Bootstrapping a Natural Language Interface to a Cyber Security Event Collection System using a Hybrid Translation Approach

Johann Roturier*, Brian Schlatter, and David Silva***
* Symantec Research Labs, Dublin, Ireland
** Symantec, Culver City, CA
{johann_roturier,brian_schlatter,david_silva}@symantec.com

Abstract

This paper presents a system that can be used to generate Elasticsearch (database) query strings for English-speaking cyber-threat hunters, security analysts or responders (agents) using a natural language interface. This system relies on a hybrid translation approach combining translation memory, information extraction and text classification techniques. The resulting queries may be used to (i) speed up the on-boarding of agents that are not (too) familiar with a specific, flexible database schema and (ii) collect question-to-query mappings with a view to train future models using a more robust framework (e.g. NMT). The system presented in this paper supports multiple data sources, including an industry-standard knowledge base and collections of existing queries provided by individual or corporate threat hunters. It allows users to ask questions about specific cybersecurity event or incident details and generates Elasticsearch query strings that can be executed against a database containing security event data. This paper presents the key components of the backend system and highlights some of the user interface design choices that were made to maximize user adoption.

1 Introduction

Studying the translation of natural language into SQL queries has a very long history. Earlier work focused on specific databases thus requiring further customization to generalize to each new database (Warren and Pereira, 1982) (Giordani and Moschitti, 2012) (Tamas and Salomie, 2016) (Wang et al., 2017). More recent work has explored the use of deep neural networks (Zhong et al., 2017) especially for Wiki-based information retrieval (Yu et al., 2018). Related areas have also been studied, such as natural language interfaces to Web APIs (Su et al., 2017), (Su et al., 2018), and dialogue-based query generation (Gur et al., 2018). However, little attention has been devoted to document-oriented databases with flexible schemas (e.g. Elasticsearch) that are often used as Security Information and Event Management (SIEM) systems. Such databases are commonly used in the context of cyber-threat hunting (or discovery) as shown for instance by the availability of the HELK stack (Rodriguez, 2019). When hunting for advanced threats, domain experts typically have to craft complex queries (Kindlund, 2018). Very often, however, little training data is available apart from API/schema descriptions and raw events so using a fully data-driven approach is often not practical. Using interactive learning in a dialogue-based scenario may alleviate this issue (Filar et al., 2017) (Filar and Seymour, 2019), but it is not clear that advanced users will have the patience to answer a long series of questions (e.g. *which user do you mean?*, *what devices are you talking about?*) that are often required to fill slots.

In the present paper, we therefore present an alternative approach to tackle the problem of bootstrapping a natural language-based system for threat hunting or security incident investigation. This hybrid approach is based on multiple techniques, including semantic search techniques (Mangold, 2007), whereby the Source Lan-

guage user input (English) is analyzed using a number of components (mostly based on rules) in order to extract entities and phrases or infer text categories that are then mapped into the Target Language's fields and terms before an actual Elasticsearch query string can be generated (ElasticsearchB.V., 2019). Using a combination of text classification and slot filling has indeed been shown to provide very competitive natural language to SQL baseline systems (Finegan-Dollak et al., 2018). Even though some of these rules have to be manually created, their precision is well suited to this task, especially as far the generation step is concerned to ensure that users are provided with valid queries. The rest of the paper is organized as follows: Section 2 describes the various data sources used to create some of the linguistic components of our backend system (either manually or automatically). Section 3 presents the actual system components with some examples. The two final sections provide some discussion of the design choices that were made to create the user interface and outline some directions for future work.

2 Data sources

In this section the main data sources used to bootstrap our system are described. The data sources include:

- An API schema file from which field descriptions, values and value mappings can be extracted for the *Event* object of an actual endpoint detection and response (EDR) system. This file required custom parsing in order to extract enumerated (and potentially mapped) values from free-text descriptions.[1]

- Raw event data from an actual EDR system that is used in production (about 1000 samples). These samples were used to automatically create linguistically-oriented entity recognition rules.

