MT-Teql: Evaluating and Augmenting Neural NLIDB on Real-world Linguistic and Schema Variations

Pingchuan Ma
Hong Kong University of Science and Technology
Hong Kong SAR, China
pmaab@cse.ust.hk

Shuai Wang
Hong Kong University of Science and Technology
Hong Kong SAR, China
shuaiw@cse.ust.hk

ABSTRACT

Natural Language Interface to Database (NLIDB) translates human utterances into SQL queries and enables database interactions for non-expert users. Recently, neural network models have become a major approach to implementing NLIDB. However, neural NLIDB faces challenges due to variations in natural language and database schema design. For instance, one user intent or database conceptual model can be expressed in *various forms*. However, existing benchmarks, using hold-out datasets, cannot provide thorough understanding of how good neural NLIDBs really are in real-world situations and its robustness against such variations. A key difficulty is to annotate SQL queries for inputs under real-world variations, requiring considerable manual effort and expert knowledge.

To systematically assess the robustness of neural NLIDBs without extensive manual effort, we propose MT-Teql, a unified framework to benchmark NLIDBs against real-world language and schema variations. Inspired by recent advances in DBMS metamorphic testing, MT-Teql implements semantics-preserving transformations on utterances and database schemas to generate their variants. NLIDBs can thus be examined for robustness utilizing utterances/schemas and their variants without requiring manual intervention.

We benchmarked nine neural NLIDBs using 62,430 inputs and identified 15,433 defects. We analyzed potential root causes of defects and conducted a user study to show how MT-Teql can assist developers to systematically assess NLIDBs. We further show that the transformed (error-triggering) inputs can be used to augment popular NLIDBs and eliminate 46.5%(±5.0%) errors made by them without compromising their accuracy on standard benchmarks. We summarize lessons from this study that can provide insights to select and design NLIDBs that fit particular usage scenarios.

PVLDB Reference Format:
Pingchuan Ma and Shuai Wang. MT-Teql: Evaluating and Augmenting Neural NLIDB on Real-world Linguistic and Schema Variations. PVLDB, 15(3): 569 - 582, 2022.
doi:10.14778/3494124.3494139

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/MTTeql/MT-Teql and http://bit.ly/MT-Teql-data.

Proceedings of the VLDB Endowment, Vol. 15, No. 3 ISSN 2150-8097.
doi:10.14778/3494124.3494139

Corresponding author: Shuai Wang {shuaiw@cse.ust.hk}.

1 INTRODUCTION

Natural Language Interface to Database (NLIDB) aims to directly retrieve data from relational databases using natural language utterances. It is expected to provide a handy interface with which users can access a database without knowledge of SQL grammar or how data is stored in the database. Recent advances show the potential of modern NLIDBs in synthesizing complex nested cross-domain cross-table SQL queries [1, 4, 7, 14, 15, 31, 34, 42].

To date, neural NLIDB becomes a major research line in this area, given the strong power of neural models on language comprehension and generation [16]. Despite the spectacular development of neural NLIDB, however, recent advances [27] in the NLP community have pointed out that only using hold-out datasets is insufficient to faithfully benchmark NLP models. We hence suspect that neural NLIDB, which is generally based on NLP primitives, could face primary challenges due to the high flexibility of natural language utterances and database schemas in the wild. For instance, one user's intent can be expressed in multiple lexically different utterances. Similarly, one conceptual model (in the form of an entity-relationship diagram) can be implemented in multiple structurally different schemas. Users adopt different normal forms for diverse purposes or may leverage an inadequate albeit functional database schema (e.g., lacking foreign key constraints). Given *semantically equivalent* utterances or schemas in different forms, neural NLIDB may yield *inconsistent* outputs. These errors exhibit the weakness of current neural NLIDBs on generalization ability and robustness in realistic noisy scenarios.

Despite the aforementioned concerns, datasets that can comprehensively benchmark the robustness of neural NLIDBs in the wild have not been developed properly. Most benchmarks usually provide queries either on a single table or from one single domain [8, 11, 15, 16, 44, 51, 54]. The authors in [16] point out various implicit assumptions and limitations in different benchmarks. For instance, WikiSQL [54] only contains single-table queries from diverse domains. ATIS [8, 24] and GeoQuery [51] do not consider grouping and ordering operations and focus on single-domains. While MAS [17] contains a wider range of queries, including joining and grouping, linguistic diversity of its utterances is limited, of which all utterances start with "return me". A recent benchmark, Spider [50], covers the most commonly-used query types and makes reasonable assumptions on utterances. Despite the progress made by Spider, this study (Sec. 5) will show that it still does not suffice for evaluating NLIDB systems under linguistic and schema design variations and accuracy benchmarked by Spider often does not align with users' real preferences. Weir et al. [44] manually craft a single-domain single-table benchmark, ParaphraseBench, which particularly focuses on evaluating linguistic variations. In summary, the above assumptions and limitations make existing benchmarks insufficient to evaluate the NLIDB real-world performance.

As a complement to existing benchmarks, we aim to comprehensively benchmark neural NLIDBs in terms of *robustness* on real-world linguistic and schema variations. With recent advances in testing DBMS [28–30], software metamorphic testing-based approaches have become popular in assessing database systems. In general, software metamorphic testing [33] is an invariant property-based testing method that relies on mutation rules referred to as metamorphic relations (MRs; see Sec. 2.2). Metamorphic testing alleviates the difficulty of determining the expected outputs of test inputs (which is often prohibitively expensive), by verifying the target software's behavior consistency under MR-mutated test inputs. As a result, software testing becomes substantially more flexible. As for testing DBMS, existing methods design *semantics-preserving* MRs to transform SQL queries and check output consistency w.r.t. original and transformed queries. Similarly, (neural) NLIDB also has its own query language (natural language utterances) and yields SQL queries for given schemas. Hence, we envision that applying semantics-preserving transformations and checking output consistency can presumably assess NLIDBs without manual labeling.

However, adapting similar testing schemes to evaluate NLIDBs is non-trivial, and the reasons are three-fold. First, transformation schemes for SQL cannot be directly adapted to transforming natural language utterances. Designing tailored semantics-preserving transformations for utterances, without loss of question clarity, is challenging. Second, the problem of NL-SQL translation is generally ill-posed, in the sense that some text inputs are not translatable into SQL queries. Some existing methods, by substituting certain words with synonyms or complex paraphrasing, can likely induce such "unnatural" and "illegitimate" inputs which are undesirable in assessing NLIDB performance in the wild [52] and fail to expose realistic errors. According to our empirical observations on popular NLP behavioral testing tools [27, 45], they are not fully qualified for assessing neural NLIDBs, given that their transformation rules can frequently break the semantics of utterances; see Sec. 6.2. Third, modern neural NLIDBs usually take the joint representations of user utterances and database schemas. Despite the fact that some prior benchmarks, e.g., ParaphraseBench, have shed a light on manually transforming utterances while preserving semantics, NLIDB performance under diverse schemas is still under-explored.

In this paper, we propose MT-Teql, a metamorphic testing (MT)-based framework to extensively benchmark NLIDB without manual effort. MT-Teql transforms seed inputs (utterance-schema pairs) via a comprehensive set of metamorphic relations (MRs), where each MR specifies a *semantics-preserving* transformation scheme toward either utterances or schemas. By comparing SQL queries derived from the original and transformed inputs, we assess the *consistency* of NLIDB outputs under test.

We extensively studied nine popular neural NLIDBs. We used MT-Teql to generate a large amount of transformed inputs to extend the standard benchmark, Spider-dev, by 60× and enrich distinct schemas by 113×. With synthetic inputs, we detected on average $1,908(\pm 390)$ errors from each NLIDB system, showing that neural NLIDB generally suffers from relatively low robustness w.r.t. real-world utterance and schema variants. We further summarized NLIDB erroneous output patterns revealed in our study and demonstrated how MT-Teql can be used as assessment criteria to help users select proper NLIDB that fits their particular usage scenarios. On the other hand, after augmentation, $46.5\%(\pm 5.0\%)$ errors could be eliminated without compromising benchmark accuracy.

Key Contributions. 1) We launch the first empirical and comprehensive study to benchmark the robustness of neural NLIDBs on real-world linguistic and schema variations. 2) Inspired by recent advances in DBMS testing, our framework, MT-Teql, delivers model-agnostic testing toward (neural) NLIDBs by launching semantics-preserving linguistic and schema transformations without requiring manual efforts. 3) We show surprising findings that popular neural NLIDB models notably suffer from low robustness, which has never been systematically explored. Furthermore, they generally manifest inconsistent robustness in front of MT-Teql. We conduct a user study to show how MT-Teql can assist developers to systematically assess NLIDBs. We also summarize lessons that can be used to diagnose and improve models in real-life usage. 4) We further show that the transformed (error-triggering) inputs can be employed to extend the training dataset of neural NLIDBs and effectively augment their performance.

Open Source. We release the source code of MT-Teql at https://github.com/MTTeql. Models, experimental data and user study are at http://bit.ly/MT-Teql-data.

2 PRELIMINARIES

2.1 Neural NLIDB

Natural Language Interface to Database (NLIDB) has been studied by the database community and the natural language processing community for decades. In the early stages of the area, rule-based systems have been applied to translate natural language to query language [1, 17, 23, 31, 43]. In addition, to date, neural models have become the standard and dominant approach for NLIDB [3–5, 14, 41, 42, 46, 47, 49, 50, 54]. The typical neural architecture leverages a sequence-to-sequence model (Seq2Seq) to translate an input natural language query (i.e., utterance) u into a target SQL query q w.r.t. a schema s. The model generally consists of two encoders of different functionalities and a decoder that generates SQL queries (or domain-specific language for post-translation), each of which extensively uses RNN (or its variants) as building blocks. We now review primary design considerations for neural NLIDB.

