



Assessing the utility of text-to-SQL approaches for satisfying software developer information needs

Mihaela Tomova¹ · Martin Hofmann¹ · Constantin Hütterer¹ · Patrick Mäder^{1,2}

Accepted: 25 July 2023 / Published online: 5 December 2023
© The Author(s) 2023

Abstract

Software analytics integrated with complex databases can deliver project intelligence into the hands of software engineering (SE) experts for satisfying their information needs. A new and promising machine learning technique known as text-to-SQL automatically extracts information for users of complex databases without the need to fully understand the database structure nor the accompanying query language. Users pose their request as so-called natural language utterance, i.e., question. Our goal was evaluating the performance and applicability of text-to-SQL approaches on data derived from tools typically used in the workflow of software engineers for satisfying their information needs. We carefully selected and discussed five seminal as well as state-of-the-art text-to-SQL approaches and conducted a comparative assessment using the large-scale, cross-domain Spider dataset and the SE domain-specific SEOSS-Queries dataset. Furthermore, we study via a survey how SE professionals perform in satisfying their information needs and how they perceive text-to-SQL approaches. For the best performing approach, we observe a high accuracy of 94% in query prediction when training specifically on SE data. This accuracy is almost independent of the query's complexity. At the same time, we observe that SE professionals have substantial deficits in satisfying their information needs directly via SQL queries. Furthermore, SE professionals are open for utilizing text-to-SQL approaches in their daily work, considering them less time-consuming and helpful. We conclude that state-of-the-art text-to-SQL approaches are applicable in SE practice for day-to-day information needs.

Keywords Software analytics · Database querying · Natural language processing · Text-to-SQL · Machine learning · Complex queries

Communicated by: Fabio Palomba

✉ Mihaela Tomova
mihaela-todorova.tomova@tu-ilmenau.de

¹ Technische Universität Ilmenau, Ilmenau 98693, Germany

² Faculty of Biological Sciences, Friedrich Schiller University, Jena 07745, Germany

1 Introduction

In their daily work, software developers utilize various tools to support project management processes, e.g., Assembla (2023); Maven (2023); Atlassian JIRA (2023), implementation processes, e.g., IBM Rational DOORS (Requirements management products 2023; Github 2023; Selenium site 2023), and development support processes, e.g., Git (2023). These tools are supposed to help them in regards to organization, management, and collaboration, even more in cases where time and geographical distance are an issue. Furthermore, data from such tools can help software developers to satisfy information needs they may have during development and maintenance. Research shows that software developers have various information needs, e.g., regarding code, importance of bug reports, reproducibility of failures, and code changes awareness (Lohar et al. 2015; Ko et al. 2007; Rath et al. 2018; Janke and Mäder 2022). Satisfying those needs can help them during decision-making, process improvement, and a myriad of other stakeholder information needs and software engineering tasks. Given, however, that the data is scattered across various tools, this may result in valuable time being wasted, searching for and tracing back the information that is needed. As a result, due to time constraints and limited knowledge critical development decisions are made based on gut feelings (Godfrey et al. 2009). Thus, a faster and easier way for software developers to find information they need is desired without the necessity for them to manually search for the information. To this end, would it not be nice for software developers to simply formulate a question and to get an answer? Given that most software tools, e.g., issue tracking systems (ITS), version control systems (VCS), or requirements management tools, can be viewed as ever growing knowledge bases that can be queried, makes developing approaches that help in retrieving the desired information a valuable field of study. In fact, the idea of retrieving data from a knowledge base by simply asking a question is not new and has been a research focus for many years. Early examples of attempts to access data from databases by only using natural language (NL) utterance date back to the 1960s–1970s, with the introduction of systems like BASEBALL (Green et al. 1961) and LUNAR (Woods 1977). While initially a rule-based approach was taken (Woods 1977; Waltz 1978; Lee et al. 2021), more sophisticated methods are available today. In recent years, machine learning has demonstrated impressive results in natural language processing (NLP) for tasks, such as machine translation, chat bots, sentiment analysis, and text summarisation. In this manner, a task, known as text-to-SQL aims to accelerate and simplify the process of querying data from databases. When applied in the context of software engineering, developers may only need to formulate an utterance in regard to the information need they have and then the text-to-SQL approach may automatically generate the SQL for the given utterance, eliminating the need for developers to formulate queries themselves. In this paper, we aim to examine how applicable and useful text-to-SQL approaches are within this context. We thus formulate the following research questions:

- RQ1 How do generic state-of-the-art text-to-SQL approaches perform when satisfying SE information needs?** We answer this RQ by performing a quantitative evaluation of text-to-SQL approaches. More specifically, we train our selected text-to-SQL approaches only on data from a generic, cross-domain dataset and evaluate them on SE data. We observe that training only on a generic, cross-domain dataset results in rather low accuracy.
- RQ2 How much can the performance of text-to-SQL approaches be improved by training with SE-specific data, i.e., utterances and queries?** We answer this RQ by performing a quantitative evaluation of text-to-SQL approaches. More specifically,

we train our selected text-to-SQL approaches on data from a generic, cross-domain dataset as well as on data from a SE dataset and evaluate them on SE data. We observe that text-to-SQL models specifically trained on SE data yield a substantially higher accuracy.

RQ3 How do SE professionals perform when satisfying their information needs via utterances and structured queries? We answer this RQ by performing a survey on the usefulness of text-to-SQL approaches in the SE domain. Our assessment suggests that participants had substantial difficulties in developing queries and that latest text-to-SQL approaches could be an actual benefit for SE professionals.

RQ4 How do SE professionals perceive text-to-SQL approaches for supporting their daily work? We answer this RQ via the same survey performed for RQ3. We observe that participants are open to the use of text-to-SQL approaches in the SE domain. Even though there is still room for improvement, especially in creating more representative training sets that allow for a better support of new and unseen queries.

In particular, our study makes the following contributions. (1) We are the first to examine text-to-SQL approaches, as a mean to satisfy information needs of SE professionals. (2) We carefully select and discuss five seminal and up-to-date text-to-SQL approaches for a systematic evaluation with generic and SE-specific training and evaluation sets. (3) We study the performance of approaches in relation to utterance specificity as well as the complexity of predicted queries. (4) Via a user survey with SE professionals, we study how they satisfy their daily information needs, how well they can express them as utterance and as structured query, and what their opinion on the utility of text-to-SQL approaches within software development is.

Our paper is structured as follows. We discuss related work in Sect. 2. In Sect. 3 we introduce the text-to-SQL task and discuss five text-to-SQL approaches to be analyzed across our experiments. In Sect. 4 we introduce and discuss the datasets that we used to perform our experiments. Section 5 explains the experimental setup and the conducted user survey. In Sect. 6 we report the results of all studies and discuss their implications. In Sect. 7 we discuss threats to validity. Finally, in Sect. 8 we conclude our work.

2 Related Work

Several studies have been conducted to understand project stakeholders' information needs during the development and maintenance of a software system.

Ko et al. (2007) observed 17 developers while performing their daily work activities and characterized the role these information needs play in developers' decision making by cataloging the source and outcome when each information type was sought. Begel and Zimmermann (2014) conducted two surveys to identify general questions that SE professionals would like to have answered by data science (e.g., "Is it really twice as hard to debug as it is to write the code in the first place?"). In result they retained 145 questions. These works imply the necessity to develop a way to automate the acquisition of knowledge satisfying developers' information needs.

In their daily work, software developers utilize a variety of software tools with the aim to organize and manage their work and to ease collaboration between team members. Portillo-Rodríguez et al. (2010) performed a survey, in which a set of software tools used by SE experts in global software development was presented. Each tool was classified according to the ISO/IEC 12207 standard and matched to a process the respective tool supports, e.g., doc-

umentation management, construction, project management. Hassan (2006) focused on the value of mining software repositories and using the mined historical data to assist managers and developers during development, maintenance, and management activities.

Bertram et al. (2010) conducted a qualitative study on the use of issue tracking systems in small, collocated software development teams and found that issue trackers other than tracking bugs, play an important role for communication and coordination between stakeholders. Lin et al. (2017) argue that information about artifacts can be found in various other data sources, such as the development environments Jazz or the Github-Jira bridge. Shang et al. (2014) show that questions regarding log lines can be answered by development knowledge present in development repositories such as ITSs. While the work of Portillo-Rodríguez et al. (2010) shows that a variety of tools exist that can be used by SE experts, the works of Hassan (2006); Bertram et al. (2010); Lin et al. (2017); Shang et al. (2014) imply how useful information from such tools can be for developers while performing their work. Given, however, the amount of data generated on daily basis, searching for the desired information can become quite difficult. Thus, automated solutions are desired.

Lin et al. (2017) propose TiQi, a tool for querying software projects based on a transform of utterances into executable SQL. The approach uses the Stanford parser, a tokenizer and seven heuristic disambiguation rules for transforming supported unstructured into structured queries intended to retrieve traceability information from a software development project. Störrle (2011) proposed the Visual Model Query Language (VMQL), which allows to query development models, e.g., UML, in a graphical way. The above mentioned works, however, require an understanding about the structure of all the queried artifacts. Thus, a solution is desired that can allow SE experts who do not know the structure of all queried artifacts or who have minimal or no formal technical training in a specific query language to still issue queries and get a desired answer. Abdellatif et al. (2020) propose to use a bot to automate and ease the process of extracting useful information from software repositories. They test the performance of their solution on a list of 15 questions covering developers' typical information needs. The chatbot uses Google's Dialogflow engine, which utilizes a NLU (natural language understanding) model, that is first trained on examples containing intents and entities manually created by the authors given a posed question. That is each type of supported question needs to be tediously prepared in this solution.

Fritz and Murphy (2010) propose an information fragment model that automates the composition of different kinds of information and provides developers with different options how to display the composed information. Abdellatif et al. and Fritz and Murphy simplify the way developers seek a desired information need, however, we argue that their proposed solutions are constrained to a specific set of questions or a project.

3 Text-to-SQL Approaches

Constructing structured queries can be very complex and error-prone (Lu et al. 1993). Recent research has made substantial contributions in developing semantic parsing solutions based on neural networks that aim to map a natural language utterance to a SQL query. These models, however, are not specifically trained to construct SQL queries for a specific field such as SE. Instead, they provide a way to simplify how databases are queried.

The task of generating a SQL query given an utterance, often in the form of a question, via a neural network is referred in the literature as text-to-SQL (Zhong et al. 2017). Researchers aim to train a model predicting correct SQL queries while taking into consideration only

a given utterance and the database’s schema including table, column, attribute and relation information (cp. Figure 1). Figure 1 exemplarily shows a user posing an utterance, in the form of a question to satisfy an information need arising from her or his current development task. Ideally, concepts named in the utterance can be mapped to tables and column headers of a DB’s schema. A text-to-SQL approach, receiving this utterance and the respective DB schema as input learns to link both information and generates a SQL query to retrieve the desired answer for the user. Some approaches also consider the actual database content for the prediction. However, this may pose scalability (depending on the size of the database) or privacy (depending on the context and application) issues (Yu et al. 2018; Zhang et al. 2020).

We used the well-known Spider Leaderboard (2023) to select text-to-SQL approaches for our research. First, we systematically screened the text-to-SQL approaches from the leaderboard and then further filtered them based on significant characteristics such as architecture, available code, and popularity to restrict focus to the most prominent candidates from a line of research and based on their applicability for our study. By examining the architecture, we aimed to consider the different methodologies applied throughout the years for generating SQL from a given NL utterance (cp. row key techniques in Table 2). Furthermore, we aimed to examine whether the architecture played a role in the evaluation of benchmark datasets. Given the amount of approaches in the Spider leaderboard, we further constrained the eventual set of text-to-SQL approaches based on the impact they had in addressing the text-to-SQL task as well as their position in the leaderboard, i.e., approaches with higher accuracy were preferred. Finally, we checked for additional software and hardware requirements to allow for a comparative evaluation of the approach on SE-specific data. Large Language Models (LLMs) do not meet those requirements and were therefore excluded from our comparative study. However, we performed a preliminary evaluation of OpenAI ChatGPT (2023); GitHub Copilot (2023) without fine-tuning on either Spider or the SEOSS-Queries datasets. Eventually, we selected the following five approaches for comparative evaluation: SQLNet (Xu et al. 2017), RatSQL (Wang et al. 2020), LGESQL (Cao et al. 2021), SmBoP+GraPPa (Rubin and Berant 2021), and T5 + PICARD (Scholak et al. 2021).

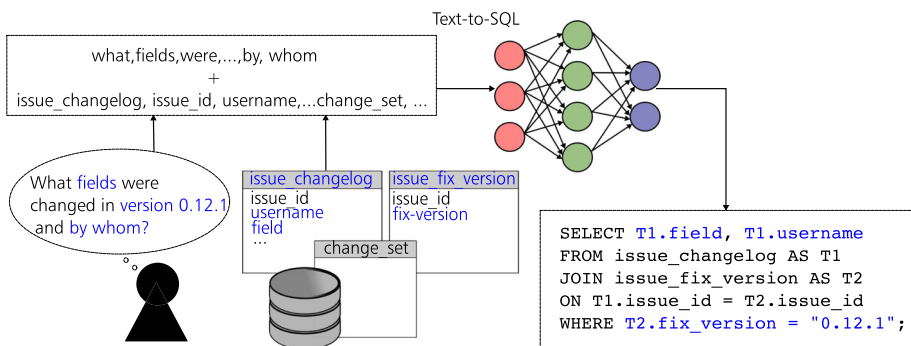


Fig. 1 An example from the SEOSS-Queries dataset (Tomova et al. 2022) to illustrate the text-to-SQL task, in which the inputs to a text-to-SQL model are a DB schema and an utterance and the output is supposed to be a correctly predicted SQL query. To construct the SQL query, the model needs to identify possible tables, table columns, and values in the utterance and based on these concepts, needs to join the columns “issue_changelog” and “issue_fix_version” on the common column “issue_id”

3.1 Text-to-SQL Method 1: SQLNet

SQLNet by Xu et al. (2017) is a seminal and frequently employed baseline method for text-to-SQL approaches. The method employs a sketch, i.e., a template or skeleton of a query, and considers the generation of the eventual SQL query as a slot-filling task. Therefore, it is only necessary to predict the content used within a SQL sketch rather than predicting the query's grammar. Figure 2 shows the sketch used in SQLNet, consisting of slots that can be aggregation operators, columns, values, or one of the following operator symbols: $>$, $<$, $=$.