- 49 manually curated pairs of queries and descriptions pertaining to the underlying EDR system. This set was extracted by parsing a PDF file containing threat discovery guidance.[2]

- The enterprise matrix from MITRE's cyber-threat intelligence (CTI) dataset, which is an industry-standard knowledge base providing some information for each of the 223 techniques defined in the ATT&CK model (Mitre, 2019). A file in STIX format is parsed in order to extract relevant information.[3]

- 285 manually curated pairs of queries and descriptions originating from the Sigma project (Roth and Patzke, 2019). This set was extracted by parsing YAML files.

- 230 manually curated pairs of queries and descriptions originating from the Lolbas project (LOLBAS, 2019). This set was extracted by parsing YAML files.

- A subset of the annotations from 39 Advanced Persistent Threat (APT) reports (containing 6,819 sentences) with attribute labels from the Malware Attribute Enumeration and Characterization (MAEC) vocabulary (Lim et al., 2017).[4]

2.1 The API Schema dataset

In order to define the domain covered by our system, we rely primarily on an API schema file based on the OpenAPI specification (OpenAPI, 2019). This file, which is available in JSON format, contains a full-fledged description of multiple methods and objects pertaining to an actual endpoint detection and response (EDR) system. Some of these methods are very narrow in scope (e.g. how to contain an endpoint) and can be covered by simple OpenC2 commands (OASIS, 2019). Others, however, can be much more complex (e.g. querying a database system to find specific security events based on several search criteria). Examples of fields pertaining to the *Event* object include (i) an integer representing a port number, (ii) a specific type ID, whose integer value maps to a textual description (e.g. *8000* for a session event), (iii) a host name string for the client computer, (iv) a MITRE tactic string corresponding to one of the 11 tactics defined by the MITRE's ATT&CK model (Mitre, 2019), or (v) an overloaded integer value mapping to multiple descriptions depending on the value of another field. The overall number of fields for the

[1] https://help.symantec.com/bucket/SymantecEDR_4.0/lists_of_all_symantec_edr_event_schemas

[2] https://support.symantec.com/us/en/article.doc11273.html

[3] https://oasis-open.github.io/cti-documentation/stix/intro

[4] https://github.com/MAECProject/schemas/blob/master/vocabs.json

Event object is very large (more than 500) but the actual number of fields will vary depending on the type of event (e.g. a file reputation request event or a session event). Such events tend to include a few dozens fields so the goal of this system is to cover those that appear the most frequently in the data (raw events).

2.2 The threat discovery guide dataset

While the previous section focused on specific fields and associated values, threat hunters often need to craft more advanced queries whose textual mappings do not include specific field values or descriptions. For instance, a question such as *show me the outbound traffic occurring on non-standard ports* has to be matched with:

```
type_id:8007 AND
-target_ip:["192.168.0.0/16" OR
"10.0.0.0/8" OR "172.16.0.0/12"
OR "127.0.0.0/8"] AND
-target_port:[80 OR 443].
```

In this example, the phrase *non-standard port* cannot be found in the API Schema but obviously a domain expert is able to associate it with all ports apart from *80* and *443* in the context of HTTP traffic. The same applies to the phrase *outbound traffic*.

Since the number of descriptions/queries pairs is quite small in this dataset (less than 50), these complex mappings cannot be learnt using a data-driven approach. However, we can try and detect some of these phrases in the user input and either (i) apply a translation memory technique to retrieve partial (or fuzzy) matches or (ii) offer suggestions via the user interface. These two approaches will be described in sections 3.1 and 4 respectively.

2.3 The MITRE CTI dataset

The MITRE cyber-threat intelligence (CTI) dataset provides some textual information for each of the 223 techniques defined in the ATT&CK model. We leverage this dataset in two ways: First, we use the technique names to detect their mentions as entities in user input (as described in Section 3.2). Second, we use additional information in order to train a multi-class text categorizer. This information includes a brief description of the techniques as well as guidance on how to detect and mitigate against such attacks. Some examples of malware or attack group using this technique may also be provided. All of this information can be parsed and split into sentences. The initial data set, referred to as *mitre-6.5k*, is quite small (6500 sentences) and not very balanced as most techniques (140/223) (classes) contain less than 23 sentences. In order to deal with this data shortage issue, we can supplement the training data with sentences extracted from external references (e.g. web pages, PDF documents) cited in the MITRE pages. This strategy allows us to increase the size of the training data to about 60K sentences (referred to as *mitre-60k*). This text classifier will be presented in Section 3.5.

