SIDeQ: Complex Program Annotations by Asking Non-Experts Simple Informative Denotation Questions

Anonymous ACL submission

Abstract

We propose a new framework, **SIDeQ**, that enables non-experts to indirectly annotate the meanings of natural language utterances by answering **S**imple **I**nformative **De**notation **Q**uestions. We take Text-to-SQL as a case study. Given a natural-language database query, SIDeQ generates a prior over SQL candidates by running a seed semantic parser (e.g., Codex), but it does not show these candidates to the annotators. Instead, it asks them to evaluate the natural-language query on various concrete databases, and upweights the candidates that are consistent with their responses. For efficient interactions, we synthesize these databases to maximize the expected information gain of knowing the correct evaluations, while keeping the question simple by reducing the database size. We build an interface based on SIDeQ and recruit non-experts to annotate a random subset of 240 utterances from the SPIDER development set. Our system with non-experts achieves the same annotation accuracy as the original SPIDER expert annotators (75%) and significantly outperforms the top-1 accuracy of Codex (59%). Finally, we analyze common mistakes by database experts without SIDeQ and those by non-experts unfamiliar with databases.

1 Introduction

The goal of semantic parsing is to map a natural language utterance u to a program s, which can be executed in an environment or possible world w (Dahl, 1989; Berant et al., 2013; Andreas et al., 2020). For example, given a user utterance u "*How old is the youngest person*," we can map it to the SQL $s =$ `SELECT MIN(AGE) FROM PEOPLE`, execute it on a database w about people, and return a value v to the user.[1] However, it is challenging to scalably collect program annotations for natural language utterances, since this requires experts in the target programming language.

The programming by examples (PBE) framework (Lavrac and Dzeroski, 1994) opens up a possibility: even though the non-experts cannot produce a program s that implements u, they can produce example input-output pairs (w, v) such that $v = s(w)$. Then a program synthesis algorithm can guess the target program s based on the examples. This framework has been applied to synthesize regular expressions (Gulwani, 2011), SQL queries (Wang et al., 2017), and visualization programs (Wang et al., 2021), among others. However, it might take our non-expert annotators a lot of effort to write down a sufficient set of example pairs for each utterance; moreover, some of these examples might not be necessary to determine the semantic parse.

We propose a new framework, **SIDeQ**, which combines the semantic parsing and the PBE paradigms and enables non-experts to annotate complex programs by answering **S**imple **I**nformative **De**notation **Q**uestions. Given an utterance u, we generate a prior p over program candidates; then we reduce our uncertainty as to the correct program by synthesizing a possible world w (e.g. database), evaluating the candidates on w, and asking the annotators which return value is correct. Since it is sometimes infeasible to pin down the correct candidate with one w, we iteratively reduce the entropy by interactively asking further questions in the same way. If annotators may make errors, even more questions are needed to achieve a small entropy, with redundant questions being useful (Figure 1 left).

For efficient interactions, we synthesize w at each step that 1) maximizes the expected information gain of knowing the correct return value v^* and 2) is simple enough that the annotators can easily evaluate u (Figure 1 right). Using information gain spares the annotators from spending their time on uninformative examples. Our framework is an instantiation of active learning (Settles, 2011), where

[1]Thus, s can be regarded as a function that maps an input w to an output v.

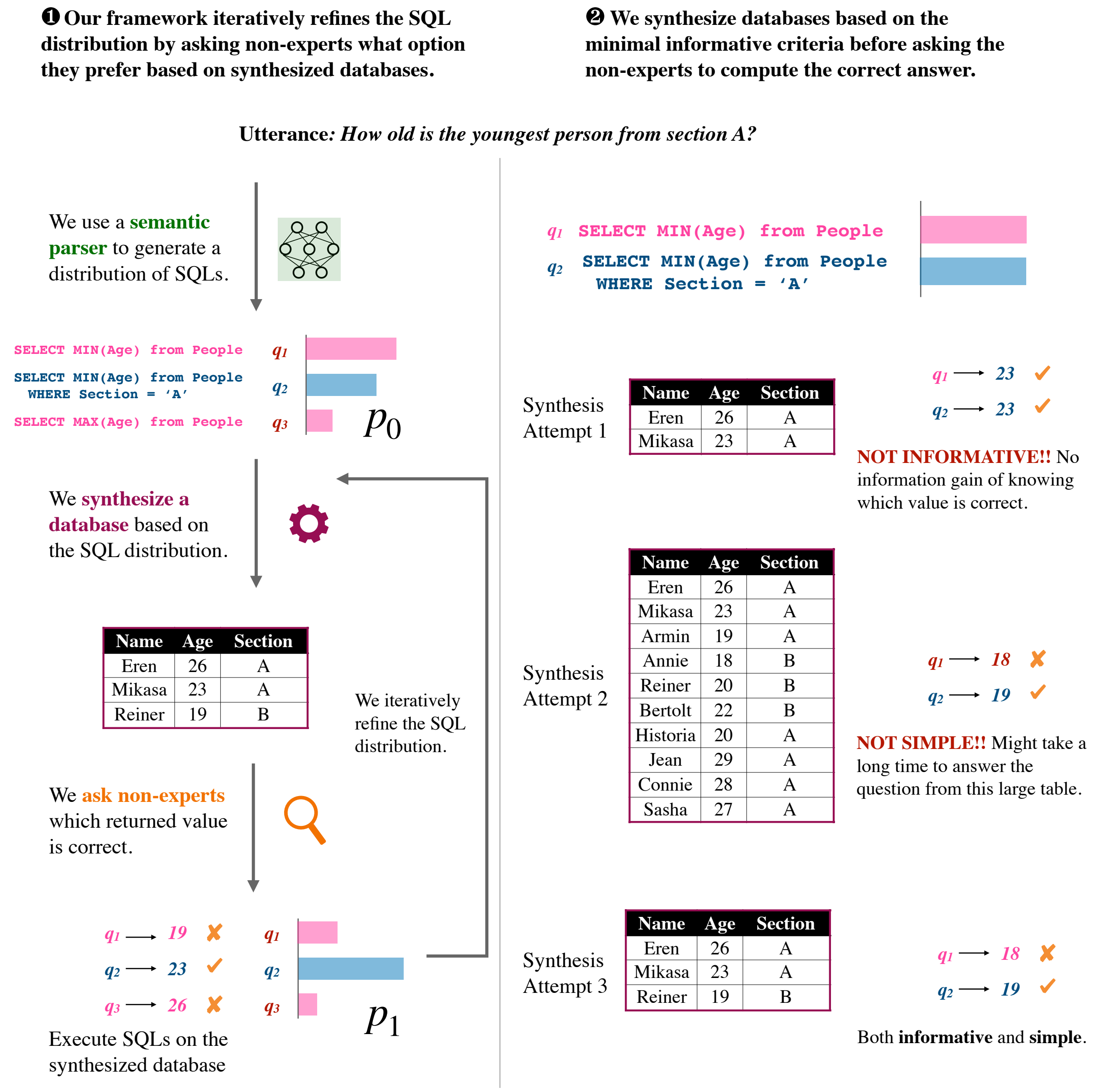

Figure 1: **Left**: Our SIDeQ framework for annotating complex programs with non-experts. **Right**: We optimize the database content to generate a simple and informative question about the denotation.

we maintain a prior over the function space and actively query the annotators with function input that maximizes information gain.

We apply our framework to annotate Text-to-SQL data, where each s is a SQL query, p is generated by Codex (Chen et al., 2021a) , w is a database, and the annotators' effort is approximated by the number of records in w. §3 proposes a practical optimization algorithm that maximizes the information gain of a small database. Using the optimized database w, a simulated perfect annotator with SIDeQ can achieve 91% accuracy on SPIDER by inspecting two databases per utterance with on average 6 records, while using the original database from SPIDER can only achieve 86% accuracy with on average 30K records (§5).

We built an interface designed to be user-friendly based on SIDeQ and evaluated its practical value (§6). We select a random subset of 240 utterances from the SPIDER development set, improve the SQL annotation with SIDeQ, and treat them as our gold standard. We recruit 11 English-speaking non-expert participants to annotate them with our interface, with each utterance examined by 2.5 non-experts on average. Their annotations allow us to achieve the same accuracy as the original SPIDER annotation performed by database experts (75%), which significantly outperforms the top-1 accuracy of Codex (59%). Finally, §7 analyzes errors made by database experts without SIDeQ and by our non-experts unfamiliar with databases.

