A Natural Language Interface for Database: Achieving Transfer-learnability Using Adversarial Method for Question Understanding

Wenlu Wang
Auburn University
wenluwang@auburn.edu

Yingtao Tian
Stony Brook University
yittian@cs.stonybrook.edu

Haixun Wang
WeWork Research
haixun.wang@wework.com

Wei-Shinn Ku
Auburn University
weishinn@auburn.edu

***Abstract*—Relational database management systems (RDBMSs) are powerful because they are able to optimize and execute queries against relational databases. However, when it comes to NLIDB (natural language interface for databases), the entire system is often custom-made for a particular database. Overcoming the complexity and expressiveness of natural languages so that a single NLI can support a variety of databases is an unsolved problem. In this work, we show that it is possible to separate data specific components from latent semantic structures in expressing relational queries in a natural language. With the separation, transferring an NLI from one database to another becomes possible. We develop a neural network classifier to detect data specific components and an adversarial mechanism to locate them in a natural language question. We then introduce a general purpose transfer-learnable NLI that focuses on the latent semantic structure. We devise a deep sequence model that translates the latent semantic structure to an SQL query. Experiments show that our approach outperforms previous NLI methods on the WikiSQL [49] dataset, and the model we learned can be applied to other benchmark datasets without retraining.**

I. Introduction

The majority of business data is relational data. Many applications are built on relational databases, including customer relationship management systems [29], financial fraud detection systems [28], and systems for knowledge discovery in medicine [6], etc. In order to make databases more accessible to the general public, much effort has been devoted to the study of natural language interfaces to database [1], or NLIDB.

An NLIDB translates a natural language question to a structured query (e.g., an SQL query) that can be executed by a database engine. Figure 1 shows two natural language questions against two relational tables. A foremost challenge is mapping natural language expressions to database columns and values. For example, in Figure 1 (c), the term "star in" is actually a 'mention' of the Actor column in the database. In Figure 1 (d), "how many people live in" mentions column Population, which shows that mentions and database column names can be different. Furthermore, the question may not even mention columns explicitly. For example, the question in Figure 1 (d) asks about the population of a county, but it does not explicitly mention column County. An NLIDB endeavors to provide the flexibility in querying a database, but because of the complexity and idiosyncrasies of natural languages, turning questions into SQL queries is a big challenge.

The biggest challenge, however, lies in building a general purpose NLIDB, that is, an NLI that works for any database. While it might be possible to customize an NLI for a particular database, making it work for any database seems extremely difficult. In this paper, we endeavor to solve this problem by first making the following observation: The two databases in Figure 1 are in different domains, and the two questions do not share any similarity as they are asking about two different topics. However, a big surprise is that the final SQL queries are exactly the same (if we replace the column names by placeholders such as c_1 and c_2). We argue that the underlying logic or the latent semantic structure of the queries is the same. What makes the questions look so different is merely the natural language idiosyncrasies for specific types of data.

Our goal, motivated by the above observation, is to separate out data-specific components and focus on the latent semantic structure in a natural language question. The said data-specific components include the schema of the data and the usage of natural language specific to the schema of the data. Given the schema of the data and potentially a knowledge base about the schema, we may 'strip' data-specific components from a natural language question, and what remains is the latent semantic structure that is common to relational queries or relational algebra. We then use a sequence-to-sequence translation method to convert it into an SQL query.

Overview of Our Approach To achieve the above goal, we devise a framework that consists of three parts:

1) We convert a natural language question q to its annotated form q^a;
2) We use a sequence-to-sequence model to translate q^a to an annotated SQL s^a;
3) We convert the annotated SQL s^a to a regular SQL s.

Figure 1 illustrates the above steps, with two examples represented in the form of (q, q^a, s^a, s). We use placeholder c_i to denote the i-th column of a database table and v_i to denote a value that is likely to belong to the i-th column. For example, the term "directed by" in Figure 1(a) is annotated as c_2 since it is a mention of the 2nd column of the database table, and "Jerzy Antczak" is annotated as v_2 since it is a value of the 2nd column. This simple idea is powerful because it reveals

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00016

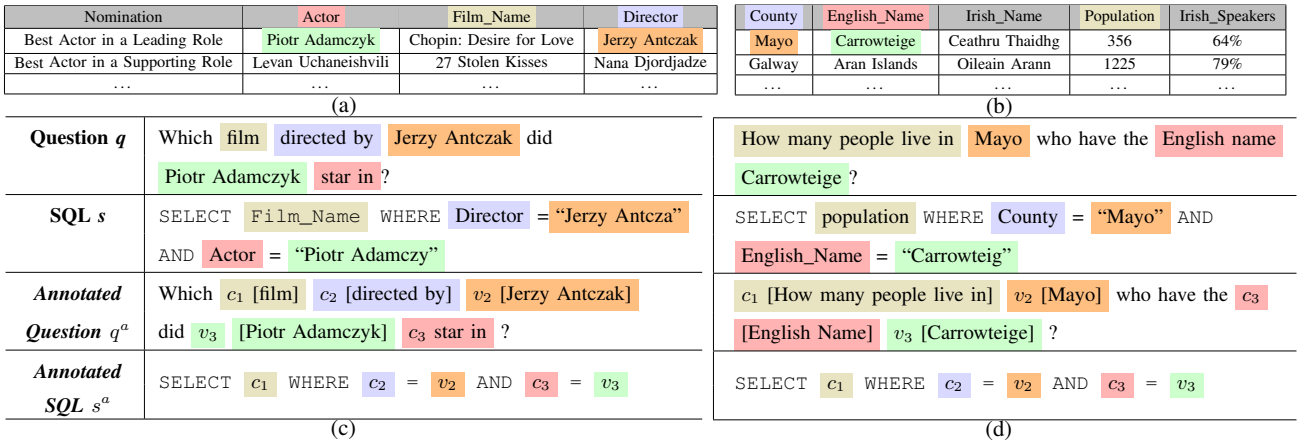

Fig. 1: Natural language questions and their corresponding SQLs against two different databases. Note that the annotated SQLs of the two different questions are the same. This figure is better viewed on media with color support.

that the two different questions in Figure 1 have exactly the same structure SELECT c_1 WHERE c_2 = v_2 AND c_3 = v_3.

The first step (annotating a question q to reveal mentions of database columns and values) is very challenging. For example, a value that appears in the query may not appear in the database. Thus, the annotation process must be able to annotate a term as a possible value of a database column. In the rest of the paper, we will discuss this and many other challenges. The second step involves a customized neural network sequence-to-sequence model, which translates a question stripped of data-specific components to an SQL. The third step (converting an annotated SQL s^a back to a regular SQL s) is deterministic. Thus, in the rest of the paper, we only focus on the first and the second step.

Our Contributions are summarized as follows

- We formalize the idea of separating data-specific components and focusing on the latent semantic structure in a natural language question to the database. This allows us to work on a question's semantics and idiosyncrasies of natural languages separately.
- We propose an automatic process that works for questions against any databases using deep neural networks and an adversarial mechanism. We further propose a deep sequence model that performs the question-to-sql translation. The combination of both leads to a high-performance NLI that enables generalization and transfer-learnabilty.
- We demonstrate the performance of our approach in empirical experiments. Our approach achieves state-of-the-art 75.6% exact query match accuracy on WikiSQL [49], and enables zero-shot transfer-ability to OVERNIGHT [44].

Paper Organization The paper is organized as follows. Section II describes the metadata we need for creating a generalizable NLIDB. Section III describes the challenges of understanding a natural language question against a database. Section IV discusses how we annotate a natural language query to reveal its semantic structure. In Section V, we describe a deep neural network that translates an annotated query to an SQL statement. Section VI covers related works, and Section VII presents empirical experiments and analysis. Section VIII concludes this paper.

II. Metadata

Metadata plays an essential role in RDBMSs. Given the metadata about a database, RDBMSs can optimize and execute queries against the database. Besides the metadata used in RDBMSs, NLIDBs need extra metadata to understand natural language expressions specific to a database. In this section, we describe the metadata we use in our work.

