Tackling Temporal Questions in Natural Language Interface to Databases

Ngoc Phuoc An Vo, Irene Manotas, Octavian Popescu,
Hangu Yeo, Elahe Khorasani, Vadim Sheinin
IBM Research
{ngoc.phuoc.an.vo, irene.manotas}@ibm.com
{o.popescu, hangu, elkh, vadims}@us.ibm.com

Abstract

Temporal aspect is one of the most challenging areas in Natural Language Interface to Databases (NLIDB). This paper addresses and examines how temporal questions being studied and supported by the research community at both levels: popular annotated dataset (e.g. Spider) and recent advanced models. We present a new dataset with accompanied databases supporting temporal questions in NLIDB. We experiment with two SOTA models (Picard and ValueNet) to investigate how our new dataset helps these models learn and improve performance in temporal aspect.

1 Introduction

Natural language interface to databases (NLIDB) is a task to bridge the gap between storing complex structured data in databases (DBs) and retrieving structured data from databases using natural language questions. An NLIDB system allows users to access information stored in a DB by using questions in natural language (e.g. English). In reality, temporal aspect is a common dimension existing in any DBs since it is realistic and practical to store data with associated date/time in DBs. However, it is challenging to retrieve information from DBs with temporal aspect due to difficulties and limitations such as 1) complex and limited association between certain entities/attributes and temporal info in DB that reflects different states of data in different time frames, and 2) complexity of natural language expressions to decipher the actual value of temporal expressions in given questions to correctly derive data from DBs. Temporal aspect is a common yet distinct and complex dimension that needs to be handled carefully in NLIDB.

Given its difficulties and complexity, temporal aspect in NLIDB is currently under-attended by the research community. First of all, it is difficult to design and create database that supports temporal dimension. Next, it is more challenging to create pairs of questions and associated SQL queries that support the learning of temporal aspect in NLIDB (discussion in Section 4). Our contribution is threefold: i) we investigated how temporal aspect was defined and annotated in the well-known Spider dataset, ii) we created the new dataset TempQ4NLIDB supporting the learning and understanding temporal questions in NLIDB, and iii) we experimented with two SOTA models - ValueNet (Brunner and Stockinger, 2021) and Picard (Scholak et al., 2021) - to understand how they learn and handle temporal questions. To the best of our knowledge, this work is one of the first attempts trying to explore this research area to support temporal questions in NLIDB.

2 Background and Related Works

NLIDB or Text-to-SQL is a long standing NLP task with many references on its complexity and achievements obtained recently (Navid et al., 2017; Popescu et al.; Yao et al., 2010). Although there is no official definition of temporal questions in NLIDB, the study (Androutsopoulos et al., 1995) mentioned about temporal questions with respect to temporal databases (Jensen and Snodgrass, 2018). In contrast, the study of temporal questions is a strong interest in Question Answering (QA) community. In the scope of this paper, we adopt a definition of temporal questions in QA for our work that is *"A temporal question is any question, which contains a temporal expression, a temporal signal, or whose answer is of temporal nature."* (Jia et al., 2018). In QA, temporal questions can be answered by temporal information embedded in semantic relations and timeline between events in corpus or taxonomy, whereas DB records associated with temporal dimension are answers in NLIDB.

The Spider dataset (Yu et al., 2018) and Spider challenge was introduced in 2018 where participating teams can have their systems evaluated on an unseen test set which is not available to public.

Proceedings of EMNLP 2022 Industry Track, pages 189–197
December 9–11, 2020.

There are two Spider sub-challenges, the first one for SQL inference without values and the second for systems that handle the values in SQL queries. Many attempts with different approaches, based on neural networks, especially on encoder-decoder architecture, have continuously improved the state of the art (SOTA). First, the sequence-to-sequence approach was introduced (Cai et al., 2018; Gehring et al., 2017; Yin et al., 2016; Rabinovich et al., 2017) in which a neural network with very good proven qualities in translation tasks translates an English query into an SQL query. These systems infer the SQL formula directly as a standard translating task from one language to another.

