Beyond Text-to-SQL for IoT Defense: A Comprehensive Framework for Querying and Classifying IoT Threats

Ryan Pavlich[1], **Nima Ebadi**[2], **Richard Tarbell**[1], **Billy Linares**[1], **Adrian Tan**[1], **Rachael Humphreys**[1], **Jayanta Kumar Das**[1], **Rambod Ghandiparsi**[1], **Hannah Haley**[1], **Jerris George**[1], [3]**Rocky Slavin**, [4]**Kim-Kwang Raymond Choo**, [4]**Glenn Dietrich**, and [4]**Anthony Rios**

[1]Data Analytics, [2]Department of Electrical and Computer Engineering,
[3]Department of Computer Science, [4]Department of Information Systems and Cyber Security
The University of Texas at San Antonio
{Ryan.Palvich, Anthony.Rios}@utsa.edu

Abstract

Recognizing the promise of natural language interfaces to databases, prior studies have emphasized the development of text-to-SQL systems. Existing research has generally focused on generating SQL statements from text queries, and the broader challenge lies in inferring new information about the returned data. Our research makes two major contributions to address this gap. First, we introduce a novel Internet-of-Things (IoT) text-to-SQL dataset comprising 10,985 text-SQL pairs and 239,398 rows of network traffic activity. The dataset contains additional query types limited in prior text-to-SQL datasets, notably, temporal-related queries. Our dataset is sourced from a smart building's IoT ecosystem exploring sensor read and network traffic data. Second, our dataset allows two-stage processing, where the returned data (network traffic) from a generated SQL can be categorized as malicious or not. Our results show that joint training to query and infer information about the data improves overall text-to-SQL performance, nearly matching that of substantially larger models. We also show that current large language models (e.g., GPT3.5) struggle to infer new information about returned data (i.e., they are bad at tabular data understanding), thus our dataset provides a novel test bed for integrating complex domain-specific reasoning into LLMs.

1 Introduction

Relational databases contain vast quantities of structured knowledge, often having trillions of rows of data, spanning diverse domains from healthcare and finance to entertainment and education. While structured query languages (SQL) provide database experts the resources to extract, manipulate, and reason over this data, many potential users remain cut off from direct access due to the steep learning curve of mastering these languages. The importance of making data more accessible and actionable for a wider audience cannot be overstated, given the growing centrality of data-driven decision-making in modern society. The vision of natural language interfaces to databases (NLIDB) is rooted in this very imperative—to allow non-experts to interact with databases using familiar, everyday language. This reinforces the importance of developing modern text-to-SQL systems that can also reason over databases.

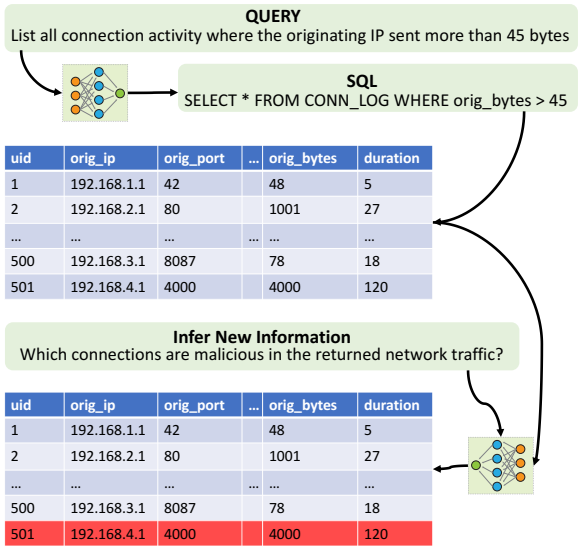

Figure 1: This figure provides an example of querying and reasoning over network traffic data.

A system that seamlessly translates natural language queries into SQL (text-to-SQL) not only democratizes access to data but also has the potential to drastically reduce the time to insights for diverse stakeholders, including managers, analysts, educators, and the general public. There have been many advances in translating natural language to SQL (Xu et al., 2017; Zhong et al., 2017; Bogin et al., 2019; Wang et al., 2018; Yu et al., 2018a; Scholak et al., 2021; Xie et al., 2022; Wang et al., 2022a; Chen et al., 2021; Sun et al., 2022). Recent work has focused on either fine-tuning transformers or on the use of pre-built large language

Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025), pages 1–12

models (e.g., ChatGPT) with prompt tuning and in-context examples. For example, Pourreza and Rafiei (2023) explored in-context learning using ChatGPT to generate SQL statements, and Dong et al. (2023) explored zero-shot text-to-SQL generation using ChatGPT. Wang et al. (2020a) developed a unified framework using fine-tuning for text-to-SQL generation, leveraging relation-aware self-attention, to tackle schema encoding, schema linking, and feature representation. Combined with BERT data augmentation, this framework yielded a remarkable exact match accuracy of 65.6% on the Spider dataset.