In summary, we propose the SIDeQ framework that enables non-experts to annotate complex programs, a practical algorithm to find a small database that maximizes information gain, a software interface to annotate Text-to-SQL data, and an annotation study with non-expert subjects. To facilitate future research, all code and data will be distributed under the CC BY-SA 4.0 license upon publication.

2 Framework

Basic Setup As our case study of SIDeQ, we will show how to synthesize a SQL query s from a natural language utterance u in the context of a database schema c.[2] s must capture the meaning of u and work properly for *any* database with schema c.

We first feed c and u to a baseline semantic parser (e.g. Codex) to generate a distribution p over SQLs. We take $p(s)$ as our prior probability that s is correct. We aim to improve p by posing useful denotation questions to non-expert annotators.

Each question is generated by the following steps: 1) synthesize a database w consistent with the schema c, 2) display u along with w and up to K most likely return values in random order, and 3) ask the annotator to choose one of the values (or "none of the above") as the appropriate return value for the natural-language question u. Given the observed response o, our posterior distribution over s is then

$$p(s \mid u, w, o) \propto p(s)\, p(o \mid u, s, w) \quad (1)$$
$$= p(s)\, p(o \mid s(w))$$

where $p(o \mid s(w))$ is our estimated probability that the annotator would have responded with o if s were the correct denotation of u and therefore $s(w)$ were the correct return value. For example, if we assume that the annotator always responds correctly, then the posterior is obtained from the prior simply by zeroing out all SQLs s that are inconsistent with o (that is, such that $s(w) \neq o$, or such that $s(w)$ is shown as a choice in the case where $o =$"none of the above") and then renormalizing.

This procedure can be iterated by asking a series of questions. We define $p_t(s) \stackrel{\text{def}}{=} p(s \mid u, w_1, o_1, \ldots, w_t, o_t) = p_{t-1}(s)\, p(o_t \mid s(w_t))$ to be the posterior after t rounds of interaction, with p_0 being the prior and p_T being our final estimate. This basic setup is illustrated at the left of Figure 1.

We output the most likely SQL in p_T as our 1-best annotation, which is then compared to a gold standard to evaluate our framework. In future work, the "soft labels" provided by the full distribution p_T could be used to retrain the semantic parser $p(s)$ as well as the annotator behavior model $p(o \mid u, s, w)$[3]—that is, the two factors of (1)—and the updated models could then be used in the same manner as below to select further questions whose answers would further reduce the models' uncertainty,[4] closing the active learning loop.

Criteria for Synthesized Databases In general, SIDeQ on round t must choose an multiple-choice denotation question and an annotator to route it to. In our case study, the annotator is fixed in advance and the question is fully determined by choosing a database w_t.

We aim to choose an *informative* question w_t. Ideally we want w_t to give different answers on different high-probability candidates s, so that the annotator's response o_t is likely to substantially reduce our entropy. We quantify this reduction as the expected information gain of w_t,

$$I_{p_{t-1}}(w_t) \stackrel{\text{def}}{=} H(p_{t-1}) - \mathbb{E}_{o_t \sim p_{t-1}}[H(p_t)] \quad (2)$$

where H returns the Shannon entropy of a distribution over the candidates s.[5]

Recall that p_t depends on the question w_t and also on the future response o_t, which we assume will be distributed as $p_{t-1}(o_t) = \sum_s p_{t-1}(s) p(o_t \mid s(w_t))$. Equivalently, the expected information gain of w_t is the mutual information under p_{t-1} between two random variables—the annotator's response O_t and the SQL S, neither of which is yet known.

[2] The schema c specifies the table and column names, along with constraints that the database must satisfy, e.g., value types, uniqueness, and foreign key.

[3] This can be regarded as an EM procedure for (locally) maximizing the incomplete-data likelihood $\sum_s p(s, o_1, \ldots, o_T \mid u, w_1, \ldots, w_T)$. Estimation of p_T for all utterances (the E step) is alternated with retraining the models on these soft labels (the M step) until convergence.

[4] By using a richer model of annotator behavior, we could estimate the error rates of individual annotators on different types of questions, which would help us to choose appropriate questions and route them to appropriate annotators.

[5] Instead of using the expected reduction in entropy, an alternative would be the expected reduction in Bayes risk, $\mathbb{E}_{o_t \sim p_{t-1}}[\mathbb{E}_{s \sim p_t}[\mathbb{E}_{\hat{s} \sim p_{t-1}}[\text{loss}(\hat{s} \mid s)] - \mathbb{E}_{\hat{s} \sim p_t}[\text{loss}(\hat{s} \mid s)]]]$. Here $\text{loss}(\hat{s} \mid s)$ quantifies the loss of using $\hat{s}$ in the application when s is correct, e.g., some measure of its expected error on a random database w. This focuses the questions on resolving consequential errors, not those where $\hat{s}(w) = s(w)$ for most w.

We also aim to keep our questions *simple*. For example, if w_t is a database with 1,000 records (rows), it may take the non-experts a long time to calculate the correct return value. We denote their effort to choose o_t from the list of $\leq K$ options as $|w_t|$ and model it crudely as being proportional to the total number of records.[6]

The right of Figure 1 illustrates these two criteria visually. §3 proposes a heuristic algorithm $\mathcal{A}$ that seeks a database w_t that is both simple and informative.[7]

Choosing u to Ask Questions If only one utterance needs annotation, we would repeatedly ask non-experts to answer questions for this utterance until our budget is exhausted. Annotator error means that we can never be 100% certain to have identified s. However, we usually need to annotate a set of utterances; we need to decide when to stop short of certainty and move on to the next utterance. In our work, we ask 1–3 questions consecutively for each utterance, stopping the interaction after round t if 1) some candidate s has $p_t(s) > 0.9$,[8] or 2) $\mathcal{A}$ fails to find a small informative database.[9]

In principle, we could stop after 0 questions if p_0 is already confident about a candidate. However, Codex is not calibrated on our task and may be *falsely* confident, so we reduce risk by asking at least one question per utterance. We could also switch freely among utterances in search of questions that yield the highest information gain per unit effort. However, followup questions about the same utterance are presumably cheaper than switching to a new utterance; this phenomenon should be reflected in the effort measure (footnote 6) and also motivates a non-myopic policy (footnote 7). We leave these refinements to future work.

[6]Of course this effort model could be enriched, to consider the number of options as well as the detailed structure of u and w_t. We could tune the parameters of such a model to predict observed annotator response times.

[7]Note that this is a so-called *myopic* (greedy) policy that only optimizes the action w_t in isolation. In principle, we could do a better job by looking ahead to future rounds. However, the myopic strategy is common in interactive protocols for information acquisition, such as adaptive quadrature or Bayesian optimization (Schulz et al., 2018).

[8]We tuned this threshold such that, under the prior p_0, an ideal annotator will end up answering an average of two questions per utterance.

[9]For example, the SQL query `SELECT B FROM TABLE LIMIT 100` returns the first 100 records of the column `B`. It cannot be distinguished from `SELECT B FROM TABLE` by any w whose `TABLE` has ≤ 100 records.

3 Optimizing the Database Content

We maximize the information gain over all databases that conform to schema c (which controls the number of tables in the database)and have at most $R = 15$ total records. Formally, we search for

$$w_t^* = \operatorname*{argmax}_{w_t : |w_t| \leq R} I_{p'}(w_t). \quad (3)$$

where p' is a truncation of p_{t-1} to just the top-16 SQL candidates (for computational efficiency). We will write $I(w)$ below, suppressing the subscripts t and p', since they are fixed throughout the optimization process.

Our algorithm can be summarized as "fuzz-then-drop." We first perform **fuzz**ing by randomly generating a large number of large databases as in Zhong et al. (2020)—see Appendix A for further details—and keep the database w^0 that maximizes the expected information gain $I(w^0)$. We then iteratively **drop** records from w^0 to attempt to satisfy the simplicity criterion.