Encoding Linguistic Feature. As shown in Fig. 1, the input of a modern neural NLIDB is a human utterance with database schemas. Usually, it first converts utterances and column/table names in schemas into numerical vectors. In contrast, conventional methods, such as NaLIR [17], ATHENA [31], and ATHENA++ [34], parse utterances with pre-defined rules into tree-based representations. More importantly, large-scale, pre-trained language models have been employed in de facto models. For instance, RAT-SQL [42], one state-of-the-art neural NLIDB, employs BERT to encode utterances and table/column names into numerical values. This way, the linguistic features of utterances and schema are obtained for subsequent SQL query generation.

Encoding Structural Feature. In addition to learn linguistic features, structural information also plays a vital role in generating SQL queries. In general, the structural information consists of an intrinsic schema structure and links between utterance tokens and

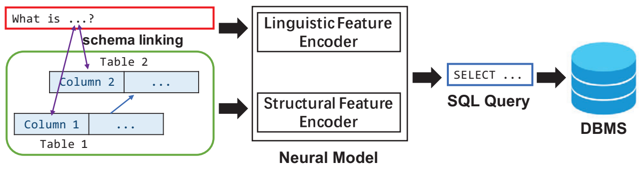

Figure 1: Workflow of modern neural NLIDBs.

schema entities. Schema structure generally includes relations between columns and tables, such as key constraints and column ownership. Compared with schema structure which is explicitly defined in inputs, schema linking is a major challenge to neural NLIDBs due to the diversity and ambiguity of natural language, which requires identifying associations between utterances and database entities, such as a table, column or even row value, as shown in Fig. 1. For a table or column, existing neural NLIDBs generally link them with simple string matching or embedding vector matching. To link a value in the database, i.e., an utterance token instance of a column, knowledge graphs and heuristics are employed to identify such relations. Different methods are used for neural NLIDBs to encode structural information. RAT-SQL and DuoRAT [32] leverage relation-aware transformers to learn different relations. Some recent advances also explore the use of graph neural network (GNN) models to learn these structural information, by treating the schemas as a graph. In particular, GNN models can help holistically encode schema entities, utterance questions, and their relations [2, 3, 5, 35, 36].

Decoder. A straightforward way is to generate queries sequentially from encoded representations by Seq2Seq models. However, given that SQL queries have syntax constraints, generating queries without controls would result in broken queries (e.g., incorrect grammar). A mature neural NLIDB would take the syntax rules of SQL queries into account. For instance, SyntaxSQLNet employs a set of sub-models to generate different clauses of a query and uses a super-model to coordinate these sub-models [49]. By doing so, the syntactical correctness of output queries is guaranteed.

In addition, instead of directly generating an SQL query, some methods opt to generate a succinct tree-based intermediate representation, e.g., AST (abstract syntax tree), and then post-translate it into a valid SQL query by pre-defined rules [14, 39]. These rules rely on some heuristics to fill in missing information to the intermediate representation and convert it into a well-formed SQL query, which circumvents complex reasoning over schemas on neural models. However, the heuristics could be tricky and do not always hold. For instance, SemQL, an intermediate representation of IRNet [14], assumes that table joining relies on the presence of an explicit foreign key constraint. Therefore, in some noisy scenarios where foreign keys are absent, these post-translations may not be able to provide coherent queries, compared with non-noisy cases.

2.2 Metamorphic Testing (MT)

Determining the correctness of SQL queries generated by NLIDBs for arbitrary utterance-schema pairs generally requires human-annotated ground truth. In contrast, MT benchmarks testing targets via **metamorphic relations** (**MRs**) without the need for ground-truth [6, 33]. Each MR denotes a general and usually *invariant property* of the testing targets. For instance, to test the implementation of $sin(x)$, instead of knowing the expected output of arbitrary floating-point input x (which requires considerable manual efforts), we assert whether the MR $sin(x) = sin(\pi - x)$ always holds when arbitrarily mutating x. A bug in $sin(x)$ is detected when input x and its mutation $(\pi - x)$ induce inconsistent outputs. To date, MT has achieved major success in detecting bugs in DBMS [28–30] and NLP-related models [19, 27, 37]. To test DBMS with metamorphic relations, for instance, NoREC transforms a query into a non-optimized form and compares whether the optimized query and non-optimized query induce identical outputs [28]. These metamorphic testing-based methods detect considerable logic bugs on well-tested DBMS, e.g., MySQL and SQLite. Likewise, metamorphic relations are applied to generate test cases for NLP-related models. The evaluation results produced by metamorphic testing are deemed as a useful complement to accuracy on hold-out datasets [27] and also help identify considerable defects on commercial NLP software. Our research further leverages MT to benchmark (neural) NLIDBs without requiring manual effort. To this end, we define a comprehensive set of MRs to conduct semantics-preserving transformations toward natural language utterances and database schemas.

Algorithm 1: MT-TEQL

```
Input: NLIDB $\mathcal{M} : \mathcal{U} \times \mathcal{S} \rightarrow \mathcal{Q}$, seed utterance $u_0$, seed schema $s_0$
Output: error-triggering inputs $E = \{(u_1, s_1), \cdots\}$
1 $T \leftarrow \mathcal{MR}_1(u_0, s_0) \cup \cdots \cup \mathcal{MR}_n(u_0, s_0)$;
2 $E \leftarrow \emptyset$;
3 foreach $(u', s') \in T$ do
4     if $\mathcal{M}(u', s') \neq \mathcal{M}(u_0, s_0)$ then
5         $E = E \cup \{(u', s')\}$
6     end
7 end
8 return $E$;
```

3 ASSESSING MODEL WITH MT-TEQL

Alg. 1 depicts the overall workflow of MT-TEQL, where given seed utterance u_0 and seed schema s_0, MT-TEQL first generates a collection T of transformed utterances and schemas based on a set of MRs (line 1). For each pair of transformed utterances and schemas (u', s'), MT-TEQL checks the consistency between queries derived from (u', s') and queries derived from (u, s) (line 4). If the consistency property is violated, (u', s') is marked as an error-triggering input (line 5), which will be collected for subsequent augmentation.

In contrast to Alg. 1, standard benchmarks usually compare the output of NLIDB with human-annotated ground truth. Inspired by DBMS metamorphic testing, MT-TEQL asserts the *consistency* between $\mathcal{M}(u, s)$ and $\mathcal{M}(u', s')$ (line 4), thus alleviating manual labeling. In addition, considering modern neural NLIDBs typically develop a *joint* understanding of utterances and schemas, MT-TEQL transforms both utterances and schemas with semantics-preserving MRs (line 1), while existing general-purpose NLP model testing tools can likely neglect certain errors as they only focus on text [27].

Table 1: Metamorphic relations in MT-Teql.

Target	Metamorphic Relations (MRs)
Utterance	Prefix Insertion
	Prefix Removal
	Prefix Substitution
	Synonym Substitution
Schema	Normalization
	Flattening
	Opaque Key
	Table Shuffle
	Column Shuffle
	Column Removal
	Column Renaming
	Column Insertion

Table 2: Categorizations of frequently-used prefixes in typical NLIDB utterances.

Type	Illustrative Examples
Common Interrogative Prefix	what is/are, which is/are
Common Declarative Prefix	tell me, return, find, list
Special Interrogative Prefix	when, where, how many
Special Declarative Prefix	count

While Alg. 1 illustrates the general workflow of MT-Teql, designing a comprehensive set of semantics-preserving mutation strategies on utterances and schemas, while preserving the "legitimacy" of synthetic inputs, is challenging. In contrast to previous works, MT-Teql does *not* mutate inputs from an adversarial perspective (e.g., injecting typos or heavily paraphrasing), nor does MT-Teql break utterances/schemas into "untranslatable" forms [52]. As listed in Table 1, we instantiate a total of 12 MRs to systematically explore realistic defects from diverse assessment criteria.

3.1 Utterance Transformation

Usually, utterances from users start with prefixes (e.g., "what is" and "tell me") and are followed by the real query body. We categorize frequently-used prefixes in utterances in Table 2. Here, common prefixes do not contain user intent while special prefixes can implicitly indicate some user intent (e.g., desired columns or aggregate functions). Observing that the choice or even the occurrence of common prefixes usually does not affect user intent, we design four MRs focusing on transforming prefixes in Table 2.

Prefix Insertion+Removal (PI/PR). In some cases, the occurrence of prefixes does not affect user intent. When an utterance starts from an explicit interrogative (either common or special) prefix, our first MR specifies inserting a common declarative prefix at the beginning. For example, "*tell me* what is the age of all singers?". Second, we observe that common prefixes do not give any information about the output query. Thus, the second MR specifies removing common prefixes. For example, "~~*what is*~~ the age of all singers?"

Prefix Substitution (PS). Likewise, the choice of common prefixes before query body also does not affect user intents. Hence, we implement an MR to replace the prefix in an utterance with another prefix, e.g., "*tell me* ~~*what is*~~ the age of all singers?"

While the above MRs are relatively easy for humans to comprehend utterances' real intents, our experiments shows that they impose considerable challenges to modern NLIDBs (see Sec. 5.1).

By observing how natural language denotes aggregate functions via utterances, we further propose an MR focusing on replacing certain tokens in an utterance with their synonyms.

Table 3: Illustrative mapping relations between aggregates and textual indicators.

Aggregate	Textual Indicator
`MIN`	minimal, minimum, lowest, smallest
`MAX`	maximal, maximum, highest, largest
`COUNT`	the (total) {number, count, amount} of
`SUM`	the (total) {sum, amount} of
`AVG`	the {mean, average} of

Synonym Substitution (SS). The choice of natural language to express an identical aggregate function should not change the corresponding SQL queries. The MR manipulates synonyms that indicate an identical aggregate function, e.g., "what is the *number* ~~*amount*~~ of singers?". Compared with general-purpose synonym substitution, our MR incorporates NLIDB domain knowledge to manipulate a collection of specific synonyms tailored for SQL language.