Database schema: Schema is part of the metadata for RDBMSs. The database schema includes, among other things, the definition of the columns of a database table. For example, the schema of the database in Figure 1(a) is defined as $\mathcal{C}$ = {Nomination, Actor, Film Name, Director}, where each column is further described by its data type and other information.

Database statistics: In RDBMSs, database statistics are important for query optimization. For example, understanding the distribution of data in each column will enable the RDBMSs to optimize the order of a join. For NLIDBs, we need statistics of the database to understand natural language queries against the database better. For example, in Figure 1 (c), for "Piotr Adamczyk", we need to determine that it is likely a mention of a value in the Actor column. Specifically, we construct and leverage database statistics that enable us to measure how likely a phrase is related to database column c for all $c \in \mathcal{C}$. In our case, we create a language model for each column. More specifically, we use pre-trained word-embeddings to decide if a particular term belongs to a particular column (the idea is that if a term is related to a column, its embedding should be close to the word-embedding space of values in the column).

Natural language expressions specific to a database: It is not trivial to know how people refer to things embodied by a database. We may understand terms such "actor' and "actress" may refer to the same column through using some simple techniques, such as synonym detection. But understanding that "how many people live in" is a mention of "Population" may need paraphrasing, which is a problem that has not been

solved. Ideally, if we have a general purpose ontology that tells us everything about how language is used to describe any entity and its features, we might simply incorporate the ontology. However, such an ontology does not exist. In this work, we introduce a new mechanism that allows us to manually introduce the knowledge of the natural language for a specific database. First, we collect database-specific natural language metadata. Specifically, for a column c, we collect phrases $\mathcal{P}_c$ that *mentions* c and expressions $\mathcal{D}_c$ that *describe* a column. Later, these phrases and expressions are used to match part of the question to provide extra candidates of *mention* (defined in Section III) of column in addition to the main algorithm (Section IV-A). For example, for c = "Population" we may collect $\mathcal{P}_c$ = {how many people live in *New York City*, density of *New York City*, ...}, and for c = "Price" we may collect $\mathcal{D}_c$ = {soar, level off, dive, ...}. Later, our method knows in a question "density of" and "level off" could mention the column "Population" and "Price" respectively. In this way, our approach provides a direct way to inject this minimal knowledge to our model, which, by the nature of merely providing extra candidates, is optional and orthogonal to the rest of the model.

III. Challenges of Question Understanding

Our goal is to understand the underlying semantic structure of a natural language question against a database. The first step toward revealing the structure is to detect the mentions of database columns and values in the question. In this section, we describe the specific challenges in this task.

Before we dive into the details, let us define a *term* to be a continuous span of words in the question. If a term refers to a database column, we say the column is mentioned by the term. For example, in question q = [*Which, film, directed, by, Jerzy, Antczak, did, Piotr, Adamczyk, star, in*], the continuous span $q[3, 4]$=[*directed, by*] is the ***mention*** of column Director. A term may also mention a value. For example, [*Piotr, Adamczyk*] could be a mention of a value in either the Director or the Actor column, as both columns contain person names. Without context, a term could be a mention of multiple columns or values in the database.

Detecting and resolving mentions are non-trivial tasks. Some mentions of database columns and values can be detected exactly as they appear in the questions. Others may require us to evaluate edit distances or semantic distances [1] between two terms. However, in many cases, mention detection relies heavily on the context. Below, we enumerate five challenges that we need to address in detecting and resolving mentions.

1) **Non-exact matching.** In the question *Who is the best actress of year 2011?* "best actress of year 2011" mentions the database column "best actor 2011". Clearly, we cannot rely on exact string matching to detect mentions.
2) **Paraphrases.** For example, the paraphrase "how many people live in ..." could be a mention of the "Population" column. We need to understand whether an expression is a paraphrase of another expression,
3) **Implicit mentions.** Consider the question in Figure 1(d): *How many people live in Mayo who have the English Name Carrowteige?* Here, "Mayo" is a county, but the question does not mention the database column County explicitly. We need to infer the column from the question, the database schema, and the database statistics.
4) **Mentions of counterfactual values.** For example, one may ask "When was Joe Biden elected U.S. president?" against a database table of U.S. presidents. But "Joe Biden" is not in the database (at least not yet). Despite that, the question is not less valid than "When was Barack Obama elected U.S. president?", and we need to handle both situations.
5) **Resolutions.** In the question "*Which film directed by Jerzy Antczak did Piotr Adamczyk star in?*" "Jerzy Antczak" and "Piotr Adamczyk" could refer to either Director or Actor. We therefore need to infer the correct resolution depending on the syntax, or the context of the question.

We address the above challenges in Section IV. Then, after we obtain an annotated query q^a, we describe a sequence translation-based approach to convert q^a into an SQL statement in Section V.

IV. Mention Detection and Resolution

In this section, we focus on the first step, namely converting a natural language question q to its annotated form q^a through mention detection and mention resolution. We propose a novel adversarial machine comprehension model for this purpose. It consists of three components that address five challenges we discussed in Section III.

1) To detect mentions for columns (Section IV-A), we propose a machine comprehension binary classifier (Section IV-B) and an adversarial text method (Section IV-C) to address column-related mention detection challenges (challenges 1, 2, and 3) that cannot be solved by edit distances and semantic distances.
2) We propose a binary classifier (Section IV-D) to address value-related mention detection (challenge 4).
3) We introduce a method (Section IV-E) for mention resolution (challenge 5).

A. Mention Detection for Columns

We describe a system that addresses three challenges described in Section III, namely, challenge 1 (non-exact matching), challenge 2 (paraphrases), and challenge 3 (implicit mentions). The 1st and the 2nd challenges ask *whether* a particular column is mentioned in the question, and the 2nd and the 3rd challenges ask *where* in the question the mention occurs. We propose to handle the task by first detecting whether a column is mentioned and then finding the term in the question that mentions the column. More specifically, we do the following:

[1] Edit distance counts the minimum number of operations (e.g., substituting, deletion, or insertion) required to transform one string into another; Semantic distance measures the L^p distance (e.g., Euclidean distance) between two words in a latent semantic space (e.g., GloVe [31]).

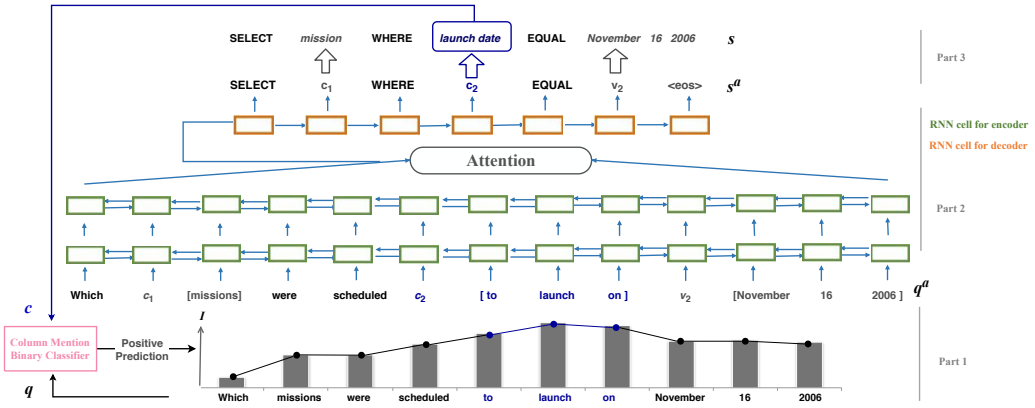

Fig. 2: Framework overview with mention detection of column "launch date" as an example.