We use superscript ℓ to denote the iteration of dropping records. Starting from $\ell = 0$, we randomly drop 5% of the records from w^ℓ to obtain $w^{\ell+1}$. If this results in a worse database, in the sense that $I(w^{\ell+1}) < I(w^\ell)$, we are willing to retry up to 20 times in hopes of randomly finding a $w^{\ell+1}$ that is not worse than w^ℓ. Once we have our final $w^{\ell+1}$ (which may or may not be worse), we repeat the procedure, continuing through $w^0, w^1, \ldots$ until we reach an empty database w^L after $L = \Theta(\log |w^0|)$ iterations. Let $\hat{w}$ be the best database smaller than R that we encountered during these iterations:

$$\hat{w} = \operatorname*{argmax}_{w \in \{w^l : \ell \in [L]\}, |w| \leq R} I(w) \quad (4)$$

Since our algorithm is randomized, we repeat it 3 times and let w^* be the $\hat{w}$ with the largest $I(\hat{w})$. Finally, we simplify w^* by dropping tables and columns that were not mentioned by any of the top-16 SQL candidates (those in p').

Our algorithm of dropping records from a large informative database is heavily inspired by Miao et al. (2019), which, given a database w such that $s_1(w) \neq s_2(w)$, provably finds the smallest subset of records in w such that s_1 and s_2 return different values. Nevertheless, their algorithm works only for a restricted family of SQLs and cannot be adapted to optimize information gain. Our algorithm does not provide any provable optimality guarantee, but is more flexible and practical.

In practice, however, applying the above algorithm naïvely can generate unnatural databases and lead to vacuous SQL execution, confusing the annotators. In Appendix A, we illustrate several typical confusions (Figure 3) and discuss how we fix them.

4 Dataset and Evaluation Metrics

We benchmark SIDeQ on the development set of SPIDER (Yu et al., 2018), an English Text-to-SQL dataset with 1034 utterance-SQL pairs distributed under the CC BY-SA 4.0 License. SPIDER is divided into *domains*, where each domain has a collection of (u, s) pairs based on the same database schema c.

4.1 Dataset

We use half of the 1034 (u, s) pairs (the *validation split*) to tune our annotator interface (§6) and our fuzz-then-drop algorithm, using a simulated annotator (§5). We use the remaining half (the *evaluation split*) to evaluate our system with simulated annotators (§5), and from these drew a random subset of 240 utterances[10] to evaluate our system with actual human annotators (§6). To make the latter evaluation less noisy, we checked and corrected the SQL annotations on these 240 utterances (§7.1), resulting in corrections to 61 of them. We also identified and fixed several issues with the SPIDER database schema and content, with details in Appendix B. The corresponding author of the SPIDER dataset endorses our re-annotation and database updates.

4.2 Obtaining the SQL Query Prior p_0

We generate a prior over SQL candidates using the Codex (Chen et al., 2021b) language model with few-shot prompting. Given an utterance u with schema c from the validation or evaluation split, we create the prompt (Figure 4) by concatenating a linearization of c, then eight (u_i, s_i) pairs from the validation split[11] associated with the same schema c, and finally the utterance u itself. Some of the examples (u_i, s_i) are chosen randomly while others are chosen because u_i has high TF-IDF similarity to u. We randomly sample 200 prompts for u by choosing different examples, and for each prompt, we ask Codex to generate 20 completions (SQL queries). Full details are given in Appendix C.1.

We then filter out non-executable candidates and merge apparently semantically equivalent ones by testing them on 1K randomly generated databases with code from Zhong et al. (2020). This merging eliminates competition among equivalent surface forms (Holtzman et al., 2021), i.e., spurious ambiguity. We define p_0 to be the empirical distribution of semantic equivalence classes in our samples; thus, each s in §2 is not actually a SQL query but an equivalence class.

Treating the original SPIDER annotation as the ground truth, the top-1 accuracy on the entire development set is 72% and the top-16 accuracy is 94%. More details are in Appendix C.2. These numbers are not comparable to prior works, which usually evaluate on unseen database domains in a zero-shot manner (harder than our setting) but do not require predicting string literals and `DISTINCT` keywords (which we need for execution).

4.3 Evaluation Metrics

As mentioned in §2, our method produces a 1-best SQL query for each utterance. We decompose its errors into three categories. First, recall from §4.2 that we only consider 4000 samples (some being duplicates), so the correct SQL might not appear in our candidate list. Second, recall from §2 that the interaction may stop before the correct candidate becomes the most probable one. Finally, the annotators sometimes respond incorrectly.

To reflect these three types of error, we calculate 1) the *candidate ceiling*—whether *any* candidate is semantically correct; 2) the *interaction ceiling*—our 1-best accuracy if the annotator always responds correctly; and 3) the *annotation accuracy*—our 1-best accuracy given the actual annotations we collected.

5 Simulated Evaluation

We benchmark SIDeQ on the evaluation split under the idealistic assumption that 1) the SQL query provided by SPIDER is always correct, and 2) our annotator always responds correctly by choosing the value returned by that SQL query.

The candidate ceiling is 96% and the interaction ceiling is 91%, which is in fact much higher than the current annotation error rate in SPIDER, as we will see in §6. We only need to interact with our idealized annotator for 1.8 rounds on average, and the databases that we present contain only 5.52 records on average. More detailed statistics can be seen in Appendix D.

While SIDeQ aims to construct simple and in-

[10] Balanced across domains as much as possible.

[11] Excluding pairs where s_i matches the correct answer s.

formative questions about utterances, the example databases that were released along with the SPIDER domains yield less informative questions: using them lowers the interaction ceiling by 5%. They are also far less simple: their median size is 72 records (and their mean size is 33,295 records due to large outliers). Human annotators cannot feasibly evaluate an utterance on such large databases.

6 Human Interaction Study

We built an interface designed to be user-friendly (§6.1). We recruited 11 non-experts to annotate them with our interface (§6.3), aggregated their responses by learning a model of annotator accuracy (§6.4), and benchmarked their performance with the newly established gold standard (§6.5).

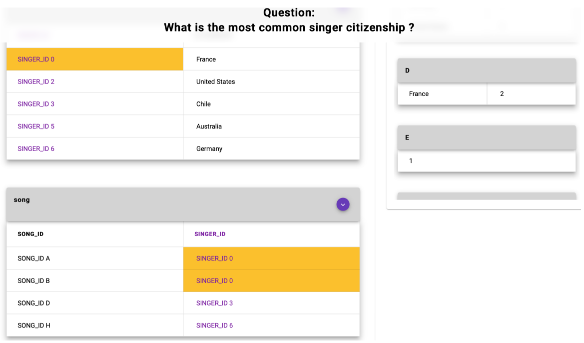

Figure 2: A screen shot of our annotation interface (§6.1). Appendix F and Figure 7 include more details.

6.1 Annotation Interface

Figure 2 shows our interface. As described in §2, the annotator needs to choose the correct value on the right of the screen based on the utterance u (top) and the database w (left). In general, a value may be a relational table. The annotator can also choose to report that the question is ambiguous/confusing or none of the choices are correct. To make it easier to reason about w, we highlight all cells in all tables that have the same value as the the cell that the cursor hovers on. Appendix F includes more details about our interface.

6.2 Expert Annotation

To establish a clean gold standard, two of the authors annotated all 240 utterances using our own SIDeQ system. Whenever our two responses to a SIDeQ question were different, we reached a consensus through discussion. We closely examined the utterances where SPIDER's SQL did not yield our consensus response on one or more questions, and corrected the SPIDER annotation if we felt that our responses were strictly better. To avoid biasing against the original annotations, we stuck to the original ones whenever there were ambiguities, and we double-checked each corrected annotation by additionally writing down reasons why it we felt it was better. As mentioned in §4.1, we ultimately corrected 61 out of the 240 SQL annotations. §7.1 analyzes these corrections in greater detail.

6.3 Non-Expert Annotation

We split the 240 utterances into 8 units, each of which contains 30 utterances across 4–5 database domains and proved to take 1–2 hours to annotate with our interface (2–4 minutes per utterance).

In this experiment, we configured our system to treat all annotators identically. Thus, all annotators for utterance u received the same first question—the questions for one annotator were not influenced by the responses from previous annotators (even though that is a more effective way to choose questions). For the annotator behavior model $p(o \mid u, s, w)$ in equation (1), we assumed that every annotator would have an 0.3 chance of responding uniformly at random, and would otherwise give the correct response.