Table 3 reports illustrative mapping rules for **SS**. Overall, one aggregate function in the SQL language can be expressed in multiple ways. For instance, the last row in Table 3 specifies that "the *mean* of" can be replaced by "the *average* of" while retaining the derived aggregate function `AVG`. Hence, by replacing tokens grouped together (e.g., "minimal" → "minimum"), the derived queries should retain the same aggregate functions, i.e., `MIN`.

Different aggregate functions may be expressed in the same form. For example, "the *amount* of" can denotes either `COUNT` or `SUM` aggregates in SQL (rows 4th and 5th in Table 3), relying on data type of corresponding columns. Thus, by replacing "the *sum* of" with "the *amount* of", NLIDBs need to consider the context to infer the correct aggregate. As such implicit indicators widely exist in practice, it is necessary to evaluate NLIDBs in this way.

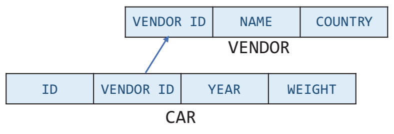

Figure 2: Original Schema.

3.2 Schema Transformation

As aforementioned, existing works generally evaluate NLIDBs by transforming natural language text [44]; nevertheless, given that utterances and database schemas are learnt jointly by modern neural NLIDBs, we identify the need to further transform database schemas to comprehensively explore potential defects.

Like MRs applied to natural language utterances, we strive to leverage a set of MRs to conduct *semantics-preserving* transformations on schemas. We propose eight MRs, which are carefully

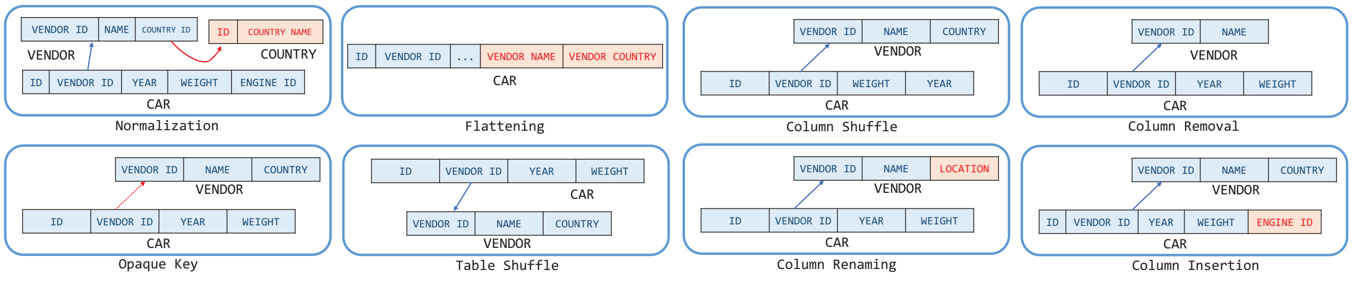

Figure 3: Illustrative examples of schema-oriented MRs.

designed to transform certain parts of schemas that do *not* change ground-truth query executions, e.g., adding extra table joining. Hence, NLIDBs are guaranteed to yield identical SQL queries.

We present illustrative examples of each schema-oriented MR in Fig. 3 (and the original schema in Fig. 2). Our observations on real-world schema variants show that users often decide to normalize some inter-dependent columns or denormalize two tables to boost performance. Also, users may not explicitly declare foreign key or primary key constraints. Furthermore, it is well known that tables/columns are stored in database without orders. Hence, the way we serialize schemas (i.e., table/column orders) as NLIDB inputs should not change outputs. However, changing orders generates database schema variants, which, as observed in our empirical study, can impose extra challenges on robustness of NLIDBs and stress the structural feature encoders. We further design four schema-level and four table-level semantics-preserving transformations.

Normalization (NO). Conducting normalization on columns that are irrelevant to associated utterances should not change output queries. In principle, it is feasible to employ well-established functional dependency mining algorithms [21] on the raw table content to pinpoint and normalize dependent columns. However, these algorithms are usually costly when we are mutating a large amount of tables to evaluate neural NLIDBs. Instead, as shown in Fig. 3, MT-TEQL launches a lightweight approach to extracting one unused column each time to form a new table with two columns (one original column and one linking column), and links the original table to the reference table with an explicit foreign key.

Flattening (FL). As a dual operation to normalization, we implement an MR to flatten a table if there exists an explicit foreign key constraint. To do so, MT-TEQL piggybacks a reference table to the main table and drops the reference table. This way, we effectively change schema structure while retaining high-level semantics.

Opaque Key (OK). Explicit foreign key constraints and primary key constraints could give hints for models to perform table joining. However, in practice, explicit key constraints are not always available due to various reasons (e.g., performance considerations). Though key constraints can be used as an indicator for table joining, plausible NLIDBs are expected to make consistent inference no matter an explicit key constraint exists or not, as key constraints may be absent in the real world. As shown in Fig. 3, the explicit foreign key constraint between `CAR.VENDOR_ID` and `VENDOR.VENDOR_ID` is removed and it is still feasible to infer their dependency based on the linguistic feature of column names and table names.

Table Shuffle (TS). While tables are theoretically unordered in the database systems, they need to be "serialized" by some orders before being fed to NLIDBs [50]. Therefore, we check if the specific order of tables in the NLIDB input would induce output changes to stress the NLIDBs. In Fig. 3, MT-TEQL implements an MR to randomly shuffle tables within a schema and assert query consistency.

Above MRs manifest holistic transformations on the entire schema. We further design four MRs to perform table-level transformations.

Column Shuffle (CS). Like **TS**, how columns are fed to NLIDBs should not induce inconsistent queries. MT-TEQL implements an MR to randomly shuffle columns within a table.

Column Removal+Renaming (CRm/CRn). The name or occurrence of an *irrelevant* column should not incur inconsistent outputs. If one column is not used in the ground-truth query, MT-TEQL implements two MRs to change its name with some synonyms (e.g., "country" → "location" in the "Column Renaming" diagram in Fig. 3) or simply drop that column.

Column Insertion (CI). By querying the knowledge graph [38], we may know what attribute the object indicated by the table may have. As shown in the "Column Insertion" diagram in Fig. 3, MT-TEQL extends the schema by inserting an extra column named `ENGINE_ID` in the `CAR` table, since the knowledge graph returns a "`has-a`" edge between "car" and "engine.", denoting an owner relationship. However, dependency on external knowledge graph may result in unrealistic insertion. For instance, knowledge graph may suggest a person has a head, while "head" is not likely to be a column of a "person" table used in daily scenarios (e.g., for class rosters). To practically enhance the realism of the mutated schemas, we only consider "high-weight" edges in the knowledge graph to extract highly-correlated items and allow users to cross-validate the inserted column in other similar tables in the dataset. Nevertheless, since this scheme is used to stress NLIDBs, we anticipate NLIDBs should retain consistency even if an "unusual" column is inserted.

Discussion. Some transformations proposed in this section are "semantics-preserving" only with respect to particular utterances. For instance, to use the **NO** scheme for schema normalization, we need to first decide columns that are irrelevant to a particular utterance; mutating other utterance-related columns will presumably lead to generating a different SQL query. Despite the potential limits, we show that such transformations are sufficient to expose a large volume of neural NLIDB inconsistencies.

Table 4: **Information of our reproduced neural NLIDBs. Acc is accuracy measured on Spider-dev.**

NLIDB	Linguistic Encoder	Structural Encoder	Acc
SyntaxSQLNet	GloVe [22]	Column Sequence	22.4
SyntaxSQLNet+aug	GloVe [22]	Column Sequence	25.8
IRNet	GloVe	Column Sequence+Linking	52.8
GNN	Bi-LSTM	Schema Graph	45.0
GlobalGNN+Linking	BERT-base [10]	Schema Graph+Linking	54.9
RAT-SQL	GloVe	Relation Encoding+Linking	53.5
DuoRAT (a)	GloVe	Relation Encoding+Linking	52.5
DuoRAT (b)	BERT-base	Relation Encoding+Linking	51.0
DuoRAT (c)	BERT-large	Relation Encoding+Linking	63.2

4 AUGMENTATION

In the case of neural NLIDBs, it is generally acknowledged that test cases for desired properties in machine learning models can be used for retraining models to improve the desired properties [40]. Our MRs can induce a large volume of synthetic utterances and schemas (60× compared with the original dataset). We also confirm that these generated synthetic inputs are well-formed and valid (see Sec. 5.1). Nevertheless, despite the promising results, it would be extremely time-consuming, if at all possible, for retraining neural NLIDBs using such a large amount of synthetic input. Hence, this section introduces two widely-used sampling methods and proposes an *error-aware* sampling method to reduce computational overhead and practically augment neural NLIDBs.

Random Sampling (RS). Suppose we have m synthetic test cases derived from the training dataset of n samples (where $m \gg n$), the random sampling (RS) scheme randomly picks n test cases from those m synthetic data. We then extend the training set of n samples with those n randomly picked samples. This way, the training cost should not be notably increased by only doubling the size of the training dataset.

Stratified Sampling (SS). To amplify MRs that are only applicable to a small amount of data (see Table 6 in Sec. 5.1), Stratified Sampling (SS) first samples $\min(m_i, n/k)$ test cases using each MR, where m_i is #cases synthesized using this MR and k is #MRs we have (k is 12 according to Table 1). In case $\sum_{i=1,\cdots,k} \min(m_i, n/k) < n$, Stratified Sampling further randomly samples $n - \sum_{i=1,\cdots,k} \min(m_i, n/k)$ inputs from the remaining test cases.