1) In Section IV-B, we develop a *Column Mention Binary Classifier*. For a question q and each column c in the database table, the classifier predicts whether c is mentioned or not in q. We devise the classifier as a machine comprehension model based on deep neural network on top of both word- and character-level representations in order to address non-exact matching, i.e., matching with semantic (different words of similar meanings) and/or lexical (character-level syntax) differences (1st challenge). The deep neural network is a bi-directional attention flow [36]. With the metadata (Section II), our approach also supports complicated natural language paraphrases (3rd challenge).
2) In Section IV-C, if the classifier determines that a column is mentioned by the question, we will identify the term that constitutes the mention. Our approach is motivated by the idea behind Adversarial Examples [9, 39]: We look for *part* of the input that is most influential in the classifier's decision. In practice, we calculate each word's influential level towards the final prediction using fast gradient method (FGM) [9], which identifies the gradient direction that is proportional to dL/dq (loss gradient with q as the input). In doing so, our approach leverages the classifier's ability to handle complex phrases (the 2nd challenge), and measuring the scale of dL/dq also handles cases where the term in question is missing (the 3rd challenge). As an extra benefit, using the adversarial method solely relies on the information learned in the aforementioned classifier and does not need extra supervision for training.

Figure 2 gives an example. We have a question q = *Which missions were scheduled to launch on November 16, 2006?* Now, the classifier predicts that column "Launch Date" is mentioned in the question. We then want to identify the term in the question that actually mentions "Launch Date". We do this by searching for a continuous span that, if changed, changes the prediction the most. Finally, to train this part, we need data in the form of (question, SQL query) pairs plus metadata including *database schema* and *natural language expressions specific to a database* as described in Section II.

B. Column Mention Binary Classifier

We train a binary classifier taking a question q, a column $c \in \mathcal{C}$, and predicting whether c is mentioned in q. We treat q and c as sequences of words, i.e., $q = [w_1^q, w_2^q, \cdots, w_n^q]$ and $c = [w_1^c, w_2^c, \cdots, w_m^c]$. Also, we use $q[i,j] = [w_i^q, ..., w_j^q]$ to denote a continuous span of words in q.

We go through words in c one by one. For each word w, the attention mechanism gets a weighted combination of words in q, which is then combined with w and fed to an LSTM. As illustrated in Figure 3, the classifier is an end-to-end sequence model that can be logically decomposed into three parts:

(i) A **word embedder** that converts a word into a vector that encodes both its semantics and syntax;
(ii) An LSTM **sequence model** for modeling questions and a bi-directional LSTM **sequence model** for columns. They consume the sequence generated by the word embedder, and produce a latent representation at each time step.
(iii) An extra LSTM **sequence model with attention mechanism** for the column. The attention mechanism is used to combine words in question q for column c's word at each step. Its output is aggregated using a multi-layer perceptron that makes predictions.

The network models the information from q and c jointly. The classifier is trained end-to-end using back-propagation.

(i) Word Embedder The word embedder $\mathrm{emb}(w)$ takes a word w and outputs an embedding (representation) of this word. As illustrated in the zoomed-in view on the left in Figure 3, the embedder contains two parts that encode semantic and lexical information, respectively: $\mathrm{E}^{word}(w)$, which is regular word embedding, and $\mathrm{E}^{char}(w)$, which is a character level model that takes the sequence of characters in w as input and produces representation through a one-dimensional convolution. The outputs of the two parts are concatenated to produce the final output $\mathrm{emb}(w) = [\mathrm{E}^{word}(w), \mathrm{E}^{char}(w)]$.

While the first part $\mathrm{E}^{word}(w)$ is straightforward, the character level model $\mathrm{E}^{char}(w)$ is worthy of more elaboration. As in Figure 4, we denote w as a sequence of characters $w = [a_1, a_2, \cdots, a_{|w|}]$. We introduce character embedding, denoted as $\mathrm{EmbChar}(ch)$, which maps a character ch to a vector. Applying character embedding on each of the characters yields a character representation matrix A where the i-th row A_i is $\mathrm{EmbChar}\,(a_j)$. Next, we use a one-dimensional convolution of width k to project each slice

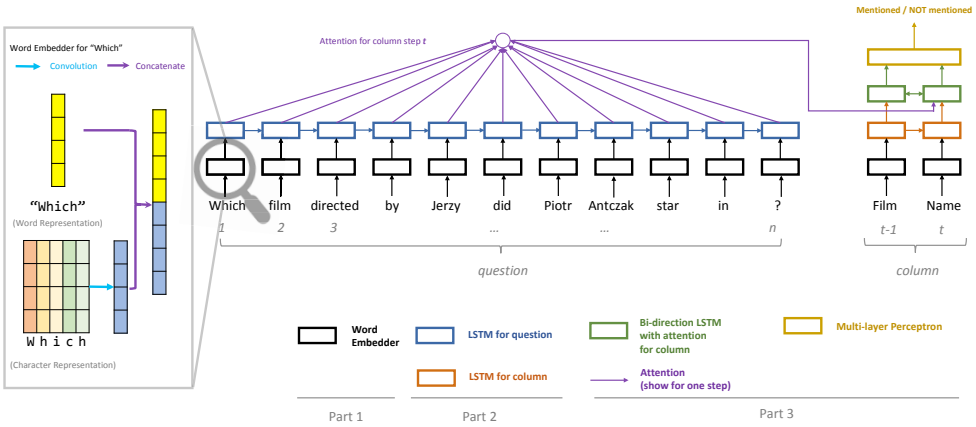

Fig. 3: Mention of Column: Classifier's overview.

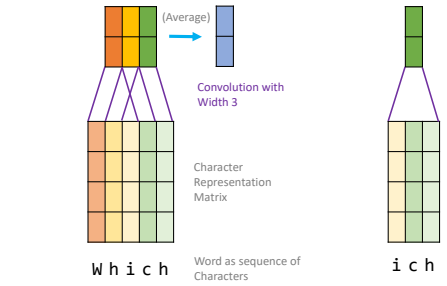

Fig. 4: Convolution Neural Network in Word Embedder

$(A_{1:k}, A_{2:k+1}, \ldots, A_{|w|-k+1:|w|})$ of A to a vector, and we compose these vectors using element-wise average to get the output, denoted as $\mathrm{E}_k^{char}(w)$. For the convolution, we pad with zeros so that at least one slice is available. Figure 4 (a) shows the whole process with a one-dimensional convolution of width $k = 3$. Figure 4 (b) highlights the application of convolution on the last slice (last k words). Note that the projection is a linear one and is shared across different slices. To capture character level syntax with different granularity, we apply the aforementioned process with different convolution width (in practice we get $\mathrm{E}_k^{char}(w)$ for $k \in \{3, 4, 5, 6, 7\}$) and concatenate these results to form $\mathrm{E}^{char}(w)$. Although different convolutions have their own projection, the character embedding $\mathrm{EmbChar}$ (by definition the character representation matrix) is shared among convolutions.

(ii) Sequence Models For question q, we stack a multi-layer recurrent neural network (RNN) on top of the word embedder, with LSTM cells to produce results at each time step (for each word in the question). Specifically, let $x_i^{(l)}$ be the input to the l-th layer in the i-th position. The hidden state and memory of the forward LSTM are computed as

$$\left[h_i{}^{(l)}, C_i{}^{(l)}\right] = \mathrm{LSTM}\left(x_i{}^{(l)}, h_{i-1}{}^{(l)}, C_{i-1}{}^{(l)}\right)$$

The input of each layer is computed as $x_i{}^{(1)} = L^1(emb(w_i^q))$ and $x_i{}^{(l+1)} = L^{(l+1)}\left(h_i{}^{(l)}\right)$, where $L^l(x) = W_0{}^{(l)}x + b_0{}^{(l)}$ is an affine transformation before each layer of RNN to keep the dimension consistent.

For column c, we apply a *separate* sequence model of the same architecture but different parameters. It stacks on top of the word embedder, which applies to each word w_i^c $(i = [1..m])$ in column c.