Recruiting Non-Experts We recruited university students who 1) are not pursuing/have not received a Computer Science degree and 2) have no prior experience with SQL to complete the annotation tasks. Each annotator could annotate any number of units (from 1 to 8) as they wished, but had to annotate them fully. For each unit we reward them with \$15 as a base payment and \$5(\$10) bonus if their response agreed with our corrected gold standard $> 85\%(95\%)$ of the time. We recruited in total 11 participants and received 20 units of annotation, and hence each utterance was examined by 2.5 participants on average. For each utterance, we asked them 1.84 denotation questions on average and the databases that we present contain only 8.71 records on average

Participation Procedure We ask each non-expert annotator to: 1) sign a consent form to participate in the study, 2) watch a 12-minute video tutorial that contains our annotation instructions and explains the basics of foreign and primary keys, 3) complete the annotation task, and 4) fill out an exit survey which collects information about their major and prior programming experiences. Our tutorial video[12] can be seen here and its transcript can be seen in Appendix G. An example unit of the annotation task can be seen here.

6.4 Learning an Annotator Accuracy Model

After the annotations were collected, we used them to improve the annotator error model of §6.3 by learning parameters α_n for each annotator n and β_d for each SPIDER domain d. For a given n and d, the chance that the annotator answers at random is no longer fixed at 0.3, but is modeled as $\sigma(\alpha_n+\beta_d+b)$, where σ is the logistic function and b is a bias term. Larger α_n and β_d predict higher error rates.

We learn the parameters as outlined in footnote 3, by optimizing the log of the incomplete-data likelihood $\sum_s p(s, o_1, \ldots, o_T \mid u, w_1, \ldots, w_T)$, summed over all utterances u. Of course, the conditional distribution shown here is now sensitive to the domain d of u and the annotator n_t who answered question w_t. Notice that just as in other adaptive crowdsourcing work (§8), we assume that we do not have access to the gold value of s, but must impute it. We will tend to learn a lower error rate for annotators who tend to agree with other annotators and with Codex, in order to explain these apparently non-random agreements.

6.5 Results

After tuning the annotator error model as above,[13] we make our 1-best SQL predictions as explained in §2. On this dataset, the candidate ceiling is 88% and the interaction ceiling is 84%. Our method achieves 75% accuracy, which significantly outperforms the top-1 candidate of Codex (59%) and is comparable to the accuracy of the original SPIDER annotation performed by database experts (75%). A breakdown is shown in Table 3.

This does not imply that non-experts with SIDeQ can necessarily replace expert annotators. At least in this experiment, our non-experts are still far from recovering the full gold standard, which was established by experts with the help of SIDeQ (§6.2). The breakdown statistics based on difficulty split can be seen in Table 3. In addition, our method relied on a baseline semantic parser p_0 that was constructed using existing expert annotations (in our case, used to prompt for Codex). Still, our hope is that if we start with any baseline semantic parser that assigns non-negligible probability to the correct denotations, then instead of improving it by ordinary supervised training on a set of imperfect expert denotations, we could instead reach or even surpass the same accuracy by running SIDeQ with experts and/or non-experts *for long enough*, perhaps even at lower total cost.

[12] Voiceover removed due to the anonymity requirement.

[13] We do not tune the semantic parser in this paper. Indeed, we do not even have the ability to fine-tune Codex (although in principle we could have built our own tunable semantic parser, which might consult Codex).

7 Analysis

7.1 Sources of Error in Expert Annotations

We discuss two representative cases below, and more in Appendix H.

Ties for Extremals For the utterance "Who is the youngest person?", the SPIDER annotation is `SELECT NAME FROM PEOPLE ORDER BY AGE LIMIT 1`. As SIDeQ discovers, in case of ties, non-experts prefer a SQL that will return *all* of the people who have the smallest age, not just return the first one. 28 out of the 61 updated annotations fall into this category.

`INNER JOIN` vs. `LEFT JOIN` Suppose the utterance is "List singer names and number of concerts for each singer." and the database contains a table of singers and a table with records (s, c) if singer s performed in concert c. The SPIDER annotation only uses `INNER JOIN` and hence fails to return singers with count 0 (who have not performed in any concert). 8 of the updates fall into this category.

Remark Since most of the Text-to-SQL models had low performance 3 years ago, Yu et al. (2018) favored short SQL annotations to make learning easier. These annotation conventions were shared between training and test sets to form a coherent structured prediction task (internal validity). Now that structured prediction is working well enough that the predictions could be used in real-world settings, we should turn to assuring that the SQL annotations actually have the desired effects (external validity). SIDeQ can help here (§6.2).

7.2 Sources of Error in Non-Expert Responses

Ambiguous Utterances Consider the utterance "What are the names of properties that are either houses or apartments with more than 1 room?" Should it be parsed as "(house) or (apartment and room > 1)", or "(house or apartment) and room > 1"? Another example: "Count the number of friends Kyle has.” What to do when there are two students named “Kyle”?

Unnatural Databases Database schemas sometimes omit to specify common-sense constraints. For example, according to common sense, “BIRTHDAY + AGE" should always yield the current year, so sorting by BIRTHDAY ascendingly is equivalent to sorting by AGE descendingly. However, SIDeQ looks for databases w that distinguish between these two strategies, and in fact it is able to synthesize them from the SPIDER example database because some of the records in that database do not conform to this unstated constraint. These databases are obviously unnatural and confuse the non-experts.

Heavy Computations It is hard for the annotator to do arithmetic, e.g., find the average of eight 9-digit values. To help SIDeQ avoid demanding such computations, we should improve our annotator effort model to recognize their difficulty.

8 Related Work

Semantic Parsing Semantic parsers have improved significantly over the past decades (Zettlemoyer and Collins, 2007; Jia and Liang, 2016; Scholak et al., 2021a). Recent large pretrained models can perform the task without task-specific architectures (Scholak et al., 2021b) or even in a zero/few-shot manner (Shin et al., 2021; Brown et al., 2020; Chen et al., 2021a). However, generating semantic parsing datasets is still challenging since it requires experts. Wang et al. (2015) addresses this by synthetically generating logical forms, using templates to explain them in natural language, and asking non-expert crowdworkers to paraphrase them. However, the paraphrases are usually restricted in linguistic diversity (Larson et al., 2020). Ideally we want non-experts to annotate programs based on naturally occurring utterances, and we predict SIDeQ will achieve higher accuracy with better seed semantic parser in the future.

Programming by Example PBE has been applied to synthesize regular expressions (Gulwani, 2011), tensor manipulation (Shi et al., 2020), data analysis (Bavishi et al., 2019), and visualization (Wang et al., 2021) programs, etc. Our work can be extended to tackle these problems as well as long as there is a seed semantic parser and we can optimize program inputs/worlds to design simple and informative denotation questions.

Some other recent works such as Ye et al. (2020); Baik et al. (2020) also try to combine semantic parsing with PBE. However, both of them require the users to provide the input output examples, which can be time-consuming to write. Pasupat and Liang (2016) asked non-experts denotation questions by synthesizing table inputs, but they did not optimize for question simplicity and focused on a simpler single-table setting.

Database Research The semantics of SQL have been extensively studied by the database research community. More related to our work, Green et al. (2007) and Chu et al. (2017b) develop methods to prove semantic equivalence of SQLs, Wang et al. (2017) synthesizes SQL from input-output examples, Chu et al. (2017a) searches for a database (counterexample) that makes two SQL return different values and Miao et al. (2019) minimizes the size of such a counterexample.

Adaptive Crowdsourcing Under SIDeQ, some questions are inherently difficult to answer and competent annotators significantly contribute towards the final accuracy. How to find the right annotators to answer the right questions and weight their responses appropriately, with as little supervision as possible? Like us, Bachrach et al. (2012) and Whitehill et al. (2009) model each individual annotator’s capability and each question’s difficulty and learn these parameters through agreement information, and Yan et al. (2011) explores an active learning setup. The line of work emerging from these papers strongly influenced our perspective.

AI-Augmented Annotation An emerging line of “human-in-the-loop” systems (which have a long history in machine translation) constructs datasets using AI-generated candidates re-ranked/filtered by (a learned model of) human preferences (Stiennon et al., 2020; Wiegreffe et al., 2021). It is increasingly important to determine human preferences over complex outputs, such as full-book summaries (Wu et al., 2021). Our work presents a strategy for an AI system to rerank complex outputs (formal representations of denoted meanings) by asking simple informative questions of annotators who do not have to understand the outputs directly. The annotators’ responses feed back to improve the system’s predictions and focus its future questions.

9 Ethical Considerations

Our human interaction study was approved by the university Institutional Review Board and our survey and interface did not collect any personal identifiable information. We note that our system is still far from perfect, so it should not be used to synthesize SQL queries or other semantic forms for high-stakes scenarios without a careful analysis of errors and the downstream harms that they might cause.