Instead of translating into SQL, another approach translates questions into a representation that captures semantics of a question, namely Intermediate Representation (IR) (Guo et al., 2019; Zhang et al., 2019; Bogin et al., 2019). The network learns a more structured and compact form of the query itself. From IR to SQL is a deterministic process: a context free grammar is used to convert one into another. The authors build on the work of (Sun et al., 2019; Cheng et al., 2019) that used Abstract Syntax Tree (AST). The decoder infers the IR as an AST representation of the query. Among others, ValueNet (Brunner and Stockinger, 2021) not only uses IR approach but also extends the context free grammar to include values and became one of the best systems in the Spider challenge in 2020. Recently, PICARD (Scholak et al., 2021), a state-of-the-art algorithm for constrained decoding was introduced and achieved top rank in Spider challenge. It relies on the structure and the content of the database as well as the knowledge encapsulated in the T5 language model (Raffel et al., 2020).

3 Discovering Temporal Aspect in Spider

Spider (Yu et al., 2018) is a popular large-scale complex and cross-domain dataset supporting Text-to-SQL task consisting of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables covering 138 different domains. Since there is no description of temporal aspect in Spider, we investigate temporal questions in this dataset.

3.1 Questions Related to Temporal Aspect

Questions related to temporal aspect are questions that query or process one or more column representing date/month/year/time in SQL queries.

Searching in Spider, among 7,000 questions in Train set, we found 504 SQL queries (7.2%) and 67 SQL queries (6.46%) among 1,036 questions in Dev set that hit our search keywords, respectively. Questions associated with these SQL queries are considered having temporal aspect.

3.2 Temporal Question Types

Among questions found in Section 3.1, we define three question types regarding temporal aspect:

Type 1: Questions querying Temporal Info. This type of question only queries for temporal information from DBs using the SELECT clause and has no logical operation with temporal columns in database, such as: {What are the first names and **birth dates** of players from the USA?}

Type 2: Questions querying Temporal Information with Grouping or Sorting. This type of question may or may not query temporal information but it has *GROUP BY* or *ORDER BY* processing on one or more temporal columns in database. These questions usually have one or more temporal adverbs like *{most recently, in the order of, youngest, oldest, longest, the latest}*. For examples: {What is the first name and country code of the **oldest** player?} or {How many total tours were there for **each ranking date**?}.

Type 3: Questions with Temporal Conditions. This type of question may (not) query temporal information but always has one ore more temporal conditions to derive required information from DB, such as: {Show the organizer and name for churches that **opened between 1830 and 1840**}. In this paper, we only focus on this type since it is the most practical and challenging type in reality. We found 201 and 19 temporal condition questions in Train/Dev sets of Spider, respectively.

4 TempQ4NLIDB Dataset for NLIDB

Since temporal dimension is practical and crucial for DBs in any real-world applications, and the lack of dataset for temporal questions in NLIDB, we created TempQ4NLIDB - a dataset of temporal condition questions for studying and experimenting with the recent advanced NLIDB models.

4.1 Accompanied Databases

We release two synthetic DBs (SQLite compatible) adopted from our real-world industry projects

	WH	HR
Temporal Train	65	46
Temporal Dev	7	4
Temporal Test	28	25
Total	100	75
Non-Temporal Train	146	99
Non-Temporal Dev	16	10
Non-Temporal Test	40	78
Total	202	187

Table 1: Temporal and Non-Temporal Questions in TempQ4NLIDB

and fully anonymized for public use. All temporal values are in standard format YYYY-MM-DD.

Human Resource (HR) is a one-table DB containing employees information such as employee number, name, birthdate, hire date, leave date, department, manager, salary, bonus.

Warehouse (WH) is a complex schema for sales activities consisting of 8 tables. More details can be found in our dataset.

4.2 Temporal Condition Questions and SQLs

We created new temporal conditional questions and SQL queries for two accompanied DBs. We also release the non-temporal questions and SQL queries for performance evaluation and comparison (Table 1). Our dataset is available here[1]. We may continue expanding this dataset with more questions of different types and complexity levels in near future.

4.2.1 An ad-hoc Date Annotator (DA)

We use an ad-hoc date annotator (a part of another rule-based NLIDB system) (Vadim et al., 2018; Popescu et al., 2019; Vo et al., 2019; Yeo et al., 2021) which was built on top of Duckling[2]. It detects temporal expressions in a given question and produces normalized values in standard YYYY-MM-DD format. For some experiment settings, it also re-writes original question by replacing original temporal expressions with their normalized values.

