jin, Joanna Siebert, Dongfang Li, and Qingcai Chen. 2022. A survey on table question answering: recent advances. In *China Conference on Knowledge Graph and Semantic Computing*, pages 174–186. Springer.

Syrine Krichene, Thomas Müller, and Julian Martin Eisenschlos. 2021. Dot: An efficient double transformer for nlp tasks with tables. *arXiv preprint arXiv:2106.00479*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653*.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A discrete hard em approach for weakly supervised question answering. *arXiv preprint arXiv:1909.04849*.

Thomas Mueller, Francesco Piccinno, Massimo Nicosia, Peter Shaw, and Yasemin Altun. 2019. Answering conversational questions on structured data without logical forms. *arXiv preprint arXiv:1908.11787*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel R Bowman. 2020. Intermediate-task transfer learning with pretrained models for natural language understanding: When and why does it work? *arXiv preprint arXiv:2005.00628*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools.

Peng Shi, Patrick Ng, Feng Nan, Henghui Zhu, Jun Wang, Jiarong Jiang, Alexander Hanbo Li, Rishav Chakravarti, Donald Weidner, Bing Xiang, et al. 2022. Generation-focused table-based intermediate pre-training for free-form question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11312–11320.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to sql queries. *arXiv preprint arXiv:2010.11246*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data?

a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 2022. Tableformer: Robust transformer modeling for table-text encoding. *arXiv preprint arXiv:2203.00274*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. *arXiv preprint arXiv:2301.13808*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845*.

Yilun Zhao, Linyong Nan, Zhenting Qi, Rui Zhang, and Dragomir Radev. 2022. Reastap: Injecting table reasoning skills during pre-training via synthetic reasoning examples. *arXiv preprint arXiv:2210.12374*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Fan Zhou, Mengkang Hu, Haoyu Dong, Zhoujun Cheng, Shi Han, and Dongmei Zhang. 2022a. Tacube: Pre-computing data cubes for answering numerical-reasoning questions over tabular data. *arXiv preprint arXiv:2205.12682*.

Yongwei Zhou, Junwei Bao, Chaoqun Duan, Youzheng Wu, Xiaodong He, and Tiejun Zhao. 2022b. Unirpg: Unified discrete reasoning over table and text as program generation. *arXiv preprint arXiv:2210.08249*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *arXiv preprint arXiv:2105.07624*.

A Appendix

A.1 Linearization

A.1.1 Using aliases

In the case of pre-order, we denote the alias for node n with α_n and use the following linearization:

$$l_{pre}(n) = \begin{cases} l(x_n) \oplus_i \alpha_{n_i} \oplus \alpha_n \oplus_i l^c_{pre}(n_i) \\ \qquad \text{if } x \in \mathcal{O} \\ \varnothing \qquad \text{else} \end{cases}$$

where $\varnothing$ denotes the empty sequence and l^c is either empty – if the operator has already been linearized – or $|| \oplus l_{pre}(n_i)$ if not. Note that the order of linearization is important, but to avoid more complicated notations we do not make it explicit here.

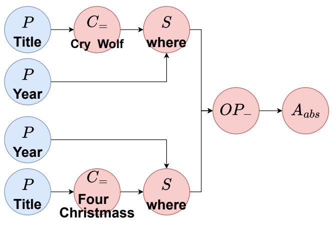
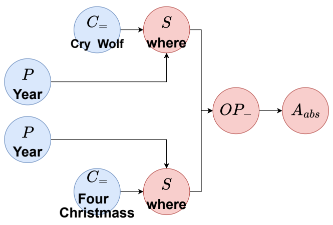
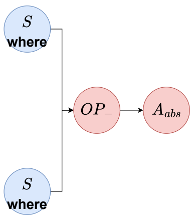
$\mathcal{O}^*$	Logical Form	Graph
{P}	abs ‖ - ‖ where ‖ 2005 | 2005 | .. ‖ = 'cry wolf' ‖ new york doll | cry wolf | .. ‖ where ‖ 2005 | 2005 | .. ‖ = 'four christmases' ‖ new york doll | cry wolf | .. ‖	 P Title C= Cry Wolf S where P Year P Year P Title C= Four Christmass S where OP− Aabs
{P, C}	abs ‖ - ‖ where ‖ 2005 | 2005 | .. ‖ f | t | .. ‖ where ‖ 2005 | 2005 | .. ‖ f | f | .. ‖	 C= Cry Wolf S where P Year P Year C= Four Christmass S where OP− Aabs
{P, C, S}	abs ‖ - ‖ 2005 ‖ 2008 ‖	
Full	3	

Table 6: Example of **Pre-order** linearization. The passage between the logical form and the graph is equivalent, in red the non executed node and in blue the executed ones. Omega control the executed nodes.
SQL : "**SELECT abs ((SELECT Year FROM w WHERE Title = 'cry wolf') - (SELECT Year FROM w WHERE Title = 'four christmases'))**".
Natural Language question :"**What is the difference in years between cry wolf and four christmases ?**".

