TURING: an Accurate and Interpretable Multi-Hypothesis Cross-Domain Natural Language Database Interface

Peng Xu[*], **Wenjie Zi**[*], **Hamidreza Shahidi**, **Ákos Kádár**, **Keyi Tang**,
Wei Yang, **Jawad Ateeq**, **Harsh Barot**, **Meidan Alon**, **Yanshuai Cao**
Borealis AI
{peng.z.xu, wenjie.zi, hamidreza.shahidi, akos.kadar, keyi.tang}@borealisai.com
{wei.yang, jawad.ateeq, harsh.barot, meidan.alon, yanshuai.cao}@borealisai.com

Abstract

A natural language database interface (NLDB) can democratize data-driven insights for non-technical users. However, existing Text-to-SQL semantic parsers cannot achieve high enough accuracy in the cross-database setting to allow good usability in practice. This work presents TURING[1], a NLDB system toward bridging this gap. The cross-domain semantic parser of TURING with our novel value prediction method achieves 75.1% execution accuracy, and 78.3% top-5 beam execution accuracy on the Spider validation set (Yu et al., 2018b). To benefit from the higher beam accuracy, we design an interactive system where the SQL hypotheses in the beam are explained step-by-step in natural language, with their differences highlighted. The user can then compare and judge the hypotheses to select which one reflects their intention if any. The English explanations of SQL queries in TURING are produced by our high-precision natural language generation system based on synchronous grammars.

1 Introduction

Today a vast amount of knowledge is hidden in structured datasets, not directly accessible to non-technical users who are not familiar with the corresponding database query language like SQL or SPARQL. Natural language database interfaces (NLDB) enable everyday users to interact with databases (Zelle and Mooney, 1996; Popescu et al., 2003; Li and Jagadish, 2014; Zeng et al., 2020). However, correctly translating natural language to executable queries is challenging, as it requires resolving all the ambiguities and subtleties of natural utterances for precise mapping. Furthermore, quick deployment and adoption for NLDB require zero-shot transfer to new databases without an in-domain text-to-SQL parallel corpus, *i.e.* cross-database semantic parsing (SP), making the translation accuracy even lower. Finally, unlike in other NLP applications where partially correct results can still provide partial utility, a SQL query with a slight mistake could cause negative utility if trusted blindly or confusing to users.

The recent Spider benchmark (Yu et al., 2018a) captures this cross-domain problem, and the state-of-the-art methods merely achieve around 70% execution accuracy at the time of this submission [2]. Meanwhile, generalization to datasets collected under different protocols is even weaker (Suhr et al., 2020). Finally, users generally have no way to know if the NLDB made a mistake except in very obvious cases. The high error rate combined with the overall system's opacity makes it hard for users to trust any output from the NLDB.

Our key observation is that our model's top-5 accuracy on Spider is 78.3%, significantly higher than the previous best single-model method at around 68%, and our own top-1 accuracy. Top-5 accuracy is the proportion of times when one of the top five hypotheses from beam-search inference is correct (in execution accuracy evaluation). For top-5 accuracy to be relevant in practice, a non-technical user needs to be able to pick the correct hypothesis from the candidate list. To this end, we design a feedback system that can unambiguously explain the top beam-search results while presenting the differences intuitively and visually. Users can then judge which, if any, of the parses correctly reflects their intentions. The explanation system uses a hybrid of two synchronous context-free grammars, one shallow and one deep. Together, they achieve good readability for the most frequent

[*] Equal contribution

[1] System demo at https://turing.borealisai.com/; video at https://vimeo.com/537429187/9a5d41f446

[2] https://yale-lily.github.io/spider

Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations, pages 298–305, August 1st - August 6th, 2021.

query patterns while near-complete coverage overall.