4.2.2 Data Variants

We create three variants (different natural language questions but the same SQL query) as follows:

***Original* Temporal Questions.** This variant is questions having original temporal expressions without normalizing by DA. It is the most challenging data format for any given model to learn by mapping between original temporal expressions in a given question and the actual DB date values in the associated SQL query.

***Full-DA* Questions.** Temporal questions that are pre-processed and re-written with Data Annotation and date values are appended at the end of questions. This variant is inspired by the mechanism that ValueNet (Brunner and Stockinger, 2021) used to learn values from given questions.

***Partial-DA* Questions.** Temporal questions that are pre-processed and re-written with Data Annotation without date values appended at the end of questions. This variant can ease the learning of a given model by mapping between the normalized date values in a question and the DB values from its associated SQL query.

Examples of question variants and same SQL.

- *Original* question: What products were sold *from 2011 to 2015*?
- *Full-DA* question: What products were sold **from 2011-01-01 to 2015-12-31**?; **2011-01-01#date#date**; **2015-12-31#date#date**
- *Partial-DA* question: What products were sold **from 2011-01-01 to 2015-12-31**
- SQL: `SELECT distinct T2.PRODUCT_ID FROM SALES AS T1 JOIN SALES_DETAILS AS T2 ON T1.SALES_ID=T2.SALES_ID WHERE T1.DATE >= '2011-01-01' AND T1.DATE <= '2015-12-31'`

4.3 Annotation Guideline

Data annotation for Text-to-SQL task is not trivial. For every data point, we must create a pair of question and associated SQL query. In addition, data annotation for Text-to-SQL with temporal aspect is even more challenging as every question needs to have at least one new temporal dimension. We define and practise the following guideline.

4.3.1 Temporal Dimension Annotation

We define the following annotation guideline for temporal questions.

Mapping Temporal Operators to SQL. We follow the standard temporal operators {`BEFORE`, `AFTER`, `ON`, `IN`, `BETWEEN...AND`} in temporal questions mapping into standard SQL operators supported by SQLite for temporal aspect {`<`, `>`,

[1] https://github.com/IBM/TempQ4NLIDB-dataset
[2] https://duckling.wit.ai/

=, >= AND <=}. Especially, the BETWEEN operator in Spider is not inclusive and mapped into (a_date > start_date AND a_date < end_date) at SQL level. Unlike Spider, our annotation for BETWEEN at SQL level is inclusive. For example:

- Spider: What roles did staff members play between '2003-04-19' and '2016-03-15'?
  SELECT role_code FROM Project_Staff WHERE date_from > '2003-04-19' AND date_to < '2016-03-15'
- Our annotation: What are employees hired between Jan 2012 and Jun 2012?
  SELECT distinct EMPNAME FROM EMPLOYEE WHERE HIREDATE >= '2012-01-01' AND HIREDATE <= '2012-06-30'

In our annotation, we support two new temporal operators SINCE for after but including, and BY for before but including (Table 2).

Mapping Temporal Expressions to SQL. For mapping from temporal expressions to date values in SQL queries, we use our Date Annotator to capture and convert all temporal expressions into standard date format YYYY-MM-DD.

If we assume that time always exists as an interval with a start_point and an end_point, we can translate any temporal expression into a time range. For example:

- Chirstmas 2000 = [2000-12-25, 2000-12-25]
- July 2020 = [2020-07-01, 2020-07-31]
- 2021 = [2021-01-01, 2021-12-31]
- Q1 of 1999 = [1999-01-01, 1999-03-31]

Depending on the combination between temporal operator and temporal expression in a given question, we define rules to convert to SQL operators and values (Table 2).

Referring to the categories of temporal expressions in the study (Jia et al., 2018), in the scope of this paper, we only focus on using explicit temporal expressions for our dataset. We do not support implicit temporal questions (e.g. *5 years ago*, *last year*) because for annotating normalized value of implicit temporal expressions, it is required to have a time anchor which we cannot embed into our data for Text-to-SQL task. For example, given the **time anchor as 2022**, *last year = 2021*. However, when it is 2023 or later, the value *2021* is no longer correct for *last year*. Due to the nature of current data format of Text-to-SQL task, we cannot embed a specific time anchor into each question and associated SQL to support implicit temporal questions.

4.3.2 Temporal Questions and SQL Creation

Next we define the following procedure to create natural language questions and SQL queries.