(iii) Sequence Model with Attention Mechanism We denote the top-layer hidden states produced by the sequence models described in Part 2 for question q and column c as

$$S^q = [s_1^q, s_2^q, \cdots, s_n^q] \qquad S^c = [s_1^c, s_2^c, \cdots, s_m^c]$$

We use a bi-directional one-layer LSTM sequence model on top of S^c with attention mechanism over questions S^q. At each step t $(1 \le t \le m)$ for column c, the forward LSTM is computed as

$$\begin{aligned}
\overrightarrow{d}_0 &= \overrightarrow{C}_0 = \mathbf{0} \\
\overrightarrow{z}_t &= \left[s_t^c \; \overrightarrow{S^q \alpha}_t{}^T\right] \\
\left[\overrightarrow{d}_t, \overrightarrow{C}_t\right] &= \overrightarrow{\mathrm{LSTM}}\left(\overrightarrow{z}_t, \overrightarrow{d}_{t-1}, \overrightarrow{C}_{t-1}\right) \\
\overrightarrow{e}_t &= v^T \mathrm{Tanh}(W_1 S^q + (W_2 s_t^c + W_3 \overrightarrow{d}_{t-1} + b) \otimes e_n) \\
\overrightarrow{\alpha}_t &= \mathrm{softmax}(\overrightarrow{e}_t)
\end{aligned}$$

where W_0, W_1, W_2, W_3, v, and b are model parameters, $\overrightarrow{d_t}, \overrightarrow{C_t}$ are the hidden states and memory of forward LSTM respectively. The outer product $(\cdot \otimes e_n)$ means repeating the vector on the left for n times.

With a backward LSTM being computed similarly, we can concatenate the forward and backward hidden state vector as $d_t = \left[\overrightarrow{d_t}; \overleftarrow{d_t}\right]$. After zero-padding d_t to the length of the maximum number of words in a column, all d_ts are concatenated and fed into a multi-layer perceptron that makes the prediction.

C. Adversarial Text Method

Assume the classifier we described above decides that column c is mentioned in question q. Then our next task is to look for the term in question q that actually mentions column c. We propose an adversarial text method to solve this problem. The method is based on the following two reasonable assumptions:

i. We assume the mention of c in q consists of a sequence of words.
ii. Drawing our inspiration from the *adversarial example* technique, we assume the mention of c is the part of q that is most influential in the classifier's decision that c is mentioned in q.

The first assumption is naturally from our domain knowledge. Arguably a model without this assumption is much more complex, so we leave the investigation of whether the extra complexity is justified to future study.

The second assumption requires some elaboration. For the background, we start with the adversarial example technique from which we draw our inspiration. An adversarial example is a carefully designed perturbation of input q that forces the aforementioned classifier in Section IV-B to make a wrong prediction. Denoting such an example as $\tilde{q} = q + \eta$, the perturbation η is small enough compared to q. In doing so, the perturbation is more significant in parts of the questions that are efficient in influencing the classifier's prediction. The effect of applying such technique can be demonstrated with the following example in which the classifier is tasked with prediction c = "*golfer*" is mentioned in the question:

q = "*Which player won the competition*?"

We denote the representation of a word as emb. Now changing the word "player" to "athlete" leads to small perturbation η = emb("*athlete*") - emb("*player*") since both words are semantically close. However for the resulted adversarial example

$\tilde{q} = q + \eta$ = "*Which athlete won the competition*?"

the perturbation η changes important features of the model (e.g., "*player*") to make the prediction regarding column c, and is highly likely to lead to a large change to the output. The combination of small perturbation and large change to the output is a good example of efficiently influencing the classifier.

We further observe that the term mentioning the column is overlapping with words whose perturbation makes an adversarial example. We hypothesize that the term which mentions column c makes the most contribution to the classifier's prediction (whether c is mentioned in the question). This concludes our second assumption to use the adversarial method for finding the term in the question.

We now formally describe our adversarial text method. Our method uses a faster adversarial method [36] to find the parts of question q that are important to the classifier's prediction by taking the gradient of loss L with respect to each word in the question. Since in natural language processing words are represented by one-hot inputs, as proposed [25], we take the gradient with respect to the representation of the words rather than the words themselves. In detail, we construct symbolic derivatives of L w.r.t. each w_i^q in q to measure the influential level of each token to the model prediction. $L = L(q, c)$ is the loss of the machine comprehension binary classifier given the column c and question q. Assuming a word embedder E (e.g., E^{word} or E^{char}) transforms a word to a high-dimensional embedding space $\mathbb{R}^{\mathrm{d}}$:

$$E(w) = [x_1, x_2, ..., x_{\mathrm{d}}]$$
$$dL/dE(w) = [dL/dx_1, dL/dx_2, ..., dL/dx_{\mathrm{d}}]$$

Then, we calculate the norm of each gradient, where $\mathrm{p}(\cdot)$ is a norm function.

$$I^{word}(w) = \mathrm{p}(dL/dE^{word}(w)) \qquad I^{char}(w) = \mathrm{p}(dL/dE^{char}(w))$$

We define $I(\cdot) = \alpha * I^{word}(\cdot) + \beta * I^{char}(\cdot)$ as the *influential level* of each token taking both word-level and character-level representation into consideration. Here, α and β are hyperparameters for balancing both the word-level and character-level information.

$$I(q) = [I(w_1^q), I(w_2^q), ..., I(w_n^q)]$$

Taking ℓ_2-norm as an example:

$$I_{\ell_2}(w_i^q) = \alpha * \|dL/dE^{word}(w_i^q)\|_2 + \beta * \|dL/dE^{char}(w_i^q)\|_2$$

Then we search for a continuous span $[a, a+1, ..., b]$ that contains the highest influential level and $b - a + 1 <$ maximum length of mentions in the question we consider. The continuous span is considered to be the term of c mentioned in q.

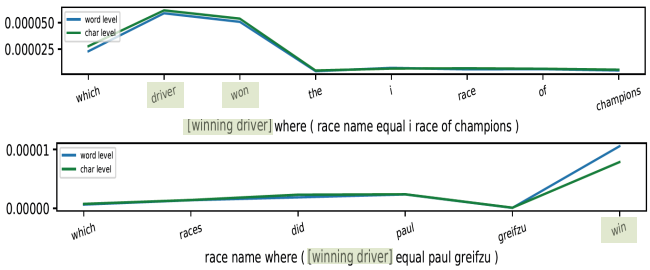

Fig. 5: Use gradient of loss with respect to each word for inferring column's mentioning term. X-axis represents the words in a natural language question, Y-axis represents influential level I of each word using ℓ_2-norm, and X-label is its corresponding SQL. Furthermore, since we use both word level and character level inputs, we plot the gradients with respect to word embedding and character embedding separately, both contributing to the model's output in a coordinated way.

In Figure 5 we show an example of detecting column "*winning driver*" in two different questions. The column name c = " [winning driver] " in SQL is the column for which we are searching the term in the question. The term highlighted in question is the term of the column mention, which corresponds to high gradient values. We can observe that word span with large gradient norms corresponds to the terms of the column in the question that a human perceives. Column "winning driver" is detected by " *driver won* " in the first question, and " *win* " in the second question. With adversarial text method, we can also address the issue of mentioning the same column in various ways.

D. Mention Detection for Values

Since the question may mention values that are counterfactual (not occurring in the table) while still being a valid question (the 2nd challenge), as a realistic setting, finding a mention of a counterfactual value should not depend on the actual content of columns, nor extra knowledge such as Freebase which is essentially another database. We leverage some aggregation that characterizes the property of columns to help value detection, which is what we described as *database statistics* in Section II. This approach avoids relying on specific values that appear in the column, thus is able to handle cases where the question mentions values that are counterfactual while still being valid.

Specifically, we propose a *Value Detection Classifier* that takes a continuous span $q[i,j]$ in the question and a column c's data statistics s_c as input, and predicts whether this $q[i,j]$ is likely to be a mention of values in column c. If the prediction is true for at least one column in the table, the span is detected as a mention of value in the table. Since the classifier requires only a column's data statistics characterizing the property of this column rather than a set of actual, concrete values in that column, it could deal with counterfactual ones.