References

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Yoram Bachrach, Thore Graepel, Thomas P. Minka, and Jo W Guiver. 2012. How to grade a test without knowing the answers - a bayesian graphical model for adaptive crowdsourcing and aptitude testing. In *ICML*.

Christopher Baik, Zhongjun Jin, Michael J Cafarella, and HV Jagadish. 2020. Constructing expressive relational queries with dual-specification synthesis. In *CIDR*.

Rohan Bavishi, Caroline Lemieux, Roy Fox, Koushik Sen, and Ion Stoica. 2019. Autopandas: neural-backed generators for program synthesis. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–27.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Anthony Chen, Pallavi Gudipati, Shayne Longpre, Xiao Ling, and Sameer Singh. 2021a. Evaluating entity disambiguation and the role of popularity in retrieval-based NLP. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4472–4485, Online. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021b. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Shumo Chu, Chenglong Wang, Konstantin Weitz, and Alvin Cheung. 2017a. Cosette: An automated prover for sql. In *CIDR*.

Shumo Chu, Konstantin Weitz, Alvin Cheung, and Dan Suciu. 2017b. Hottsql: Proving query rewrites with univalent sql semantics. *ACM SIGPLAN Notices*, 52(6):510–524.

Deborah A. Dahl. 1989. Book reviews: Computer interpretation of natural language descriptions. *Computational Linguistics*, 15(1).

Todd J Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40.

Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1):317–330.

Ari Holtzman, Peter West, Vered Shwartz, Yejin Choi, and Luke Zettlemoyer. 2021. Surface form competition: Why the highest probability answer isn't always right. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7038–7051, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Stefan Larson, Anthony Zheng, Anish Mahendran, Rishi Tekriwal, Adrian Cheung, Eric Guldan, Kevin Leach, and Jonathan K. Kummerfeld. 2020. Iterative feature mining for constraint-based data collection to increase data diversity and model robustness. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8097–8106, Online. Association for Computational Linguistics.

Nada Lavrac and Saso Dzeroski. 1994. Inductive logic programming. In *WLP*, pages 146–160. Springer.

Zhengjie Miao, Sudeepa Roy, and Jun Yang. 2019. Explaining wrong queries using small examples. In *Proceedings of the 2019 International Conference on Management of Data*, pages 503–520.

Panupong Pasupat and Percy Liang. 2016. Inferring logical forms from denotations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23–32, Berlin, Germany. Association for Computational Linguistics.

Torsten Scholak, Raymond Li, Dzmitry Bahdanau, Harm de Vries, and Chris Pal. 2021a. DuoRAT: Towards simpler text-to-SQL models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1313–1321, Online. Association for Computational Linguistics.

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

Eric Schulz, Maarten Speekenbrink, and Andreas Krause. 2018. A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, 85:1–16.

Burr Settles. 2011. From theories to queries: Active learning in practice. In *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, pages 1–18. JMLR Workshop and Conference Proceedings.

Kensen Shi, David Bieber, and Rishabh Singh. 2020. Tf-coder: Program synthesis for tensor manipulations. *arXiv preprint arXiv:2003.09040*.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7699–7715, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2020. Learning to summarize from human feedback. *arXiv preprint arXiv:2009.01325*.

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive sql queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 452–466.

Chenglong Wang, Yu Feng, Rastislav Bodik, Isil Dillig, Alvin Cheung, and Amy J Ko. 2021. Falx: Synthesis-powered visualization authoring. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier Movellan, and Paul Ruvolo. 2009. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc.

Sarah Wiegreffe, Jack Hessel, Swabha Swayamdipta, Mark Riedl, and Yejin Choi. 2021. Reframing human-ai collaboration for generating free-text explanations. *arXiv preprint arXiv:2112.08674*.

Jeff Wu, Long Ouyang, Daniel M Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. 2021. Recursively summarizing books with human feedback. *arXiv preprint arXiv:2109.10862*.

Yan Yan, Romer Rosales, Glenn Fung, and Jennifer G. Dy. 2011. Active learning from crowds. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 1161–1168, Madison, WI, USA. Omnipress.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2020. Benchmarking multimodal regex synthesis with complex structures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6081–6094, Online. Association for Computational Linguistics.

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

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague, Czech Republic. Association for Computational Linguistics.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online. Association for Computational Linguistics.

Supplementary Material

A Other Synthesis Constraints

Overall, we follow the recipe of Zhong et al. (2020) to generate large informative databases that conform to a given schema c. We draw upon an existing database in this domain (provided by the SPIDER dataset in our experiments) to obtain plausible cell values. Following Zhong et al., we first synthesize cell values for all the “parent" columns (i.e., the columns that are being referenced), and then populate child columns with random subsamples from the parent columns.

Naturalness of $\hat{w}$ As shown in Figure 3 (a), unrestricted random cell values can confuse non-expert annotators unfamiliar with databases. Therefore, we now always use individual cell values in the existing databases[14] or their minor perturbations[15] to generate large informative databases, rather than synthesizing completely random values.

Database records might also be confusing even if individual cell values are not. For example, the annotator can be confused by counterfactual information where U.S. is in Asia as shown in Figure (b); therefore, we sometimes initializes w^0 with the existing database. The annotator can also be confused by uncommon patterns where two people have the same name but different IDs; therefore, we sometimes enforce a column to contain unique values as long as the column content in the existing database satisfies the uniqueness constraint.

Non-vacuous Execution Extremely small $\hat{w}$ frequently leads to undefined denotations. For example, since the maximum value for zero element is undefined, the correct denotation is `NULL`, which confuses non-expert annotators without computer science background (Figure 3 b). Therefore, we always add a small probability mass of `RETURN NULL` to the distribution $\mathcal{S}$, which incentivizes our algorithm to produce $\hat{w}$ such that other SQL candidates will return non-`NULL` values.

Even if the returned value is well-defined, small $\hat{w}$ can lead to confusion if some operators are not needed to answer the question. For example, in Figure 3 (e), asking the maximum over one element might appear confusing, as we do not need the max operator to obtain a correct denotation. Therefore, we always add into $\mathcal{S}$ a small probability mass of “neighbor queries" (Zhong et al., 2020) obtained by dropping aggregation operators and `WHERE` clauses from SQL candidates in $\mathcal{S}$. This incentivizes our algorithm to produce $\hat{w}$ such that the SQL candidates will meaningfully use their operators.

Managing Tradeoffs between two Criteria All the above tweaks make a tradeoff between the informative and the simplicity criteria in some way: we impose restrictions on w or modify $\mathcal{S}$ to decrease the annotator effort while sacrificing information gain we can potentially achieve. How do we decide when to apply certain tweaks?

In our paper, we always add small probabilities of neighbor queries and `RETURN NULL` to $\mathcal{S}$ and use cell values from the existing database. We then consider 3 types of tweaks that we sometimes apply: 1) w_0 satisfies the uniqueness constraint, 2) w_0 is initialized with an existing database, and 3) $|\hat{w}| \leq 15$ rather than 30. We define in total $2^3 = 8$ different “configurations" $0 \leq c < 8$, each of which specifies what subset of tweaks to apply. For example, $c = 6 = B110$ means we apply tweaks 1) and 2). We enumerate from $c = 7$ to 0 until $I(\hat{w}) \neq 0$; in other words, we start by applying all the tweaks and drop the tweaks gradually until we obtain a $\hat{w}$ with positive expected information gain.

B Fixing SPIDER Databases

We found several issues with the SPIDER databases and modified them as follows:

- Some SPIDER databases do not conform to the foreign key constraint, i.e. some of the children columns contain values not in the parent columns they are referring to. We enforce the foreign key constraint by dropping the illegal records.
- We identify missing foreign key constraints under some domains and add them.
- Some `Date` typed columns are string-valued and use English words to represent values, e.g. "nov1,2021"; as a result, "dec1,2021", which is chronologically later, will be considered smaller alphabetically. We fix this by canonacalizing date representations in a “`yyyy-mm-dd`" format.
- The `voter_1` domain does not contain an appropriate foreign key design; since fixing it

[14] Text-to-SQL datasets are usually released with databases with values.

[15] E.g., ± 1 for integer values.