Step 1: Generating simple SQL queries without temporal aspect. We first look into a given database and generate SQL query based on the database structure for expected information. For example: in HR database, we can generate different simple queries to derive information from every column in EMPLOYEE table: SELECT EMPNO, SELECT EMPNAME, SELECT MGRNAME, SELECT BIRTHDATE, SELECT HIREDATE, SELECT LEAVEDATE, SELECT SALARY, SELECT DPNAME.

We also can generate SELECT for more than one columns, for example: SELECT EMPNO, EMPNAME, SALARY FROM EMPLOYEE. We define patterns of table names and column names to automatically generate simple SQL queries (Popescu et al., 2022).

Step 2: Identifying temporal column. We need to verify which column has at least one temporal aspect and establish the association between them. For example:

1. There are 3 temporal columns: BIRTHDATE, HIREDATE, LEAVEDATE
2. Columns that can have association with the 3 temporal columns: EMPNO, EMPNAME
3. Without columns in (2), these columns can have no association with the 3 temporal columns: MGRNAME, SALARY, DPNAME

Now we can attach the column having temporal aspect with corresponding simple SELECT that we created in Step 1 using following pattern: [SELECT Column_from_(2) FROM EMPLOYEE WHERE Temporal_Column_in_(1) Temporal_Operator Temporal_Value]

For example: SELECT EMPNO, EMPNAME, MGRNAME FROM EMPLOYEE WHERE LEAVEDATE = '2022-05-17'

Step 3: Searching Temporal Operator and Temporal Value from DB. For evaluation with Execution Accuracy metric, the SQL query in Step #2 needs to return a valid result from DB submission. Thus, we develop a search algorithm to find unique temporal operator and temporal value so that the SQL query will return a valid result from DB. At the end of this step, we also manually verify correctness of the result of every SQL query against the corresponding DB.

Temporal Operators and Expressions	Operators and Values in SQL
BEFORE a_temporal_expression	a_date < start_point of a time range
AFTER a_temporal_expression	a_date > end_point of a time range
BY a_temporal_expression	a_date <= end_point of a time range
SINCE a_temporal_expression	a_date >= start_point of a time range
ON | IN a_temporal_expression	a_date >= start_point AND a_date <= end_point
BETWEEN temp_exp_A AND temp_exp_B	a_date >= start_point_A AND a_date <= end_point_B
FROM temp_exp_A TO temp_exp_B	a_date >= start_point_A AND a_date <= end_point_B
BEFORE July 14th 2021	a_date < '2021-07-14'
BEFORE July 2021	a_date < '2021-07-01'
BEFORE 2021	a_date < '2021-01-01'
AFTER Oct 25th 2020	a_date > '2020-10-25'
AFTER Oct 2020	a_date > '2020-10-31'
AFTER 2020	a_date > '2020-12-31'
SINCE Mar 20th 2018	a_date >= '2018-03-20'
SINCE Mar 2018	a_date >= '2018-03-01'
SINCE 2018	a_date >= '2018-01-01'
BY Apr 7th 2018	a_date <= '2018-04-07'
BY Apr 2018	a_date <= '2018-04-30'
BY 2018	a_date <= '2018-12-31'
ON 1/1/2021	a_date >= '2021-01-01' AND a_date <= '2021-01-01'
ON Christmas 2017	a_date >= '2017-12-25' AND a_date <= '2017-12-25'
IN July 2022	a_date >= '2022-07-01' AND a_date <= '2022-07-31'
IN 3rd quarter of 2016	a_date >= '2016-07-01' AND a_date <= '2016-07-31'
BETWEEN 1990 AND 2000	a_date >= '1990-01-01' AND a_date <= '2000-12-31'
FROM 2000 TO 2010	a_date >= '2000-01-01' AND a_date <= '2010-12-31'

Table 2: Rules and Examples for Mapping Temporal Operators and Expressions into SQL Queries

Step 4: Generating Natural Language Questions. We manually generate natural language questions for SQL queries created in Step #3 by defining semantic relations between entities and attributes. We also enrich the language of questions by using semantic similarity/relatedness techniques (e.g. paraphrasing, synonyms) and different syntax structures (e.g. passive voice, relative clause, prepositional phrase) to generate various WH-questions (Popescu et al., 2018; Vo and Popescu, 2016; Vo et al., 2015; Vo and Popescu, 2015a,b). For examples:

- Employee {has | was born} BIRTHDATE.
- Employee {joined | is hired | is employed | is recruited | started working} in a DEPARTMENT on a HIREDATE.
- Employee {left | retired} a DEPARTMENT on a LEAVEDATE.