Xu et al. proposed separate models for generating slot content within a `SELECT` clause and slot content within a `WHERE` clause of a query, making use of two main techniques: sequence-to-set and column attention. The sequence-to-set approach predicts which column names appear in a subset of interest by computing probabilities given a column name and an utterance. Column attention is utilized in order to predict a specific column based on the embedding of an utterance. Before the prediction of the slot content, utterances and column names are first represented as sequence of tokens. Each token is enumerated and then encoded via a Glove (Pennington et al. 2014) word embedding. Glove is an unsupervised learning algorithm that is used to obtain vector representations of words. Afterwards, using the sequence-to-set technique the authors first predict a set of columns for the `SELECT` and `WHERE` clause. A `SELECT` clause may consist of column names and aggregate operators, while the `WHERE` clause may consist of a set of columns, operators, and values. After a column is selected for the `SELECT` clause, a probability that this column is preceded by an aggregate operator is computed. Based on the predicted columns in the `WHERE` clause, predictions about the operator(s) and value(s) in it follow. Xu et al. consider the prediction of the number of columns in the `WHERE` clause as a $(N + 1)$ -way classification problem, while the prediction of the OP slot is considered a 3-way classification since an operator can be one of the following symbols: $>$, $<$, $=$. Furthermore, the authors provide a way to derive possible cell values from an utterance by employing a sequence-to-sequence structure consisting of a biLSTM (Schuster and Paliwal 1997) encoder and a pointer network (Vinyals et al. 2015) decoder that uses a column attention mechanism. A biLSTM encoder contains two LSTM neural networks, processing the input in opposite directions. Pointer networks can compute the probability that a token from an input sequence, i.e., utterance, is part of the output sequence, i.e., value in an SQL query. Initially, SQLNet was trained on the WikiSQL dataset. For our evaluation, we used the adapted version (SQLNet, Spider version 2023) for the Spider dataset and trained it on the Spider dataset (Yu et al. 2018).

3.2 Text-to-SQL Method 2: RatsQL

RATSQL (Wang et al. 2020) proposed by Wang et al. has been a top performing method for a long time and has inspired the architecture of many more specific text-to-SQL models found in the Spider leaderboard today. The approach represents the utterance and the database schema in a joint question-contextualized schema graph. The initial representations of the column and table nodes are derived via a pre-trained Glove word embedding. In case of multi-word labels, the embeddings of the words are further processed via a biLSTM recurrent neural

Fig. 2 SQLNet sketch query syntax adapted from Xu et al. (2017)

```
SELECT $AGG $COLUMN
WHERE $COLUMN $OP $VALUE
(AND $COLUMN $OP $VALUE)*
```


network (Hochreiter and Schmidhuber 1997). Also the initial representations of the words in the utterance are processed by a separate biLSTM. Alternatively, RATSQ uses the last hidden states of the prominent BERT model rather than Glove to obtain the initial representations. The independent initial representations are represented as a set and inputted into an encoder-decoder framework. In the encoder, relation-aware self-attention is then used to produce the joint representation of columns, tables, and question of the question-contextualized schema graph. Relation-awareness is realized by a set of self-attention layers, called RAT layers between all elements of the input as well as relation embeddings between question and schema (Shaw et al. 2018). Relations between question and schema constitute edges in the question-contextualized schema graph and are derived via schema linking. Wang et al. consider two types of schema linking: named-based and value-based linking. They consider four relation types for name-based linking: QUESTION-COLUMN-M, QUESTION-TABLE-M, COLUMN-QUESTION-M, and TABLE-QUESTION-M. Thereby, M specifies whether there is an exact match, partial match or no match relation between n-grams of the utterance and names of columns or tables in the database schema. The output of the encoder is a joint representation of the column, table, and question. Value matching can be challenging as it requires access to the database content to correctly link values from an utterance to a column of the database schema not mentioned in the utterance. The authors propose value-based linking without exposing the model to the whole data of a database by adding a Column-Value relation between any word and column name if the question word occurs as a value (exact or partial) of a column. The RAT-SQL decoder generates the SQL program as an abstract syntax tree (Yin and Neubig 2017) and then an LSTM outputs one of the following decoder actions: ApplyRule (expands the last generated node into a grammar rule), SelectColumn (selects a column from the DB schema) or SelectTable (selects a table from the DB schema). Following the original publication, we evaluate RatsQL with a pre-trained Glove embedding.

3.3 Text-to-SQL Method 3: LGESQL

LGESQL (Cao et al. 2021) proposed by Cao et al. considers the topological structure of edges in a graph and employs a line graph to attend to its nodes. The authors first construct an edge-centric graph (line graph) from a node-centric graph. With the help of these two graphs, a structural query topology consisting of nodes, representing questions, tables and columns, and edges, representing the structure of a query, is captured by gathering information in the neighborhood of the nodes. LGESQL employs an encoder-decoder (Sutskever et al. 2014; Bahdanau et al. 2015) framework and an architecture that consists of three components: graph input module, line graph enhanced hidden module, and graph output module. The graph input module computes the initial embedding for nodes and edges. Two alternatives are proposed for obtaining node representations: (1) using Glove embeddings (Pennington et al. 2014) and (2) via a pre-trained language model, e.g., BERT (Devlin 2019) or ELECTRA (Clark et al. 2020). When using Glove, each word in the utterance or schema item (table or column) is first embedded via Glove. Then, these embedding vectors become input to a type-aware biLSTM to attain contextual information. The forward and backward hidden states of the biLSTM for each word in the utterance are concatenated and are then used to construct the input graph. For each table and column, the last forward and backward hidden state of the biLSTM are concatenated and also used as graph input. All node representations are eventually stacked together to create a node embedding matrix. When using a pre-trained language model rather than Glove, question words and schema items are represented as a sequence following the pattern: $[CLS]q_1q_2\dots q_lQ[SEP]t_{i_0}t_1c_{j_0}^1c_1^1\dots[SEP]$. Where t_{i_0} and

c_{j0} represent type information of a table or a column respectively that are placed before each schema item. This sequence becomes input to the pre-trained language model. Following the pre-trained language model, a sub-word attentive pooling layer is appended to obtain word-level representations. Each word vector per utterance and the schema item are then fed into a biLSTM to predict graph inputs for all nodes. The Line Graph enhanced hidden module component consists of L dual relational graph attention network layers. The aim of this component is to capture the structure of the original node-centric graph and the line graph. Similarly to RATSQL, self-attention is used and represented by computing the attention weights via a multi-head scaled dot-product. In the edge-centric graph, the relation graph attention network layers consider only local (1-hop) relations based on which the updated node representation is computed. The graph output module component performs two tasks: text-to-SQL and graph pruning. For the text-to-SQL task, the authors generate an abstract syntax tree of the target query via a depth-first-search. Then, one of the following decoder actions depending on the item is performed: ApplyRule, SelectColumn, or SelectTable. The idea of graph pruning is that by identifying the intent and a constraint form of an utterance it is possible to extract irrelevant content from the target query. For our experiments, we decided to evaluate LGESQL with Glove word embeddings since the previous approaches also rely on this embedding allowing for a more objective comparison of the actual text-to-SQL models.

3.4 Text-to-SQL Method 4: SmBoP+GraPPa

SmBoP+GraPPa by Rubin and Berant (2021) proposes, in contrast to the above described text-to-SQL approaches, a novel decoding method. The approach reuses the encoder proposed by Wang et al. (2020) (RatSQL) to encode utterances and schema information. The only difference being that the authors exchange the pre-trained language model BERT with the pre-trained encoder GraPPa (Yu et al. 2021). GraPPa is based on RoBERTa (Liu et al. 2019) and is further fine-tuned on synthetically generated utterance-query pairs by the authors. In contrast to RATSQL and LGESQL, in SmBoP no abstract syntax tree of the target query is generated but rather a semi-autoregressive bottom-up parser.

After the encoding of the utterance and schema, the authors initialize the beam with the K highest scoring trees of height 0. Each tree can include either schema constants, e.g., a table or a column name, or database values, e.g., values of a column. K is computed by scoring the schema constants and database values and then choosing the top half of each. After the initialization of the beam, the algorithm proceeds to score trees on the frontier. A beam is defined by a symbolic representation of the query tree and its corresponding vector representation. Iterative at each decoding step, attention (Vaswani et al. 2017) is used to contextualize the tree representations with information from the question representation. The question representation is used in the tree scoring-frontier allowing to construct sub-trees from beams of previous depth. Trees can be generated by applying either an unary operation, e.g., distinct, or binary operation, e.g., selection, on beam trees. For each operation, a scoring function is defined. At the end of each iteration, the top- K trees are chosen and a new tree representation is computed. For the tree representation, Rubin and Berant follow the same intuition as Guo et al. (2019) and propose relational algebra (Codd 1970) augmented with SQL operators as a formal query language to address the mismatch between utterance and SQL. To make sure that constructed trees are balanced, i.e., height of trees corresponds to the current iteration step, the authors introduce an unary KEEP operation. By doing so, the formal query will not be modified. Eventually, the authors use beam search to find the top-scoring tree that is going to be used for the final decoding.

3.5 Text-to-SQL Method 5: T5 + PICARD

Scholak et al. (2021) propose PICARD, a method for constraining auto-regressive decoders of language models through incremental parsing. At each generation step, PICARD predicts top-k highest probability tokens and excludes tokens that failed PICARD's checks. Four different checks are proposed: off (no checking), lexing, parsing without guards, and parsing with guards. In the lexing mode, partial, detokenized model output is converted to a white-space delimiter sequence of individual SQL keywords, and identifiers like aliases, tables, and columns. In this mode, PICARD is able to detect spelling errors in keywords or reject table and column names that are invalid for the given SQL schema. In the parsing without guards, the detokenized model output is parsed to a data structure that represents an abstract syntax tree. In this mode, PICARD can reject invalid query structures and detect issues with compositions of SQL expressions. Furthermore, PICARD prohibits duplicate binding of a table alias in the same select scope but permits shadowing of aliases defined in a surrounding scope. In the parsing with guards mode, PICARD performs additional checks and guards, when assembling the SQL abstract syntax tree to leave out invalid hypotheses from the beam as early as possible. Guards can match a table name used when referring to column names and bring it into a scope by adding it to the FROM clause. T5 (Text-to-Text Transfer Transformer) (Raffel et al. 2020) by Raffel et al. is used as a baseline model, in combination with PICARD. The T5 framework is based on a Transformer architecture proposed in Vaswani et al. (2017) by Vaswani et al. and is trained on the publicly available web archive Colossal Clean Crawled Corpus (C4) dataset. In general, the T5 framework follows the implementation described in Vaswani et al. (2017) with small changes such as removing the Layer Norm bias, placing the layer normalization outside the residual path, as well as implementing a relative positional embeddings (Shaw et al. 2021) instead of learned positional embeddings. Furthermore, for the text encoding SentencePiece (Kudo and Richardson 2018) is used. The main innovation of T5 is the training task. The T5 training procedure is text-to-text where all inputs and outputs are texts. In addition to the text-to-SQL task, the T5 model is successfully employed in the following downstream tasks: machine translation, question answering, abstractive summarization, and text classification (Raffel et al. 2020). Known usages of the T5 model in the software engineering domain are: code completion (Ciniselli et al. 2022), log statement generation (Mastroauro et al. 2022), code generation, code summarisation, code repair, error diagnosis (Kajiura et al. 2022).

3.6 Comparison of the Text-to-SQL Approaches

Table 1 compares the chosen quantitatively in terms of exact match accuracy, a common metric to compare text-to-SQL approaches (cp. Subsection 5.1 "Quantitative Evaluation of Text-to-SQL Approaches"), of all five chosen methods as reported by the Spider leaderboard. The Spider leaderboard provides this accuracy metric for a dev set (aka evaluation set), which is known to the researchers when developing their methods, and a test set, which remains hidden to all contributors of the leaderboard and aims to ensure a fair and unbiased comparison of methods. Higher exact match accuracy shows a better performing approach. From top to bottom we observe an increasing accuracy across the chosen approaches.

In Table 2 we compare the chosen approaches more qualitatively based on four characteristics that we consider important and discriminating: representation of DB schema and utterance, input encoding, SQL generation strategy, and overall architecture. The first two characteristics are important to understand how NL is represented in a form that can be

Table 1 Evaluated text-to-SQL models and their general performance in terms of exact match (EM) accuracy as reported by the Spider leaderboard

Model	submitted to Spider leaderboard	EM acc. [%]	
		(dev)	(test)
SQLNet	Sep, 2018	10.9	12.4
RatSQL (Glove)	Dec, 2019	62.7	57.2
LGESQL (Glove)	Feb, 2021	67.6	62.8
SmBoP+GraPPa	Mar, 2021	74.7	69.5
T5 + PICARD	May, 2022	77.2	72.4

interpreted by the neural network model. The third characteristic describes how SQL queries are generated. Eventually, the fourth characteristic gives an overview of important technical characteristics of the given approach. We observe that even though the utterance as well as the DB schema become an input to the network in all approaches, the way in which they are represented is different. For example, SQLNet and T5 + PICARD prepare them as a sequence of tokens, while the other three approaches represent them as question-schema graph. The way in which the input is then encoded has also been shown as very relevant

Table 2 Methodological comparison of evaluated text-to-SQL approaches

Criterion	SQLNet	RatSQL	LGESQL	SmBoP + GraPPa	T5 + PICARD
Utterance and schema representation	as sequence of tokens	as question-contextualized schema graph	as question-schema graph	as question-contextualized schema graph	as sequence of tokens (sentence-piece tokenization)
Input encoding	Glove	Glove, BERT	Glove, BERT, ELECTRA	GraPPa	token IDs; relative positional encoding
SQL generation strategy	sketch aligning to SQL grammar	SQL as AST + context-free grammar	SQL as AST + graph pruning	based on semi-autoregressive bottom-up parser	autoregressive decoder with multi-head attention and constrained output during beam search
Key techniques	slot prediction, sequence-to-set prediction, column attention	encoder-decoder, relational-aware self-attention	encoder-decoder, L dual relational graph attention network layers	RatSQL + GraPPa encoder, semi-autoregressive decoder, cell values via prob. computation	self-attention, transfer learning, constrained decoder, cell values similar to bridging Nan et al. (2022)

and we observe a varying encoding of textual inputs into vector form. The approaches either rely on word embeddings, e.g., Glove, or pre-trained language models, e.g., BERT, ELECTRA. Word embeddings and pre-trained language models can provide additional information which is supposed to support the linking of concepts between the utterance and the respective DB schema. Furthermore, we observe that all approaches but SQLNet propose an encoder-decoder framework. Since SQLNet describes a sketch-based approach, it generates the SQL query by predicting slots. RatSQL and LGESQL generate an abstract syntax tree (AST), while SmBoP and T5 + PICARD propose a semi-autoregressive bottom-up parser and autoregressive decoder with multi-head attention respectively. These characteristics are expected to have an influence on the performance of the discussed text-to-SQL approaches also when applied to SE tasks.