Our system, TURING, is not only interpretable, but also a highly accurate cross-domain NLDB. Our semantic parser is based on the one in Xu et al. (2020), which does not handle value prediction like many other previous state-of-the-art models on Spider. Compared to previous executable semantic parsers, we achieve significant gains with a number of techniques, but predominantly by drastically simplifying the learning problem in value prediction. The model only needs to identify the text span providing evidence for the ground-truth value. The noisy long tail text normalization step required for producing the actual value is offloaded to a deterministic search phase in post-processing.

In summary, this work presents two steps towards a more robust NLDB:

1. A state-of-the-art text-to-SQL parsing system with the best top-1 execution accuracy on the Spider development set.

2. A way to relax usability requirement from top-1 accuracy to top-k accuracy by explaining the different hypotheses in natural language with visual aids.

2 System Overview

As shown in Figure 1, TURING's interface has two main components: the database browser showing schema and selected database content, and the search panel where the users interact with the parser. Figure 1 caption describes the typical user interaction using an example.

Behind the front-end interface, TURING consists of an executable cross-domain semantic parser trained on Spider that maps user utterances to SQL query hypotheses, the SQL execution engine that runs the queries to obtain answers, and the explanation generation module that produces the explanation text and the meta-data powering explanation highlighting. The next sections will describe the semantic parsing and explanation modules.

3 Semantic Parser

The backbone of TURING is a neural semantic parser which generates an executable SQL query T given a user question Q and the database schema $\mathcal{S}$. We follow the state-of-the-art system (Xu et al., 2020), but extend it to generate executable SQL query instead of ignoring values in the SQL query, like many other top systems (Wang et al., 2019; Guo et al., 2019) on the Spider leaderboard.

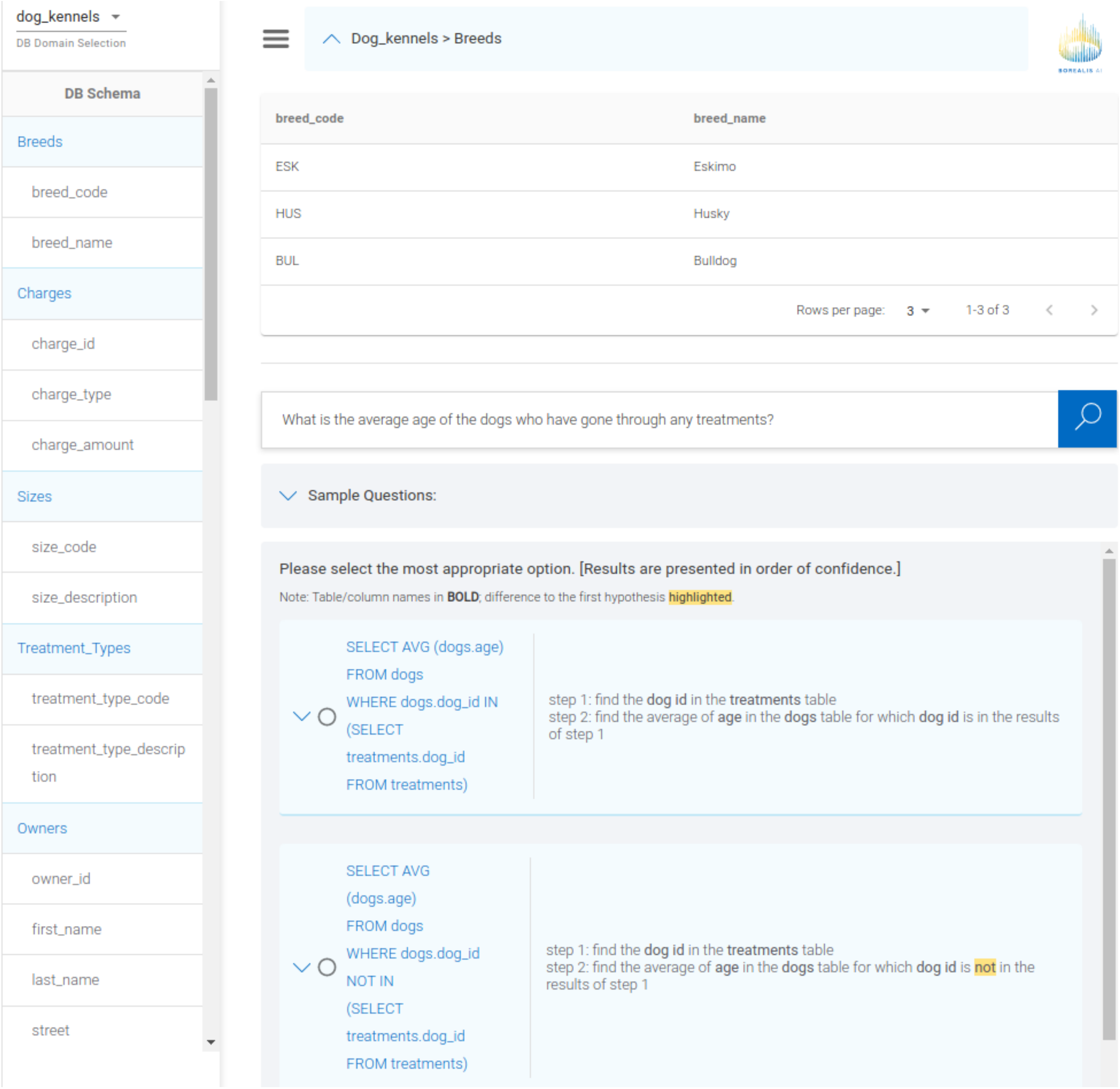
On the high-level, our SP adopts the grammar-based framework following TranX (Yin and Neubig, 2018) with an encoder-decoder neural architecture. A grammar-based transition system is designed to turn the generation process of the SQL abstract syntax tree (AST) into a sequence of tree-constructing actions to be predicted by the parser. The encoder f_{enc} jointly encodes both the user question $Q = q_1 \ldots q_{|Q|}$ and database schema $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\}$ consisting of tables and columns in the database. The decoder f_{dec} is a transition-based abstract syntax decoder, which uses the encoded representation $\mathcal{H}$ to predict the target SQL query T. The decoder also relies on the transition system to convert the AST constructed by the predicted action sequences to the executable surface SQL query.