We carefully generate explicit temporal expressions from the temporal filters created in Step #3 and attach to our natural language questions to generate temporal questions. For examples:

1. How many employees were hired *in December 2012*?
2. Which employees left *before April 2010*?
3. What employees joined Marketing department *after Christmas 2010*?
4. Show me employees hired for Manufacturing department *in 1970* and retired *in 2000*.
5. What are the Manufacturing employees with birthdays *between 1939 and 1945*?
6. What are the employees in Sales department that have birthdays *before 1970*?

4.3.3 Data Annotation Validation

The annotation is semi-automatic and then data is manually curated by one worker. We not only examine the correctness of SQL syntax for every given question, we also submit every SQL query to corresponding DB and examine the result returned from DB. Thus, we ensure that every question always has a valid result returned from submitting its SQL query to corresponding DB. Finally the data is examined and validated by other two workers.

5 Experiments with SOTA Models

We define three settings to experiment with two recent SOTA models: ValueNet (Brunner and Stockinger, 2021) and Picard (Scholak et al., 2021) for temporal questions.

Setting 1: Original Models trained on Spider. We evaluate original ValueNet and Picard models (that were trained only on Spider dataset) on temporal questions in Test set of our new dataset.

Setting 2: Only Temporal Questions. We train and evaluate ValueNet and Picard (which were already pretrained on Spider) on temporal questions in Train and Test set of our dataset, respectively.

Setting 3: Blended Questions (NT+Full-DA). We mix Full-DA temporal questions with other Non-Temporal questions of the same database for training (on top of Spider as in Setting 2) and test.

Picard. We fine-tuned T5-large with the data splits described in settings 1 through 3 above for each DB in the TempQ4NLIDB dataset. For settings 2 and 3, we fine-tuned first on the Spider dataset, took the best performing model, and continue the training on the corresponding training data combination using the temporal data for each schema. Each model was fine-tuned on 448 epochs. Training the model for more epochs did not improve the model performance on the validation set. We used Adafactor (Shazeer and Stern, 2018), a learning rate of 10^-5, and a batch size per device of 5. During testing, we enabled Picard with the highest parsing mode.

Table 3 shows evaluation results of HR and WH temporal test sets with Picard. For Setting 1, models trained without temporal data (using the default Spider dataset for training) show very poor understanding of temporal questions. For Setting 2, models trained on temporal data understand temporal questions much better. Setting 3 results show a declining performance for models to handle both temporal and non-temporal questions.

ValueNet (VL). We use the default configuration for VL experiments. VL evaluation only reports execution accuracy. For Setting 1 and 2, the VL models performed poorly on the temporal questions. The model trained on the Spider dataset only, but also adding the available trying, obtained less than 1% accuracy. The main reasons is that the VL encoder receives no information about the type of the columns and values, and when there are more than two values in the SQL query, the system systematically confound them. Because VL does not implement any control over the correctness of SQL formula, many of the SQL queries with multiple values are wrong because the values are switched between themselves or the wrong operator is used, like "=" instead of ">=". As the temporal questions are at least two values with multiple operators the inferred SQL was always wrong. For Setting 3 experiments (Table 4), as some of the questions themselves were not multi-values, the accuracy was significantly higher.

Observations. We learn the following lessons:

1. Only training on Spider is insufficient for understanding temporal condition questions.
2. Additional training on TempQ4NLIDB significantly helps models to improve the understanding of temporal condition questions.
3. It is challenging for models to understand temporal expressions in given questions then generate corresponding temporal filters with normalized values (error type 2). Rewriting questions with normalized values of temporal expressions (Partial-DA and Full-DA variants) will help to generate temporal filters with correct values.
4. We mix temporal and non-temporal questions for both training and testing to increase the complexity (e.g. multi-table joins, multiple selects, multiple values/filters, more complex language and sentence structure, etc). It is more challenging to handle both temporal and non-temporal questions than just one type.
5. More works are needed to expand the coverage for other types of temporal questions (e.g. implicit one).

6 Error Analysis

We present four error types in predictions made by Picard and ValueNet for temporal questions.

Type 1. Models cannot detect temporal values in question, thus, no corresponding filter created.