Figure 3 depicts for the given utterance: "Return the issue ids of issues of type Bug", how each approach handles cell value predictions during the SQL query generation. As we can see, SQLNet, RatSQL, and LGESQL do not integrate cell value prediction during the SQL generation process, while SmBoP + GraPPa and T5 + PICARD do. This difference is essential when executing a predicted query since for approaches that do not predict cell values, a way to replace cell value placeholders needs to be developed and applied. For the evaluation of the five approach, the exact match is computed without considering cell values.

4 Datasets

In this section, we briefly discuss the two datasets that we used to perform our experiments. Table 3 gives an overview of their extend in terms of utterances, queries, databases and

NL utterance: Return the issue ids of issues of type Bug	
Gold SQL: <code>SELECT issue_id FROM issue WHERE type = 'Bug'</code>	
SQLNet Prediction:	<code>select t2.git_repository_id from issue_attachment as T1 join git_repository as T2 where t1.mime_type = "VALUE"</code>
RATSQL Prediction:	<code>SELECT issue.issue_id FROM issue WHERE issue.type = 'terminal'</code>
LGESQL Prediction:	<code>SELECT issue.issue_id FROM issue WHERE issue.type = "value"</code>
SmBoP + GraPPa Prediction:	<code>SELECT issue.issue_id FROM issue WHERE issue.type = 'Bug'</code>
T5 + PICARD Prediction:	<code>SELECT issue.issue_id FROM issue WHERE issue.type = 'Bug'</code>

Fig. 3 Example showing how the five evaluated text-to-SQL approaches handle cell value prediction given an utterance. Text marked in orange indicates tokens considered when computing the exact match accuracy

Table 3 Overview of the Spider and our proposed SEOSS-Queries text-to-SQL datasets

Dataset	# utterances	# queries	# DBs	# domains	# tables per DB	# ORDER BY	# GROUP BY	# nested queries	# HAVING
Spider (Yu et al. 2018)									
– all subsets	10,181	5,693	200	138	5.1	1,335	1,491	844	388
– software engineering subset	49	24	1	1	6	12	8	12	0
SEOSS-Queries	1,162	166	1	1	13	16	49	15	8

covered topic domains, while an in-depth discussion below will refer to their complexity shown in the remaining part of the table.

4.1 The Spider Dataset

The Spider dataset is proposed by Yu et al. (2018) and is a large-scale, cross-domain dataset that containing 10,181 utterances and 5,693 unique SQL queries. The dataset is split into a train set and a dev set. Additionally, a non-publicly available test set exists. The SQL queries are split into four hardness levels, i.e., easy, medium, hard, and extra hard, referring to 138 different domains separated into 200 databases. The hardness level is determined based on the number of SQL components, selections, and conditions found in the queries. For example, SQL queries that contain nested sub-queries' concepts, such as `GROUP BY`, `JOIN`, `LIMIT`, and `ORDER BY`, are considered harder to predict (Yu et al. 2018). By splitting the SQL queries into different hardness levels, Yu et al. provide a way to better understand a model's performance regarding queries' complexity. In comparison to another large and popular text-to-SQL dataset, WikiSQL (Zhong et al. 2017), Spider contains more complex SQL queries covering SQL operators such as `JOIN` and `GROUP BY` (Yu et al. 2018). For this reasons, we chose Spider as training set in transfer learning experiments. We also chose Spider, since WikiSQL restricts query generation to solely one table. This means that predicted queries merely consist of the `SELECT` and the `WHERE` clause. In the text-to-SQL task we envision and study within the SE domain, solely querying one table when gathering and relating data from different software tools is unrealistic.

4.2 The SEOSS-Queries Dataset

In a preliminary study, we found that SE data are almost not present in the cross-domain Spider dataset (Yu et al. 2018). The small SE subset of Spider merely consists of artificially generated data reflecting software defect tracking information (cp. Table 3). Furthermore, the accompanying queries are few and not derived from actual project stakeholder needs. Therefore, we previously proposed the dedicated SEOSS-Queries dataset (Tomova et al. 2022). In a nutshell, to create the SEOSS-Queries dataset, we reviewed works summarizing information needs of software engineers (Ko et al. 2007; Fritz and Murphy 2010; Abdellatif et al. 2020; Begel and Zimmermann 2014), derived through interviews and by studying software developers' day-to-day information needs, as well as by examining typical data queried in software tools such as ITSs. Our literature review gave us a general overview of the different information needs software engineers have. As next, we further filtered our findings by asking the question whether a given information need can be satisfied with data gathered from a software tool. This was necessary since some information needs, e.g., "Is it really twice as hard to debug as it is to write the code in the first place?" Begel and Zimmermann (2014), "Which coding guidelines/patterns have the most effect on code quality (e.g. small functions, lower complexity metrics, removal of duplicate code)?" Begel and Zimmermann (2014), "Did I make any mistakes in my new code?" Ko et al. (2007), "Is the problem worth fixing?" Ko et al. (2007), cannot be easily answered and the answer may vary depending on different factors and situations. Furthermore, to create the SEOSS-Queries dataset, we referred to real-world, open-source project data, reflecting actual information needs of software developers, collected from the ITS, i.e. JIRA, and VCS, i.e. GIT, of the Apache PIG project (Apache Pig project 2023), and persisted into a SQLite database by Rath and Mäder (2019).

SEOSS-Queries is publicly available under the following link (SEOSS-Queries Repository 2023). In total, the dataset consists of 1,162 utterances translating into 166 queries. Thereby, each query relates to four precise utterances, referred to as specific, and three more general ones, referred to as non-specific. The four specific ones include relevant column and table names appearing in the corresponding SQL query while the non-specific ones are formulated less precisely. To define non-specificity in terms of our concrete case we first agreed upon the following rules: the utterance must exclude columns' and tables' names; column names, table names, DB values need to be expressed by using synonyms (e.g., "When was issue X resolved?" = "When was issue X solved?"; "Who is assigned to issue..." = "Who is responsible for issue...") or by paraphrasing them (e.g., "between 2014-10-01 and 2014-10-31" = "in the month of October"); the utterance must not be syntactically correct, i.e., utterances without a verb (e.g., "Any bugs?"). Non-specific utterances were generated with respect of the above rules, making sure that at least one rule was employed. Each author acted as an evaluator to the utterances generated by the other authors. Examples of specific and non-specific utterances are shown in Fig. 4. By manually creating multiple specific and non-specific utterances

SQL Hardness Level: Easy

specific NL: Give me a list with all **issue ids** which **created date** is between '2017-01-01' and '2017-03-31'

non-specific NL: Find all **issue ids** of issues that were **reported** between '2017-01-01' and '2017-03-31'

SQL: SELECT **issue_id** FROM issue
WHERE **created_date** BETWEEN '2017-01-01' AND '2017-03-31'

SQL Hardness Level: Medium

specific NL: Return **issues** that have **assignee** set to Daniel Ward and **status** set to 'Open'

non-specific NL: Find **open issues** where Daniel Ward is **responsible for** them

SQL: SELECT * FROM issue
WHERE **assignee** = "Daniel Ward" AND **status** = 'Open'

SQL Hardness Level: Hard

specific NL: Search for **issues** that belong to grunt **component** and parser **component**

non-specific NL: Return **issues** that are **both in the grunt and** parser component

SQL: SELECT * FROM **issue** AS T1
JOIN issue_component AS T2 ON T1.**issue_id** = T2.**issue_id**
JOIN issue_component AS T3 ON T1.**issue_id** = T3.**issue_id**
WHERE T2.**component** = 'grunt' AND T3.**component** = 'parser' AND T2.**issue_id** = T3.**issue_id**

SQL Hardness Level: Extra Hard

specific NL: Return **assignee** and **count of assigned issues** of **status** "Open" of the developer **with the most issue ids assigned to him/her**

non-specific NL: **Who is working on most of the issues** in the Apache Pig project?

Give me his/her **name** and **the count of issues** he/she is **working on**

SQL: SELECT **assignee**, **Count(issue_id)** FROM issue
WHERE **status** = "Open"
GROUP BY **assignee**
ORDER BY **Count(issue_id)**
DESC LIMIT 1

Fig. 4 Examples from the SEOSS-Queries dataset showing specific and non-specific utterances and their corresponding SQL query for each of the four differentiated SQL query hardness levels. With the same color we highlighted in the utterances and SQL queries words that express the same content, with a green border we highlight exact match to a column or a table from the SQL query, with a red border we highlight words that do not exactly match a column or a table from the SQL query

per SQL query instead of machine-generating ones, we aimed for a more realistic scenario representing the diversity of NL and were interested in how the models handle utterances, expressing the same intent written with different wording. The non-specific utterances can be used as a way to measure how well a model can interpret the utterances. Furthermore, given that some information in the utterances is missing, they can be used to evaluate whether a model can correctly deduce the missing information.

We used Yu et al. (2018) evaluation script to categorize each SQL query according Spider's four hardness levels (easy, medium, hard, and extra hard). During the categorization we found that 33 utterances-queries pairs of the SEOSS-Queries dataset could not be processed and categorized due to SQL syntax constructs not supported by Yu et al.'s script. For example, SQL containing functions such as `strtime` or specific SQL keywords such as `NOT`, `CASE` could not be processed, returning an exception. Further examples of incompatible queries are shown in Table 4. Aiming for comparability to Spider's assessment, we decided not to adapt the evaluation script, but to remove incompatible queries, resulting in 931 utterances and 133 SQL queries for performing our experiments. These 133 queries are distributed across the four complexity levels as follows: 56 easy, 54 medium, 11 hard, and 12 extra hard. Figure 4 shows per hardness level an example of an utterance and its corresponding SQL query.

To examine the generalization capabilities of the evaluated approaches, it is common practice to split the data into training splits ('train') and kept-away evaluation splits ('dev'). SEOSS-Queries provides different splits for evaluation. The utterance-based split selects three specific and three non-specific utterances from each SQL query with four of those being selected for training and two for evaluation. This split aims to shed light on how well approaches can predict under optimal conditions where they have been exposed to all queries during training and are only exposed to unknown utterances relating to these queries during evaluation. In contrast, the query-based split selects an SQL query with all its utterances belonging to either the training, 80% of all queries, or the test set, remaining 20% of the queries.

5 Evaluation of Text-to-SQL Approaches for SE Tasks

Below, we evaluate text-to-SQL approaches regarding their applicability and utility in satisfying SE information needs. Therefore, we perform quantitative experiments using the five methods introduced before and qualitative assessment via a survey with software engineers.

5.1 Quantitative Evaluation of Text-to-SQL Approaches

We conducted three experiments to evaluate the five selected text-to-SQL approaches regarding their utility for satisfying SE information needs (cp. Sec. 3). A reproduction package detailing how we trained models in these experiments and allowing for follow-up experiments is available here SEOSS-Queries Repository (2023). Experiment 1 aimed to evaluate how well the standard large-scale cross-domain Spider dataset is suited to train the examined text-to-SQL approaches when satisfying SE information needs, i.e., we used Spider for training and SEOSS-Queries' utterance-based split for evaluation. Experiment 2 and 3 aimed to evaluate how the examined text-to-SQL approaches perform when trained and tested on software engineering data, i.e., using the SEOSS-Queries dataset for training and evaluation. In Experiment 2, we used the utterance-based split that had all queries already in the training and tested novel utterances for them, while in Experiment 3 by using the query-based split

Table 4 Examples of utterances and SQL queries that are incompatible to the Spider grammar

Problematic SQL query	Incompatibility
<pre>SELECT CASE WHEN EXISTS (SELECT * FROM change_set_link WHERE issue_id = 'PIG-4092' AND commit_hash = '[...]') THEN 'True' ELSE 'False' END</pre>	CASE statement
<pre>SELECT issue_id FROM issue WHERE strftime('%Y-%m-%d', created_date) = DATE('now', '-1 day')</pre>	strftime() function
<pre>SELECT * FROM issue AS T1 WHERE T1.issue_id NOT IN (SELECT T2.issue_id FROM issue_attachment AS T2)</pre>	NOT in WHERE clause
<pre>SELECT * FROM issue WHERE type = 'Bug' and (status = 'Resolved' OR status = 'Closed') AND (resolution= 'Fixed' or resolution = 'Done')</pre>	brackets in WHERE clauses
<pre>SELECT T1.* FROM issue AS T1 JOIN issue_fix_version AS T2 ON T1.issue_id = T2.issue_id WHERE T2.fix_version = '0.12.1'</pre>	T1.* in SELECT
<pre>SELECT AVG(count_per_issue) FROM (SELECT Count(DISTINCT username)/AS count_per_issue FROM issue_comment GROUP BY issue_id)</pre>	keyword AS in SELECT
<pre>SELECT Count(*) FROM issue WHERE description IS NULL</pre>	NULL in WHERE clause

Table 5 Training and evaluation instances per experiment

utterances	Experiment 1		Experiment 2		Experiment 3	
	Train (Sp. + SQ.)	Test (SQ.)	Train (Sp. + SQ.)	Test (SQ.)	Train (Sp. + SQ.)	Test (SQ.)
specific	8659 + 0	133	8659 + 532	133	8659 + 742	108
non-specific	8659 + 0	133	8659 + 532	133	8659 + 742	81
all	8659 + 0	266	8659 + 532	266	8659 + 742	189

Sp. = Spider, SQ = SEOSS-Queries

exposed the model at evaluation to queries not seen in the training (cp. Sec. 4.2). All three experiments were evaluated for all utterances, specific and non-specific, together as well as for specific and non-specific utterances separately.