In detail, the data statistics of column c, referred to as s_c, is a feature vector representing the property of this column. It is the dimension-wise average of the feature vectors of all cells in that column, where a cell's feature vectors are the dimension-wise average of all its words' embedding $\mathrm{emb}(w_i) = \alpha * E^{word}(w_i) + \beta * E^{\mathrm{char}}(w_i)$. Formally, this means

$$s_c = \frac{1}{|P_c|} \sum_{p \in P_c} \left(\frac{1}{|p|} \sum_{w \in p} \mathrm{emb(w)} \right)$$

with the slight abuse of symbols denoting all cells in column as P_c, (of multiple cells) a single cell in P_c as p, and (of multiple words) a word in p as w. Note that the design of s_c ensures that the data statistics contains only $O(1)$ amount of information regardless of the number of cells in a column. Similarly, the statistics of $q[i,j]$, referred to as $s_{q[i,j]}$, is also the dimension-wise average of all its words' embedding:

$$s_{q[i,j]} = \frac{1}{|j-i+1|} \sum_{i \le k \le j} \mathrm{emb}\left(q[k]\right).$$

On top of that, the classifier is implemented as a two-layer MLP (multi-layer perceptron) defined as:

$$y = \mathrm{Sigmoid}\left(W_2 \cdot \mathrm{ReLU}\left(W_1 \cdot \left[s_c - s_{q[i,j]}, s_c \cdot s_{q[i,j]}\right] + b_1\right) + b_2\right)$$

where the input is from concatenating $s_c - s_{q[i,j]}$ and $s_c \cdot s_{q[i,j]}$, $\mathrm{Sigmoid}(x) = \frac{1}{1+e^{-x}}$ and $\mathrm{ReLU}(x) = \max(x, 0)$. The output y, by the definition of Sigmoid, is continuous and is in the range $[0, 1]$, thus serving as a likelihood function. The classifier predicts true if $y > 0.5$.

Furthermore, it is natural to assume that a value should be a short multi-word entity, so in training and inference, we only consider a limited set of candidates of spans that do not contain stop words. Formally this means we consider $q[i,j]$ only if $\nexists k : i \le k \le j, q[k] \in \mathrm{StopWords}$.

E. Mention Resolution

It is possible to have many candidate mentions of columns and values (5th challenge). For example, in question "Which film directed by Jerzy Antczak did Piotr Adamczyk star in?" the values "*Jerzy Antczak*" and "*Piotr Adamczyk*" could be mentions of either "*director*" or "*actor*". Both are valid unless the syntax and context of the natural language question are taken into consideration . The goal of mention resolution is to figure out globally, what is the most likely subset of mentions that are consistent.

Inspired by [20], we leverage the question's dependency tree to help reduce the number of candidate mentions. In general, the proposed mention resolution strategy favors value and column pairs that are structurally close to each other in the said tree. We observe that in the question's dependency tree, a value is often the closest child node of the paired column. Therefore, we use the distance of two nodes (denoted as $\mathrm{dist}(\cdot,\cdot)$) in the question's dependency tree as a measure of structural closeness. Specifically, for value v and column c, and their mentions m_v and m_c (in the question), the optimal pair identified by structural closeness is $\min_{m_v \in q, m_c \in q} \mathrm{dist}(m_v, m_c)$. We only consider matched pairs (v, c) that have the optimal structural closeness.

V. Sequence to Sequence Translation

In this section we describe the second step of our framework that translates q^a to an annotated SQL s^a. As shown in examples Figure 1 (a) and (b), after the first step that provides the mentioned paired columns and values to a question in the form of an annotated question q^a, this second step does the actual work of translating q^q into an annotated SQL s^a.

The sequence-to-sequence [38] model (or seq2seq as it is commonly referred to) takes input in the form of a sequence of tokens and produces output also as a sequence. Such a model has been enjoying massive success in many natural language processing applications (see Related Work section for more). Motivated by such success, we represent both the input (annotated question q^a) and the output (annotated SQL s^a) as lists of sequences, and devise a seq2seq model that converts the former to the later. We describe how we represent our annotated question and SQL query as sequences in Section V-A, followed by presenting the very seq2seq model that does the actual translation in Section V-B.

A. Representation of Annotated Sequence

There are many options in representing annotations in a form that can be leveraged by a seq2seq model. For example, in Figure 1(a), "*directed by*" is annotated as the mention of c_2. We can either replace "*directed by*" by c_2 or insert c_2 following "*directed by*" in the question. We exploit different annotation encoding methods and propose our optimal annotation to separate information related to a schema from questions without loss of information.

*1) **Insert symbols**:* Intuitively, the annotation should enable schema separation that strips off schema-specific information from natural language questions by substituting schema-specific information with symbols. However, replacing the mentions of columns with unified symbols (e.g., c_i and v_i) hurts the semantic expressiveness of the question. Therefore, we propose to insert the symbols around the mentions rather than substituting them to leverage the semantics of column texts. We name such approach as "**column name appending**". Figure 6a shows the differences between the two approaches, and highlights that our proposed approach provides more semantic information to the downstream sequence model.

*2) **Table Header Encoding**:* When a column in SQL is not mentioned in the question explicitly, we can only rely on the deep model and the context to infer the column. For example, in Figure 6b, column name "Nomination Date" is not explicitly mentioned. Most of the columns (e.g., "Nomination Date") consist of multiple tokens, which are hard to generate correctly by a sequence model token by token. To encourage the correct inference of multi-token columns, we append all

Question	*What position did the player LeBron James play?*
Symbol Appending	*What* c_1 *[position] did the* c_2 *[player]* v_2 *[LeBron James] play?*
Symbol Substitution	*What* c_1 *did the* c_2 v_2*?*

(a) Annotation Format.

q^a	*When* v_1 *[Piotr Adamczy] was nominated as* c_1 *[Best Actor in a Leading Role]?*
s^a	`SELECT` Nomination Date `WHERE` c_1 `=` v_1
q^a	*When* v_1 *[Piotr Adamczy] was nominated as* c_1 *[Best Actor in a Leading Role]* g_1 *[Nomination]* g_2 *[Actor]* g_3 *[Film Name]* g_4 *[Director]* g_5 *[Nomination Date]*
s^a	`SELECT` g_5 `WHERE` c_1 `=` v_1

(b) Table Header Encoding.

Fig. 6: Representation of Annotated Sequence

the headers $g \in \mathcal{C}$ to the end of the annotated question, so that even if a multi-token column is not mentioned in the question, it could be inferred as g_i by the sequence model.

Figure 6b shows an example where "g_1 *[Nomination]* g_2 *[Actor]* g_3 *[Film Name]* g_4 *[Director]* g_5 *[Nomination Date]*"is appended to the annotated question. and thus simplifies the annotated SQL as "`SELECT` g_5 `WHERE` c_1 `=` v_1", where multi-token column name "Nomination Date" is simplified as "g_5".

B. Sequence Translation Model

For formality, we denote a natural language question in annotated form as $q^a = (q_1^a, q_2^a, ..., q_N^a)$, and the corresponding annotated SQL query as $s^a = (s_1^a, s_2^a, ..., s_M^a)$. We train a seq2seq model to estimate $p(s^a|q^a)$, which captures the conditional probability of

$$p(s^a|q^a) = \prod_{j=1}^{M} p(s_j^a|q_a, s_{1:j-1}^a)$$

Encoder is implemented as a stacked bi-directional Gated Recurrent Unit (GRU) [4]. To keep the dimensions consistent, we add an affine transformation before each layer of GRU, defined as the follows ${y_i}^{(l)} = {W_0}^{(l)} x_i^{(l)} + {b_0}^{(l)}$, where ${x_i}^{(l)}$ is the input of the l-th layer at the i-th position. ${W_0}^{(l)}$ and ${b_0}^{(l)}$ are model parameters. The hidden state of the forward GRU and backward GRU are computed as:

$$\overrightarrow{h_i}^{(l)} = \overrightarrow{\text{GRU}}({y_i}^{(l)}, \overrightarrow{h_{i-1}^{(l)}}) \qquad \overleftarrow{h_i}^{(l)} = \overrightarrow{\text{GRU}}({y_i}^{(l)}, \overleftarrow{h_{i-1}^{(l)}})$$