2.4 The Sigma and Lolbas datasets

While the previous two datasets are directly associated with the EDR system of interest, other datasets could be relevant to users of that system. For instance, the Sigma project contains a number of detection rules that may be used to query security information and event management system logs. Some experienced cyber-security professionals may actually be more familiar with these rules (and their default fields) than the schema presented in section 2.1. Similarly, the Lolbas project makes a number of detection rules available to identify anomalous or suspicious usage of legitimate system or administration tools. Such rules are associated with a description and/or a title whose words could appear in user input. These rules from these datasets are therefore parsed by our system and made available to users via interactive suggestions and partial matching. One of the challenges with these datasets is that the queries or commands they expose are not fully compatible with our target Elasticsearch index. For instance, the following Lolbas command includes a *Command* field that must be mapped to a *process.cmd_line* field:

```
Command: rundll32.exe
shdocvw.dll,OpenURL
"C:\test\calc.url"
```

This example also includes a test parameter `"C:\test\calc.url"` that would have to be replaced with a proper value in a successful query. In order to deal with this problem, this test parameter can be easily filtered out by parsing the command in order to generate the following query:

```
process.cmd_line:["rundll32.exe"
AND "shdocvw.dll,OpenURL"]
```

Interestingly, both of these data sources also include references to MITRE technique IDs in their rules. This is useful for two reasons. First, when

the query includes fields that cannot easily be mapped (e.g. *Event ID 13: RegistryEvent (Value Set)* generated by Sysmon (Russinovich and Garnier, 2019)), the MITRE technique ID may be suggested to the user who would then be able to filter out the results interactively. Second, we can use 420 labelled sentences from these datasets (referred to *test-set*) to evaluate our system's text classification component trained on the actual MITRE data.

3 System Overview

Our system is implemented on top of a Spacy pipeline (Honnibal and Montani, 2017) using a number of default and custom components. Specifically, we rely on Spacy's default tokenizer, tagger, named entity recognizer and dependency parser for English.[5] We also designed a number of custom components to address some of the issues described in the previous section. These six components, which are shown in Figure 1, are described in the next sections.

3.1 Index-based Phrase Matcher

This component relies on the descriptions/queries pairs described in the previous section. The descriptions are first lemmatized, lower-cased and split into n-grams (with a minimum length of 3). These n-grams are then stored as the keys of an inverted index in order to point to relevant queries (and associated metadata, such as description, source, etc.). When user input is submitted, this index is used to find an exact match or partial match, by computing a similarity score (between 0 an 1) using the n-gram overlap technique for text reuse described by Clough et al. (2002). When an exact match is found, the other steps from the pipeline may be skipped.

3.2 Named Entity Recognition

When a partial match or no match is found, the user input is processed by the modules performing named entity recognition. Default entity types such as *ORG* or *GPE* are considered as they proved relevant on the sample of raw events. For instance, some fields contain country or organization names when the event has been enriched with a domain's WHOIS information or when it includes file-based digital certificate information. The *PERSON* entity also proved relevant for fields containing user names such as email addresses or host names. In other cases, however, custom entity types have to be defined. Since we wanted to automate this process as much as possible, we performed some analysis on the raw event samples to find some fields whose values contained specific word shapes or word lemmas. For instance, we found that device names followed a specific pattern that could be learnt from specific word shapes. Since device naming will vary from one organization to the next, using this approach provides a lot of flexibility. This approach also allows to support variants or synonyms, for instance to handle adversary groups that tend to be named in multiple ways in the industry. Finally, when ambiguity is present (i.e. when a value occurred across multiple fields, such as "suspicious") or when the pattern seems too complex to learn automatically, we rely on the rules-based phrase matcher which is presented next.