Experimental Setup In general, we followed the training and evaluation setup, i.e., epochs, hyperparameter, proposed by the original models, intending to mimic the results from the Spider leaderboard and to ensure that we performed the training correctly. In all of the five approaches, we adjusted the training, dev, and evaluation dataset files, i.e., for Experiment 2 and 3, as well as the table.json file to include data from the SEOSS-Queries dataset and DB schema. Furthermore, we extended the Spider databases with the Apache Pig database coming from the SEOSS-Queries dataset. The five text-to-SQL approaches were trained on machines with the following specifications: 2 TB RAM, 8 x NVIDIA A100, 2 x AMD EPYC 7742 64-core processor. During the training early stopping was used as a convergence stop criteria. Table 5 presents the number of training and evaluation instances per experiment.¹

Evaluation Metrics Traditionally, the accuracy of predicted SQL queries is evaluated via their returned result (execution accuracy) or comparing the queries word by word (exact match accuracy). However, considering cases in which the retrieved value may result in queries that return a correct result, i.e., zero or empty value, but with a SQL query describing different intent, lead to exact match accuracy becoming the standard performance measure. Comparing the gold (ground truth) and the predicted queries seems intuitive and easy to execute. However, creating the WHERE clause becomes vague because a sequence of Boolean expressions is not necessarily unique. The literature refers to this ambiguity as an "ordering issue". Comparing sub-components of SQL queries' WHERE-expressions solve this problem by matching sub-expressions first, comparing them in a second step. Zhong et al. published an example of the exact match accuracy calculation procedure (Evaluation script spider 2023).

Comparing the results of the queries in terms of exact match is a precise and informative measure, and therefore the evaluation of large text-to-SQL datasets is performed that way. Another reason is that not all text-to-SQL approaches generate executable SQL queries, which would be the precondition for a result-based evaluation. However, datasets consist of samples that are hard to recognize because small questions can lead to complicated queries with several nested queries and specific values to be found in the question and put in the right place for the query. The solution to this problem is to rank the difficulty into difficulty levels based on the type and number of SQL concepts with different levels: easy, medium, hard, and extra-hard (Yu et al. 2018). We, therefore, calculated one exact match accuracy for each difficulty level separately to better understand the performance of the text-to-SQL approaches studied.

¹ In Experiment 2 the training set consists of 532 examples since we excluded one specific utterance to balance out the number of specif and non-specific utterances during training.

5.2 Survey on Usefulness of Text-to-SQL Approaches in the SE Domain

Furthermore, we performed an extensive user survey evaluating how text-to-SQL approaches and their results are perceived in the SE domain and by what kind of SE experts text-to-SQL approaches may be used.

Before conducting this survey, we asked a colleague to act as a pilot and to perform the survey in an initial form. After completion, we asked him to assess the interpretability of the descriptions and the complexity of the tasks in it, under consideration of the time we planned giving our participants. Taking into account the insights from the pilot, we refined descriptions and reduced the questions for the final survey.

Next, we reached out to people known to the authors via email and social media. These potential participants were individuals working in the software engineering domain, encompassing both academia and industry. In the email, we provided a generic link to the online survey for their convenience and ease of access. Since our models were trained on data primarily dealing with the solution of programming issues, we explicitly stated in the emails that the participants need to have some programming experience and be familiar with issue tracking systems and/or version control systems. Furthermore, we asked these contacts to forward the email to other possible participants, resulting in a snowball effect across a larger population of potential participants.

The estimated time for taking the survey was not more than 60 min and the survey was accessible for a time period of three weeks. Our survey consisting of 20 questions split into five sections. Four of these questions (Q13, Q15, Q16, Q17) consisted of tasks for which we had formed two groups. Thereby, roughly half of the participants received the first treatment while the other received the second treatment of tasks.

Section 1, comprising eight questions, inquired about general, mostly demographic, information of the participant like age and experience with respect to the focus of the study.

Section 2 aimed to uncover how a participant satisfied her or his development information needs, specifically focused on ITS information. Additionally, we randomly chose per hardness level two SQL queries (eight in total) from the SEOSS-Queries dataset as well as one specific and one non-specific utterance accompanying those. Participants were asked to rate the understandability of the presented utterances, construction complexity of presented queries, and had to match utterances to respective queries. Within this section, we aimed to assess how representative SEOSS-Queries utterances were and what skills in terms of satisfying their information needs participants had.

In Sect. 3, each participant received two verbally described scenarios, e.g., “The data analyst Eve would like to find the version of a given software project in which the highest number of issues were fixed.”, and was asked to formulate an utterance and to construct the respective SQL query. These scenarios were derived from SEOSS-Queries’ evaluation sets and were supposed to describe different complexity levels of SQL queries. Thereby, half of the participants received scenarios supposed to yield a query of hardness level medium and one of hardness level hard, while the other half received a scenario supposed to yield one easy and one extra hard query. Furthermore, the scenarios described utterances for which our highest performing model correctly generated queries. Similar to the text-to-SQL approaches, participants were provided the database schema of the underlying Apache Pig project data. Eventually, the aim of this task was a comparison between text-to-SQL approaches and human developers in generating SQL queries regarding information needs.

In Sect. 4, we provided participants four utterance-SQL query pairs that were not successfully generated by some of the best performing text-to-SQL approaches in our study,

aiming to introduce participants with some of the limitations of the trained models, letting them decide whether the wrongly generated queries can be useful or not to them. We asked participants to identify problems in the given queries and to judge how difficult it would be for them to correct the query.

In the last Section, we aimed to gather participants’ general perception on text-to-SQL approaches and their applicability to the SE domain in their current state. A list of all asked questions as well as snapshots of their representation during the survey is provided in the [Appendix](#).

To summarize, Sect. 1 of our survey provides us with demographic information about our participants, Sect. 2 gives us overview of the participant’s experience in satisfying their information needs. With Sect. 3 we tend to assess the query creation capabilities of our participants, aiming to see whether they can create SQL queries as well as the highest performing text-to-SQL approach in our study. Section 4 presents to our participants limitations of text-to-SQL models in the form of incorrectly constructed queries generated by some of the best performing models in our study. Section 5 concludes our survey and provides us with information about the position of our participants regarding the use of text-to-SQL approaches for SE tasks.

6 Evaluation Results and Discussion

Fig. 5 provides a schema of the research strategy we applied in our work, as well as an overview of which steps in our schema are linked to which RQs.

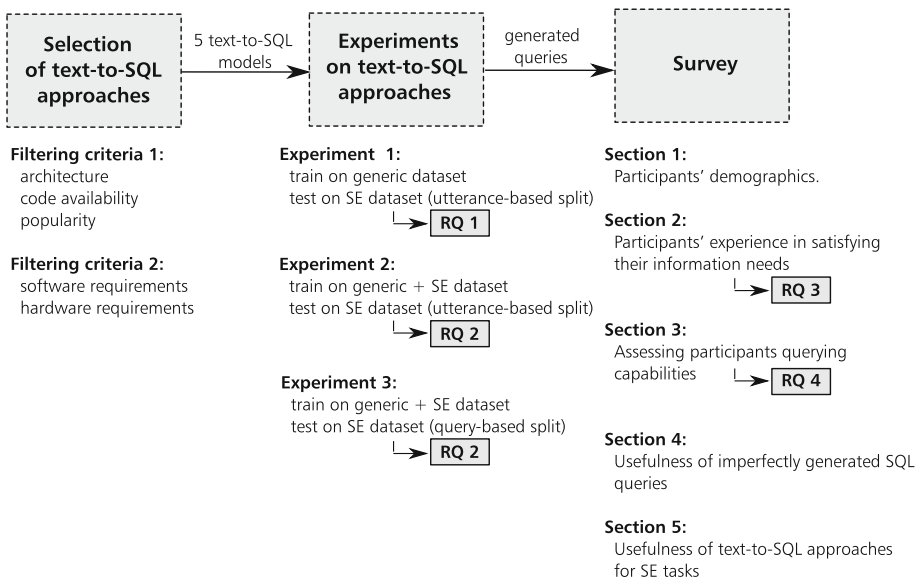


Fig. 5 Overview of the research strategy used in the form of three-step schema. Horizontally, we show the order in which each step was applied, as well as, depicted above arrows, significant results from the steps. Vertically, we present additional information about each of the three steps, linking RQs to specific points in the step

Table 6 reports the results of our experiments with the five previously introduced approaches and the two datasets. Thereby, the first column refers to the respective approach and the following three groups of five columns refer to the three different trainings with these approaches, the first on the cross-domain Spider dataset, the second on the utterance-based split of SEOSS-Queries, and the third on the query-based split of SEOSS-Queries. Thereby, the five columns report exact match accuracy on the respective test set separated into the previously discussed query complexity levels easy to extra hard (Xhard) and additionally across all test samples. Furthermore, the table reports results across three row groups from top to bottom that refer to just the specific utterances in the test set (top row group), just the non-specific utterances (middle row group), and all tested utterances together (lower row group).

6.1 Utilizing Text-to-SQL Out of the Box for SE Tasks (Experiment 1)

The best performing model in the first experiment is the one trained with T5 + PICARD, which was at the time of conducting our study also the top performing text-to-SQL approach with available code on the Spider leaderboard. SQLNet is the worst performing text-to-SQL approach. That is not surprising since the approach was initially designed for the less complex WikiSQL dataset and we had selected SQLNet mostly as a baseline. Comparing the performance of the worst and best performing text-to-SQL approaches, we observe an improvement of 44% exact match accuracy between the seminal SQLNet and the currently best performing T5 + PICARD. This approach predicts 61% of the easy queries correctly. However, this performance degrades to merely 9% correctly predicted extra hard queries. This growing share of unsuccessfully predicted queries suggests that there are utterance-query pairs in the SEOSS-Queries test set that are uncommon for the model trained on Spider. We observed that this happens due to specific vocabulary and concepts used in SE data, e.g., bug, improvement, feature, commit, change set, hash, or due to SQL concepts that are less present in Spider and therefore not properly trained, such as, nested queries, queries containing a HAVING clause. SQL queries marked as extra hard were the hardest to predict and only a very small share of them were correctly predicted by LGESQL, SmBoP+GraPPa, and T5 + PICARD (cp. Table 6 specific utterances). Non-specific utterances of hardness level extra hard (Xhard) could not be predicted by any approach suggesting that due to missing information like column and table names and due to performing the training without the SEOSS-Queries DB schema, it is much more difficult for the text-to-SQL approaches to link DB values to the corresponding table and columns.

Answer to RQ1 RQ1 aimed to answer how well generic state-of-the-art text-to-SQL approaches handle NL utterances from the SE domain. In general, we observe that models trained on a generic, cross-domain dataset such as Spider can handle SE domain-specific utterances and queries that are somewhat similar to other domains but fail to handle additional information specific to the SE domain and can therefore only correctly predict 46% of the tested utterance-query pairs (RQ1). We consider this rate too low for an actual integration of text-to-SQL approaches into a productive environment.

6.2 Text-to-SQL Specifically Trained for SE Needs (Experiment 2)

T5 + PICARD remains the top performing and SQLNet the worst performing text-to-SQL approach when training on the SE-domain-specific SEOSS-Queries dataset. We observe substantial improvements between 20% and 61% for the different approaches and specifically

Table 6 Performance of the five selected text-to-SQL approaches measured as exact match accuracy (em accuracy) of the predicted queries for evaluation utterances of the SEOSS-Queries dataset reported in total (all) and per complexity level of the resulting query, i.e., easy to extra hard (Xhard). Thereby the models in Experiment 1 were trained on the cross-domain Spider dataset, the models in Experiment 2 were trained on the utterance-based split of SEOSS-Queries, and the models in Experiment 3 were trained on the query-based split of SEOSS-Queries. Bold values depict the models with highest accuracy per experiment

Model	Experiment 1 Spider Training						Experiment 2 SEOSS-Queries Training						Experiment 3						
	em accuracy % ↑			em accuracy [%]↑			em accuracy [%]↑			em accuracy [%]↑			em accuracy [%]↑			em accuracy [%]↑			
	easy	med	hard	Xhard	all	count	easy	med	hard	Xhard	all	count	easy	med	hard	Xhard	all	count	
<i>specific utterances</i>																			
count	56	54	11	12	133	56	54	11	12	133	56	54	11	12	133	20	56	12	108
SQLNet	0.071	0.000	0.000	0.000	0.030	0.464	0.093	0.182	0.000	0.248	0.000	0.248	0.182	0.000	0.248	0.000	0.018	0.083	0.019
RatSQL	0.375	0.167	0.000	0.000	0.226	0.893	0.907	0.545	0.417	0.827	0.900	0.482	0.583	0.200	0.519	0.900	0.482	0.583	0.200
LGESQL	0.554	0.352	0.182	0.083	0.398	0.929	1.000	0.818	1.000	0.955	0.950	0.554	0.833	0.200	0.593	0.950	0.554	0.833	0.200
SmBoP+GraPPa	0.536	0.481	0.182	0.083	0.444	0.821	0.852	0.636	0.333	0.774	1.000	0.518	0.667	0.200	0.565	1.000	0.518	0.667	0.200
T5 + PICARD	0.696	0.556	0.273	0.182	0.561	0.964	0.981	0.909	1.000	0.970	0.950	0.661	0.667	0.250	0.639	0.950	0.661	0.667	0.250
<i>non-specific utterances</i>																			
count	56	54	11	12	133	56	54	11	12	133	56	54	11	12	133	15	42	9	81
SQLNet	0.018	0.000	0.000	0.000	0.008	0.268	0.148	0.182	0.000	0.188	0.067	0.000	0.222	0.000	0.037	0.067	0.000	0.222	0.037
RatSQL	0.143	0.167	0.091	0.000	0.135	0.839	0.704	0.636	0.250	0.714	0.533	0.190	0.667	0.067	0.284	0.533	0.190	0.667	0.067
LGESQL	0.286	0.204	0.182	0.000	0.218	0.875	0.852	1.000	0.833	0.872	1.000	0.405	0.556	0.000	0.457	1.000	0.405	0.556	0.000
SmBoP+GraPPa	0.375	0.222	0.091	0.000	0.256	0.750	0.611	0.455	0.000	0.602	0.667	0.190	0.556	0.000	0.284	0.667	0.190	0.556	0.000
T5 + PICARD	0.518	0.315	0.182	0.000	0.364	0.893	0.926	1.000	0.917	0.917	0.667	0.381	0.667	0.200	0.432	0.667	0.381	0.667	0.200

Table 6 continued

Model	Experiment 1 Spider Training				Experiment 2 SEOSS-Queries Training				Experiment 3						
	em accuracy % ↑				em accuracy [%]↑				em accuracy [%]↑						
	easy	med	hard	Xhard	all	easy	med	hard	Xhard	all	easy	med	hard	Xhard	all
<i>all utterances</i>															
count	112	108	22	24	266	112	108	22	24	266	35	98	21	35	189
SQLNet	0.045	0.000	0.000	0.000	0.019	0.366	0.120	0.182	0.000	0.218	0.029	0.010	0.143	0.000	0.026
RatSQL	0.259	0.167	0.045	0.000	0.180	0.866	0.806	0.591	0.333	0.771	0.743	0.357	0.619	0.143	0.418
LGESQL	0.420	0.278	0.182	0.042	0.308	0.902	0.926	0.909	0.917	0.914	0.971	0.490	0.714	0.114	0.534
SmBoP+GraPPa	0.455	0.352	0.136	0.042	0.350	0.786	0.731	0.545	0.167	0.688	0.857	0.378	0.619	0.114	0.444
T5 + PICARD	0.607	0.426	0.227	0.087	0.457	0.929	0.954	0.955	0.958	0.944	0.829	0.541	0.667	0.229	0.550

for T5 + PICARD an improvement of nearly 50% over the model trained on Spider (Experiment 1). Figure 6 exemplarily visualizes what this improvement means by contrasting five utterances for which the T5 + PICARD model trained on Spider (Experiment 1) predicted the wrong query, while the T5 + PICARD model trained on SEOSS-Queries (Experiment 2) predicted correctly. The reason potentially being that partially missing or abstract information in non-specific utterances has a significant effect when linking utterance concepts to DB schema concepts during prediction. For example, a user may search for issues of priority 'Blocker' or 'Critical' via the non-specific utterance "Return issues that are either blocking or critical", which may not be expressive enough since the concept "blocking" or "critical" does not exist in the DB's schema. The SEOSS-Queries dataset contains two kinds of utterances: specific and non-specific (cp. Sec. 4.2) enabling us to examine the influence of an utterance's specificity on the accuracy of the predicted query. We observe that specific utterances allow for a 5% to 17% more accurate query prediction than non-specific utterances (cp. Table 6, specific vs. non-specific utterances across all experiments). Taking a closer look at results split across the different query complexity levels to be predicted, easy to extra hard (Xhard), we observe that all approach but SQLNet are at least able to predict some queries per complexity level. Thereby, the share of successfully predicted queries grows with the maturity of the approach and the best performance in each level yields T5 + PICARD while LGESQL performs just slightly worse.