To alleviate unnecessary burden on the decoder, we introduce two novel modifications to the transition system to handle the schema and value decoding. With simple, but effective value-handling, inference and regularization techniques applied on this transition system, we are able to push the execution accuracy much higher for better usability.

3.1 Transition System

Our transition system has four types of action to generate the AST, including (1) **ApplyRule**$[r]$ which applies a production rule r to the latest generated node in the AST; (2) **Reduce** which completes the generation of the current node; (3) **SelectColumn**$[c]$ which chooses a column c from the database schema $\mathcal{S}$; (4) **CopyToken**$[i]$ which copies a token q_i from the user question Q.

There are two key distinctions of our transition system with the previous systems. First, our transition system omits the action type **SelectTable** used by other transition-based SP systems (Wang et al., 2019; Guo et al., 2019). This is made possible by attaching the corresponding table to each column, so that the tables in the target SQL query can be deterministically inferred from the predicted columns. Second, we simplify the value prediction by always trying to copy from the user question, instead of applying the **GenToken**$[v]$ action (Yin and Neubig, 2018) which generates tokens from a large vocabulary or choose from a pre-processed picklist (Lin et al., 2020). Both of the changes constrain the output space of the decoder to ease the



dog_kennels

DB Domain Selection

DB Schema

- Breeds
  - breed_code
  - breed_name
- Charges
  - charge_id
  - charge_type
  - charge_amount
- Sizes
  - size_code
  - size_description
- Treatment_Types
  - treatment_type_code
  - treatment_type_description
- Owners
  - owner_id
  - first_name
  - last_name
  - street

Dog_kennels > Breeds

breed_code	breed_name
ESK	Eskimo
HUS	Husky
BUL	Bulldog

Rows per page: 3 1-3 of 3

What is the average age of the dogs who have gone through any treatments?