Adaptive Sampling (AS). Given that NLIDBs can have errors, one might wonder about the feasibility of using error-triggering inputs for augmentation. However, we note that the total amount of error-triggering inputs are *not comparable* to the size of the standard training dataset, due to overfitting issues on the training set. Hence, augmentation with only error-triggering inputs is not realistic. Instead, we design Adaptive Sampling (AS), as a practical *error-aware* sampling scheme. In particular, we first randomly split the standard training set in ten folds forming a nine-fold training set S_t and a one-fold validation set S_v. An NLIDB M_0 is then trained on S_t using *half of the standard training epochs*. Then, we transform test cases in S_v using our MRs, and evaluate M_0 in terms of its error rate r_i (see for Sec. 5 the definition of r_i) using the synthetic test cases generated by each MR. We then normalize r_i to $\hat{r_i}$ such that $\sum \hat{r_i} = 1$. Then, similar to SS, we sample $\min(m_i, \hat{r_i}n)$ test cases from the synthetic data set generated by each MR and further sample from the remaining test cases to obtain a total of n cases. We re-train NLIDB M_0 with n original inputs and those n new inputs sampled under the awareness of errors.

5 EXPERIMENTS

Sec. 5.1 assesses the quality of the MT-Teql-generated utterance and schema variants. These utterance/schema inputs are used to evaluate neural NLIDBs in Sec. 5.2. We then investigate erroneous cases found in Sec. 5.2, and summarize typical error patterns in Sec. 5.3. To demonstrate the versatile usage of MT-Teql outputs, a user study in Sec. 5.4 shows that MT-Teql can help developers to differentiate and assess NLIDB models. Furthermore, Sec. 5.5 leverages MT-Teql to extend the training dataset of neural NLIDBs and enhance their robustness. In what follows, we first discuss the experiment setup.

Dataset. While MT-Teql is capable of testing NLIDBs with unlabeled data, we synthesize our test cases from Spider-dev [50], which is widely used in practice. In terms of augmentation, we synthesize training data from Spider-train. Using the Spider dataset also allows us to compare the accuracy of augmented NLIDBs.

NLIDBs under Testing. We follow the official instructions to reproduce NLIDB implementations which are summarized in Table 4. We reproduce nine NLIDBs [2, 3, 5, 14, 32, 42, 49] for evaluation, which feature diverse natural language encoders and schema learning modules and, presumably, manifest distinct defects in front of different MRs.

NLIDBs with Augmentation. We use techniques presented in Sec. 3 for NLIDB augmentation with test cases synthesized from Spider-train. Therefore, there is *no overlap between test data (which is derived from Spider-dev) and training data.* We design three extra groups (i.e., RS$^+$, SS$^+$ and AS$^+$) for a fair comparison that sample more synthetic data and make the size of its augmented training set equal to "SyntaxSQLNet+aug" [49].

Evaluation Metrics. Three metrics are used in our evaluation, namely, naturalness, error rate, and accuracy. **Naturalness** is defined as the fluency score of an utterance. It is constructed from a large-scale human corpus and is useful to quantify grammatical correctness and coherence of synthetic texts without manual effort. We refer readers to [13] for the detailed definition. **Error Rate** is defined as $r_i = \frac{\sum \mathbb{1}_{eval(\mathcal{M}(u,s),\mathcal{M}(u',,s'))}}{|\mathcal{S}|}$, where $eval(\cdot,\cdot)$ checks the equivalence of SQL queries yielded by $\mathcal{M}(u,s)$ and $\mathcal{M}(u',s')$; and $|\mathcal{S}|$ denotes the size of transformed input. Aligned with previous works [50], we use exact set matching (EM) for equivalence checking. We follow conventions to report NLIDB **Accuracy** on a standard benchmark (i.e., Spider-dev in our experiment).

5.1 Comprehensiveness and Naturalness

As shown in Table 6, MT-Teql synthesizes a considerable amount of utterance-scheme pairs based on each MR. We also report that Prefix Removal (PR) and Synonym Substitution (SS) are highly subject to the structure of utterances, which result in a smaller amount of generated utterance-scheme pairs.

Comprehensiveness. We report the statistics of popular NLIDB benchmarks in Table 5. For those datasets that contain a training set and a test (or dev) set, we only pick the test (or dev) set, since they are used by previous works to benchmark NLIDBs [44]. We also deem the comparison as fair, because MT-Teql generates its

Table 5: Comparison of NLIDB benchmarks.

Dataset	Schema Domain	Query Type	Method	#Utterance	#Schema	#Table per Schema	Naturalness
WikiSQL-dev [54]	Cross-domain	Single-table	crowdsourcing	8,421	2630	1	0.143
WTQ-test [16]	Cross-domain	Single-table	crowdsourcing	2,955	2102	1	N/A
GeoQuery-dev [11, 15, 51]	Single-domain	Cross-table	hand-crafted	49	1	7	0.144
ATIS-dev [8, 11, 15]	Single-domain	Cross-table	hand-crafted	486	1	25	0.133
MAS [17]	Single-domain	Cross-table	hand-crafted	194	1	8	0.128
FIBEN [34]	Single-domain	Cross-table	hand-crafted	300	1	152	0.127
ParaphraseBench [44]	Single-domain	Single-table	hand-crafted	399	1	1	0.145
Spider-dev [50]	Cross-domain	Cross-table	hand-crafted	1,034	20	4.05	0.155
Spider-Realistic [9]	Cross-domain	Cross-table	hand-crafted	508	≈20	≈4.05	N/A
MT-Teql	Cross-domain	Cross-table	auto-generated	62,430	2,273	4.97	0.148

Table 6: Distribution of (transformed) inputs of each MR.

MR	# Inputs	MR	# Inputs
Prefix Insertion	6,370	Opaque Key	3,697
Prefix Removal	199	Table Shuffle	2,575
Prefix Substitution	8,266	Column Shuffle	6,675
Synonym Substitution	639	Column Removal	8,707
Normalization	8,707	Column Renaming	11,775
Flattening	2,018	Column Insertion	2,802
Total	62,430		

benchmark from Spider-dev. Table 5 categorizes benchmarks by schema domains and query types. Cross-domain benchmarks denote the schemas collected from various domains. For instance, WikiSQL collects schemas from Wikipedia web tables, which cover multiple domains. In contrast, GeoQuery only uses an identical schema about geography information for all queries in the benchmark. Cross-table benchmarks denote that the SQL queries may have joining operations to connect different tables within a schema. For WikiSQL, since its schemas are derived from standalone web tables, all queries only retrieve data from one single table. Among all these benchmarks, only Spider, Spider-Realistic and MT-Teql are cross-domain cross-table benchmarks, which indicate a practical and challenging setting.

MT-Teql synthesizes 62,430 utterance-schema pairs from 1,034 data samples in Spider-dev [50]. As shown in Table 5, the size of the MT-Teql testcase surpasses other benchmarks by orders of magnitude. In addition, MT-Teql synthesizes considerable distinct cross-table schemas compared with other benchmarks, which can presumably uncover more defects.

Naturalness. As previously mentioned, instead of heavily transforming utterances (e.g., replacing certain words with typos) which likely induce ill-formed utterances, we aim to synthesize *natural* utterances whose incurred errors likely denote real-world defects that normal users frequently encounter. That is, we advocate naturalness as an important metric to assess the quality of synthetic utterances. We follow [19, 26] to evaluate the naturalness of synthetic utterances and report the naturalness in Table 5.[1] In Suppl. Material [18], we empirically illustrate that the employed naturalness score aligns well with human perception. MT-Teql synthetic utterances exhibit comparable naturalness with hand-crafted Spider-dev utterances and even better than other benchmark datasets. Based on our observations, ParaphraseBench paraphrases utterances with uncommon phrases (e.g., "whose age is 15" ⟶ "are exactly as old as 15"), which induces relatively unnatural utterances. Some benchmarks, such as MAS and FIBEN, exhibit relatively low naturalness. We inspect the utterances with the low naturalness of these benchmarks and report that many utterances are grammatically incorrect. For instance, an utterance in FIBEN states "*find all stocks has a last traded value Greater than 1500.*"

[1]We only consider synthetic utterances in MT-Teql for naturalness evaluation. We do not evaluate the naturalness of WTQ-test and Spider-Realistic since they are not publicly available at the time of writing.

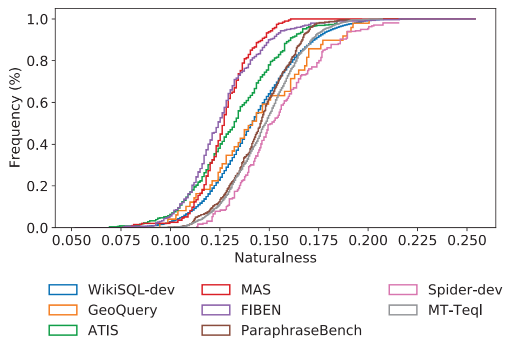

Figure 4: Cumulative distributions of naturalness.

Fig. 4 further reports the cumulative naturalness distributions of utterances in Spider-dev, ParaphraseBench and MT-Teql. Particularly, in the bottom 25%, the naturalness distributions of Spider-dev utterances and utterances generated by MT-Teql are very close, illustrating promising results that our MRs do not notably impede the readability of worse-case utterances in the original dataset. MT-Teql synthesized utterances manifest comparable naturalness compared with other hand-crafted benchmarks. It also generally outperforms WikiSQL, a crowdsourced benchmark, on naturalness.