Spider Training (T5 + PICARD, Experiment 1)	SEOSS-Queries Training (T5 + PICARD, Experiment 2)
<p>Utterance: Any issues assigned to Daniel Ward</p> <p>Pred. SQL: ❌</p> <pre>select t1.issue_id from issue as t1 join issue_changelog as t2 on t1.issue_id = t2.issue_id [join is not necessary] where t1.assignee = "Daniel Ward"</pre>	<p>Pred. SQL: ✅</p> <pre>select * from issue where assignee = 'daniel Ward'</pre>
<p>Utterance: Select the commit hashes of change sets which committed date is between the created date and resolved date of issue id 'PIG-3592'</p> <p>Pred. SQL: ❌</p> <pre>select t1.commit_hash from code_change as t1 join issue_comment as t2 on t1.commit_hash = t2.created_date + t2.created_date_zoned [no SQL about resolved date] where t2.issue_id = 'PIG-3592'</pre>	<p>Pred. SQL: ✅</p> <pre>select commit_hash from change_set where committed_date >= (select created_date from issue where issue_id = 'PIG-3592') and committed_date <= (select resolved_date from issue where issue_id = 'PIG-3592')</pre>
<p>Utterance: Count issue links by occurrence</p> <p>Pred. SQL: ❌</p> <pre>select [missing column 'name'] count(*) from issue_link group by count(*) [wrong group by]</pre>	<p>Pred. SQL: ✅</p> <pre>select name, count(*) from issue_link group by name</pre>
<p>Utterance: Return the number of assignees that are assigned to the component impl</p> <p>Pred. SQL: ❌</p> <pre>select count(*) [missing info. about assignees] from issue_component where component = 'impl'</pre>	<p>Pred. SQL: ✅</p> <pre>select count(assignee) from issue as t1 join issue_component as t2 on t1.issue_id = t2.issue_id where t2.component = "impl"</pre>
<p>Utterance: Show me the issue types of source and target issues that are linked and which link name is set to Cloners</p> <p>Pred. SQL: ❌</p> <pre>select t1.type [missing type of target issue] from issue as t1 join issue_link as t2 on t1.issue_id = t2.source_issue_id [missing join for target issue] where t2.name = 'Cloners'</pre>	<p>Pred. SQL: ✅</p> <pre>select distinct t2.type, t3.type from issue_link as t1 join issue as t2 on t1.source_issue_id = t2.issue_id join issue as t3 on t1.target_issue_id = t3.issue_id where t1.name = "Cloners"</pre>

Fig. 6 Five examples of utterances that resulted in not correctly predicted queries when trained on the generic Spider dataset (Experiment 1) but were correctly predicted by the model trained on the SEOSS-Queries dataset (Experiment 2). The queries were predicted by the approach performing best in both experiments, i.e., T5 + PICARD

Answer to RQ2 RQ2 aimed at evaluating performance differences when training our models with and without SE-specific data. We conclude that text-to-SQL models specifically trained for the SE domain yield a substantially higher accuracy than models trained on generic datasets (RQ2). Across all test utterances, T5 + PICARD and LGESQL achieve the highest accuracy with 94% and 91% respectively. They demonstrate this accuracy almost independently of a queries' complexity levels making them ready for an integration into the daily work of SE experts.

6.3 Predicting Non-Trained SE Utterance-Query Pairs (Experiment 3)

In a third experiment, we trained SEOSS-Query's query-based split, i.e., we kept entire queries with all their respective utterances out of training and solely tested them. This has some similarity to the first experiment in which we trained with the cross-domain Spider dataset but tested on SEOSS-Queries. However, now a model was trained with SE-domain queries. Again, T5 + PICARD performs best across all test utterances yielding a 9% higher accuracy compared to Experiment 1 but a 40% decreased accuracy compared to Experiment 2. That is the domain-specific training allows the model to better generalize to new an unseen queries from the same domain, while still roughly every second query is not correctly predicted. Especially queries of complexity extra hard (Xhard) suffer from a tremendous accuracy reduction over the utterance-based split (96% vs. 23%).

Experiment 2 vs. Experiment 3 Overall, the difference in performance between Experiment 2 and 3 suggests that models' generalization capability would benefit from more training data. Additionally, we performed McNemar's test that showed no statistically significant difference in the disagreements between the top two highest performing models, i.e., LGESQL and T5 + PICARD. The contingency tables for both experiments are depicted in Table 7. The computed odds ratios for Experiments 2 and 3 were 5.54 and 10.2. In both cases, the odds ratio was greater than one, suggesting a positive association or dependency between the models. The α -values of 0.38 and 0.77 for Experiment 2 and Experiment 3, respectively, indicate no significant superiority of any models at a significance level of 0.05. Together with the odds ratio greater than one, we conclude that both methods are performing equally well with a high probability of coming to the same results.

6.4 Large-Language-Models as Alternative to Text-to-SQL Models.

Large language models (LLM) became very popular in assisting day-to-day business tasks from preparing emails to drafting text documents and have also been trained on a considerable corpus source code. OpenAI as one of the leaders in this field does not only offer its general

Table 7 The table shows the contingency tables of the top two highest performing approaches in Experiment 2 and 3 (LGESQL, depicted vertically, and T5 + PICARD, depicted horizontally), upon which a McNemar's test is performed. The entries 'True' indicate correct queries, while 'False' represents incorrect queries

LGESQL/T5+PICARD	Experiment 2 (p -value: 0.377)		Experiment 3 (p -value: 0.766)	
	True	False	True	False
True	228	13	80	21
False	19	6	24	64

purpose GPT models, e.g., for interactive use in the ChatGPT (OpenAI ChatGPT 2023) form, but also models like Codex specifically aimed at generating source code, e.g., available as GitHub Copilot (2023). We did not consider such models as first class citizens in our study, primarily to ensure consistency in the analysis by comparing similar approaches. Furthermore, we objected on these LLMs in an industrial environment. More specifically, privacy may be an issue, since query and database information would be shared with the provider of the services. We argue as well that the text-to-SQL approaches from our analysis provide a more cost-efficient solution. At the time of writing this manuscript, Copilot as well as ChatGPT require paid subscriptions for continued use.

Recognizing, however, the impact of LLMs, we decided to still evaluate ChatGPT and GitHub Copilot on the utterance-based and query-based test splits of the SEOSS-Queries dataset (cp. Table 8). We are not allowed to fine-tune LLMs such as ChatGPT and Github Copilot on our own machines and therefore excluded them from the comparative evaluation; however, we performed a zero-shot evaluation since those models are designed for a zero-shot setting (Kojima et al. 2022). Zero-shot means that the model was not trained for the task it is meant to solve; however, we found that ChatGPT is capable of reproducing samples from the Spider dataset if asked (cp. prompt, Appendix, Sect. 1) In comparison to our Experiments 2 and 3, we evaluated ChatGPT and Github Copilot on the utterance-based split and the query-based split and compared the results with T5-PICARD fine-tuned. In general, the findings demonstrate that without any training their results are useable to some extent; however, their performance is not on par with the purposely trained T5+PICARD model discussed before. Looking at the generated SQL queries, we observe that both ChatGPT and Copilot did not manage to fully follow our prompt guidelines (cp. Appendix, Sect. 1), e.g., when asked to reference joined tables via specifically named aliases. We also observed that especially Copilot added additional columns and tables, not present in the DB schema.

6.5 Understanding SE users information needs and text-to-SQL utility (Survey)

Participants' demographics In total, 26 participants completed our survey. One participant acted as a pilot and was therefore omitted from the evaluation. We had 20% female participants and the average participant was 31 years old. The majority of our participants rated their SQL experience as basic (40%) or good (36%) and reported a programming experience between 5 and 10 years. Less represented were participants that rated their SQL experience as middle (16%) or very good (8%) and reported a programming experience of more than 10 years. A majority of 56% participants reported responsibility for multiple roles in SE, while 44% are either solely developers, researchers, or managers. We also asked participants to rate how often they solve tasks by themselves and how often they ask a colleague for help, both on a Likert scale from 1 (seldom) to 5 (often). On average, participants solve their tasks themselves (4.64) and roughly half of the time they ask colleagues for help (2.44).

Participants' experience in satisfying their information needs A majority of 76% participants had experience with ITS such as either JIRA, GitLab or both. Additionally, 12% were experienced in other ITSs, such as YouTrack and Bugzilla. Merely 12% did not have any experience in using an ITS and we decided to exclude them from further analysis. 72.2% of the participants gained initial experience in ITSs through research projects, 36.3% through student projects, 36.3% through company projects and 22.7% through open source projects. A majority of 68% participants satisfies their information needs in ITSs via a search bar that provides filtering options, 5% of the participants prefers a search solely via query, e.g., Atlassian's Jira Query language (JQL) or GitLab's Elasticsearch syntax, and 23% of the

Table 8 Performance of the ChatGPT (v3.5) and GitHub Copilot (v1.96.263) measured as exact match accuracy (em accuracy) of the predicted queries for evaluation utterances of the SEOSS-Queries dataset reported in total (all) and per complexity level of the resulting query, i.e., easy to extra hard (Xhard). Results are based on the evaluation of the models on the utterance-based and on the query-based splits from the SEOSS-Queries dataset. In the case of T5+PICARD, the results were taken from Experiment 2 provided in Tab. 6

Model	SOESS Queries Utterance-Based Split				SOESS Queries Query-Based Split				[%]↑	all	Xhard	all		
	em accuracy	easy	med.	hard	em accuracy	easy	med.	hard						
<i>specific utterances</i>														
count	56	54	11	12	133	133	15	15	42	20	56	12	20	108
ChatGPT	0.643	0.500	0.000	0.083	0.481	0.083	0.481	0.000	0.750	0.071	0.071	0.000	0.200	0.213
GitHub Copilot	0.411	0.241	0.000	0.167	0.286	0.167	0.286	0.000	0.950	0.143	0.143	0.000	0.000	0.250
T5 + PICARD	0.964	0.981	0.909	1.000	0.970	1.000	0.970	0.667	0.950	0.661	0.661	0.667	0.250	0.639
<i>non-specific utterances</i>														
count	54	11	12	133	15	133	15	15	42	9	9	15	81	0.086
ChatGPT	0.554	0.204	0.000	0.167	0.331	0.167	0.331	0.000	0.267	0.071	0.071	0.000	0.000	0.160
GitHub Copilot	0.429	0.278	0.000	0.000	0.293	0.000	0.293	0.000	0.400	0.167	0.167	0.000	0.000	0.432
T5 + PICARD	0.893	0.926	1.000	0.917	0.917	0.917	0.917	0.667	0.667	0.381	0.381	0.667	0.200	0.432
<i>all utterances</i>														
count	112	108	22	24	266	266	266	266	35	98	98	21	35	189
ChatGPT	0.598	0.352	0.000	0.125	0.406	0.125	0.406	0.000	0.543	0.071	0.071	0.000	0.114	0.159
GitHub Copilot	0.420	0.259	0.000	0.083	0.289	0.083	0.289	0.000	0.714	0.153	0.153	0.000	0.000	0.212
T5 + PICARD	0.929	0.954	0.955	0.958	0.944	0.958	0.944	0.667	0.829	0.541	0.541	0.667	0.229	0.550

participants use both options. Among the participants, 90% use an ITS to inquire information about their daily tasks, e.g., issues assigned to them, 59% inquire information about issues assigned to others, 18% search for contact information, and 9% gather information about the progress of a project, e.g., definition of done for feature requests or team velocity.

Assessing SEOSS-Queries' content When training text-to-SQL models it is necessary to consider the quality of the data used for training. Hence, we assessed whether utterances of the SEOSS-Queries dataset are understandable and representative, i.e., describe a relevant information need of the participant. We asked each participant to rate eight utterances on a five level Likert scale ranging from 1 = not understandable or not representative to 5 = understandable or representative respectively. We observe an average representativeness rating of 3.81 and an average understandability rating of 4.32. Thereby, participants with less than ten years of experience (68%) considered the given utterances less relevant (3.63), while more experienced ones consider them more relevant (4.2). In contrast, the understandability rating was almost independent of a participant's programming experience. We also asked each participants to rate eight SQL queries, two per complexity level easy to extra hard, in order to assess the validity of the query complexity concept used when analyzing the results of text-to-SQL approaches. They rated the queries of the different complexities as follows: easy (84% easy, 16% medium), medium (76% easy, 22% medium, 2% hard), hard (6% easy, 42% medium, 32% hard, 20% extra hard), and extra hard (8% easy, 40% medium, 22% hard, 30% extra hard). We observe that participants tend to rate the complexity of queries automatically assessed as medium to extra hard roughly one level easier than Yu et al.'s evaluation script (cp. Sec. 4.2). For example, the hardness level of the SQL query `SELECT * FROM issue AS T1 WHERE T1.issue_id IN (SELECT T2.issue_id FROM issue_attachment AS T2)` was automatically assessed as hard, while a majority of participants rated it as medium. This tendency towards less complex judgements is interesting since we also assessed participant's capabilities in creating queries below and will report about substantial deficiencies. We argue that query understanding differs from query creation in complexity, potentially explaining this divergence. Overall, even if rated somewhat easier participant's mostly identified the same relative complexity rating of queries giving the measure credibility.