Sample Questions:

Please select the most appropriate option. [Results are presented in order of confidence.]

Note: Table/column names in **BOLD**; difference to the first hypothesis highlighted.

○

```
SELECT AVG (dogs.age)
FROM dogs
WHERE dogs.dog_id IN
(SELECT
treatments.dog_id
FROM treatments)
```

step 1: find the **dog id** in the **treatments** table
step 2: find the average of **age** in the **dogs** table for which **dog id** is in the results of step 1

○

```
SELECT AVG
(dogs.age)
FROM dogs
WHERE dogs.dog_id
NOT IN
(SELECT
treatments.dog_id
FROM treatments)
```

step 1: find the **dog id** in the **treatments** table
step 2: find the average of **age** in the **dogs** table for which **dog id** is not in the results of step 1

Figure 1: TURING system in action: the user selected database "Dog_kennels"; the left and top panels show the database schema and table content. The user then entered "What is the average age of the dogs who have gone through any treatments?" in the search box. This question is run through the semantic parser producing multiple SQL hypotheses from beam-search, which are then explained step-by-step as shown. The differences across the hypotheses are highlighted. The tokens corresponding to table and columns are in bold. If there were more valid hypotheses, a "Show more" button would appear to reveal the additional ones.

learning process, but the latter change unrealistically assumes that the values are always explicitly mentioned in the question. To retain the generation flexibility without putting excessive burden on the decoder, we propose a conceptually simple but effective strategy to handle the values next.

3.2 Handling Values

Value prediction is a challenging, but crucial component of NLDBs, however, only limited efforts are committed to handling values properly in the current cross-domain SP literature. Value mentions are usually noisy, if mentioned explicitly at all, requiring commonsense or domain knowledge to be inferred. On the other hand, the number of possible values in a database can be huge, leading to sparse learning signals if the model tries to choose from the possible value candidates.

Instead of attempting to predict the actual values directly, our SP simply learns to identify the input text spans providing evidence for the values. As mentioned earlier, we introduce the **CopyToken** action to copy an input span from the user question, indicating the clues for this value. The ground-truth

CopyToken$[i]$ actions are obtained from a tagging strategy based on heuristics and fuzzy string matching between the user question and the gold values. As a result, the decoder is able to focus on understanding the question without considering other complexities of the actual values which are difficult to learn. If the values are only implicitly mentioned in the user question, nothing is copied from the user question. We leave the identification of the actual values to a deterministic search-based inference in post-processing, after the decoding process. This yields a simpler learning task as the neural network does not need to perform domain-specific text normalization such as mapping "female" to "F" for some databases.

Given the schema, the predicted SQL AST and the database content, the post-processing first identifies the corresponding column type (*number, text, time*), operation type (*like, between, >, <, =, ...*), and aggregation type (*count, max, sum, ...*). Based on these types, it infers the type and normalization required for the value. If needed, it then performs fuzzy-search in the corresponding column's values in the database. When nothing is copied, a default value is chosen based on some heuristics (*e.g.*, when there exist only two element "Yes" and "No" in the column, the default value is "Yes"); otherwise, the most frequent element in the column is chosen. Searching the database content can also be restricted to a picklist for privacy reasons like previous works (Zeng et al., 2020; Lin et al., 2020).

Another benefit of this simple value handling strategy is the ease to explain. The details are presented in the Sec. 4.

3.3 Encoder-Decoder

Our encoder architecture follows Xu et al. (2020). The encoder, f_{enc}, maps the user question Q and the schema $\mathcal{S}$ to a joint representation $\mathcal{H} = \{\phi_1^q, \dots, \phi_{|Q|}^q\} \cup \{\phi_1^s, \dots, \phi_{|\mathcal{S}|}^s\}$. It contextualizes the question and schema jointly through both the RoBERTA-Large model similar to (Guo et al., 2019), as well as through the additional sequence of 24 relation-aware transformer (RAT) (Wang et al., 2019) layers. As mentioned in Section 3.1, tables are not predicted directly but inferred from the columns, so we augment the column representations by adding the corresponding table representations after the encoding process.