5.2 Assessing NLIDB Inconsistency

We evaluate the reproduced NLIDBs in Table 4 with utterance-schema pairs synthesized by MT-Teql. We report the error rate of NLIDBs in Table 7. Overall, MT-Teql successfully finds erroneous predictions $(1,908(\pm 380))$ from all the evaluated NLIDBs. We interpret that utterance MRs are all highly effective to expose defects with an average error rate of 9.5%. **SS** is particularly effective, given its high flexibility in perturbing natural language

Table 7: Error rate of NLIDBs. PI: Prefix Insertion; PR: Prefix Removal; PS: Prefix Substitution; SS: Synonym Substitution; NO: Normalization; FL: Flattening; OK: Opaque Key; TS: Table Shuffle; CS: Column Shuffle; CRm: Column Removal; CRn: Column Renaming; CI: Column Insertion.

NLIDB	PI	PR	PS	SS	NO	FL	OK	TS	CS	CRm	CRn	CI	All
SyntaxSQLNet	8.2	8.0	6.7	13.3	2.4	2.7	1.3	2.1	1.1	1.5	1.7	0.8	3.2
SyntaxSQLNet+aug	9.2	3.5	8.2	16.7	2.1	2.2	1.4	2.3	1.1	1.4	1.6	1.1	3.4
IRNet	7.5	11.2	7.0	37.8	2.4	6.6	5.2	2.8	2.1	2.1	1.6	2.1	4.0
GNN	4.6	5.5	3.4	25.4	3.4	4.3	5.3	0.0	0.0	1.0	0.5	0.2	2.4
GlobalGNN+Linking	4.0	7.5	3.3	18.3	5.8	6.9	6.1	0.4	0.2	2.9	1.1	3.0	3.2
RAT-SQL	2.4	5.0	2.6	21.8	3.0	4.9	3.8	0.2	0.04	1.3	0.7	0.0	1.9
DuoRAT (a)	4.7	6.0	4.6	29.9	2.5	5.9	6.1	1.0	0.1	2.0	1.0	0.6	2.9
DuoRAT (b)	3.5	2.5	4.4	14.1	2.5	6.5	5.0	2.0	0.2	2.5	0.9	1.2	2.7
DuoRAT (c)	5.0	8.0	4.3	12.4	5.3	7.0	3.4	4.8	2.1	4.8	1.4	1.8	3.8
Average	5.5	6.4	4.9	21.1	3.3	5.2	4.2	1.7	0.8	2.1	1.2	1.2	3.1

utterances, of which 21% transformed utterances trigger errors. Relatively mundane prefix-oriented transformations (i.e., **PI**, **PR**, and **PS**), which impose easy challenges for humans, are also effective for testing NLIDBs (on average 5.6%). Schema-based transformations also achieve reasonable performance with an average error rate of 2.5%. **NO**, **FL** and **OK**, by manipulating the holistic structure of schema, are more useful at triggering errors (4.2%). In contrast, **TS**, **CS**, **CRm**, **CRn** and **CI**, which are generally trivial for humans, still induce certain errors (1.4%).

For NLIDB-wise comparison in Table 7, we see that the performance of NLIDBs is largely influenced by the design of NLIDBs (e.g., specific natural language encoders and schema learning modules as shown in Table 4), which is aligned with our intuition. For example, SyntaxSQLNet and IRNet use the sequences of columns as input and are generally more sensitive to **TS** and **CS**. In contrast, the other NLIDBs learn from schema structural features (e.g., graph-based method and relation encoding) and become more resilient to these MRs. We also observe that the adopted natural language encoders can induce distinct errors in terms of **SS**. For example, DuoRAT (a) uses GloVe to encode utterances and has 2.4× more errors on **SS** compared with its variant, DuoRAT (c), which uses BERT-large as the natural language encoder. It is also difficult for NLIDBs to overcome schema-related MRs. As shown in Table 7, while GNN-based methods (i.e., GNN and GlobalGNN+Linking) are almost resilient to **TS** and **CS**, holistic structural changes (e.g., **NO**) can still impose considerable challenges.

We view Table 7 illustrates the strength of MT-Teql by exposing numerous defects from well-trained neural NLIDBs. By applying MT-Teql as assessment criteria, developers can obtain finer-grained diagnosis to NLIDB and allow them to further improve it on specific criteria, compared to the standalone accuracy on hold-out datasets.
Lessons. Overall, our studies show that *high benchmark accuracy does **not** necessarily indicate better robustness*. As illustrated in Table 4 and Table 7, models with high accuracy do not necessarily induce better consistency (robustness). We interpret that, for gaining high accuracy on the standard benchmark, models inevitably learn more fine-grained information from both utterances and schemas, which become sensitive to subtle changes in the meantime. For instance, DuoRAT (c) encodes structural information in a finer-grained manner (vs. RAT-SQL). Therefore, while it achieves higher standard accuracy, its robustness downgrades simultaneously.

In addition, we find that *powerful pretraining techniques may be resilient to linguistic variations*. As shown above, BERT-large has a better capability of capturing user intents from noisy utterances generated by Synonym Substitution. Hence, we interpret that powerful pretraining techniques, which are customized for NLIDB, may provide a normalized “embedding” of utterances for the follow-up layers to proceed, thus achieving higher robustness in terms of linguistic variations. It is worth noting that task-oriented pretraining techniques already show effectiveness in boosting accuracy [9, 48].

Furthermore, we presume that *it is generally hard for models to capture schema invariance*. Given the flexibility of schema design, it is non-trivial for models to comprehend and be robust to such variations. For instance, although GNN-based methods (i.e., GNN and GlobalGNN+Linking) manifest high robustness toward Table Shuffle and Column Shuffle, more holistic structural changes (e.g., Normalization) can still impose challenges. Hence, we envision that data augmentation techniques can mitigate the issue by training models on diverse schemas, as we show in Sec. 5.5.

Table 8: Empirical distribution of error types.

Error Type	# Errors	Error Type	# Errors
Column Prediction	101	Table Joining	171
Aggregate Function	77	Complex Query	48
Operator	73	Others	30

5.3 Error Analysis

We manually checked 500 pairs of errors and we present the error type distribution in Table 8. We also notice that there are no false negative cases in the errors and presume the overall false negative should be lower than error cases reported by standard metrics. For example, it is reported that exact set matching has a 2.5% false negative rate on average and 8.1% in the worst case [53]. In the following, we discuss five major sources of errors with representative cases in Table 9 and Fig. 5, corresponding to utterance-related and schema-related error cases, respectively. They are derived by transforming the `Singer, Pet, Car` schemas in the Spider-dev set or transforming utterances associated with these schemas. In Table 9, we report the original and transformed utterances and corresponding SQL queries, respectively. In Fig. 5, we present the SQL query derived from the transformed schemas (`Pred@T`) and the query derived from the original schemas (`Pred@O`), accordingly.

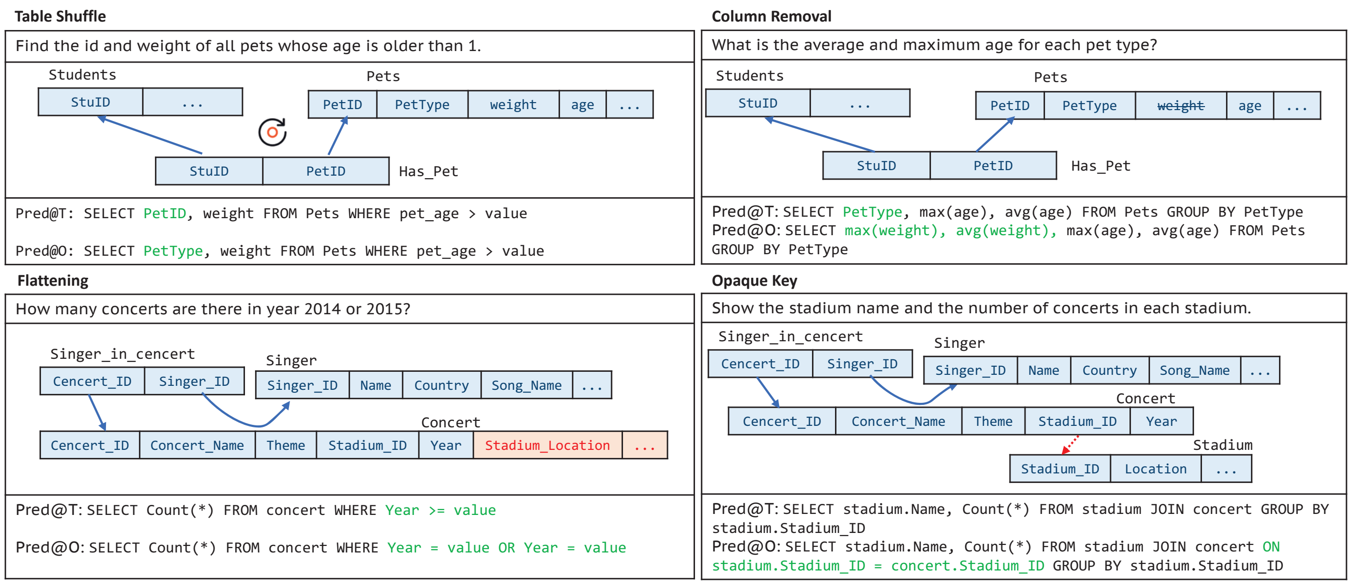

Table Shuffle

Find the id and weight of all pets whose age is older than 1.

```
Pred@T: SELECT PetID, weight FROM Pets WHERE pet_age > value

Pred@O: SELECT PetType, weight FROM Pets WHERE pet_age > value
```

Column Removal

What is the average and maximum age for each pet type?

```
Pred@T: SELECT PetType, max(age), avg(age) FROM Pets GROUP BY PetType
Pred@O: SELECT max(weight), avg(weight), max(age), avg(age) FROM Pets
GROUP BY PetType
```

Flattening

How many concerts are there in year 2014 or 2015?

```
Pred@T: SELECT Count(*) FROM concert WHERE Year >= value

Pred@O: SELECT Count(*) FROM concert WHERE Year = value OR Year = value
```

Opaque Key

Show the stadium name and the number of concerts in each stadium.

```
Pred@T: SELECT stadium.Name, Count(*) FROM stadium JOIN concert GROUP BY
stadium.Stadium_ID
Pred@O: SELECT stadium.Name, Count(*) FROM stadium JOIN concert ON
stadium.Stadium_ID = concert.Stadium_ID GROUP BY stadium.Stadium_ID
```

Figure 5: Schema-related errors. Foreign key constraints are annotated by arrows. Transformations are in red and errors are in green. `Pred@T` denotes the prediction on the transformed inputs and `Pred@O` denotes the prediction on the original inputs.