3.3 Rules-based Phrase Matcher

Since this component requires the creation of manual rules to annotate specific token sequences from the input text, it is reserved to a small set of significant fields that may rely on ambiguous values. For instance, the *operation* field can refer to a creation event that may take a different value depending on the context (e.g. deleting a registry key value or deleting a file). Using a high-precision rule to match such token sequences is well suited to tackle this problem. Besides, it allows us to quickly add variants (e.g. synonyms) to extend the rule's coverage. In order to speed up the identification of variants for specific verbs and nouns, we rely on the annotations from the MalwareTextDB dataset (Lim et al., 2017) for those annotations whose MAEC attribute label overlaps with some of the Event field descriptions (e.g. "delete file"). This strategy allows to recover a number of variants, such as "delete", "clean", "wipe", "remove", or "destroy" for "delete file", or "connect", "communicate", "establish", "initiate" for "send network packet".

3.4 Entity Relation Extractor

This component relies on additional manually created rules that cannot easily be expressed using the formalism of the rules-based phrase matcher. This module makes use of dependency parsing in-

[5] In our experiments, we used their *small* model trained on OntoNotes5.

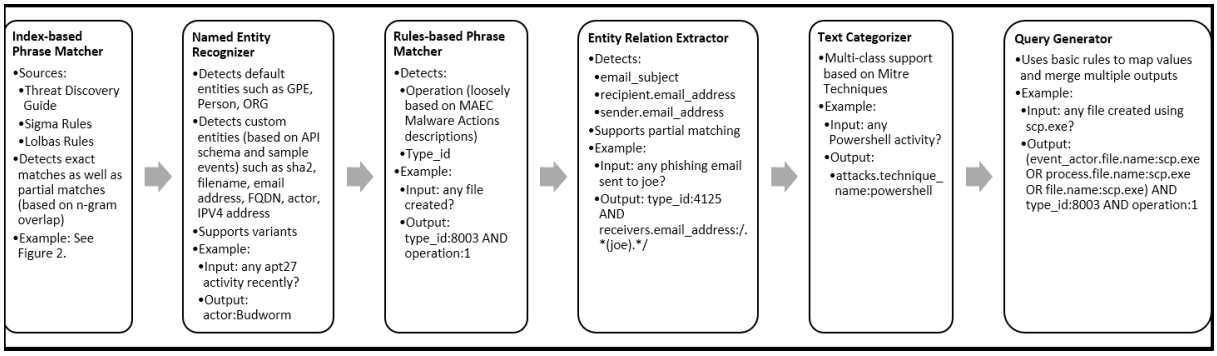

Figure 1: Custom components included in the pipeline

formation to target specific event types. Specifically, email-related events (including phishing and spear-phishing) tend to involve a number of entities (i.e. sender, recipients(s)) and properties (e.g. subject, attachment, message status) that can be very challenging to handle even when substantial training data is available. For instance, Su et al. (2017) report 57% accuracy for generating valid *GET-Messages* API calls using Microsoft's email search API when these calls contain 1 to 4 parameters. One of the rules used by our component navigates the parse tree of the user input and looks for subtrees starting with prepositions such as *to* (for the email recipient), *from* or *by* (for the sender), *about* (for the subject), and *with* (for the attachment). The content of the subtree is then analyzed to determine whether they contain specific entities (such as email addresses, person names or filenames). When such entities are found, they can be associated with the actual field names.

3.5 Text Classification

When no entities or matches are found in the user input, we rely on a fall-back component. This component is completely different from the previous ones as it does not rely on rules. Instead, we train a multi-class (223) categorization model (a stacked ensemble of a bag-of-words model and a convolutional neural network model) using the MITRE technique training data sets described in Section 2. Even if the performance of this bootstrapped model is poor (16% on *test-set* when trained on *mitre-6.5k* and 24% accuracy when trained on *mitre-60k*) it allows for the initial labelling of user input with a MITRE technique ID (i.e. when the class probability is greater than 0.5) even when the actual MITRE technique name is not explicitly present. The performance of the model is expected to improve as usage data is collected.