We concatenate forward state vector and backward state vector as ${h_i}^{(l)} = [\overrightarrow{h_i}^{(l)}, \overleftarrow{h_i}^{(l)}]$, $i = [1..N]$. The input of each layer is computed as: (ϕ is the word embedding lookup function)

$${x_i}^{(1)} = \phi(q_i^a) \qquad {x_i}^{(l+1)} = {h_i}^{(l)}$$

Decoder is an *attentive* GRU with *copy mechanism*. We use Bahdanau's attention [2] as follows: At each time step i in the *decoder*, the decoding step is defined as:

$$\begin{aligned}
d_0 &= \text{Tanh}(W_1[\overrightarrow{h_N^{(l)}}, \overleftarrow{h_1^{(l)}}]) \\
d_i &= \text{GRU}([\phi(s_{i-1}^a), \beta_{i-1}], d_{i-1}) \\
e_{ij} &= v^T \text{Tanh}(W_2 h_j^{(l)} + W_3 d_i) \\
\alpha_{ij} &= e_{ij} / \sum_{j'} e_{ij'} \\
\beta_i &= \sum_{j=1}^{M} \alpha_{ij} {h_j}^{(l)}
\end{aligned}$$

where W_0, W_1, W_2, W_3, and v are model parameters, $(d_1, ..., d_N)$ is the hidden states of the decoder, and j the index enumerating all the positions in *encoder*. In the NLI task, tokens in the output often correspond directly from the input natural language question. To encourage the model to favor tokens that appear in the input, we make our own implementation of copy mechanism that samples output token s_a^t as

$$\begin{aligned}
p(s_i^a|q^a, s_{1:i-1}^a) &\propto \exp(U[d_i, \beta_i]) + M_i \\
M_i[s_j^a] &= \exp(e_{ij})
\end{aligned}$$

Note that this is different from the vanilla copy mechanism where the output is sampled through softmax over entire word vocabulary as $p(s_i^a|q^a, s_{1:i-1}^a) \propto exp(U[d_i, \beta_i])$.

VI. Related Works

A. Natural Language Interface to Database (NLIDB)

Natural Language Interface to Database aims to provide an interactive bridge between users and machines, where users can issue a natural language question, which would then be translated to a structured query that is executable by a database engine. [1] first explores this task with concrete examples defining this problem and highlights the separation between linguistic and database-derived information. Later [33] proposes to identify questions whose answers are tractable solely from the database, and [7] incorporates tree kernels in ranking candidate queries. Many recent advances can be categorized into two groups. The first group uses semantic parsing [17, 30, 44] as well as some extensions that support cross-domain semantic parsing [14, 37]. However, most works in this group are confined in narrow domains because of the challenges brought by natural languages. The other group relies on neural-based methods where sequence to sequence models are leveraged to translate questions to SQL queries, optionally combined with the help of user feedback [16], reinforcement learning [49], and external semantic models [21]. On the system side, NaLIR [20] and NaLIX [22] demonstrate the feasibility of implementing a research-oriented NLIDB approach as an interactive, industry-level system involving real-world database querying and language parsing. NaLIR parses a question to a tree structure as the internal representation and then translated into SQL. PRECISE [32] defines a Semantic Tractability Model, and works for semantically tractable questions only. DBPal [3, 40] allows users to build an NLI for a new database, which is able to generate a synthetic training set with wide linguistic variations from a given database schema.

B. Slot Filling in Dialogue System

Dialogue system aims at communicating with a user in a session with multiple turns of dialogues, where *state*, or conceptually what the session is talking about, needs to be tracked for dialogue system to achieve good performance [47]. Commonly the dialogue system identifies and tracks entities that appear across turns as slots in a process called *slot filling*. These slots and entities that can fill in these slots are usually specific to the domain that the dialogue system is focusing on. For example, slots can be food, airport, or city names, and therefore are from a pre-defined, externally crafted list of possible values. Traditionally, classifier-based feature engineering that identifies slots is used in this task, such as methods proposed in [19, 34]. Recently, neural-based approaches have

been proposed for tracking state: [13] proposes a simple recurrent network that predicts the probability of each word in the dialog being one of the pre-defined slots, which is extended by [26], a hierarchical model that can handle cases where entities can be from one of the multiple domains. To specifically provide better tracking for ranking slot values in dialog, Belief Tracker [27] sums up separating representations of system output, user feedback, and candidate slot values. To further improve the performance, [45] considers a policy network that arbitrates the outputs from multiple models, including the aforementioned belief tracker, a sequence model that encodes user input, and a generation network that produces system output. Notably with engineering consideration for real-world scenarios with multiple databases, [12] proposes two separate classifiers, that classify not only what slot should be used for filling, but also slots' corresponding databases.

Closest to our proposed work is [46], which employs a sketch-based approach that represents an SQL as a template with slots, and the model predicts values from a limited candidate set to be filled in each slot. This is different from our work that focuses on annotation and does not restrict SQL to a particular template-based form. Another close work is TypeSQL [48] that enriches the inference of columns and values using a domain-specific knowledge-based model that searches five types of entities on Freebase, an extra large database, which is in contrast to our work, which does not rely on extra database knowledge.

C. Adversarial Text Methods

Adversarial samples have been extensively studied since their first discovery [39]. In the image space, after adding a small intentionally crafted perturbation to the original sample, a deep model might make a false prediction with high confidence, while the small perturbation causes subtle visual differences to humans. In the text domain, after altering a few words or characters of an input sentence, the deep model might also be fooled with high confidence. Even though adversarial mechanisms on text domain are not as widely discussed as on image domain, several adversarial text attacks are proposed [18, 23, 35] following a similar strategy: select characters, words or phrases that are the most influential to the predictions, then perturb them while monitoring the success of an adversarial sample generation. To increase the success rate of adversarial sample generations, we use the gradient of the loss function to select features for perturbation. For example, [23] uses Fast Gradient Method [9] on a character-level Convolutional Neural Network (CNN), and [8] uses Fast Gradient Method combined with kNN search in the embedding space to select perturbation candidates.

D. Sequence-to-sequence Generation model

Sequence to Sequence learning (referred to as seq2seq in the rest of the paper) [38] has led to many advances in neural semantic parsing models. Notable advances in sequence learning include attention mechanism [2] and pointer network [41] that boost performance for sequence learning and enable it to handle long sequence and rare words. They have seen successful applications on language model [24], text summarization [11], text understanding [45], and neural computing [10]. Our model also benefits from these techniques since our model needs to see both information packed in a long sequence and rare words that only appear in some tables.

VII. Experiments and Analysis

We conduct experiments [2] on two scenarios: (1) an in-domain scenario of NLIDB trained and evaluated on WikiSQL dataset [49], and (2) a cross-domain scenario where we evaluate our zero-shot learning (trained on WikiSQL) performances on OVERNIGHT dataset [43].

We use three metrics for evaluating the query synthesis accuracy: **(1) Logical-form accuracy**. We compare the synthesized SQL query against the ground truth for whether they agree token-by-token in their logical form, as proposed in [49]. **(2) Query-match accuracy**. Like logical-form accuracy, except that we convert both synthesized SQL query and the ground truth into canonical representations before comparison. **(3) Execution accuracy**. We execute both the synthesized query and the ground truth query and compare whether the results agree, as proposed in [49].

A. WikiSQL

WikiSQL contains 87673 records of natural language questions, SQL queries, and 26521 database tables. Since tables are not shared among the train/validation/test splits, models evaluated on WikiSQL are supposed to generalize to new questions and database schemas.

1) Mention Detection Performance: We use string match with edit distances and semantic distances to detect mentions that are context-free, and adversarial binary classifier using ℓ_2-norm, $\alpha = 1$, $\beta = 0$ (Section IV-A) to detect mentions that heavily rely on the context. Note that database-specific knowledge is not used in WikiSQL for fair comparisons. First of all, we compare our mention detection performance with TypeSQL, which employs a template-based approach to formalize the task into a slot filling problem. The sketch is:

```
SELECT $AGG $SELECT_COL
WHERE $COND_COL $OP $COND_VAL(AND $COND_COL $OP $COND_VAL)*
```

where each component, such as `$AGG`, `$SELECT_COL`, and `WHERE` clause (including operator `$OP`), is identified separately.