Table 9: Utterance-related error cases. u, u' denote original and transformed utterances. q, q' represent SQL queries generated from u and u', respectively.

MR	Utterance and SQL Query
PI	u = what is the model for the car with a weight smaller than the average? q = `SELECT CarName.Model FROM CarName JOIN [. . . ] WHERE CarData.Weight < (SELECT avg(Weight) FROM CarData)`
	u' = return what is the model for the car with a weight smaller than the average? q' = `SELECT ModelList.Model FROM ModelList JOIN [. . . ] WHERE CarData.Weight < (SELECT Model FROM CarName)`
PR	u = what is the year that had the most concerts? q = `SELECT Year FROM concert GROUP BY Year ORDER BY count(*) DESC LIMIT 1`
	u = ~~what is~~ the year that had the most concerts? q = `SELECT SongReleaseYear FROM singer GROUP BY SongReleaseYear ORDER BY count(*) DESC LIMIT 1`
PS	u = show countries where a singer above age 40 and a singer below 30 are from. q = `SELECT Country FROM singer WHERE Age > value INTERSECT SELECT Country FROM singer WHERE Age < value`
	u' = let me know countries where a singer above age 40 and a singer below 30 are from. q' = `SELECT Song_Name FROM singer WHERE Age > value INTERSECT SELECT Song_Name FROM singer WHERE Age < value`
SS	u = Find the number of pets whose weight is heavier than 10. q = `SELECT count(*) FROM Pets WHERE weight > value`
	u' = Find the amount of pets whose weight is heavier than 10. q' = `SELECT weight FROM Pets WHERE weight > value`

Column Predictions. As shown in **PS** of Table 9, the NLIDB is misled to choose an incorrect column Song_Name instead of Country, even though the correct column name is explicitly referred to in the utterance. In contrast, as reported by the error analysis on the standard dataset [14], most column prediction errors are due to the fact that the ground-truth column names are not explicitly (or merely partially) mentioned in the utterances. Similarly, after shuffling table orders or removing an irrelevant column, the predicted columns become inconsistent in "Table Shuffle" and "Column Removal" of Fig. 5. According to our observation, column prediction errors not only exist in the SELECT clause, but also occur in the GROUP BY and ORDER BY clauses. According to our manual study of 500 errors, erroneous column predictions account for 20.2% of errors.

Aggregate Function. Among 500 manually checked errors, we report that 15.4% of errors are caused by incorrect aggregate functions. As discussed in Sec. 3.2, changing aggregate function indicators can impose challenges to NLIDBs. For instance, in **SS** of Table 9, by replacing "number of" with "amount of", the NLIDB under test fails to comprehend the transformed utterance and uses an aggregate function count(*) in q'.

Operator. Predicting operators is a major step towards identifying intended contents. Our study on the 500 erroneous cases reports that 14.6% of errors are due to incorrect operator predictions. For example, a >= operator may be mistakenly predicted as =. As shown in the "Flattening" case in Fig. 5, some irreverent changes in schema structure can result in an incorrect prediction on the operator in the WHERE clause. Besides, we also observe errors in the HAVING clauses of the group-by operation. Overall, our manual study shows that such erroneous operators widely exist in almost all MRs.

Table Joining. Linking multiple tables in one query is challenging for NLIDBs and is a major cause of errors (34.2%). As an important table joining indicator, foreign key constraints help NLIDBs link two tables to a large degree. Without explicit foreign key constraints in the schema, NLIDBs may lack the guidance to link two tables. As shown in the "Opaque Key" case of Fig. 5, though the NLIDB captures the need for table joining, it fails to make predictions on the joining condition (i.e., the ON clauses). Besides "Opaque Key", our manual study shows that prefix- and column-based transformations can also trigger a considerable number of table joining errors.

Complex Query. Besides the four basic types of errors, we also observe that some "extra hard" queries in the Spider-dev set, with multiple table joining and complex conditions, are lengthy and fragile, even for human experts, to translate. For instance, **PI** in Table 9, by changing the prefix of an utterance, reports a drastic change in the generated queries, including incorrect column, table, join operation and nested sub-query. In these cases, even subtle changes in the input can induce multiple errors in the corresponding output. Nevertheless, as reported in Sec. 5.5, the dataset augmented by our transformed inputs can effectively enhance NLIDB performance toward "Extra Hard" challenges. Among the 500 sampled pairs, we observe that about 9.6% of predictions have more than one error.

Misc. & Lessons. We have shown common weaknesses of neural NLIDB. Besides the aforementioned types of errors, we also find some errors are less common, e.g., erroneous table predictions and unwanted conditions in `WHERE` clauses, which comprise 6.0% of errors. More importantly, our manual investigation reveals that *human-in-the-loop revision, such as human annotation or interactive correction, may help to correct a substantial amount of inconsistencies.* On one hand, if users can explicitly specify their desired aggregate functions or columns in addition to the input utterances, the output space of the model can be effectively reduced, presumably enhancing the model accuracy. On the other hand, if MT-Teql pinpoints inconsistent outputs under given utterances, NLIDB can ask users to revise particularly inconsistent clauses for correction. Hence, we envision adopting human-in-the-loop methods to further disambiguate user intent and prune output space.

5.4 MR as Assessment Criteria

Motivation. MT-Teql features an NLIDB model-agnostic design; we can smoothly benchmark representative neural NLIDB models and effectively detect their erroneous outputs automatically. Overall, we interpret that our study and findings in Table 7 have suggested an important usage scenario of this work: *when different NLIDBs distinctly perform w.r.t. MRs, users can select the most appropriate NLIDB to use, according to their preference or particular scenarios, which can be reflected from certain MRs.*

Domain-General Criteria vs. Domain-Specific Criteria. In general, MRs proposed in this research can be treated as domain-general assessment criteria, where we benchmark the consistency of their general functionality and expose their differences. Table 7 has shown that MT-Teql can effectively expose differences of NLIDBs. NLIDBs of different architectures and data preprocessing techniques show largely distinct robustness against different MRs. Holistically, we suspect that these differences are informative enough to facilitate users selecting NLIDBs. For instance, when users mostly face single table queries (e.g., web table query), DuoRAT would be desirable despite its low robustness against schema variations. In contrast, if users frequently encounter sophisticated database schemas, RAT-SQL, due to higher robustness, would be more suitable despite its relatively low accuracy compared with DuoRAT.

Furthermore, we also envision the potential of designing domain-specific MRs as the assessment criteria. For instance, a user, particularly concerning multi-lingual scenarios, can design MRs to benchmark the consistency of NLIDB regarding a query in different languages. This way, users particularly concerning multi-lingual scenarios can opt for NLIDB that performs the best over the MRs.

User Study. In the rest of this section, we present a user study: 1) to assess whether MT-Teql's MRs are informative enough to help users select NLIDBs, and 2) to assess the feasibility for users to design domain-specific MRs, as a plugin of MT-Teql.

First, we pick a collection of NLIDBs from Table 4 and put MT-Teql's MRs into two main categories, in total seven capabilities to reflect robustness. We then use Formula 1 (smaller is better; see below) to summarize NLIDB robustness. Here, 🎉 denotes "good", 😑 denotes "neutral" and 🤢 denotes "bad".

$$\mathsf{F}(r) = \begin{cases} \text{🎉}, r < \mathsf{avg} - 0.5 \times \mathsf{std} \\ \text{😑}, \mathsf{avg} - 0.5 \times \mathsf{std} \le r \le \mathsf{avg} + 0.5 \times \mathsf{std} \\ \text{🤢}, r > \mathsf{avg} + 0.5 \times \mathsf{std} \end{cases} \qquad (1)$$

In Formula 1, `avg` denotes average error rate and `std` denotes standard error. Each NLIDB's robustness is thus summarized in Table 11 over two categories and seven capabilities. The last two rows are models' "Overall Robustness" and "Overall Accuracy". Note that "Overall Accuracy" is summarized from models' accuracy on hold-out datasets (which is from Table 4). "Overall Robustness" is computed by averaging the above seven scores of capabilities.

We recruit seven participants (who have taken at least one graduate database and NLP course or experience on developing relevant software) to join the assessment procedure. The user study is designed to validate the following questions:

❶ *Are capabilities implied by MT-Teql's MRs valid?*
❷ *Would MT-Teql's evaluation affect users' preferences?*
❸ *Can developers use MT-Teql to assess NLIDBs in their domain-specific scenarios?*

❶ serves as a "pre-study" to measure the validity of our asserted capabilities and MRs. To answer ❶, we invite participants to assign *validity scores* of each capability based on their own scenarios. Participants also need to assign *correlation scores* between derived capabilities and MRs. Each score ranges from 1 to 3. Higher scores denote higher validity or correlation. Ideally, an MR should manifest high correlation with its implied capability, and the capability itself should be deemed as valid by our participants.

To answer ❷, we invite participants to assign scores (in the form of 🎉, 😑 and 🤢) to each NLIDB and compare users' choices with the "Overall Accuracy" row in Table 11. This comparison can convincingly reflect the utility of MRs in better assessing model difference, by comparing models' accuracy on hold-out datasets with participants' decisions. After being aware of models' performance under MT-Teql, users tend to take the model robustness into their considerations rather than simply employ the most accurate model.