Assessing participants querying capabilities We asked each participant to satisfy two information needs verbally described as scenarios by developing an utterance and a respective query (cp. Sec. 5.2). In total, the set consisted of four scenarios that were expected to result in a query of complexity easy to extra hard respectively. We randomly assigned either the easy and the extra hard scenario or the medium and the hard scenario to a participant. When evaluating participants' utterances and queries, we considered for an utterance whether it correctly described the given scenario and for a query whether it correctly reflected the given scenario and whether it was executable. Participants had mostly no problem in expressing utterances for the easy and medium scenarios with 92% of them matching the given scenario. The remaining 8% utterances either did not correctly mention the concept they were interested in or did not match the intent of the scenario, e.g., rather than retrieving the column 'fix_version' from the table 'issue_fix_version', they retrieved the column 'component' from the table 'issue_component'. For the hard and extra hard scenarios we evaluated 72% of the developed utterances as exactly matching the given scenario, while the remaining ones did not perfectly match the intent of the information need, e.g., participants asked for the names of developers and the count of issues while they were supposed to retrieve the name of the most productive developer. Looking at the formulated participants' utterances we speculate that in some cases participants failed to formulate an utterance with the expected intent because they were not familiar enough with the DB schema in our usecase or did not pay enough attention in the

wording of the given scenario. To this end, we acknowledge that in most cases, the formulated utterances, while not exactly matching the entire intent of the given scenario, are still partially correct. For example, in the case where the most productive developer needs to be retrieved, the expected utterance aims to output the name of a single developer while the formulated utterances aim to retrieve a list with each developer's name and the issue she or he is assigned to. Participants can then find the name of the intended developer by viewing the results from the executed query. Alternatively, they can refine their utterance in cases in which they are not satisfied by the result. A conversational aspect can help in cases in which the user needs to be informed that she or he is searching for a table or column that does not exist (caused for example from a spelling mistake) or in cases where a column, table or value partially matches one or more cases in a DB. However, such aspect will require the utterances used for training to be reformulated into conversational form.

Regarding queries, we found that 56% of the developed queries for the scenarios of complexity level easy or medium were matching the intent and were executable. Main mistakes were missing quotes (24%), spelling mistakes in table and column names (16%), missing operators in the `WHERE` clause (4%). Eventually, only a minor group of 28% of the participants correctly constructed executable queries for the scenarios of complexity hard and extra hard. Queries were incomplete (32%), contained syntax errors (28%), or did not correctly reference the required database concepts (12%) (cp. Fig. 7). This success rate is dependent on participants' experience with 54% of the experienced ones, i.e., SQL experience level good and very good, and merely 7% of the less experienced ones, i.e., SQL experience level basic and middle, writing a correct query. When asked for which form of retrieval was more difficult to create for the easy and medium scenarios, participants responded: query (56%), utterance (3%), and both equally (36%). For the hard and extra hard scenarios, a majority (76%) considers query creation more difficult, while 4% consider the utterance more difficult to create and the remaining ones are indifferent. Based on the results of Experiment 2 (cp. Table 6), we selected the best performing text-to-SQL approach, T5 + Picard, to validate utterances that we would have developed for the given scenario and used them to predict queries. We assessed all predicted queries to be correct.

Answer to RQ3 RQ3 aims to examine how well SE professionals can construct NL utterances and SQL queries. In conclusion, our findings suggest that participants had substantial difficulties in developing queries and that latest text-to-SQL approach could be an actual benefit for SE professionals. On the other hand, we observe that developed utterances, though of higher quality, are also not always correct potentially suggesting that professionals would need some form of and time to get acquainted with such an approach or that they had trouble in understanding the expressed scenario in the survey.

Is imperfectly generated SQL query still useful? We aimed to understand, to what extent participants can assess imperfectly generated SQL queries in terms of three potential problems: syntactically incorrect queries, incomplete queries in terms of SQL concepts, and incomplete queries in terms of database concepts, i.e., tables and columns. First, we asked participants to assess four presented queries, split into clauses such as `SELECT` and `WHERE`, whether it was syntactically correct and found that none of them did correctly differentiate all correct and incorrect clauses in the queries. Separated into two clusters, we found that 84% of the participants identified at least 60% of the incorrect clauses in queries, while the remaining 16% identified less than 30% of the mistaken SQL clauses. Second, we asked participants to decide per presented query whether it was complete and found that 12% of them correctly answered this question for all four queries, while 36% did not assess a single one correctly. Finally, we asked participants to decide per presented query whether it referred to all necessary column and table names. We found that 40% of the participants correctly



Fig. 7 Five examples of mistaken queries developed by survey participants in order to satisfy an information need expressed as scenario in the survey

solved this task for all four provided queries, while 12% did not correctly decide for a single one. These results suggest, that in cases, where column or table names are incorrect, users can still use the wrongly generated SQL queries. However, this is not the case when the SQL query is incomplete, e.g., has missing SQL clauses. In such cases users are expected to need more time to complete the SQL. Here, we argue, that it may take SE experts less time, however, in comparison to writing the SQL from scratch, since the result from text-to-SQL already focuses on important tables and columns from the DB schema and partially generates a SQL.

Usefulness of text-to-SQL approaches for SE tasks A majority of 52% participants vote for using a combination of text-to-SQL approaches while still being able to manually write queries to satisfy their information needs, 32% would solely use text-to-SQL models rather than writing queries themselves and 16% prefer to write SQL themselves. A vast majority of 92% participants answered that they needed multiple tries to construct a desired and correct query, which suggests that they lose precious development time when constructing queries manually. Furthermore, we observe that utterances are easier to construct for the participants,

especially in cases where the resulting query is of complexity hard or extra hard. Based on the very good performance observed in Experiment 2 (cp. Table 6) and considering the challenges that participants face in constructing queries themselves, we argue that text-to-SQL approaches are ready to help SE experts in satisfying their daily information needs and thereby improving their development projects. Asked, whether they find text-to-SQL approaches useful and why, nineteen participants (76%) answered that they find text-to-SQL approaches helpful for people with basic or no knowledge in SQL, seventeen (68%) consider the usage of text-to-SQL when constructing SQL queries less time-consuming and three (12%) see the usage as a good starting point when writing SQL even beyond the application in a question answering scenario. Four participants (16%) find text-to-SQL error-prone, one (4%) participant time-consuming, and one (4%) participant expresses concern that wrongly generated SQL can lead to overestimation and misinterpretation in cases when the user is not knowledgeable in SQL.

Answer to RQ4 RQ4 aims to examine how open SE professionals are, after knowing the capabilities of text-to-SQL models, in using such approaches. In general, participants are open to the use of text-to-SQL approaches in the SE domain. Even though there is still room for improvement, especially in creating more representative training sets that allow for a better support of new and unseen queries.

7 Threats to Validity

There are several potential threats to the validity in our study in regard to the performed experiments and the performed survey.

Construct validity deals with the evaluation of measurements that are used to evaluate the validity of an investigated method. SQL queries in the SEOSS-Queries dataset were manually created by the authors of the paper. In cases, in which alternative SQL queries would have lead to identical results and satisfied the given utterance, only one query was chosen and included in the dataset. Exact match accuracy was used to evaluate generated SQL queries by models. This type of accuracy metric implies that deviations in the SQL keywords used in the gold and predicted SQL queries would lead to lower accuracy, even in cases in which the outputted result matches the one in the gold evaluation set. A further threat to consider is the SQL grammar that is accepted by text-to-SQL approaches. Not all SQL queries, e.g., SQL containing functions such as `strtime` or specific SQL keywords such as `NOT`, `CASE`, from the SEOSS-Queries dataset could be processed, meaning that we could not use the entire SEOSS-Queries dataset for model evaluation. Nevertheless, without the incompatible SQL queries, we still had a substantial amount of data upon which to base our experiments. A concern may be expressed regarding the expressiveness of the SE dataset used in our study, considering that not all SQL grammar can be handled by text-to-SQL approaches from our study. Since our main goal is to assess whether text-to-SQL approaches can be considered applicable and useful in the SE domain, we decided not to extend the currently accepted SQL grammar, as to give a more realistic representation of the current capabilities of such approaches.

Internal validity represents the confidence with which a cause-and-effect relationship from a study cannot be explained by other factors. The way data for training and evaluation was chosen may have affected our findings. We performed three experiments aiming to investigate on one hand the importance of what data is used for training and on the other hand the performance of text-to-SQL approaches on different evaluation sets. We had a rather small number of 25 participants in our survey potentially not being sufficiently represen-

tative. We argue however, that though the sample could clearly be larger, we had a large and representative variety of participants and observed fairly consistent result allowing us to draw conclusions on the benefit of text-to-SQL approaches for the SE domain. Three of the authors evaluated the user survey results. Cases, in which the evaluation of the results differed between the authors, were discussed and agreed upon.

External validity concerns with the ability to generalize the results beyond the studied context. The experiments we performed were based on open-source data only. One can argue that these data not entirely represent data found in commercial projects and can be considered a potential threat to validity. Given, however, that it is gathered from software tools used by SE professionals on a daily manner, we argue that it reflects daily information needs of SE professionals and provides a much more realistic scenario than data generated by solely artificial means. The work assumes that a database exists in which data from various tools are stored, without going into further details how such database can be created. Integrating data from different sources is a challenging task, acknowledged for decades. However, this is not the main focus of our work and we refer to the following relevant works, e.g., Mäder and Cleland-Huang (2013); Hassan (2008); Rath et al. (2017); Keivanloo et al. (2012); Kolovos et al. (2019) and existing tool integration solutions like Apache Builder,² dealing with this challenging task.

8 Conclusions

During software development, SE experts can use information stored in development tools' repositories as a source for satisfying their information needs. Recent advances in machine learning for NLP lead to a variety of text-to-SQL approaches that can be utilized for satisfying these information needs based on simply posing a NL question (aka utterance). Our extensive experiments show impressive results when training latest approaches on SE-specific training data. In an accompanying user survey, we found that SE professionals have substantial deficits in satisfying their information needs via SQL queries, but perform much better when expressing them as utterance demonstrating the benefit that these approaches can offer in the SE domain. Apart from methodological advancements, a key enabler are representative training sets. Given the variety of language that can be used when formulating an utterance, e.g., synonyms, there is clearly a need for further research and the orchestration of richer datasets. Our survey shows that SE professionals are open to be supported by text-to-SQL approaches. They find them less time-consuming and expert-level professionals consider them a good starting point when creating their own SQL queries. We conclude that text-to-SQL approaches have a place in the SE domain and are ready to be provided to practitioners.

There are several key areas, however, that could contribute to the further improvement and adoption of text-to-SQL models in the SE domain. It is crucial to have rich and diverse datasets that encompass a wide range of NL expressions and SQL syntax. With the abundance of SE tools available today, it seems possible to create datasets that cover manifold information needs and deliver the required large quantities of training data. The development of text-to-SQL models would also benefit from established guidelines assisting in the construction of comprehensive training and test sets, and in selecting appropriate evaluation criteria. Finally, the effective integration of text-to-SQL models into the daily work of SE professionals needs to be further studied. This involves careful consideration of how these models can be seamlessly integrated into existing workflows, how to assess the correctness of provided results and how to deliver results in the most useful form for the developer.

² provides a way to download artifacts from remote repositories into a local repository.

Appendix

Evaluation on Large Language Models

Familiarity of ChatGPT with the spider dataset.

When asked about what is the spider dataset, ChatGPT gave the following response:

- *“As of my last update in September 2021, there is no widely known or standardized "spider dataset." However, it's possible that new datasets have been created or popularized after my last update.*

Up until that point, one notable dataset related to spiders is the "Spider" dataset, which is used for the evaluation of natural language interfaces to databases (NLIDBs). The Spider dataset was introduced by researchers from Stanford University and contains complex SQL queries paired with corresponding natural language questions. It was designed to test and benchmark the performance of various natural language understanding and SQL generation models on the task of converting natural language questions into SQL queries for relational databases.

If there are any newer datasets or specific contexts you are referring to beyond my last update, I won't have information on them. I recommend checking recent research papers, academic databases, or data repository websites for the latest datasets related to spiders or any other topics of interest.”

Further inquiry about whether ChatGPT can provide examples of complex SQL queries inside the Stanford dataset, we received the following response:

- *“As an AI language model, I don't have direct access to specific datasets, including the Stanford "Spider" dataset, or its contents. However, I can provide you with an example of a complex SQL query that could be present in such a dataset based on the general understanding of the type of queries typically found in NLIDB evaluation datasets. Suppose we have a database schema with two tables: "Employees" and "Departments." The "Employees" table contains information about employees, such as their ID, name, department ID, and salary. The "Departments" table contains department information, including the department ID and the department name. A complex SQL query in this context might be something like:*

Natural Language Question:

List the department names and the average salary of all departments with more than 100 employees.

SQL Query:

```
SELECT d.department_name, AVG(e.salary) AS average_salary
FROM Departments d
JOIN Employees e ON d.department_id = e.department_id
GROUP BY d.department_name
HAVING COUNT(e.employee_id) > 100;
```

In this example, the SQL query retrieves the department names and the average salary of departments that have more than 100 employees. It involves a join operation between the "Departments" and "Employees" tables and uses aggregation functions (AVG) along

with the GROUP BY and HAVING clauses to filter the results based on the number of employees in each department.

Keep in mind that the actual queries in the Stanford "Spider" dataset could be more complex and diverse, covering a wide range of SQL functionalities and challenging natural language expressions. To get access to the actual dataset, you should refer to the official sources provided by Stanford or the research papers related to the dataset."