- How many iphones were sold *since October 2013* in shops located in New York?
  Prediction: `select sum(t2.quantity) from products as t1 join sales_details as t2 on t1.product_id = t2.product_id join sales as t3 on t2.sales_id = t3.sales_id join shops as t4 on t3.shop_id = t4.shop_id where t4.address = 'New York' and`

DB	Test	Match(%)	Exec(%)
	Setting 1		
HR	Original	4.35	0.00
	PartialDA	0.00	0.00
	FullDA	0.00	0.00
WH	Original	3.57	10.71
	PartialDA	0.00	10.71
	FullDA	0.00	10.71
	Setting 2		
HR	Original	82.61	82.61
	Partial-DA	95.65	95.65
	Full-DA	100.00	100.00
WH	Original	67.86	60.71
	Partial-DA	89.29	92.86
	Full-DA	89.29	92.86
	Setting 3		
HR	NT + Full-DA	58.42	66.34
WH	NT + Full-DA	72.00	75.00

Table 3: Picard's Performance for Setting 1, 2, and 3.

		Setting 3	(ValueNet)
DB	Test	Match(%)	Exec(%)
HR	NT + Full-DA	N/A	47%
WH	NT + Full-DA	N/A	39%

Table 4: ValueNet's Performance for Setting 3.

```
t1.type = 'IPHONE'
```
Actual:
```
select distinct sum(T3.QUANTITY) from SHOPS AS T1 JOIN SALES AS T2 on T1.SHOP_ID = T2.SHOP_ID JOIN SALES_DETAILS AS T3 on T2.SALES_ID = T3.SALES_ID JOIN PRODUCTS AS T4 on T3.PRODUCT_ID = T4.PRODUCT_ID where T2.DATE >= '2013-10-01' and T4.TYPE = 'IPHONE' and T1.ADDRESS = 'New York'
```

Type 2. Original temporal expressions used in predicted query without normalizing values.

- Who were recruited in Marketing department *since March 2012*?
  Prediction:
  ```
  select distinct empname from employee where dptname = 'Marketing' and hiredate > 'March 2012'
  ```
  Actual:
  ```
  select distinct EMPLOYEE.EMPNO from EMPLOYEE where EMPLOYEE.DPTNAME = 'Marketing' and EMPLOYEE.HIREDATE >= '2012-03-01'
  ```

Type 3. Missing temporal filters in prediction in case question has multiple temporal expressions.

- What are the names of Sales employees who were hired *after 1977-12-31* and departed *before 2020-01-01*; *1977-12-31#date#date; 2020-01-01#date#date*
  Prediction:
  ```
  SELECT DISTINCT T1.empname FROM employee AS T1 WHERE T1.dptname = '1977-12-31' and T1.hiredate < 'Sales'
  ```
  Actual:
  ```
  select distinct EMPLOYEE.EMPNAME from EMPLOYEE where EMPLOYEE.HIREDATE > '1977-12-31' and EMPLOYEE.LEAVEDATE < '2020-01-01' and EMPLOYEE.DPTNAME = 'Sales'
  ```

Type 4. Incorrect filters (matching with wrong DB columns) created for temporal values.

- How many iphones were sold since 2013-10-01 in shops located in New York; *2013-10-01#date#date*
  Prediction:
  ```
  SELECT DISTINCT sum(DISTINCT T1.quantity) FROM sales_details AS T1 JOIN products AS T2 ON T1.product_id = T2.product_id JOIN stock AS T15 ON T2.product_id = T15.product_id JOIN shops AS T3 ON T15.shop_id = T3.shop_id JOIN sales AS T4 ON T3.shop_id = T4.shop_id WHERE T2.type = '2013-10-01' and T3.address = '2013-10-01' and T4.date = '2013-10-01'
  ```
  Actual:
  ```
  select distinct sum(T3.QUANTITY) from SHOPS AS T1 JOIN SALES AS T2 on T1.SHOP_ID = T2.SHOP_ID JOIN SALES_DETAILS AS T3 on T2.SALES_ID = T3.SALES_ID JOIN PRODUCTS AS T4 on T3.PRODUCT_ID = T4.PRODUCT_ID where T2.DATE >= '2013-10-01' and T4.TYPE = 'IPHONE' and T1.ADDRESS = 'New York'
  ```

7 Conclusions and Future Work

In this paper, we attempt to tackle the temporal aspect in NLIDB. This is a practical and challenging topic in real-world applications for which there are not many studies. Being motivated by the necessity and lack of available dataset for temporal questions, we created the new dataset TempQ4NLIDB. We also experimented with two SOTA models in NLIDB and show that they benefit from our dataset for better learning temporal questions. In future, we will increase the size of our dataset and expand different types of temporal questions (e.g. implicit).