To answer ❸, we describe two domain-specific scenarios and desired capabilities. Participants are then asked to design corresponding MRs for MT-Teql to evaluate the given capabilities. In contrast to the general functionality-oriented capabilities in Table 11, we evaluate the extensibility of MT-Teql under domain-specific scenarios. Details of the user study are released in our artifact.

Table 10: Error rate of augmented NLIDBs. aug: SyntaxSQLNet standard data augmentation; RS: Random Sampling; SS: Stratified Sampling; AS: Adaptive Sampling; Cmp: Relative Change on ALL.

NLIDB	PI	PR	PS	SS	NO	FL	OK	TS	CS	CRm	CRn	CI	All	Cmp
SyntaxSQLNet	8.2	8.0	6.7	13.3	2.4	2.7	1.3	2.1	1.1	1.5	1.7	0.8	3.2	N/A
SyntaxSQLNet+RS	5.2	4.0	5.1	8.1	1.2	2.2	0.9	1.6	0.5	0.9	1.0	0.3	2.0	-37.5%
SyntaxSQLNet+SS	4.7	3.5	4.6	7.8	0.9	2.0	0.8	1.4	0.3	0.5	0.9	0.4	1.8	-43.8%
SyntaxSQLNet+AS	3.9	6.5	4.3	7.2	0.8	1.4	0.3	1.4	0.2	0.6	0.8	0.9	1.6	-50.0%
SyntaxSQLNet+aug	9.2	3.5	8.2	16.7	2.1	2.2	1.4	2.3	1.1	1.4	1.6	1.1	3.4	N/A
SyntaxSQLNet+RS$^+$	5.3	3.0	4.4	6.9	1.4	1.7	0.7	1.6	0.3	0.5	0.7	0.1	1.8	-47.0%
SyntaxSQLNet+SS$^+$	3.8	4.5	4.2	4.4	1.5	1.7	0.4	1.6	0.2	0.6	0.7	0.2	1.6	-52.9%
SyntaxSQLNet+AS$^+$	3.5	4.5	3.8	2.2	1.5	2.4	1.1	1.8	0.8	1.1	0.9	0.2	1.8	-47.1%
IRNet	7.5	11.2	7.0	37.8	2.4	6.6	5.2	2.8	2.1	2.1	1.6	2.1	4.0	N/A
IRNet+RS	4.3	7.6	4.7	14.9	1.0	3.4	2.8	0.5	0.5	0.9	0.5	0.1	1.9	-52.5%
IRNet+SS	5.2	8.1	5.5	17.3	1.2	4.7	3.6	1.1	0.7	1.0	0.8	0.3	2.4	-40.0%
IRNet+AS	4.6	8.1	5.7	10.2	1.0	2.5	3.0	0.6	0.4	0.7	0.9	0.1	2.1	-47.5%

Table 11: MRs as assessment criteria. SSN denotes SyntaxSQLNet, IRN denotes IRNet, GGL denotes GlobalGNN+Linking, RAT denotes RAT-SQL, and Duo denotes DuoRAT (c).

Capability	**SSN**	**IRN**	**GGL**	**RAT**	**Duo**
Utterances of diverse linguistic habits	😑	🤢	😑	🎉	🎉
start with diverse prefixes.	🤢	🤢	🎉	🎉	😑
indicate aggregates in diverse forms.	🎉	🤢	😑	😑	🎉
express an attribute in diverse ways.	🤢	🤢	🎉	🎉	😑
Schemas with diverse design styles	🎉	😑	🤢	🎉	🤢
follow diverse normal forms.	🎉	😑	🤢	🎉	🤢
lack foreign key constraints.	🎉	🤢	🤢	😑	😑
stores in diverse orders.	😑	🤢	🎉	🎉	🤢
contains/misses irrelevant columns.	🎉	😑	🤢	🎉	🤢
Overall Robustness	😑	🤢	😑	🎉	🤢
Overall Accuracy	🤢	😑	😑	😑	🎉

Feedback of ❶ shows that participants confirm the validity of our designed MRs. In particular, they agree that all the summarized capabilities in Table 11 are reflected by our designed MRs (2.45 ± 0.22). More importantly, participants confirm that all capabilities are valid for evaluation (2.53 ± 0.13). Furthermore, in ❷, most participants' choices (74.3%) are not consistent with the row of "Overall Accuracy" in Table 11. That is, when users have finer-grained assessment criteria, their preferences are not likely to be solely influenced by accuracy on hold-out datasets. The promising results illustrate that our MRs can serve as informative assessment criteria, and they also indicate the limits of using only accuracy for NLIDB assessment. In ❸, we prepare two domain-specific scenarios (e.g., multi-lingual NLIDB) for evaluation. Six out of seven participants successfully designed at least one MR on each given capability. With manual inspections, we report that all MRs are valid and effective for assessment and can be leveraged in MT-Teql. This shows the feasibility and easiness of extending MT-Teql in evaluating other domain-specific capabilities of NLIDBs.

5.5 MT-Teql for Augmentation

We report the error rates in Table 10, subsuming base NLIDBs and also NLIDBs augmented by three schemes. Overall, Table 10 shows that with augmentation, errors of existing NLIDBs are notably reduced in all settings (on average −46.5%). Consistent with our intuition, the adaptive sampling scheme (and AS$^+$), which entails an "error-aware" augmentation, exhibits a stable enough performance to reduce prediction errors in general, where 48.2%(±1.3%) errors are effectively eliminated. Table 10 shows that the standard augmentation in SyntaxSQLNet (i.e., SyntaxSQLNet+aug) impedes the performance under six MRs (out of 12), resulting in an increase in the overall error rate (see "**All**" column). The lower robustness of SyntaxSQLNet+aug further exhibits that potential limitations of other template-based augmentations, e.g., DBPal [44], in comparison to MT-Teql (mutation-based augmentation).[2] We also observe that MT-Teql surpasses other standard NLP augmentation techniques, as will be shown in Suppl. Material [18].

Table 12: Accuracy of augmented NLIDBs. All*: accuracy on "Hard" and "Extra" questions. Cmp: relative change on All*.

NLIDB	Easy	Medium	Hard	Extra	All	All*	Cmp
SyntaxSQLNet	41.9	19.3	17.8	6.6	22.4	12.4	N/A
SyntaxSQLNet+RS	37.9	20.6	23.0	6.0	22.8	14.7	+19%
SyntaxSQLNet+SS	41.1	18.8	20.1	9.0	22.8	14.7	+19%
SyntaxSQLNet+AS	37.5	18.4	19.5	7.2	21.4	13.5	+9%
SyntaxSQLNet+aug	42.7	24.2	22.4	8.4	25.8	15.6	N/A
SyntaxSQLNet+RS$^+$	46.0	20.0	22.4	7.8	24.6	15.3	-1%
SyntaxSQLNet+SS$^+$	39.5	21.1	25.3	7.8	24.1	16.8	+8%
SyntaxSQLNet+AS$^+$	42.3	21.7	21.3	8.4	24.5	15.0	-4%
IRNet	72.3	53.6	42.0	32.9	52.8	37.6	N/A
IRNet+RS	70.3	52.9	47.7	30.5	52.6	39.3	+5%
IRNet+SS	70.7	53.8	45.9	35.9	52.7	41.1	+9%
IRNet+AS	70.3	54.3	44.8	29.3	52.5	37.3	-1%

Table 12 further reports the accuracy on Spider-dev (the **All** column), which is slightly reduced by 0.6% on average after augmentation. On one hand, it is generally acknowledged that NLIDBs with relatively higher robustness can exhibit lower accuracy to hold-out datasets. On the other hand, we note that the performance of NLIDBs is generally enhanced on "Hard" and "Extra" questions compared with base NLIDBs (the **All*** and **Cmp** columns). In particular, SyntaxSQLNet, after augmented by the SS$^+$ scheme, achieves better accuracy for "Hard" and "Extra" questions compared with

[2]We are unable to evaluate DBPal since its augmentation is not released.

MR	Utterance and SQL Query
PR	u = return me the homepage of PVLDB.
	q = `SELECT DISTINCT journal.homepage FROM journal WHERE journal.name = "PVLDB"`
	u' = ~~return me~~ the homepage of PVLDB.
	q' = `N/A`
PS	u = return me the paper with more than 200 citations.
	q = `SELECT DISTINCT publication.title FROM publication WHERE publication.reference_num > 200`
	u' = tell me the paper with more than 200 citations.
	q' = `SELECT DISTINCT publication.citation_num FROM publication WHERE publication.reference_num > 200`
SS	u = return me the number of the organizations.
	q = `SELECT DISTINCT count(organization.name) FROM organization`
	u' = return me the amount of the organizations.
	q' = `SELECT DISTINCT publication.title FROM publication, organization, writes, author WHERE publication.pid = writes.pid AND writes.aid = author.aid AND author.oid = organization.oid`

Table 13: Erroneous outputs of NaLIR found by MT-Teql. u, u' denote original and transformed utterances. q, q' represent SQL queries generated from u and u', respectively.

"SyntaxSQLNet+aug." More importantly, it also achieves much better robustness as shown in Table 10. Hence, when envisioning that "Hard" or "Extra" questions are mostly encountered in usage scenarios, users can leverage SS$^+$ scheme to augment NLIDBs for simultaneously better accuracy and robustness.