(a)

Name	Age	Section
asdg	102	^&(#@
qwerty	200	pqogen

Unnatural values

(b)

Country	Continent
U.S.	Asia
Canada	Africa

Counterfactual content

(c)

ID	Name	Age
1	Eren	26
2	Eren	23

Uncommon pattern

Utterance: *How old is the youngest person from section A?*

(d)

Name	Age	Section
Reiner	26	B

Vacuous answer ⟶ `NULL`

(e)

Name	Age	Section
Eren	26	A

Vacuous operator ⟶ `26`

Figure 3: Examples of unnatural databases (above) and vacuous execution (below), which motivates several tweaks in Appendix A. In (a) the individual cell values are unnatural. In (b) the records contradict world knowledge. In (c) the database contains two persons with the same name, which is atypical (but possible). In (d) the denotation of the utterance is undefined, since we cannot take the maximum over zero element. In (e) we do not need the max operator to obtain the correct denotation, since there is only one person in section A.

would require an effort of re-annotating all 15 associated SQLs, we chose to exclude them from our evaluation.

We update the test suite for semantic evaluation (Zhong et al., 2020) accordingly based on the new database schema.

C Generating SQL Candidates

C.1 Prompting Codex

As sketched in §4.2, we obtain SQL program candidates through few-shot prompting, where the database schema is followed by 4 or 8 (with 50% probability) pairs of natural language utterances with their corresponding SQL queries from the SPIDER development set from the subset of utterance-SQL pairs associated with the same database schema. To select each in-context example, with probability 50% we choose a random example that has not been selected from the validation split, and with probability 50% we choose the most similar example that has not been selected based on TF-IDF similarity. Finally we append the target natural language utterance u to be annotated, and ask Codex to continue generating text after this prompt, which generates a candidate SQL program corresponding to u. An example prompt can be seen in Figure 4. We sampled 200 different prompts, which varied in their selected examples, and for each prompt we sampled 20 candidates from Codex with temperature=1.0 and top_p=0.95.

k	easy	medium	hard	extra	all
1	0.87	0.80	0.56	0.45	0.72
2	0.94	0.89	0.74	0.63	0.84
4	0.96	0.93	0.87	0.70	0.89
8	0.97	0.95	0.95	0.78	0.92
16	0.98	0.96	0.98	0.81	0.94
32	0.98	0.96	0.98	0.85	0.95

Table 1: The top-k accuracy for the SQL candidates generated by Codex on SPIDER (Yu et al., 2018), calculated on each split.

	easy	med	hard	extra	all
Candidate	0.99	0.97	0.98	0.88	0.96
OurDB	0.93	0.95	0.97	0.75	0.91
OrigDB	0.98	0.90	0.83	0.66	0.86

Table 2: The accuracy ceilings on each difficulty split. "med" stands for medium difficulty. Candidate means the candidate ceiling. OurDB means the accuracy ceiling achieved by querying a simulated annotator with the small informative databases we synthesize. OrigDB means the accuracy ceiling by querying with the databases released by the SPIDER dataset.

C.2 Top-K Accuracy

We report the top-k accuracy from the prior p_0 over SQL queries (§4.2) in Table 1, and graph the top-k accuracy curve in Figure 5.

D Simulated Interaction Statistics

The breakdown statistics of candidate and interaction ceiling (see §4.3) can be seen in Table 2, and the distribution database sizes and number of round of interaction can be seen in Figure 6.

E Human Annotation

We provide breakdown statistics on the annotation accuracy of different sources based on difficulty in Table 3.

```
CREATE TABLE Highschooler(
        ID int primary key,
        name text,
        grade int)
CREATE TABLE Friend(
        student_id int,
        friend_id int,
        primary key (student_id,friend_id),
        foreign key(student_id) references Highschooler(ID) ON DELETE CASCADE,
        foreign key (friend_id) references Highschooler(ID) ON DELETE CASCADE
)
[Other table schema omitted]

Write a query that answers "Count the number of high schoolers."
SELECT count(*) FROM Highschooler

Write a query that answers "What are the names of high schoolers who have 3 or more
friends?"
SELECT T2.name FROM Friend AS T1 JOIN Highschooler AS T2 ON T1.student_id  =  T2.id
GROUP BY T1.student_id HAVING count(*)  >=  3

[6 More examples omitted]

Write a query that answers "Find the average grade of all students who have some
friends."
_________[Models' Completion] ______
```

Figure 4: An example prompt we use for the Codex API. We obtain SQL program candidates through 4/8-shot prompting, where the database schema (orange) is followed by 4/8 pairs of natural language utterance and their corresponding SQL queries from the SPIDER development set, randomly sampled from the subset of queries associated with the same database schema. Finally we concatenate the target natural language utterance u to be annotated, and ask Codex to complete the prompt, which results in a candidate SQL program corresponding to u.

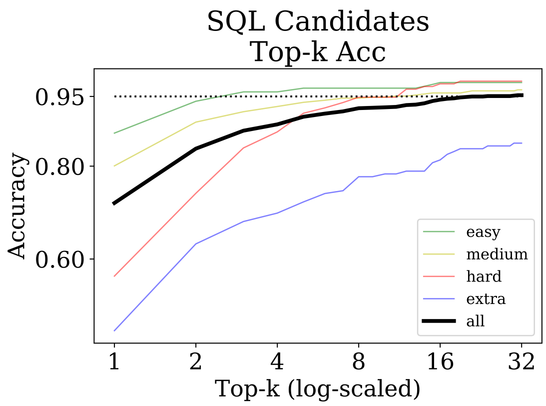

Figure 5: The top-k accuracy for the candidate SQL programs generated by Codex, after filtering and merging candidates. On each difficulty split we plot the curve of top-k accuracy (y-axis) and k (x-axis, log-scaled). The numbers can be seen in Appendix Table 1.

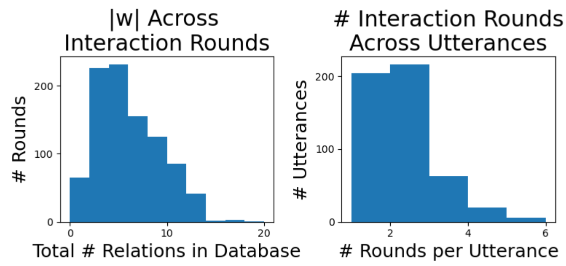

Figure 6: The interaction statistics using the SPIDER annotation to simulate an ideal annotator. **Left**: the distribution of the size of the database c across each round of interaction. **Right**: the distribution of the number of rounds across difference utterance.

	easy	medium	hard	extra	all
Candidate Ceiling	0.91	0.91	0.92	0.69	0.88
SPIDER	0.93	0.78	0.63	0.50	0.75
Codex	0.78	0.65	0.43	0.36	0.59
SIDeQ ^{r}	0.75	0.71	0.65	0.49	0.67
SIDeQ ^{m}	0.83	0.80	0.73	0.47	0.75

Table 3: 1-best accuracy of various SQL prediction methods, broken down by the difficulty level of the utterance (as categorized by SPIDER). Codex returns the most probable SQL according to p_0. SIDeQ r does the same, after eliminating SQLs that are inconsistent with the responses of a single randomly chosen annotator. SIDeQ ^{m} is our full method, which returns the SQL with the highest posterior probability after we fit our model of annotator error.