3.6 Query Generator

Once all of the analysis components have been executed on the user input, their output is combined in order to generate an Elasticsearch query string following the syntactic constraints of this mini language. Some field/value combinations are much easier to handle than others (e.g. actor:actor_name). In some cases, however, a mapping is required to restore the expected value. When multiple values are present for a given field, these values are joined together with an *OR* operator in a list. All fields/terms pairs are currently joined with the *AND* operator, unless field ambiguity must be handled (e.g. an *IP address* entity may refer to multiple fields so the various options must be present in the query). Finally, we have introduced some basic support for regular expression patterns when specific entities are found instead of others. As mentioned in Section 3.4 a person name may be used to search for specific email events. Since an email address is expected for this field, the query will not return the expected results unless the person name (e.g. `John`) is turned into a pattern such as `/John.*/`).

4 User Interface

When designing the user interface, four main requirements were taken into account:

1. The interface should be familiar to users who may have been relying on *conversion* systems to convert their queries from one SIEM system to another. Tools such as *Uncoder*[6] or *Sigma UI*[7] allow users to write, save and convert queries in a number of formats, but do not

[6] `https://uncoder.io/`
[7] `https://github.com/socprime/SigmaUI`

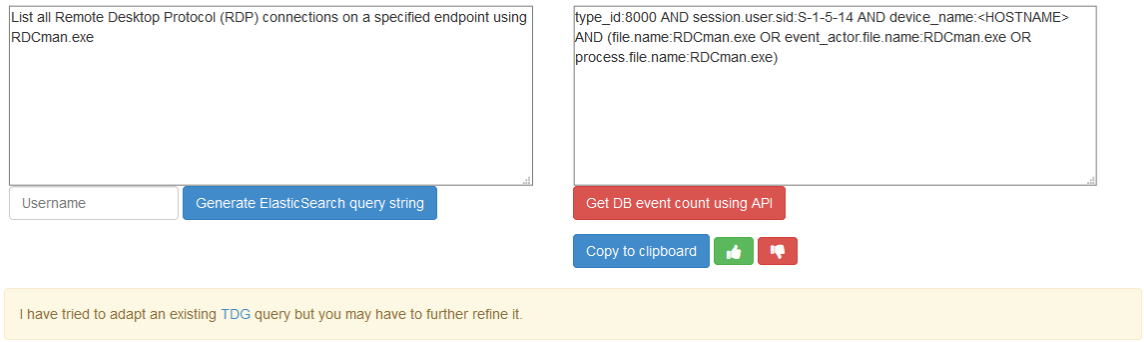

Figure 2: Web-based standalone user interface

allow users to use natural language to generate a new query. Using a 2-column format where (source) user input and (target) generated query are displayed side by side in the middle of the screen is a well-known layout for (machine) translation or conversion applications. Using this layout prevents usability issues that are typically associated with chat widgets as they only use a fraction of the screen.

2. The interface should be as intuitive as possible, so the number of UI elements should be limited to the following functions: (i) generating a query after entering some text (e.g. question), (ii) executing the query against the DB (possibly via an API call, (iii) copying the query to the clipboard, (iv) indicating whether the query was useful (or not), and (v) giving the user a chance to access the source of a leveraged rule via a hyperlink.

3. To minimize user frustration due to misinterpretation, question suggestions should be made available as a drop-down selection once the user has typed at least one word. These suggestions are based on those question/query pairs described in Section 2 that have not been rated negatively by users.

4. The interface should be able to be deployed as a standalone application or as a widget within an existing Web application (e.g. a Kibana instance).[8] In the latter scenario, a bookmarklet can be made available to users and some contextual application can be leveraged to influence the query generation step. For instance, when a user name or incident ID is detected on the page of the host application, this information can be passed to the query generator component to (i) skip queries that may have been rated as poor by a given user or (ii) disambiguate some entities.

These four main requirements led to the creation of a simple interface whose standalone version is shown in Figure 2. Since this interface can be easily integrated within existing applications, it allows for a seamless collection of question/query/rating triplets that we plan to make use of in the future as explained in the next section.