We use a LSTM decoder f_{dec} to generate the action sequence A. Formally, the generation process can be formulated as $\Pr(A|\mathcal{H}) = \prod_t \Pr(a_t|a_{<t}, \mathcal{H})$ where $\mathcal{H}$ is the encoded representations outputted by the encoder f_{enc}. The LSTM state is updated following Wang et al. (2019): $\boldsymbol{m}_t, \boldsymbol{h}_t = f_{\text{LSTM}}([\boldsymbol{a}_{t-1} \| \boldsymbol{z}_{t-1} \| \boldsymbol{h}_{p_t} \| \boldsymbol{a}_{p_t} \| \boldsymbol{n}_{p_t}], \boldsymbol{m}_{t-1}, \boldsymbol{h}_{t-1})$, where $\boldsymbol{m}_t$ is the LSTM cell state, $\boldsymbol{h}_t$ is the LSTM output at step t, $\boldsymbol{a}_{t-1}$ is the action embedding of the previous step, $\boldsymbol{z}_{t-1}$ is the context representation computed using multi-head cross-attention of $\boldsymbol{h}_{t-1}$ over $\mathcal{H}$, p_t is the step corresponding to the parent AST node of the current node, and $\boldsymbol{n}$ is the node type embedding. For **ApplyRule**$[r]$, we compute $\Pr(a_t = \textbf{ApplyRule}[r]|a_{<t}, \mathcal{H}) = \text{softmax}_r(g(\boldsymbol{z}_t))$ where $g(\cdot)$ is a 2-layer MLP. For **SelectColumn**$[c]$, we use the memory augmented pointer network following Guo et al. (2019). For **CopyToken**$[i]$, a pointer network is employed to copy tokens from the user question Q with a special token indicating the termination of copy.

3.4 Column Label Smoothing

One of the core challenges for cross-domain SP is to generalize to unseen domains without overfitting to some specific domains during training. Empirically, we observe that applying uniform label smoothing (Szegedy et al., 2016) on the objective term for predicting **SelectColumn**$[c]$ can effectively address the overfitting problem in the cross-domain setting. Formally, the cross-entropy for a ground-truth column c^* we optimize becomes $(1-\epsilon) * \log p(c^*) + \frac{\epsilon}{K} * \sum_c \log p(c)$, where K is the number of columns in the schema, ϵ is the weight of the label smoothing term, and $p(\cdot) \triangleq \Pr(a_t = \textbf{SelectColumn}[\cdot]|a_{<t}, \mathcal{H})$.

3.5 Weighted Beam Search

During inference, we use beam search to find the high-probability action sequences. As mentioned above, column prediction is prone to overfitting in the cross-domain setting. In addition, value prediction is dependent on the column prediction, that is, if a column is predicted incorrectly, the associated value has no chance to be predicted correctly. As a result, we introduce two hyperparameters controlling influence based on the action types in the beam, with a larger weight $\alpha > 1$ for **SelectColumn** and a smaller weight $0 < \beta < 1$ for **CopyToken**.

4 Explanation Generation

The goal of the explanation generation system is to unambiguously describe what the semantic parser understands as the user's command and allow the user to easily interpret the differences across the multiple hypotheses. Therefore, unlike a typical dialogue system setting where language generation diversity is essential, controllability and consistency are of primary importance. The generation not only needs to be 100% factually correct, but the differences in explanation also need to reflect the differences in the predicted SQLs, no more and no less. Therefore, we use a deterministic rule-based generation system instead of a neural model.