Prompting ChatGPT and GitHub Copilot

Both ChatGPT and GitHub Copilot can be employed for the text-to-SQL task. In both cases a prompt describing the DB schema needs to be first included before querying NL utterances. Additional information such as what SQL syntax to use and how to display the generated SQL queries was included in the prompt. For the experiments we used ChatGPT 3.5 and GitHub Copilot v1.96.263

– Prompt:

```
You are a helpful assistant for generating
syntactically correct read-only SQL to answer a given
NL utterance.
```

```
Database: apache-pig
```

```
The following is the schema of the tables you can
query:
```

```
$ db-schema
```

```
If it is necessary to use AS then use it like T1 T2
..., but if the alias name is not going to be used in
the query again, then do not use.
```

```
If in using COUNT(*) and COUNT(COLUMN) there is no
difference then use COUNT(*).
```

```
Generate SQL with SQLite syntax.
```

– DB schema

```
PRAGMA foreign_keys = ON;
CREATE TABLE IF NOT EXISTS "issue_fix_version" (
  "issue_id" text NOT NULL,
  "fix_version" text NOT NULL,
  FOREIGN KEY("issue_id") REFERENCES "issue"("issue_id")
);
```



```
CREATE TABLE IF NOT EXISTS "issue_component" (  
  "issue_id" text NOT NULL,  
  "component" text NOT NULL,  
  FOREIGN KEY("issue_id") REFERENCES "issue"("issue_id")  
);  
CREATE TABLE IF NOT EXISTS "issue_comment" (  
  "issue_id" text NOT NULL,  
  "username" text,  
  "display_name" text,  
  "created_date" text,  
  "created_date_zoned" text,  
  "message" text,  
  FOREIGN KEY("issue_id") REFERENCES "issue"("issue_id")  
);  
CREATE TABLE IF NOT EXISTS "issue_changelog" (  
  "issue_id" text NOT NULL,  
  "username" text,  
  "display_name" text,  
  "created_date" text,  
  "created_date_zoned" text,  
  "group_id" integer,  
  "field_type" text,  
  "field" text,  
  "from_value" text,  
  "from_string" text,  
  "to_value" text,  
  "to_string" text,  
  FOREIGN KEY("issue_id") REFERENCES "issue"("issue_id")  
);  
CREATE TABLE IF NOT EXISTS "issue_attachment" (  
  "issue_id" text NOT NULL,  
  "username" text,  
  "display_name" text,  
  "created_date" text,  
  "created_date_zoned" text,  
  "mime_type" text,  
  "content" text,  
  "filename" text,  
  "size_bytes" integer,  
  FOREIGN KEY("issue_id") REFERENCES "issue"("issue_id")  
);  
CREATE TABLE IF NOT EXISTS "issue" (  
  "issue_id" text NOT NULL UNIQUE,  
  "type" text,  
  "created_date" text,  
  "created_date_zoned" text,  
  "updated_date" text,  
  "updated_date_zoned" text,  
  "resolved_date" text,
```



```
"resolved_date_zoned" text,
"summary" text,
"description" text,
"priority" text,
"status" text,
"resolution" text,
"assignee" text,
"assignee_username" text,
"reporter" text,
"reporter_username" text,
PRIMARY KEY("issue_id")
);
CREATE TABLE IF NOT EXISTS "code_change" (
"commit_hash" text NOT NULL,
"file_path" text,
"old_file_path" text,
"change_type" text,
"patch_type" text,
"is_deleted" integer,
"sum_added_lines" integer,
"sum_removed_lines" integer,
FOREIGN KEY("commit_hash") REFERENCES "change_set"
("commit_hash")
);
CREATE TABLE IF NOT EXISTS "change_set_link" (
"commit_hash" text NOT NULL,
"issue_id" text NOT NULL,
FOREIGN KEY("commit_hash") REFERENCES "change_set"
("commit_hash"),
FOREIGN KEY("issue_id") REFERENCES "issue"("issue_id")
);
CREATE TABLE IF NOT EXISTS "change_set" (
"commit_hash" text NOT NULL UNIQUE,
"git_repository_id" integer,
"committed_date" text,
"committed_date_zoned" text,
"message" text,
"author" text,
"author_email" text,
"is_merge" integer,
PRIMARY KEY("commit_hash")
);
CREATE TABLE IF NOT EXISTS "meta" (
"key" text NOT NULL UNIQUE,
"value" text NOT NULL
);
CREATE TABLE IF NOT EXISTS "git_repository" (
"git_repository_id" integer NOT NULL UNIQUE,
"name" text NOT NULL,
```

```

    "url" text NOT NULL,
    "checkout_hash" text NOT NULL
);
CREATE TABLE "issue_link" (
    "source_issue_id" text NOT NULL,
    "target_issue_id" text NOT NULL,
    "name" text,
    "outward_label" text,
    "is_containment" integer,
    FOREIGN KEY("source_issue_id") REFERENCES "issue"
        ("issue_id"),
    FOREIGN KEY("target_issue_id") REFERENCES "issue"
        ("issue_id"));
CREATE TABLE IF NOT EXISTS "jira_repository" (
    "key" text NOT NULL UNIQUE,
    "base_url" text NOT NULL
);
COMMIT;

```

Overview of questions asked for the user survey

Section 1: Participants' demographics.

- Q 1: How old are you? (cp. Fig 8)
- Q 2: What is your gender? (cp. Fig 8)
- Q 3: Are you in a noisy environment? (cp. Fig 8)
- Q 4: What is your highest level of education? (cp. Fig 8)
- Q 5: How many years of programming experience do you have? (cp. Fig 8)
- Q 6: How experienced are you in writing SQL queries? (cp. Fig 8)
- Q 7: What is/are your role(s) in software projects? (cp. Fig 9)
- Q 8: How often do you resolve work problems/tasks by yourself and how often by asking and discussing with a colleague? (cp. Fig 9)

Section 2: Participants' experience in satisfying their information needs.

- Q 9: What Issue Tracking Systems are you familiar with? (cp. Fig 9)
- Q 10: In what kind of projects did you use Issue Tracking Systems? (cp. Fig 10)
- Q 11: How do you extract information from Issue Tracking Systems? (cp. Fig 10)
- Q 12: What information do you typically retrieve from an Issue Tracking System? (cp. Fig 10)
- Q 13: How understandable, i.e., is the intent clear, and relevant for SE professionals are the following NL utterances? (cp. Fig 11)
- Q 14: How hard are the following SQL queries to construct? (cp. Fig 12)
- Q 15: Match the following natural language utterances to their corresponding SQL query. (cp. Fig 13)

Section 3: Assessing participants querying capabilities

- Q 16: Given a DB schema write the natural language (NL) and SQL based on the given scenario and rate which (SQL, NL) was harder to construct. (cp. Fig 14)

Section 4: Is an imperfectly generated SQL query still useful?

- Q 17: Given the given natural language (NL) utterance and the given wrongly generated SQL queries, can you identify where the mistakes are and rate how difficult it would be to correct the query? (cp. Figure 15)

Section 5: Usefulness of text-to-SQL approaches for SE tasks

- Q 18: Would you prefer to use a text-to-SQL approach rather than writing SQL queries yourself for satisfying your SE information needs? (cp. Fig 16)
- Q 19: When writing SQL do you need multiple tries to get a desired result? (cp. Fig 16)
- Q 20: Do you find text-to-SQL models useful? (cp. Fig 16)

Detailed Survey Questions

1. How old are you?

2. What is your gender?

female

male

other

prefer not to respond

3. Are you in a noisy environment?

yes

no

4. What is your highest level of education?

no formal education

Bachelor

Master

PhD

other

5. How many years of programming experience do you have?

6. How experienced are you in writing SQL queries?

1 (low)

2

3

4

5

6 (high)

Fig. 8 Survey questions 1 to 6

7. What is/are your role(s) in software projects?

manager

researcher

developer

tester

other (please specify)

8. How often do you resolve work problems/tasks by yourself and how often by asking and discussing with a colleague?

by myself

1 (seldom) 2 3 4 5 (often)

by asking and discussing with a
colleague

1 (seldom) 2 3 4 5 (often)

3 Experience in querying data

Experience in querying data

1. What Issue Tracking Systems are you familiar with?

JIRA

GITLAB

other (please specify)

no experience in Issue Tracking Systems, I use instead...

Fig. 9 Survey questions 7 to 9

2. In what kind of projects did you use Issue Tracking Systems?

- research project
- open source project
- company project
- student project
- prefer not to respond

3. How do you extract information from Issue Tracking Systems?

- via a query language
- via a search option
- another way (please specify how)

4. What information do you typically retrieve from an Issue Tracking System?

- information about my daily tasks (What are my tasks?, Any comments made by others on my issues?)
- information about issues solved by others (Who solved what?)
- contact information of assignees/reporters
- other (please specify)

4.1 Experience in querying data NL

In the following section, the natural language utterances you will see, are used for training and evaluation of text-to-SQL models.

5.1. How understandable, i.e., is the intent clear, and relevant for SE professionals are the following NL utterances?

List all issues with attachments

1 (not understandable) 2 3 4 5 (understandable)

Fig. 10 Survey questions 10 to 12

1 (not relevant) 2 3 4 5 (relevant)

4.3 Experience in querying data NL 2

In the following section, the natural language utterances you will see, are used for training and evaluation of text-to-SQL models.

5.3. How understandable, i.e., is the intent clear, and relevant for SE professionals are the following NL utterances?

Find all issue ids of issues that were reported between '2017-01-01' and '2017-03-31'

1 (not undestandable) 2 3 4 5 (understandable)

1 (not relevant) 2 3 4 5 (relevant)

Who is the person in charge of issue PIG-3599

1 (not undestandable) 2 3 4 5 (understandable)

1 (not relevant) 2 3 4 5 (relevant)

4.4 Experience in querying data NL 3

In the following section, the natural language utterances you will see, are used for training and evaluation of text-to-SQL models.

Fig. 11 Excerpt of survey question 13

Text-to-SQL approaches can successfully generate SQL queries of different complexity levels (easy, medium, hard, extra hard). The difficulty level of the SQL queries varies depending on the number of SQL keywords, clauses, and JOINS used to construct them. An easy query for example selects single column from a single table, while an extra hard SQL query selects one or more columns and contains multiple clauses of the same or different kind (e.g. WHERE, HAVING, GROUP BY).

In the following section you will be provided with SQL queries that were successfully generated by a state-of-the-art text-to-SQL approach.

6.3. How hard are the following SQL queries to construct?

SELECT assignee FROM issue WHERE issue_id = "PIG-3599"

1 (easy) 2 (medium) 3 (hard) 4 (extra hard)

SELECT type, Count(*) FROM issue GROUP BY type HAVING type = 'Bug' OR type = 'Improvement'

1 (easy) 2 (medium) 3 (hard) 4 (extra hard)

4.8 Experience in querying data SQL 3

Text-to-SQL approaches can successfully generate SQL queries of different complexity levels (easy, medium, hard, extra hard). The difficulty level of the SQL queries varies depending on the number of SQL keywords, clauses, and JOINS used to construct them. An easy query for example selects single column from a single table, while an extra hard SQL query selects one or more columns and contains multiple clauses of the same or different kind (e.g. WHERE, HAVING, GROUP BY).

In the following section you will be provided with SQL queries that were successfully generated by a state-of-the-art text-to-SQL approach.

6.4. How hard are the following SQL queries to construct?

SELECT T1.source_issue_id, T1.target_issue_id FROM issue_link AS T1 JOIN issue AS T2 ON T1.source_issue_id = T2.issue_id JOIN issue AS T3 ON T1.target_issue_id = T3.issue_id WHERE T2.type = 'Improvement' AND T3.type = 'Bug' ORDER BY T1.source_issue_id DESC

1 (easy) 2 (medium) 3 (hard) 4 (extra hard)

Fig. 12 Excerpt of Survey question 14

SELECT issue_id FROM issue WHERE created_date BETWEEN '2017-01-01' AND '2017-03-31'

1 (easy) 2 (medium) 3 (hard) 4 (extra hard)

4.9 Experience in querying data Karussell

7. Match the following natural language utterances to the their corresponding SQL query.

Start by looking at the SQL queries and then make your choice. In total 8 natural language utterances must be matched.

				<pre>SELECT T1.source_issue _id, T1.target_issue_ id FROM issue_link AS T1 JOIN issue AS T2 ON SELECT * FROM issue AS T1 WHERE T1.issue_id IN (SELECT T2.issue_id FROM issue_attachmen t AS T2) SELECT assignee FROM issue WHERE issue_id = "PIG-3599" SELECT T1.source_issue _id = T2.issue_id JOIN issue AS T3 ON T1.target_issue_ id = T3.issue_id WHERE T2.type = 'Improvement' AND T3.type = 'Bug' ORDER BY T1.source_issue _id DESC</pre>	No matching SQL
Find issues where Daniel Ward is the assignee and the status is 'Open'	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
List all issues with attachments	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Who is the person in charge of issue PIG-3599	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Give me the source and target issues ids of linked issues where the source issue id is of type bug and the target issue id is of type improvement	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Return open issues assigned to Daniel Ward	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Give me all issues that have an attachment	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Give me the assignee to whom issue id PIG-3599 is assigned	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Give me ids of linked issues where the source issue is of type bug and the target issue is of type improvement	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

4.10 Complexity in querying data

Complexity in querying data

Fig. 13 Excerpt of survey question 15

8. Given a DB schema write the natural language (NL) and SQL based on the given scenario and rate which (SQL, NL) was harder to construct.

Use the DB schema to chose appropriate table and column names when constructing the SQL queries. Aim to construct a correct SQL. You can search on the Internet regarding SQL syntax if necessary.

The issue Tracker Jira provides different values for the following:

- **priority can be Minor, Trivial, Major, Critical, Blocker;**
- **status can be Resolved, Closed, Open, Patch Available, Reopened, In Progress;**
- **resolution can be Duplicate, Fixed, Done, Incomplete, Resolved;**
- **issue links can be Duplicate, Blocker, Reference, Cloners, Required;**
- **issue types can be Bug, Improvement, Task, New Feature, Wish;**

The researcher Alice works on a project dealing with examining traceability in Issue Tracking Systems (ITSs). As trace links can be considered links between issues. The first thing Alice wants to find out is what issue links exist in an ITS.

NL:	<input type="text"/>
SQL:	<input type="text"/>

Which was harder to construct?

NL

SQL

both were equally hard to construct

The manager Bob would want to know who is the most productive developer in his team. A good indication of the productivity of each developer is the number of issues each one of them is working on. Help Bob find the most productive developer.

NL:	<input type="text"/>
SQL:	<input type="text"/>

Fig. 14 Excerpt of survey question 16

	right	wrong
SELECT clause	<input type="radio"/>	<input type="radio"/>
FROM clause	<input type="radio"/>	<input type="radio"/>
GROUP BY	<input type="radio"/>	<input type="radio"/>
ORDER BY	<input type="radio"/>	<input type="radio"/>
assuming column and table names are correct, the SQL syntax is NOT correct or is NOT complete (e.g., missing clauses, missing important SQL keywords)	<input type="radio"/>	<input type="radio"/>
all relevant column and table names for the construction of a SQL query leading to a desired result were present and correct	<input type="radio"/>	<input type="radio"/>

How difficult will it be to correct the SQL query?