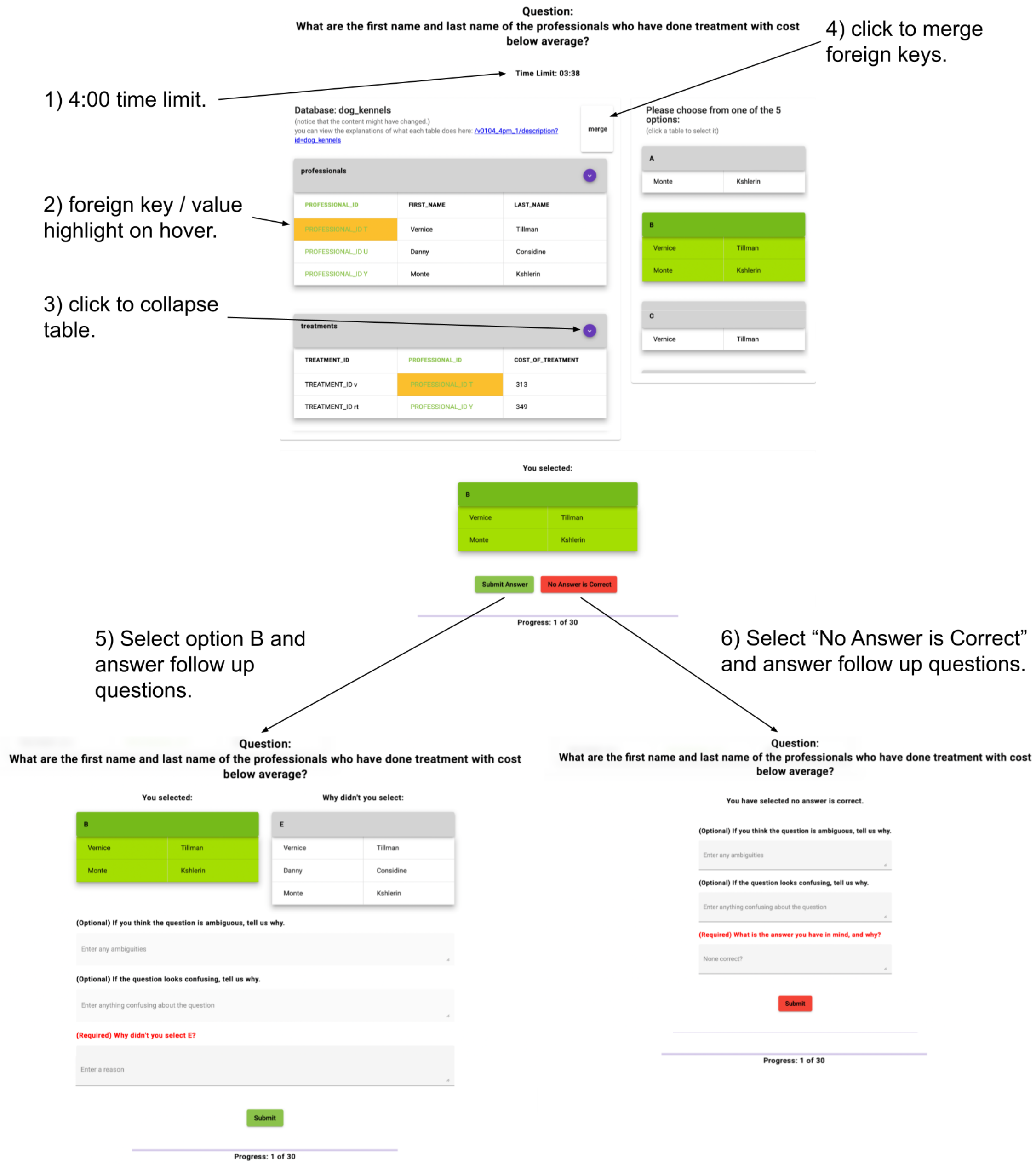

Figure 7: A detailed screenshot of our interface, and the logical flow of follow up questions.

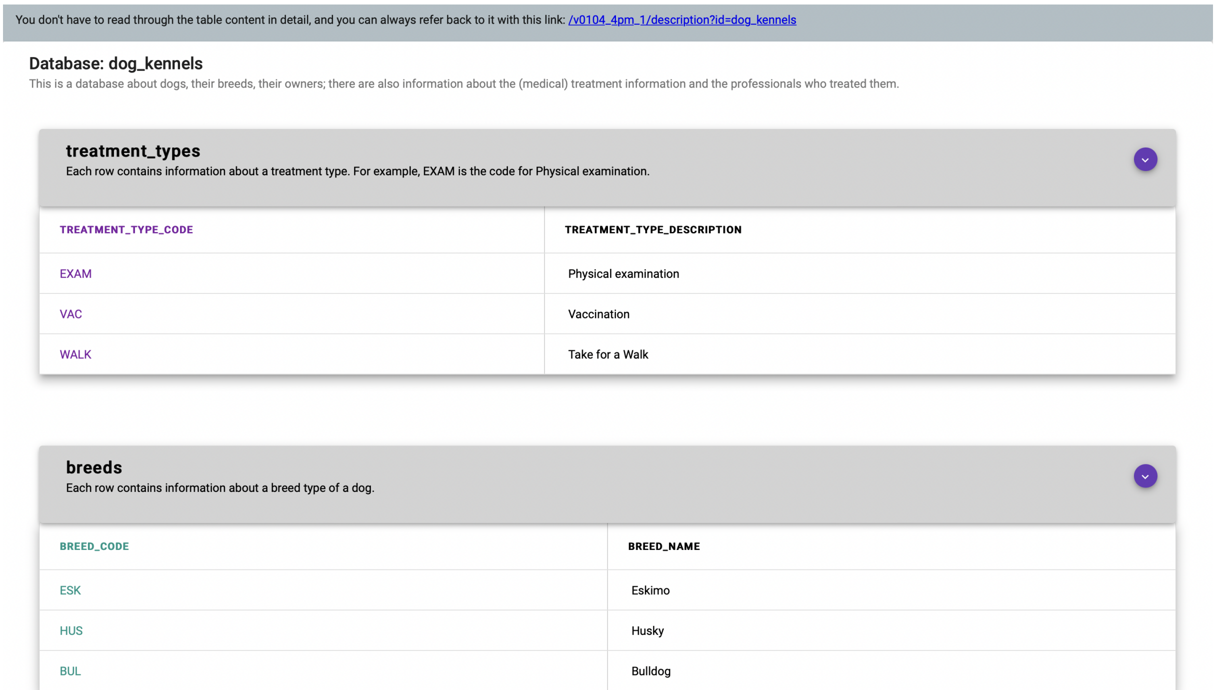

Figure 8: An example database description page presented to users before they start answering questions for that database.

F Interface

See Figure 7 for a detailed screenshot of our interface. We implemented the front-end of our interface with Angular and the back-end was built with flask and Redis. Users presented with a sequence of 40 distinct questions, and each question may have multiple rounds. For each round, the user is given a 4 minute time-limit before the interface automatically transitions to the next question. Before being asked questions on a new database, the user is presented with a page displaying all the tables in the database alongside descriptions we wrote for each table (see Figure Figure 8 for an example screenshot). When answering questions, the user is given a link back to this page for reference.

The user can either select one of the multiple choice questions presented or select "No Answer is Correct", and depending on their selection the user is presented with a differing set of followup questions. Regardless of their selection, we always ask the user two optional followups: "if you think the question is ambiguous, tell us why." and "if the question looks confusing, tell us why." In addition to these optional questions, we sometimes ask required followup questions. Specifically, if the user is on their final round and selects an answer which does not agree with the SPIDER annotation, we ask them why they didn't select the correct answer according to spider. Or if the user selects "No Answer is Correct", we ask "What is the answer you have in mind and why?" We use the users' answers to these followups to collect information on the users' reasoning in answering questions and to determine issues with the SPIDER dataset.

We implemented a number of features in our interface to minimize the annotator effort. One of the largest challenges in this task is answering questions across several foreign keys. We implement two distinct mechanisms to make this easier for users. Firstly we highlight all table values or foreign keys matching the value the mouse is currently hovering over. Secondly, we give the user the option to merge all foreign keys into a single table by pressing a "merge" button. We allow the users to choose when to merge because there is a trade-off; while merged mode can make reasoning about foreign keys easier, it also can significantly increase the width of the tables visible to the user.

Sometimes there are tables presented to the user that are not necessary for answering the question, so we give users the option to collapse tables to simplify their display.

G Video Transcript

Page 1 In this task, you will be asked to answer questions from several tables.

Page 2 Here is the overall idea. You will be given a question on the top of the page, several tables on the left of the page, and you need to choose one of the options on the right, that corresponds to the correct answer. In this question, you are asked to “Show name, country, age for all singers ordered by age from the oldest to the youngest.”. Therefore, we expect the correct option to list the information about Joe Sharp first, since he is older. We look at the options and B is correct. Notice that A is wrong because it does not list the information for all singers, and C is wrong because it lists the singers from the youngest to the oldest.

After you submit the answer, our system will ask you whether there is anything that appears ambiguous or confusing. We don’t need it for this question now.

Page 3 Let’s go through some more examples.

Page 4 In this question you are asked “How many singers do we have?” This is a tricky question. First notice that the tables have changed from before, so you need to re-read the table. Secondly, there are actually two singers, but they have the same name. You should consider them to be two different persons with the same name but different SSN, and hence choose B.

There is a time limit shown at the top of the page, and after 4 minutes the system will move on to the next question.

Page 5 Takeaways:

- Names are different from IDs. Two different people can have the same name.
- There is a time limit of 4 minutes for each question.