Our explanation generator is a hybrid of two synchronous context-free grammar (SCFG) systems combined with additional heuristic post-processing steps. The two grammars trade off readability and coverage. One SCFG is shallow and simple, covering the most frequent SQL queries; the other is deep and more compositional, covering the tail of query distribution that our SP can produce for completeness. The SCFG can produce SQL and English explanation parallel. Given a SQL query, we parse it under the grammar to obtain a derivation, which we then follow to obtain the explanation text. At inference time, for a given question, if any of the SQL hypotheses cannot be parsed using the shallow SCFG, then we move onto the deep one.

4.1 Details of the Grammars

Using the deep SQL syntax trees allows almost complete coverage on the Spider domains. However, these explanations can be unnecessarily verbose as the generation process faithfully follows the re-ordered AST without 1.) compressing repeated mentions of schema elements when possible 2.) summarizing tedious details of the SQL query into higher level logical concepts. Even though these explanations are technically correct, practical explanation should allow users to spot the difference between queries easily. To this end, we design the shallow grammar similarly to the template-based explanation system in Elgohary et al. (2020), which simplifies the SQL parse trees by collapsing large subtrees into a single tree fragment. In the resulting shallow parses production rules yield non-terminal nodes corresponding to 1.) anonymized SQL templates 2.) UNION, INTERSECT, or EXCEPT operations of two templates 3.) or a template pattern followed by ORDER-BY-LIMIT clause. Our shallow but wide grammar has 64 rules with those non-terminal nodes. The pre-terminal nodes are place-holders in the anonymized SQL queries such as Table name, Column name, Aggregation operator and so on. Finally, the terminal nodes are the values filling in the place holders. The advantage of this grammar is that each high-level SQL template can be associated with an English explanation template that reveals the high level logic and abstracts away from the details in the concrete queries. To further reduce the redundancy, we make assumptions to avoid unnecessarily repeating table and column names. Table. 1 showcases some rules from the shallow SCFG and one example of explanation. In practice, around 75% of the examples in the Spider validation set have all beam hypotheses from our SP model parsable by the shallow grammar, with the rest handled by the deep grammar. The deep grammar has less than 50 rules. But because it is more compositional, it covers 100% of the valid SQLs that can be generated by our semantic parser. Some sample explanation by the deep grammar can be found in Table. 2.

Finally, whenever the final value in the query differs from original text span due to post-processing, a sentence in the explanation states the change explicitly for clarity. For example, "*'Asian' in the question is matched to 'Asia' which appears in the column Continent.*"

5 Quantitative Evaluations

Implementation Details. We apply the DT-Fixup technique from (Xu et al., 2020) to train our semantic parser and mostly re-use their hyperparamters. The weight of the column label smoothing term ϵ is 0.2. Inference uses a beam size of 5 for the beam search. We set the column weight as $\alpha = 3$ and the value weight as $\beta = 0.1$.

Dataset. We use Spider (Yu et al., 2018b), a complex and cross-domain Text-to-SQL semantic parsing benchmark, which has 10,180 questions, 5,693 queries covering 200 databases in 138 domains. All our experiments are evaluated based on the development set. We use the execution match *with* values (**Exec**) evaluation metrics.

Results on Spider. We compare TURING with the top systems on the Spider execution leaderboard that have published reports with execution accuracy on the development set as well. As seen from Table 3, our single model significantly outperforms the previous state of the art in terms of **Exec** accu-

```
S -> P
S -> P UNION P
P -> (SELECT <T_0>.<C_0> FROM <T_1> GROUP BY <T_2>.<C_1> HAVING <AOps_0> ( <T_3>.<C_2> ) <WOps_0> <L_0>,
     find the different values of the {<C_0>} in the {<T_1>} whose {<AOps_0>} the {<C_2>} {<WOps_0>} {<L_0>})
```

```
step 1: find the average of product price in the products table
step 2: find the different values of the product type code in the products table
        whose average of the product price is greater than the results of step 1
```