6 DISCUSSION

6.1 Validity to Non-Neural NLIDB

While the NLIDBs assessed in Sec. 5.1 are based on deep learning, we note that MT-Teql can also be used to evaluate non-neural NLIDBs. To demonstrate MT-Teql's ability to non-neural NLIDBs, we reproduce NaLIR [17], a popular non-neural NLIDB, and evaluate it with MT-Teql on linguistic variations.[3]

We use the MRs defined in Sec. 3.1 to transform 194 utterances from the MAS benchmark [17] and obtain 2,924 utterances. We set up NaLIR according to the official instructions and test it against mutated utterances. We show that MT-Teql stresses NaLIR in terms of parsing natural language utterances. NaLIR generates inaccurate inquiries in front of several utterances mutated by **SS** and **PR**. We achieve 92.1% and 69.4% error rates. Particularly, 38.1% of errors is caused by simply changing the prefix of utterances (using **PS**). Other MRs are not applicable due to the characteristics of the MAS benchmark. Table 13 lists some NaLIR errors regarding different MRs. We show that after transforming utterances using **PR**, NaLIR fails to deliver valid queries in most cases. By changing the prefixes, NaLIR typically picks the wrong column. SS can effectively stress the testing target, as seen when benchmarking neural NLIDBs. We presume that replacing "number" to "amount" may drastically impact the SQL query. We also observe that **PI** is not applicable because all MAS benchmark utterances begin with *"return me,"* making it impossible to add a declarative prefix. We interpret the results as reasonable. According to a relevant survey [16], the performance of NaLIR largely depends on its mapping rules. Incomprehensive rules prevent NaLIR from smoothly processing real-world linguistic variants, resulting in erroneous SQL queries.

[3]Non-neural NLIDBs don't normally follow training-testing paradigms. They do not ship with a "training dataset" for MT-Teql to augment. It is also impractical to evaluate schema variants because it needs substantial database interactions. Nevertheless, MT-Teql can assess translations from utterances to SQL queries.

6.2 Comparison with NLP Behavioral Testing

NLP behavioral testing becomes popular in evaluating NLP model robustness and consistency [12, 20, 27, 45]. It often generates controlled counterfactuals using pre-trained language models (e.g., GPT-2 [25]). At this step, we launch empirical study to explore if existing NLP behavioral testing techniques can be leveraged in our research context. We report detailed evaluation results in Suppl. Material [18]. In short, we find that while sentences generated by representative behavioral testing techniques like CheckList [27] and Polyjuice [45] share seemingly similar meaning with the original utterances, many of the transformed utterances are no longer semantics-preserving in the context of NLIDB. Our findings thus illustrate the overall inadequacy of existing NLP behavioral testing in assessing NLIDBs. We leave designing ML-based NLIDB behavioral testing for future exploration.

7 RELATED WORK

NLIDB is a challenging topic that requires the ability to translate human utterances into corresponding SQL queries on given relational databases. It has been actively studied by both database and NLP communities for decades [16]. Generally, there are two research lines to tackle this problem. The first line leverages the strong power of deep learning models to generate complex SQL queries in an end-to-end manner [3–5, 14, 41, 42, 44, 46, 47, 49, 50, 54]. Another research line aims to extract key information by pre-defined parsing rules and then convert it into SQL queries [17, 31, 34]. Various benchmarks and evaluation methods are proposed to evaluate the performance of NLIDBs [8, 9, 11, 15–17, 34, 44, 50, 51, 54]. In particular, Spider benchmark advocates a focus on generating cross-table queries from cross-domain utterances [50], which take an important step for real-world adoption of NLIDBs. ParaphraseBench crafts a benchmark with 399 utterances on a single table with diverse linguistic variations to stress NLIDBs [44]. Suhr et al. pinpoint the general challenge on linguistic variations, novel database, query structure, and conventions in different datasets [39]. Data augmentation techniques are thus extensively used to enrich utterances and achieve higher performance [44, 48, 49].

8 CONCLUSION

In this paper, we present a comprehensive empirical study of neural NLIDBs. We introduce MT-Teql, an automatic framework to benchmark NLIDBs on real-world linguistic and schema variations. MT-Teql delivers a model-agnostic design and enables thorough assessment without manual efforts. We conduct fair and systematic evaluations on neural NLIDBs using MT-Teql. We further launch user studies to illustrate how MT-Teql helps developers assess NLIDB, and also augment NLIDB with findings of MT-Teql. We summarized several important findings, which can assist users to better benchmark and select neural NLIDBs in real-world development and usage scenarios.

REFERENCES

[1] Christopher Baik, Hosagrahar V Jagadish, and Yunyao Li. 2019. Bridging the semantic gap with SQL query logs in natural language interfaces to databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 374–385.

[2] Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 4560–4565. https://doi.org/10.18653/v1/P19-1448

[3] Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Global Reasoning over Database Structures for Text-to-SQL Parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3650–3655.

[4] Ursin Brunner and Kurt Stockinger. 2021. ValueNet: a natural language-to-SQL system that learns from database information. In *International Conference on Data Engineering (ICDE), Chania, Greece, 19-22 April 2021*. IEEE.

[5] Sanxing Chen, Aidan San, Xiaodong Liu, and Yangfeng Ji. 2020. A Tale of Two Linkings: Dynamically Gating between Schema Linking and Structural Linking for Text-to-SQL Parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*. 2900–2912.

[6] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 1998. *Metamorphic testing: a new approach for generating next test cases*. Technical Report. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong ….

[7] 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. *arXiv preprint arXiv:2004.03125* (2020).

[8] Deborah A Dahl, Madeleine Bates, Michael K Brown, William M Fisher, Kate Hunicke-Smith, David S Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *HUMAN LANGUAGE TECHNOLOGY: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

[9] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2020. Structure-Grounded Pretraining for Text-to-SQL. *arXiv preprint arXiv:2010.12773* (2020).

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1)*.

[11] Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. *arXiv preprint arXiv:1806.09029* (2018).

[12] Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, et al. 2020. Evaluating Models' Local Decision Boundaries via Contrast Sets. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*. 1307–1323.

[13] Tao Ge, Furu Wei, and Ming Zhou. 2018. Fluency boost learning and inference for neural grammatical error correction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1055–1065.

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

[15] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760* (2017).

[16] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.

[17] Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment* 8, 1 (2014), 73–84.

[18] Pingchuan Ma and Shuai Wang. 2021. MT-Teql: Evaluating and Augmenting Neural NLIDB on Real-world Linguistic and Schema Variations. Supplementary Material. https://bit.ly/MT-Teql-sm.

[19] Pingchuan Ma, Shuai Wang, and Jin Liu. 2020. Metamorphic Testing and Certified Mitigation of Fairness Violations in NLP Models. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. 458–465.

[20] Nishtha Madaan, Inkit Padhi, Naveen Panwar, and Diptikalyan Saha. 2021. Generate Your Counterfactuals: Towards Controlled Counterfactual Generation for Text. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 13516–13524.

[21] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment* 8, 10 (2015), 1082–1093.

[22] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[23] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*. 141–147.

[24] Patti Price. 1990. Evaluation of spoken language systems: The ATIS domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

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

[26] Marco Tulio Ribeiro, Carlos Guestrin, and Sameer Singh. 2019. Are red roses red? evaluating consistency of question-answering models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 6174–6184.

[27] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 4902–4912. https://doi.org/10.18653/v1/2020.acl-main.442

[28] Manuel Rigger and Zhendong Su. 2020. Detecting optimization bugs in database engines via non-optimizing reference engine construction. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1140–1152.

[29] Manuel Rigger and Zhendong Su. 2020. Finding bugs in database systems via query partitioning. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–30.

[30] Manuel Rigger and Zhendong Su. 2020. Testing Database Engines via Pivoted Query Synthesis. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 667–682.

[31] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Özcan. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1209–1220.

[32] Torsten Scholak, Raymond Li, Dzmitry Bahdanau, Harm de Vries, and Chris Pal. 2020. DuoRAT: Towards Simpler Text-to-SQL Models. *arXiv preprint arXiv:2010.11119* (2020).

[33] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on software engineering* 42, 9 (2016), 805–824.

[34] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. ATHENA++ natural language querying for complex nested SQL queries. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2747–2759.

[35] Peter Shaw, Philip Massey, Angelica Chen, Francesco Piccinno, and Yasemin Altun. 2019. Generating Logical Forms from Graph Representations of Text and Entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 95–106.

[36] Meina Song, Zecheng Zhan, and E Haihong. 2019. Hierarchical schema representation for text-to-SQL parsing with decomposing decoding. *IEEE Access* 7 (2019), 103706–103715.

[37] Ezekiel Soremekun, Sakshi Udeshi, and Sudipta Chattopadhyay. 2020. Astraea: Grammar-based Fairness Testing. *arXiv preprint arXiv:2010.02542* (2020).

[38] Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, Satinder P. Singh and Shaul Markovitch (Eds.). AAAI Press, 4444–4451.

[39] 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*. 8372–8388.

[40] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.

[41] Bailin Wang, Mirella Lapata, and Ivan Titov. 2020. Meta-Learning for Domain Generalization in Semantic Parsing. *arXiv preprint arXiv:2010.11988* (2020).

[42] 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* (2019).

[43] David HD Warren and Fernando CN Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *American journal of computational linguistics* 8, 3-4 (1982), 110–122.

[44] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. 2020. DBPal: A Fully Pluggable NL2SQL Training Pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2347–2361.

[45] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. 2021. Polyjuice: Generating Counterfactuals for Explaining, Evaluating, and Improving Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

[46] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436* (2017).

[47] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769* (2018).

[48] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing. *arXiv preprint arXiv:2009.13845* (2020).

[49] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018. SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 1653–1663.

[50] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium.

[51] 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*. 1050–1055.

[52] Jichuan Zeng, Xi Victoria Lin, Caiming Xiong, Richard Socher, Michael R Lyu, Irwin King, and Steven CH Hoi. 2020. Photon: A Robust Cross-Domain Text-to-SQL System. *arXiv preprint arXiv:2007.15280* (2020).

[53] Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. *arXiv preprint arXiv:2010.02840* (2020).

[54] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR* abs/1709.00103 (2017).