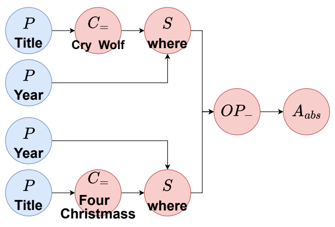
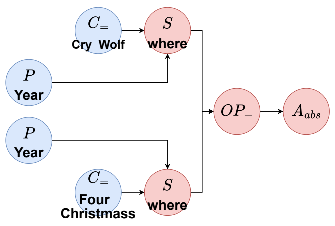
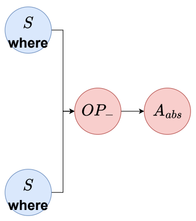
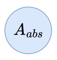
$\mathcal{O}^*$	Logical Form	Graph
{P}	new york doll | cry wolf | .. || = 'four christmases' || 2005 | 2005 | .. || where || new york doll | cry wolf | .. || = 'cry wolf' || 2005 | 2005 | .. || red || - || abs ||	
{P, C}	f | f | .. || 2005 | 2005 | .. || where || f | t | .. || 2005 | 2005 | .. || where || - || abs ||	
{P, C, S}	2008 || 2005 || - || abs ||	
Full	3	

Table 7: Example of **Post-order** linearization. The passage between the logical form and the graph is equivalent, in red the non executed node and in blue the executed ones. Omega control the executed nodes.
SQL : "**SELECT abs ((SELECT Year FROM w WHERE Title = 'cry wolf') - (SELECT Year FROM w WHERE Title = 'four christmases'))**".
Natural Language question :"**What is the difference in years between cry wolf and four christmases ?**".

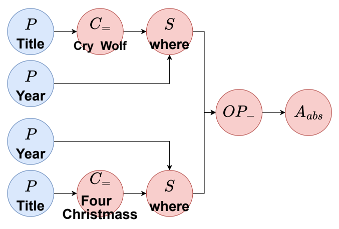
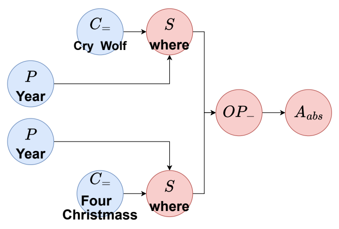
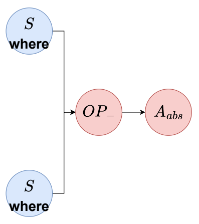
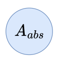
$\mathcal{O}^*$	Logical Form	Graph
{P}	N16 abs N25 || N25 - N9 N3 || N9 where N4 N23 || N4 = ’cry wolf’ N19 || N3 where N28 N23 || N28 = ’four christmases’ N19 ||| N23 2005 | 2005 | .. || N19 new york doll | cry wolf | .. ||	 P Title $C_=$ Cry Wolf S where P Year P Year P Title $C_=$ Four Christmass S where OP_- A_{abs}
{P, C}	N28 abs N25 || N25 - N38 N20 || N38 where N13 N18 || N20 where N1 N18 ||| N18 2005 | 2005 | .. || N13 f | t | .. || N1 f | .. ||	 $C_=$ Cry Wolf S where P Year P Year $C_=$ Four Christmass S where OP_- A_{abs}
{P, C, S}	N37 abs N7 || N7 - N2 N23 ||| N2 2005 || N23 2008 ||	
Full	3	

Table 8: Example of **Pre-order-Alias-End** linearization. The passage between the logical form and the graph is equivalent, in red the non executed node and in blue the executed ones. Omega control the executed nodes.
SQL : "**SELECT abs ((SELECT Year FROM w WHERE Title = ’cry wolf’) - (SELECT Year FROM w WHERE Title = ’four christmases’))**".
Natural Language question :"**What is the difference in years between cry wolf and four christmases ?**".

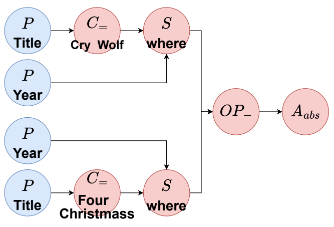
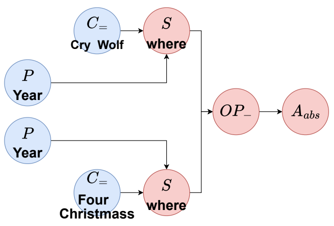
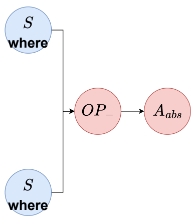
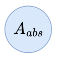
$\mathcal{O}^*$	Logical Form	Graph
{P}	N11 2005 | 2005 | .. || N22 new york dolll cry wolf | .. ||| N20 abs N30 || N30 - N4 N36 || N4 where N25 N11 || N25 = 'cry wolf' N22 || N36 where N10 N11 || N10 = 'four christmases' N22 ||	 P Title $C_=$ Cry Wolf S where P Year OP_- A_{abs} P Year P Title $C_=$ Four Christmass S where
{P, C}	N2 2005 | 2005 | .. || N34 f | t | .. || N38 f | f | .. ||| N6 abs N3 || N3 - N7 N11 || N7 where N34 N2 || N11 where N38 N2 ||	 $C_=$ Cry Wolf S where P Year OP_- A_{abs} P Year $C_=$ Four Christmass S where
{P, C, S}	N31 2005 || N29 2008 ||| N33 abs N26 || N26 - N31 N29 ||	
Full	3	

Table 9: Example of **Pre-order-Alias-Start** linearization. The passage between the logical form and the graph is equivalent, in red the non executed node and in blue the executed ones. Omega control the executed nodes.
SQL : "**SELECT abs ((SELECT Year FROM w WHERE Title = 'cry wolf') - (SELECT Year FROM w WHERE Title = 'four christmases'))**".
Natural Language question :"**What is the difference in years between cry wolf and four christmases ?**".