Table 1: Sample shallow grammar production rules and one example explanation.

```
Step 1: find the entries in the employee table whose age is less than 30.0.
Step 2: among these results, for each city of the employee table,
        where the number of records is more than 1, find city of the employee table.
---------------
"30" in the question is converted to 30.
"one" in the question is converted to 1.
```

```
Step 1: find combinations of entries in the employee table, the hiring table and the shop table
        for which employee id of the employee table is equal to employee id of the hiring table
        and shop id of the hiring table is equal to shop id of the shop table.
Step 2: among these results, for each shop id of the shop table,
        find the average of age of the employee table and shop id of the shop table.
```

Table 2: Examples of explanation by the deep grammar. The first example also showcases the additional explanation for value post-processing.

Model	Exec
GAZP + BERT (Zhong et al., 2020)	59.2
Bridge v2 + BERT (Lin et al., 2020)	68.0
Bridge v2 + BERT (ensemble)	70.3
Turing + RoBERTa	**75.1**(best), 73.8 ± 0.7

Table 3: **Exec** accuracy on the Spider development set.

Model	Exec
Turing + RoBERTa	73.8 ± 0.7
w/o. value post-processing	67.2 ± 0.8
w/o. column label smoothing	73.1 ± 1.2
w/o. weighted beam search	73.5 ± 0.7
top 3 in the beam	77.3 ± 0.4
top 5 in the beam	78.3 ± 0.3

Table 4: Ablation study on various techniques used in TURING. We use 5 runs with different random seeds.

racy on the development set.

Ablation Study. Table 4 shows an ablation study of various techniques in TURING. We can see that removing the value post-processing decreases the accuracy significantly, showing that copying alone is not enough due to the mismatch in linguistic variation and the schema specific normalization. The effectiveness of the proposed column label smoothing and weighted beam search are also reflected by the **Exec** accuracy on Spider. Furthermore, simply adding more hypotheses in the beam can significantly boost the coverage of the correct predictions, leading to 4.5% accuracy gain over the top one accuracy. By combining all these techniques together, TURING achieves an overall performance gain above 10% over the previous best single model system (68.0% of Bridge v2). [3]

6 Related Work

Executable Cross-database Semantic Parsing. Early NLDB systems use rule-based parsing (Zelle and Mooney, 1996; Li and Jagadish, 2014) and cannot handle the diversity of natural language in practice. Neural semantic parsing is more promising for coverage but is still brittle in real-world applications where queries can involve novel compositions of learned patterns (Finegan-Dollak et al., 2018; Shaw et al., 2020). Furthermore, to allow plug-and-play on new databases, the underlying semantic parser may not be trained on in-domain parallel corpus but needs to transfer across domains in a zero-shot fashion.

Executable cross-database semantic parsing is even more challenging. Many of the previous work only tackle the cross-domain part, omitting the value prediction problem required for executable queries (Guo et al., 2019; Wang et al., 2019; Choi et al., 2020; Xu et al., 2020). Unlike the output space of predicting the SQL sketch or columns,

[3] Rubin and Berant (2020) updated a version (April 11th 2021) around the time of this submission with a dev accuracy of 75% (missing from the first version), and a test accuracy of 71.1% significantly higher than the original 60.5%.

the value prediction output space is much less constrained. The correct value depends on the source question, the SQL query, the type information of the corresponding column, as well as the database content. This complexity combined with limited training data in standard benchmark datasets like Spider makes the task very difficult. Some previous works directly learn to predict the values (Yin and Neubig, 2018; Guo and Gao, 2020) on WikiSQL (Zhong et al., 2017), but does not generalize in cross-domain settings. On Spider, Zeng et al. (2020) and Lin et al. (2020) build a candidate list of values first and learn a pointer network to select from the list. TURING instead learns a pointer network to identify the input source span that provides evidence for the value instead of directly the value as previously described. Identification of the actual value is offloaded to post-processing. From a system perspective, it is also simpler for a power user of the NLDB to upload a domain-specific term description/mapping which can extend the heuristic-search-based value post-processing instantly rather than relying on re-training.