Page 6 In this question you are asked to find the song names of the singers above the average age. The average age is the mean of these 4 numbers, which is 34.5. The singers John and Rose have age above 34.5, so we can find their songs, which are sun and gentle man, which is D. Use a calculator if you need to!

Also, notice that there are other tables, but they are not relevant to the question. Feel free to ignore them. You can also choose to collapse them if that makes it easier, and you can click the button again to view it.

Page 7 Takeaways:

- Use a calculator if you need to.
- Not every table is needed.

Page 8 Here’s the same question and the same table. Let’s say somehow Sun and Gentleman is not one of the options, and then you should report that no answer is correct. Then we will ask you why you think no answer is correct. For example, you can write “gentleman and sun is correct. the average age is 34.5, John and Rose are above this age and have song gentleman and Sun”.

The system asks us why we didn’t choose A, we can answer “sun is by Rose, who is older than 34.5”. Please tell us enough information so that we can know why your choice is correct - for example if you just say “sun is also a correct answer”, it only describes the difference between the two options rather than explaining why it is correct. Giving us more information can help you win more bonus.

Page 9 Takeaways:

- Choose no option is correct and tell us why when you think no options are correct
- Tell us why you didn’t choose an answer when we ask you to do so.

Page 10 The question is “What are the full names of all players, sorted by birth date?” First, notice that there are a lot of answers in this case, and you need to scroll down to read through all of them. Secondly, there are a lot of ambiguities: for example, the question didn’t mention whether we should sort from youngest to oldest, or the reverse; secondly, the question does not mention whether the first and last name should be in the same column. For these reasons, A, B are both correct. C, D are wrong because the question does not ask for birthday information; F is wrong because it only lists one player and G is wrong for including birthday information. Then we can write in the response: “ABE are all correct; not sure if we should sort them from the oldest to youngest or reverse; also not sure whether to put the first and last name into the same column.” But still, make your best guess, let’s say, A.

Then we click submit, and the system asks us why we didn't choose C. We explain that "the question does not ask us for the birthday and it contains redundant information".

Page 11 Takeaways:

- There can be a lot of options. Make sure to read through every of them
- When the question is ambiguous and multiple answers are plausible, tell us why it is ambiguous and what are the plausible answers. But still, first make your best guess and submit.

Page 12 The question is "Give the names of countries that are in Europe and have a population equal to 80000." In this fictitious table, Brazil is in Europe and has a population of 80,000. Therefore, the correct answer is A, even though we know that Brazil is in fact in South America. However, it still cannot stop us from answering the question based on the table. Finally, there are many more countries in the world, beyond these three countries in the table, but we should pretend that there are only three countries in the world here.

Page 13 Takeaways:

- Try accepting the information from this table as much as possible and focus on the part useful for answering the question.
- If something is not present in the tables, pretend that it does not exist.

Page 14 Here are some more difficult tables.This is a database that contains information about battles and death. The overall description of the databases can be seen at the top of the page, which says: This database contains information about battles, death events, and ships. And then each table has its own description as well. For example, in the ship table, each row contains information about a ship, the 4th row means the ship D was lost in battle with ID 4, and you can look up information about battle 4 in the battle table. To make it convenient for you, whenever you move your cursor to a value, all the same values will be highlighted. Here we notice that according to the 5th row, Ship E was also lost in battle 4.

To view multiple tables at the same time, you can choose to zoom out, like this. Then you can zoom back in, like this. You can typically find this option in the Help panel of your browser. Again, if you think some tables are irrelevant, just collapse them like this.

You don't have to study the tables in detail, since they will probably change for the next question.

Page 15 Takeaways:

- You don't have to study the table content in great detail, since they will be changing.
- Zoom-in/out if you need to. You can find them in the helper panel of your browser.

Page 16 This question is "Show names, results and bulgarian commanders of the battles with no ships lost in the 'English Channel".

The question asks for certain battles namely, those that did not lose ships in the English Channel [pause]. Let's start by finding the battles that did lose ships in the English channel [pause]. Only Battle 5 did; it lost ship C there. So the other battles, Battles 0 and 7, lost no ships there. In fact, Battle 0 lost no ships at all, which is why it doesn't show up in the second table. We find the names of Battle 0 and 7, along with their other information. Therefore, the answer is E. One very common mistake people make is that they ignored the word "no", and they chose the battles that lost the ship. Be careful and pay close attention to every word!

Notice that there was originally the death table. We removed it from the display to make it easier for you.

The phrase 'Bulgarian commander' might send you looking for a table that tells you each commander's nationality. But actually, Bulgarian_commander is a column in the battles table. Presumably this table lists battles that Bulgaria fought. Each battle had two sides, and this column is naming the commander for the Bulgarian side. You don't have to fully understand how the tables are set up, but you should figure out enough to answer the question.

Just to repeat, to make it easier for you to process this information, whenever your cursor moves to an ID or a piece of text, its counterpart in other tables will light up; whenever you click on a text, the counterpart in the answer will also be highlighted.

You can also choose to merge the tables. After you merge the table, there will still be two tables. Each of the rows in the battle table will still contain information about a battle, and each of the rows in the ship table will still contain information about a ship. However, the battle information where the ship is lost is merged into the ship table. Notice that battle 0 will not appear in the ship table, because no ship is lost in the battle, so be careful when you try to interpret the merged table. Click unmerge to recover to the original view.

Finally, if you forgot what each table means, you can always view them here.

Page 17 Takeaways:

- Pay close attention to how the question is being asked. They might lead to different options. Many mistakes people make are because they did not read the questions carefully.
- Sometimes we choose not to show you certain tables and columns if we know for sure they are not needed.
- Use the highlight functionality if that helps you to reason across tables.
- Use the merge functionality if you need to. Each table will contain information about the same object/entity, but the information about its related objects will be pooled in.

Page 18 The question is “List the name and date of the battle that has lost the ship named ’Lettice’ and the ship named ’HMS Atalanta’”. Since there is no ship named “HMS atlanta”, there is no battle that lost both of these ships. So you should choose A, “no result found".

Page 19 Takeaways: Choose no_result_found if no answer satisfies the question.

Page 20 To summarize, here are a couple of things you need to remember to answer the questions correctly:

- Pay close attention to how the question is asked; most mistakes are made because of not reading the question carefully.
- Accept the information in the table even if they are changing and might be different from the knowledge you have for the real world
- IDs are different from names
- Some questions might have a lot of options to choose from and you need to read through all of them.

Page 21 To make it easier for you to answer the questions:

- Use the highlight and merge operations when you need to
- Use a calculator if you need to
- Zoom out to fit the tables into the screen and prevent scrolling.
- Not all table or column is needed to answer the questions

Page 22 For freeform response:

- Reporting ambiguities or tell us why the question is confusing only if you need to
- Explaining why you did not choose another option when we ask you. Giving us more information can you help you win more bonus.

H More Analysis on Updated Annotations

Interpreting Database Schema Properly If each row contains information about an orchestra, the year it was founded, and its associated recording company, when user asks “Which recording company was founded the earliest?", our system should response “Not enough information to tell", rather than finding the recording company of the earliest-founded orchestra.

Accounting for all Allowed Values The annotated SQL should account for all legal cell values, either specified by the database schema or a hidden generation process of the database. For example, when we are asked about the maximum value in a column that allows `NULL` value, we prefer SQL that returns the actual maximum value rather than the `NULL` value. For another example, if the utterance is “How many countries have a republic government form?", the where clause `GOVERNMENT = "Republic"` will ignore any countries with the government form “Federal Republic", and hence the correct annotation should be `GOVERNMENT LIKE "%Republic%"`.

Nevertheless, it is difficult to handle arbitrary cell values allowed by the schema. For example, if a user asks how many dogs are there, an ideal annotated SQL might need to account for cell values like Chihuahua, Husky, etc, which requires common sense reasoning and is hard to implement with a logical form. Therefore, we either need to make stronger assumptions about what cell values are allowed in a database, or introduce additional modules to handle common sense. We were particularly lenient when evaluating the SPIDER annotated SQLs and only test them on cell values that appear in their released database, even though their database schema allows for a much larger set of possible cell values.

I Computation

The simulation evaluation on the evaluation split in §5 takes around 240 CPU hours. Finding an informative small database can take up to several minutes; therefore, to support real-time interaction, we pre-compute the databases for all possible choices a participant might choose. Pre-computing the choices for all 240 utterances takes around 100 CPU hours.