1 (easy)
2
3
4
5 (hard)

NL: Find all issue ids of issues solved on the 2013-10-23

SQL: SELECT issue.issue_id FROM issue WHERE issue.status = "2013-10-23"

	right	wrong
SELECT clause	<input type="radio"/>	<input type="radio"/>
FROM clause	<input type="radio"/>	<input type="radio"/>
WHERE clause	<input type="radio"/>	<input type="radio"/>
assuming column and table names are correct, the SQL syntax is NOT correct or is NOT complete (e.g., missing clauses, missing important SQL keywords)	<input type="radio"/>	<input type="radio"/>
all relevant column and table names for the construction of a SQL query leading to a desired result were present and correct	<input type="radio"/>	<input type="radio"/>

How difficult will it be to correct the SQL query?

1 (easy)
2
3
4
5 (hard)

4.13 Usage of text-to-SQL approaches 3

Fig. 15 Excerpt of survey question 17

Text-to-SQL approaches show promising results, however they still need to be improved. From the previous section you got a feeling of what kind of NL utterances and SQL queries a text-to-SQL model can understand and construct.

1. Would you prefer to use a text-to-SQL approach rather than writing SQL queries yourself for satisfying your SE information needs?

- use text-to-SQL
- write SQL myself
- both

2. When writing SQL do you need multiple tries to get a desired result?

- one try
- multiple tries

3. Do you find text-to-SQL models useful?

<input type="checkbox"/> yes	<input type="checkbox"/> no
why?	why?
<input type="checkbox"/> less error-prone	<input type="checkbox"/> error-prone
<input type="checkbox"/> less time-consuming	<input type="checkbox"/> time-consuming
<input type="checkbox"/> helps stakeholders with little knowledge in SQL	<input type="checkbox"/> not helpful
<input type="checkbox"/> other reasons?	<input type="checkbox"/> other reasons?
<input type="text"/>	<input type="text"/>

7 Endseite

Thank you for your participation!

Fig. 16 Survey questions 18 to 20

Acknowledgements Our work is funded by DFG grant: MA 5030/3-1 and BMBF grant: 16DHB4009.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability The datasets analysed during the current study are available in a Github repository (Spider Leaderboard 2023) (Spider) and a Figshare repository (SEOSS-Queries Repository 2023) (SEOSS-Queries).

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Abdellatif A, Badran K, Shihab E (2020) Msrbot: Using bots to answer questions from software repositories. *Empir Softw Eng* 25(3):1834–1863
- Apache Pig project. <https://pig.apache.org/>. Accessed 12 January 2023
- Evaluation script spider (2023) <https://github.com/taoyds/spider>. Accessed 12 January 2023
- Assembla (2023) <https://get.assembla.com/>. Accessed 12 January 2023
- Atlassian JIRA (2023) <https://www.atlassian.com/de/software/jira>. Accessed 12 January 2023
- Bahdanau D, Cho K, Bengio Y (2015) Neural machine translation by jointly learning to align and translate. In: ICLR
- Begel A, Zimmermann T (2014) Analyze this! 145 questions for data scientists in software engineering. In: ICSE, pp. 12–23. ACM
- Bertram D, Voida A, Greenberg S, Walker RJ (2010) Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In: CSCW, pp. 291–300. ACM
- Cao R, Chen L, Chen Z, Zhao Y, Zhu S, Yu K (2021) LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. 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), pp. 2541–2555. Association for Computational Linguistics, Online. <https://doi.org/10.18653/v1/2021.acllong.198>. <https://aclanthology.org/2021.acl-long.198>
- Ciniselli M, Cooper N, Pascarella L, Mastropaolo A, Aghajani E, Poshyanyk D, Penta MD, Bavota G (2022) An empirical study on the usage of transformer models for code completion. *IEEE Trans Software Eng* 48(12):4818–4837
- Clark K, Luong M, Le QV, Manning CD (2020) ELECTRA: pre-training text encoders as discriminators rather than generators. In: ICLR OpenReview net
- Codd EF (1970) A relational model of data for large shared data banks. *Commun. ACM* 13(6):377–387
- Devlin J, Chang M, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (1):4171–4186. Association for Computational Linguistics
- Fritz T, Murphy GC (2010) Using information fragments to answer the questions developers ask. In: ICSE (1):175–184. ACM
- Git (2023) <https://git-scm.com/>. Accessed 12 January 2023
- Github (2023) <https://github.com/>. Accessed 12 January 2023
- GitHub Copilot (2023) <https://github.com/features/copilot>. Accessed 18 Juli 2023
- Godfrey MW, Hassan AE, Herbsleb JD, Murphy GC, Robillard MP, Devanbu PT, Mockus A, Perry DE, Notkin D (2009) Future of mining software archives: A roundtable. *IEEE Softw* 26(1):67–70. <https://doi.org/10.1109/MS.2009.10>
- Green BF, Wolf AK, Chomsky C, Laughery K (1961) Baseball: An automatic question answerer. In: Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference, IRE-AIEE-ACM'61 (Western), vol. 19, pp. 219-224. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/1460690.1460714>
- Guo J, Zhan Z, Gao Y, Xiao Y, Lou J, Liu T, Zhang D (2019) Towards complex text-to-sql in cross-domain database with intermediate representation. In: ACL (1):4524–4535. Association for Computational Linguistics
- Hassan AE (2006) Mining software repositories to assist developers and support managers. In: ICSM, pp. 339-342. IEEE Computer Society
- Hassan AE (2008) The road ahead for mining software repositories. In: 2008 IEEE International Conference on Software Maintenance 48–57. <https://doi.org/10.1109/FOSM.2008.4659248>
- Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural Comput* 9(8):1735–1780
- Janke M, Mäder P (2022) Graph based mining of code change patterns from version control commits. *IEEE Trans Software Eng* 48(3):848–863

- Kajiura T, Souma N, Sato M, Takahashi M, Kuramitsu K (2022) An additional approach to pre-trained code model with multilingual natural languages. In: APSEC 580–581. IEEE
- Keivanloo I, Forbes C, Hmood A, Erfani M, Neal C, Peristerakis G, Rilling J (2012) A linked data platform for mining software repositories. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR) 32–35. <https://doi.org/10.1109/MSR.2012.6224296>
- Ko AJ, DeLine R, Venolia G (2007) Information needs in collocated software development teams. In: ICSE 344–353. IEEE Computer Society
- Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y (2022) Large language models are zero-shot reasoners. In: NeurIPS. <http://papers.nips.cc/paperfiles/paper/2022/hash/8bb0d291acd4ac06ef112099c16f326-Abstract-Conference.html>
- Kolovos D, Neubauer P, Barmpl K, Matragkas N, Paige R (2019) Crossflow: A framework for distributed mining of software repositories. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR) 155–159. <https://doi.org/10.1109/MSR.2019.00032>
- Kudo T, Richardson J (2018) Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In: EMNLP (Demonstration) 66–71. Association for Computational Linguistics
- Lee C, Gottschlich J, Roth D (2021) Toward code generation: A survey and lessons from semantic parsing. CoRR [arXiv:2105.03317](https://arxiv.org/abs/2105.03317)
- Lin J, Liu Y, Guo J, Cleland-Huang J, Goss W, Liu W, Lohar S, Monaikul N, Rasin A (2017) Tiqui: a natural language interface for querying software project data. In: ASE 973–977. IEEE Computer Society
- Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, Stoyanov V (2019) Roberta: A robustly optimized BERT pretraining approach. CoRR [arXiv:1907.11692](https://arxiv.org/abs/1907.11692)
- Lohar S, Cleland-Huang J, Rasin A, Mäder P (2015) Live study proposal: Collecting natural language trace queries. In: R. Matulevicius, T. Weyer, P. Forbrig, A. Herrmann, M. Daneva, J. Dörr, A. Hoffmann, A. Kalenborn, M. Trapp, G. Herzwurm, W. Pietsch, A. Lenz, S. Schockert, M. Daun, C. Palomares, I. Morales-Ramirez, B. Tenbergen, B. Paech, R.J. Wieringa, E. Knauss, A. Perini (eds.), Joint Proceedings of REFSQ-2015 Workshops, Research Method Track, and Poster Track co-located with the 21st International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2015), Essen, Germany, March 23, 2015, CEUR Workshop Proceedings 1342:207–210. CEUR-WS.org. <http://ceur-ws.org/Vol-1342/preface-RMT.pdf>
- Lu H, Chan HC, Wei KK (1993) A survey on usage of SQL. SIGMOD Rec 22(4):60–65
- Mäder P, Cleland-Huang J (2013) A visual language for modeling and executing traceability queries. *Softw. Syst. Model.* 12(3):537–553. <https://doi.org/10.1007/s10270-012-0237-0>
- Mastrolopolo A, Pascarella L, Bavota G (2022) Using deep learning to generate complete log statements. In: ICSE 2279–2290. ACM
- Maven (2023) <https://maven.apache.org/>. Accessed 12 January 2023
- Nan L, Hsieh C, Mao Z, Lin XV, Verma N, Zhang R, Kryscinski W, Schoelkopf H, Kong R, Tang X, Mutuma M, Rosand B, Trindade I, Bandaru R, Cunningham J, Xiong C, Radev DR (2022) Fetaqa: Free-form table question answering. *Trans Assoc Comput Linguistics* 10:35–49
- OpenAI ChatGPT (2023) <https://openai.com/blog/chatgpt>. Accessed 18 Juli 2023
- Pennington J, Socher R, Manning CD (2014) Glove: Global vectors for word representation. In: EMNLP 1532–1543. ACL
- Portillo-Rodríguez J, Vizcaino A, Ebert C, Piattini M (2010) Tools to support global software development processes: A survey. In: ICGSE 13–22. IEEE Computer Society
- Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu PJ (2020) Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* 21(140):1–67. <http://jmlr.org/papers/v21/20-074.html>
- Rath M, Mäder P (2019) The seoss 33 dataset - requirements, bug reports, code history, and trace links for entire projects. *Data in Brief* 25:104005. <https://doi.org/10.1016/j.dib.2019.104005>
- Rath M, Rempel P, Mäder, P (2017) The ilmseven dataset. In: 2017 IEEE 25th International Requirements Engineering Conference (RE) 516–519. <https://doi.org/10.1109/RE.2017.18>
- Rath M, Rendall J, Guo JLC, Cleland-Huang J, Mäder P (2018) Traceability in the wild: automatically augmenting incomplete trace links. In: ICSE 834–845. ACM
- Requirements management products (2023) <https://www.ibm.com/dede/products/requirements-management>. Online; accessed 12 January 2023
- Rubin O, Berant J (2021) Smbop: Semi-autoregressive bottom-up semantic parsing. In: NAACL-HLT 311–324. Association for Computational Linguistics
- Scholak T, Schucher N, Bahdanau D (2021) PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In: Proceedings of the 2021 Conference on Empirical Methods in Natural

- Language Processing, pp. 9895–9901. Association for Computational Linguistics. <https://aclanthology.org/2021.emnlp-main.779>
- Scholak T, Schucher N, Bahdanau D (2021) PICARD: parsing incrementally for constrained auto-regressive decoding from language models. In: EMNLP (1):9895–9901. Association for Computational Linguistics
- Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. *IEEE Trans Signal Process* 45(11):2673–2681
- Selenium site (2023) <https://www.selenium.dev/>. Accessed 12 January 2023
- SEOSS-Queries Repository (2023) <https://figshare.com/s/e2190f2d32798ce1d0fd>. Accessed 12 January 2023
- Shang W, Nagappan M, Hassan AE, Jiang ZM (2014) Understanding log lines using development knowledge. In: ICSME 21–30. IEEE Computer Society
- Shaw P, Chang MW, Pasupat P, Toutanova K (2021) Compositional generalization and natural language variation: Can a semantic parsing approach handle both? 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), pp. 922–938. Association for Computational Linguistics, Online. <https://doi.org/10.18653/v1/2021.acl-long.75>
- Shaw P, Uszkoreit J, Vaswani A (2018) Self-attention with relative position representations. In: NAACL-HLT (2):464–468. Association for Computational Linguistics
- Spider Leaderboard (2023) <https://yale-lily.github.io/spider>. Accessed 12 January 2023
- SQLNetSpider version (2023) <https://github.com/taoyds/spider/tree/master/baselines/sqlnet>. Accessed 12 January 2023
- Störrle H (2011) VMQL: A visual language for ad-hoc model querying. *J Vis Lang Comput* 22(1):3–29
- Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: NIPS 3104–3112
- Tomova M, Hofmann M, Mäder P (2022) Seoss-queries - a software engineering dataset for text-to-sql and question answering tasks. *Data in Brief* 42:108211. <https://doi.org/10.1016/j.dib.2022.108211>
- Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: NIPS 5998–6008
- Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. In: NIPS 2692–2700
- Waltz DL (1978) An english language question answering system for a large relational database. *Commun. ACM* 21(7):526–539. <https://doi.org/10.1145/359545.359550>
- Wang B, Shin R, Liu X, Polozov O, Richardson M (2020) RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. In: ACL, pp. 7567–7578. Association for Computational Linguistics
- Woods W (1977) Lunar Rocks in Natural English: Explorations in Natural Language Question Answering 5:521–569. North-Holland
- Xu X, Liu C, Song D (2017) Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR arXiv:1711.04436*
- Yin P, Neubig G (2017) A syntactic neural model for general-purpose code generation. In: ACL (1):440–450. Association for Computational Linguistics
- Yu T, Li Z, Zhang Z, Zhang R, Radev DR (2018) Typesql: Knowledge-based type-aware neural text-to-sql generation. In: NAACL-HLT (2):588–594. Association for Computational Linguistics
- Yu T, Wu C, Lin XV, Wang B, Tan YC, Yang X, Radev DR, Socher R, Xiong C (2021) Grappa: Grammar-augmented pre-training for table semantic parsing. In: ICLR. Open-Review net
- Yu T, Zhang R, Yang K, Yasunaga M, Wang D, Li Z, Ma J, Li I, Yao Q, Roman S, Zhang Z, Radev DR (2018) Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In: EMNLP 3911–3921. Association for Computational Linguistics
- Zhang X, Yin F, Ma G, Ge B, Xiao W (2020) F-SQL: fuse table schema and table content for single-table text2sql generation. *IEEE Access* 8:136409–136420
- Zhong V, Xiong C, Socher R (2017) Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR arXiv:1709.00103*

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.