5 Future work

One of the next steps is to make the system available to real users and study how they would benefit from using such system. Once the system is deployed, we would also like to investigate the feasibility of using an NMT framework to leverage the user feedback. Specifically, we would like to further improve training data generation and create robust sequence to sequence models so that both novice and expert users can perform their work in a proficient manner. Also, we would like to support additional input languages, including the possibility to generate natural language descriptions of queries entered by expert users. Additional future work also includes a better handling of specific query conditions (e.g. NOT) that are not currently covered by our rules, thus requiring additional user input.

[8] https://www.elastic.co/products/kibana

References

Clough, Paul, Robert Gaizauskas, Scott S. L. Piao, and Yorick Wilks. 2002. Meter: Measuring text reuse. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 152–159, Stroudsburg, PA, USA. Association for Computational Linguistics.

ElasticsearchB.V. 2019. The elasticsearch query string mini-language. https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html.

Filar, Bobby and Rich Seymour. 2019. Going deep with artemis 3.0. https://www.endgame.com/blog/technical-blog/going-deep-artemis-30. Last accessed: 2019-04-15.

Filar, Bobby, Richard Seymour, and Matthew Park. 2017. Ask me anything: A conversational interface to augment information security workers. *CoRR*, abs/1707.05768.

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

Giordani, Alessandra and Alessandro Moschitti. 2012. Translating questions to SQL queries with generative parsers discriminatively reranked. In *Proceedings of COLING 2012: Posters*, pages 401–410, Mumbai, India, December. The COLING 2012 Organizing Committee.

Gur, Izzeddin, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. Dialsql: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1339–1349, Melbourne, Australia, July. Association for Computational Linguistics.

Honnibal, Matthew and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. https://github.com/explosion/spaCy.

Kindlund, Darien. 2018. Using natural language searches for fast incident response. https://insightengines.com/blog/using-natural-language-searches-fast-incident-response/. Last accessed: 2019-04-15.

Lim, Swee Kiat, Aldrian Obaja Muis, Wei Lu, and Chen Hui Ong. 2017. Malwaretextdb: A database for annotated malware articles. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1567, Vancouver, Canada, July. Association for Computational Linguistics.

LOLBAS. 2019. Living off the land binaries and scripts. https://github.com/LOLBAS-Project/LOLBAS.

Mangold, Christoph. 2007. A survey and classification of semantic search approaches. *Int. J. Metadata Semant. Ontologies*, 2(1):23–34, September.

Mitre. 2019. Cyber threat intelligence repository expressed in stix 2.0. https://github.com/mitre/cti.

OASIS. 2019. The open command and control (openc2) language specification. http://docs.oasis-open.org/openc2/oc2ls/v1.0/oc2ls-v1.0.html.

OpenAPI. 2019. The openapi specification. https://github.com/OAI/OpenAPI-Specification.

Rodriguez, Roberto. 2019. The hunting elk. https://github.com/Cyb3rWard0g/HELK.

Roth, Florian and Thomas Patzke. 2019. Sigma: Generic signature format for siem systems. https://github.com/Neo23x0/sigma.

Russinovich, Mark and Thomas Garnier. 2019. Sysmon v9.0. https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon.

Su, Yu, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. 2017. Building natural language interfaces to web apis. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 177–186, New York, NY, USA. ACM.

Su, Yu, Ahmed Hassan Awadallah, Miaosen Wang, and Ryen W. White. 2018. Natural language interfaces with fine-grained user interaction: A case study on web apis. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, pages 855–864, New York, NY, USA. ACM.

Tamas, Ionut and Ioan Salomie. 2016. Artemis - an extensible natural language framework for data querying and manipulation. In *IEEE 12th International Conference on Intelligent Computer Communication and Processing, ICCP 2016, Cluj-Napoca, Romania, September 8-10, 2016*, pages 85–91.

Wang, Chenglong, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive sql queries from input-output examples. *SIGPLAN Not.*, 52(6):452–466, June.

Warren, David H.D. and Fernando C.N. Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

Yu, Tao, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *CoRR*, abs/1809.08887.

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