Our model has a pre-processing step where schema-related information is detected through mention detection, `$COND_COL` and `$COND_VAL` that are emphasized in the sketch involve schema-related information. We evaluate the accuracy of canonical representation matches on `$COND_COL` and `$COND_VAL` between the synthesized SQL and the ground truth, and our accuracy is 91.8%, which outperforms the state-of-the-art TypeSQL 87.9%.

Our NLIDB model is a seq2seq model, and mention detection is serving as a pre-processing component, which is only part of our contribution. Since our method learns the structure

[2] Our code is publicly available at https://github.com/VV123/NLIDB_gradient

Column	Question
date	***When did*** *the Baltimore Ravens play at home?*
venue	***Where*** *was the game played on 20 May?*
player	*Who* ***is the golfer*** *that golfs for Northern Ireland?*
competition description	*What was her final score on* ***the ribbon apparatus****?*

TABLE I: Mention detection using adversarial text method.

of the output by a seq2seq model itself, the `$AGG` and `$OP` are inferred by the seq2seq model (we believe they are part of the structure).

Case Studies There are many questions in WikiSQL that do not have straightforward indicators of column names. To prove that our method can detect mentions by semantic meaning, we present four real examples of mention detection by our adversarial method in Table I. Our method is able to identify "*date*" by its question word "*when did*", and "*venue*" by its question word "*where*". Column "*player*" can be detected by its synonyms "*golfer*", and implicitly mentioned column "*competition description*" can also be detected.

We further justify that our adversarial method is able to pinpoint the term of a mention accurately by its adversarial gradient. Even a column name is mentioned as a combination of several discontinuous words, and our method is able to detect the mention terms by their semantic meaning. In Figure 7, we show three examples of adversarial gradients with respect to both word-level input and character-level input. Both word-level input and character-level input share the same trend. In the first example, our model is able to identify column " [year] " is in fact mentioned in the question and the mention term is around the term of " *2008* ". In the second example, column " [candidates] " is mentioned by its singular form " *candidate* ". The gradient norm of all the words are small except " *candidate* ", which means our method is able to pinpoint the column mention precisely. In the third example, column " [years in toronto] " is mentioned by highest gradient words " *toronto* " and " *2006-07* " ("*toronto team in 2006-07*" as a continuous span). Even though " *year* " is not explicitly mentioned in the question, our model is able to infer the meaning of " *year* " by " *2006-07* ".

2) Training Details: For the encoder and decoder of our sequence-to-sequence model, each has one layer of GRU with a hidden state size of 400 and $2 * 400$, respectively. The tied embedding weights are shared in the input and output layers of both encoder and decoder. We initialize the embedding layer with pre-trained GloVe embedding (dimension $D = 300$). Symbols introduced by annotation (e.g., c_1 and v_1) are also treated as tokens, each of them being represented by the concatenation of the embeddings of an annotation type (e.g, c and v) and an index. Also, the embeddings of an annotation type and an index are randomly initialized with $D' = 150$, so the concatenation has a dimension of $D = 300$. The other out-of-vocabulary token is initialized with a random vector of $D = 300$. We use gradient clipping with a threshold 5.0 for training and beam search with width 5 for inference.

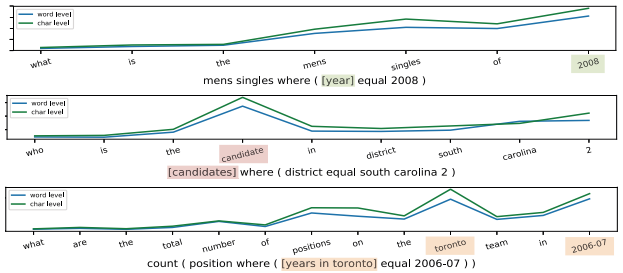

Fig. 7: Examples in WikiSQL defined in Figure 5

3) Evaluation: We compare our method with previous methods through three aforementioned metrics: accuracies in terms of logical form exact match, exact query match, and the results of query execution. As shown in Table II, our result outperforms these previous methods, including the state-of-the-art TypeSQL. This demonstrates that our method is able to generalize to unseen tables, since in WikiSQL database, tables are not shared among train and test splits.

We note that TypeSQL achieves high accuracy in the "content-sensitive" setting where it queries Freebase when handling natural language questions in training as well as in inferencing, while our method achieves higher accuracy without querying Freebase.

4) Ablation: In Table II, we demonstrate our contribution by performing ablation with different components of our model. Removing each component of our method leads to a decrease in performance: The removal of (1) half of GRU hidden size (hidden size 200 for encoder and 400 for decoder), (2) copy mechanism, (3) column name appending (e.g., using column substitution instead), and (4) encoding of table header, each respectively decreases performance on the test set.

Since the annotation and sequence modeling are separated in our framework, we test our annotation method combined with the transformer model [3], an alternative and state-of-the-art architecture for sequence modeling such as machine translation. With the same annotation, the transformer model shows worse performance. We hypothesize that the reason behind this is the difference between NLIDB task and translation tasks: NLIDB has a considerable difference between vocabulary sizes in source space and target space.

We also report the transformation error incurred by the annotation process. As shown in Table III, we report the exact query match accuracy Acc_{qm} before and after the annotation recovery step (transferring s^a to s). Our experiments have shown that our automatic annotation will not hurt the performance; on the contrary, it increases the accuracy.

B. Zero-shot transfer-ability

For cross-domain evaluation, we evaluate the transferability of our model that is trained on one domain (WikiSQL) and tested on other unseen domains to assess its transfer-ability. This task is challenging since the model is required to model domains not seen before.

[3]We use transformer from https://github.com/tensorflow/tensor2tensor

		Dev			Test		
		Acc_{lf}	Acc_{qm}	Acc_{ex}	Acc_{lf}	Acc_{qm}	Acc_{ex}
Previous Method	Seq2SQL [49]	52.5%	53.5%	62.1%	50.8%	51.6%	60.4%
	SQLNet [46]	-	63.2%	69.8%	-	61.3%	68.0%
	PT-MAML [15]	63.1%	-	68.3%	62.8%	-	68.0%
	Coarse2Fin [5]	-	-	-	71.7%	-	78.5%
	TypeSQL* [48]	-	79.2%	85.5%	-	75.4%	82.6%
Annotation	Annotated Seq2seq (Ours)	75.5%	75.4%	83.1%	**75.6%**	**75.6%**	**83.6%**
	– Half Hidden Size	74.8%	74.8%	82.5%	75.0%	75.0%	82.9%
	– Column Name Appending	74.6%	74.5%	81.9%	74.5%	74.5%	82.1%
	– Copy Mechanism	74.2%	74.2%	81.4%	74.4%	74.4%	81.9%
	– Table Header Encoding	74.9%	74.8%	81.8%	74.6%	74.6%	81.8%
	– seq2seq + Transformer	68.8%	68.9%	77.4%	69.1%	69.2%	78.4%

TABLE II: Comparison of models. lf, qm, ex represent logical forms, exact query match, and query execution accuracy, respectively. Performances on the first block are copied from the corresponding papers. "–" and "+" mean removing or adding one component from our best approach respectively for ablation. *We report the results of content sensitive TypeSQL for fair comparisons.

	Dev		Test	
	Acc_{before}	Acc_{after}	Acc_{before}	Acc_{after}
Annotated Seq2seq (Ours)	74.8%	75.4%	75.0%	75.6%
– Half Hidden Size	74.5%	74.8%	74.6%	75.0%
– Table Header Encoding	74.5%	74.8%	74.2%	74.6%
– Column Name Appending	74.1%	74.5%	74.0%	74.5%
– Copy Mechanism	73.7%	74.2%	73.8%	74.4%

TABLE III: "Recovery" performances on WikiSQL dataset. Acc_{before} (Acc_{after}) denotes query match accuracy before (after) annotation recovery.

1) OVERNIGHT: OVERNIGHT [43] is generated from a grammar and annotated with natural language through crowdsourcing. Since OVERNIGHT is not equipped with predefined databases, we use the annotated OVERNIGHT dataset from [42], where the queries are converted to SQL.