References

Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. *arXiv preprint arXiv:1905.06241*.

Ursin Brunner and Kurt Stockinger. 2021. Valuenet: A natural language-to-sql system that learns from database information. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2177–2182. IEEE.

Ruichu Cai, Boyan Xu, Xiaoyan Yang, Zhenjie Zhang, Zijian Li, and Zhihao Liang. 2018. An encoder-decoder framework translating natural language to database queries.

Jianpeng Cheng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. 2019. Learning an executable neural semantic parser. *Computational Linguistics*, 45(1):59–94.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*.

Christian S Jensen and Richard T Snodgrass. 2018. Temporal database.

Zhen Jia, Abdalghani Abujabal, Rishiraj Saha Roy, Jannik Strötgen, and Gerhard Weikum. 2018. Tempquestions: A benchmark for temporal question answering. In *Companion Proceedings of the The Web Conference 2018*, pages 1057–1062.

Yaghmazadeh Navid, Wang Yuepeng, Dillig Isil, and Thomas Dillig. 2017. Query synthesis from natural language. *International Conference on Object-Oriented Programming*.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases.

Octavian Popescu, Ngoc Phuoc An Vo, Vadim Sheinin, Elahe Khorashani, and Hangu Yeo. 2019. Tackling complex queries to relational databases. In *Asian Conference on Intelligent Information and Database Systems*, pages 688–701. Springer.

Octavian Popescu, Irene Manotas, Ngoc Phuoc An Vo, Hangu Yeo, Elahe Khorashani, and Vadim Sheinin. 2022. Addressing limitations of encoder-decoder based approach to text-to-SQL. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1593–1603, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Octavian Popescu, Ngoc Phuoc An Vo, and Vadim Sheinin. 2018. A large resource of patterns for verbal paraphrases. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901. Association for Computational Linguistics.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2019. Semantic parsing with syntax and table aware sql generation. *56th Annual Meeting of the Association for Computational Linguistics*.

Sheinin Vadim, Khorasani Elahe, Yeo Hangu, Xu Kun, An Vo Ngoc, Phuoc, and Octavian Popescu. 2018. Quest: A natural language interface to relational databases. *LREC*.

Ngoc Phuoc An Vo, Simone Magnolini, and Octavian Popescu. 2015. FBK-HLT: A new framework for semantic textual similarity. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 102–106, Denver, Colorado. Association for Computational Linguistics.

Ngoc Phuoc An Vo and Octavian Popescu. 2015a. Learning the impact and behavior of syntactic structure: A case study in semantic textual similarity. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 688–696, Hissar, Bulgaria. INCOMA Ltd. Shoumen, BULGARIA.

Ngoc Phuoc An Vo and Octavian Popescu. 2015b. A preliminary evaluation of the impact of syntactic structure in semantic textual similarity and semantic relatedness tasks. In *Proceedings of the 2015*

Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop, pages 64–70, Denver, Colorado. Association for Computational Linguistics.

Ngoc Phuoc An Vo and Octavian Popescu. 2016. A multi-layer system for semantic textual similarity. In *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 56–67.

Ngoc Phuoc An Vo, Octavian Popescu, Vadim Sheinin, Elahe Khorasani, and Hangu Yeo. 2019. A natural language interface supporting complex logic questions for relational databases. In *International Conference on Applications of Natural Language to Information Systems*, pages 384–392. Springer.

Ziyu Yao, Yu Su, Huan Sun, and Wen tau Yih. 2010. Model-based interactive semantic parsing: A unified framework and a text-to-sql case study. *IJCNLP*.

Hangu Yeo, Elahe Khorasani, Vadim Sheinin, Ngoc Phuoc An Vo, Octavian Popescu, and Petros Zerfos. 2021. Programmatic database language generation for big data applications. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 2848–2856. IEEE.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016. Neural enquirer: Learning to query tables with natural language.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based sql query generation for cross-domain context-dependent questions. *arXiv preprint arXiv:1909.00786*.