Query Explanation. Explaining structured query language has been studied in the past (Simitsis and Ioannidis, 2009; Koutrika et al., 2010; Ngomo et al., 2013; Xu et al., 2018). Full NLDB systems can leverage explanations to correct mistakes with user feedback (Elgohary et al., 2020), or to prevent mistakes by giving clarifications (Zeng et al., 2020). However, these methods can only handle cases where the mistake or ambiguity is about the table, column, or value prediction. There is no easy way to resolve structural mistakes or ambiguities if the query sketch is wrong. TURING, on the other hand, offers the potential to recover from such mistakes if the correct query is among the top beam results. This is an orthogonal contribution that could be integrated with other user-interaction modes. Finally, the NaLIR system (Li and Jagadish, 2014) has a similar feature allowing the user to pick from multiple interpretations of the input question. However, NaLIR's interpretation is based on syntactical parses of the question rather than interpreting the final semantic parses directly. A rule-based semantic parser then maps the selected syntactic parse to SQL. As the syntactic parse is not guaranteed to be mapped to the correct SQL, this interpretation does not completely close the gap between what the NLDB performs and what the user thinks it does.

7 Conclusion

We presented TURING, a natural language interface to databases (NLDB) that is accurate, interpretable, and works on a wide range of domains. Our system explains its actions in natural language so that the user can select the right answer from multiple hypotheses, capitalizing on the much higher beam accuracy instead of top-1 accuracy. TURING provides a complementary way to resolve mistakes and ambiguities in NLDB.

Acknowledgments

We appreciate the ACL demo anonymous reviewers for their valuable inputs. We would like to thank Mehrsa Golestaneh and April Cooper for their work on the improved front-end version, `https://turing-app.borealisai.com`, which is not in the scope of this publication. We also would like to thank Wendy Tay and Simon J.D. Prince for their general support.

References

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases.

Ahmed Elgohary, Saghar Hosseini, and Ahmed H. Awadallah. 2020. Speak to your parser: Interactive text-to-sql with natural language feedback. In *Annual Conference of the Association for Computational Linguistics (ACL 2020)*.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

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. *ACL*.

Tong Guo and Huilin Gao. 2020. Content enhanced bert-based text-to-sql generation.

G. Koutrika, A. Simitsis, and Y. E. Ioannidis. 2010. Explaining structured queries in natural language. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 333–344.

F. Li and H. V. Jagadish. 2014. Nalir: an interactive natural language interface for querying relational databases. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *arXiv preprint arXiv:2012.12627*.

Axel-Cyrille Ngonga Ngomo, Lorenz Bhmann, Christina Unger, Jens Lehmann, and Daniel Gerber. 2013. Sorry, i don't speak sparql: translating sparql queries into natural language. In *WWW*, pages 977–988.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157.

Ohad Rubin and Jonathan Berant. 2020. Smbop: Semi-autoregressive bottom-up semantic parsing.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*.

Alkis Simitsis and Yannis E. Ioannidis. 2009. Dbmss should talk back too. *CoRR*, abs/0909.1786.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online. Association for Computational Linguistics.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

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

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

P. Xu, Wei Yang, W. Zi, Keyi Tang, Chengyang Huang, J. Cheung, and Yanshuai Cao. 2020. Optimizing deeper transformers on small datasets: An application on text-to-sql semantic parsing. *ArXiv*, abs/2012.15355.

Pengcheng Yin and Graham Neubig. 2018. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. *arXiv preprint arXiv:1810.02720*.

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. 2018a. 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*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

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

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Jichuan Zeng, Xi Victoria Lin, Steven C.H. Hoi, Richard Socher, Caiming Xiong, Michael Lyu, and Irwin King. 2020. Photon: A robust cross-domain text-to-SQL system. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 204–214, Online. Association for Computational Linguistics.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

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