We evaluate the transfer learning ability of our model that trained on WikiSQL to be evaluated on five sub-domains of the annotated OVERNIGHT dataset (include both train and test splits). Since OVERNIGHT SQL sketch is highly variant and different from WikiSQL sketch, we make a reasonable assumption that only the sketch compatible ones are considered in the transfer-ability evaluation. Table IV(a) presents our transfer-ability performance. Transfer accuracy is calculated over sketch-compatible records and non-compatible records/operations are discarded. Our model exhibits high transfer-ability with zero-shot learning.

Since the transfer model is trained on WikiSQL only, we further test our model's learning ability on OVERNIGHT separately. We train our model on OVERNIGHT train split, and evaluate on test split, and achieve an accuracy of 81.4%, which shows the robustness of our model.

2) ParaphraseBench: ParaphraseBench [40] is a benchmark to evaluate the robustness of NLIDBs and explicitly test different linguistic variations. We adopt the same setting as Section VII-B1 (non-compatible operations excluded) and our transfer evaluations (Table IV(b)) justify the robustness of our design.

Sub-domain	BASKETBALL	CALENDAR	HOUSING	RECIPES	RESTAURANTS	OVERALL
Acc_{qm}	39.7%	76.3%	51.5%	81.8%	79.3%	60.6%

(a) OVERNIGHT

Paraphrase	NAIVE	SYNTACTIC	LEXICAL	MORPHOLOGICAL	SEMANTIC	MISSING
Acc_{qm}	96.49%	92.98%	57.89%	87.72%	56.14 %	43.86%

(b) ParaphraseBench

TABLE IV: Transfer Accuracy

VIII. CONCLUSION

In this work, we propose an NLIDB converting natural language questions to structured queries (e.g., SQL) for relational databases. The main contribution of our work is to separate database-specific information from the natural language questions themselves and learn knowledge of the natural language and database-specific knowledge separately. Our experimental analysis ascertains the effectiveness of our approach over the state-of-the-art approaches on multiple datasets.

REFERENCES

[1] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(01):29–81, 1995.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

[3] F. Basik, B. Hättasch, A. Ilkhechi, A. Usta, S. Ramaswamy, P. Utama, N. Weir, C. Binnig, and U. Cetintemel. Dbpal: A learned nl-interface for databases. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1765–1768. ACM, 2018.

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of EMNLP*, pages 1724–1734, 2014.

[5] L. Dong and M. Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of ACL*, Melbourne, Australia, 2018.

[6] N. Esfandiari, M. R. Babavalian, A.-M. E. Moghadam, and V. K. Tabar. Knowledge discovery in medicine: Current issue and future trend. *Expert Systems with Applications*, 41(9):4434–4463, 2014.

[7] A. Giordani and A. Moschitti. Translating questions to sql queries with generative parsers discriminatively reranked. *COLING*, pages 401–410, 2012.

[8] Z. Gong, W. Wang, B. Li, D. Song, and W.-S. Ku. Adversarial texts with gradient methods. *arXiv preprint arXiv:1801.07175*, 2018.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[10] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

[11] J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of ACL*, volume 1, pages 1631–1640, 2016.

[12] R. C. Gunasekara, D. Nahamoo, L. C. Polymenakos, J. Ganhotra, and K. P. Fadnis. Quantized-dialog language model for goal-oriented conversational systems. *arXiv preprint arXiv:1812.10356*, 2018.

[13] M. Henderson, B. Thomson, and S. Young. Word-based dialog state tracking with recurrent neural networks. In *Proceedings of SIGDIAL*, pages 292–299, 2014.

[14] J. Herzig and J. Berant. Neural semantic parsing over multiple knowledge-bases. In *Proceedings of ACL 2017*, pages 623–628, 2017.

[15] P.-S. Huang, C. Wang, R. Singh, W.-t. Yih, and X. He. Natural language to structured query generation via meta-learning. In *Proceedings of NAACL HLT 2018*, 2018.

[16] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. Learning a neural semantic parser from user feedback. In *Proceedings of ACL*, volume 1, pages 963–973, 2017.

[17] R. Jia and P. Liang. Data recombination for neural semantic parsing. In *Proceedings of ACL*, 2016.

[18] R. Jia and P. Liang. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of EMNLP 2017*, pages 2021–2031, 2017.

[19] S. Lee. Structured discriminative model for dialog state tracking. In *Proceedings of the SIGDIAL 2013 Conference*, pages 442–451, 2013.

[20] F. Li and H. 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. ACM, 2014.

[21] J. Li, W. Wang, W.-S. Ku, Y. Tian, and H. Wang. Spatialnli: A spatial domain natural language interface to databases using spatial comprehension. *arXiv preprint arXiv:1908.10917*, 2019.

[22] Y. Li, H. Yang, and H. Jagadish. Nalix: an interactive natural language interface for querying xml. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 900–902. ACM, 2005.

[23] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi. Deep text classification can be fooled. In *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI*, pages 4208–4215, 2018.

[24] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

[25] T. Miyato, A. M. Dai, and I. Goodfellow. Adversarial training methods for semi-supervised text classification. *ICLR*, 2017.

[26] N. Mrkšić, D. Ó. Séaghdha, B. Thomson, M. Gasic, P.-H. Su, D. Vandyke, T.-H. Wen, and S. Young. Multi-domain dialog state tracking using recurrent neural networks. In *Proceedings of ACL*, volume 2, pages 794–799, 2015.

[27] N. Mrkšić, D. Ó. Séaghdha, T.-H. Wen, B. Thomson, and S. Young. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of ACL*, volume 1, pages 1777–1788, 2017.

[28] E. Ngai, Y. Hu, Y. Wong, Y. Chen, and X. Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3):559–569, 2011.

[29] E. W. Ngai, L. Xiu, and D. C. Chau. Application of data mining techniques in customer relationship management: A literature review and classification. *Expert systems with applications*, 36(2):2592–2602, 2009.

[30] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of ACL*, pages 1470–1480, 2015.

[31] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[32] A.-M. Popescu, A. Armanasu, O. Etzioni, D. Ko, and A. Yates. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, page 141. Association for Computational Linguistics, 2004.

[33] A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM, 2003.

[34] H. Ren, W. Xu, Y. Zhang, and Y. Yan. Dialog state tracking using conditional random fields. In *Proceedings of the SIGDIAL 2013 Conference*, pages 457–461, 2013.

[35] S. Samanta and S. Mehta. Towards crafting text adversarial samples. *arXiv preprint arXiv:1707.02812*, 2017.

[36] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *ICLR*, 2017.

[37] Y. Su and X. Yan. Cross-domain semantic parsing via paraphrasing. In *Proceedings of EMNLP*, pages 1235–1246, 2017.

[38] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[39] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[40] P. Utama, N. Weir, F. Basik, C. Binnig, U. Cetintemel, B. Hättasch, A. Ilkhechi, S. Ramaswamy, and A. Usta. An end-to-end neural natural language interface for databases. *arXiv preprint arXiv:1804.00401*, 2018.

[41] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[42] W. Wang, Y. Tian, H. Xiong, H. Wang, and W.-S. Ku. A transfer-learnable natural language interface for databases. *arXiv preprint arXiv:1809.02649*, 2018.

[43] W. Y. Wang and D. Yang. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using #petpeeve tweets. In *EMNLP*, pages 2557–2563, 2015.

[44] Y. Wang, J. Berant, and P. Liang. Building a semantic parser overnight. In *Proceedings of ACL*, pages 1332–1342, 2015.

[45] T.-H. Wen, D. Vandyke, N. Mrkšić, M. Gasic, L. M. R. Barahona, P.-H. Su, S. Ultes, and S. Young. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of EACL*, volume 1, pages 438–449, 2017.

[46] X. Xu, C. Liu, and D. Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

[47] S. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech & Language*, 24(2):150–174, 2010.

[48] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. R. Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of NAACL-HLT*, pages 588–594, 2018.

[49] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.