Much of the prior work on text-to-SQL generation has focused on simply generating SQL statements from the input text queries. Some recent work has expanded on standard studies by exploring conversational text-to-SQL tasks (Yu et al., 2019). Intuitively, Yu et al. (2019) developed a system that can ask follow-up questions to answer ambiguous queries better, verify returned results, and notify users of unanswerable queries. However, there is limited work that can query a database and make inferences (understand) the returned data. Follow-up questions may involve making inferences and returning results that are not directly within the database. Hence, translating natural language to SQL is only half the challenge. The true power of such a system lies in its ability to retrieve and infer new information about the data returned. This ensures that the insights drawn from databases are accurate and meaningful. For instance, in an educational context, a student might not only ask for the number of historical events in a given time but might also want to know their significance or interconnections, requiring a depth of reasoning beyond retrieval.

At a high level, our work combines two lines of research not explored in previous papers: tabular data classification and question answering using transformers (Badaro et al., 2023) and text-to-SQL generation. There has been some recent work about predicting various aspects of tabular data. For example, Yang and Zhu (2021) predicts whether a claim is true or false given an input table. Likewise, Deng et al. (2022) developed a system to inform missing or corrupted data within a table. However, much of this work assumes the table is provided. Hence, we develop a new text-to-SQL dataset to make predictions/inferences about the data and query the data using a single model. An example of our task is provided in Figure 1. As a case study, our dataset consists of Internet-of-Things (IoT) data from a smart building setting. Specifically, we assume a centralized database that captures both network traffic about the IoT devices and sensor readings (temperature, humidity, CO2 levels, etc.). The SQL statements query the IoT databases to return relevant data. The reasoning component of our dataset is specific to the network data. We classify the network traffic as malicious (e.g., DDoS attacks, botnet activity, etc.) or benign (non-malicious activity). Our decision to use IoT data is due to the following reasons. First, IoT data has a huge temporal component (Acar et al., 2020). There have been limited text-to-dataset resources that contain many temporal-related queries (e.g., Spider is based on SQLite databases and does not support datetime columns). Second, making inferences about network traffic data is non-trivial and has not been explored in the NLP community.

In summary, the contributions of this paper are as follows: **(i)** We introduce a new IoT-SQL dataset containing 10,985 unique text-SQL pairs and 239,398 rows of network traffic activity from Zeek logs with annotations for malicious and non–malicious activity (e.g., DDoS attacks). This dataset provides a new test bed for text-to-SQL models and LLMs towards both querying data an actually understanding it. Specifically, current state-of-the-art LLMs GPT3.5 fail to perform well on this dataset for the reasoning component.[1] **(ii)** We evaluate the performance of text-to-SQL models that can jointly query and reason about the data (i.e., predict whether specific network traffic is malicious). Our results suggest that modeling both tasks together substantially improves text-to-SQL performance with limited impact on network-traffic malicious activity detection. **(iii)** We perform error analysis and provide examples of how jointly training to query and understand the data improved SQL generation.

2 Related Work

Text-to-SQL Datasets. Recent momentum has grown in evaluating text-to-SQL systems, especially their generalizability, with less focus on the medical domain. Text-to-SQL translates text into machine-readable formats. Several datasets exist for this task: ATIS (Dahl et al., 1994; Srinivasan Iyer and Zettlemoyer, 2017) (airline queries), Geography (Zelle and Mooney, 1996;

[1]Dataset: https://zenodo.org/records/15000588.

Srinivasan Iyer and Zettlemoyer, 2017) (geographical data), Restaurants (Giordani and Moschitti, 2013; Tang and Mooney, 2000; Popescu et al., 2003) (restaurant details), WikiSQL (Zhong et al., 2017), Spider (Yu et al., 2018b), and IMDB and Yelp (Navid Yaghmazadeh and Dillig, 2017) (movie and business data). The Spider dataset emerges as a cornerstone resource in the text-to-SQL benchmarks landscape. Designed to evaluate text-to-SQL systems rigorously, Spider boasts impressive extensiveness and diversity, featuring over 10,000 questions from over 200 databases. Its strength lies in its volume and the complexity of its queries.

Recent efforts have also been made to develop new datasets beyond traditional text-to-SQL pairs. Yu et al. (2019), for example, collected a conversation-like corpus where a system can ask follow-up questions to answer ambiguous queries better, verify returned results, and notify users of unanswerable queries. Similarly, researchers have also focused on curating data (text-SQL pairs) that capture items missing in previous datasets (e.g., temporal-related queries). For example, Vo et al. (2022) introduced a new dataset called TempQ4NLIDB that contains 389 temporal-related question-SQL pairs to overcome limitations in existing datasets (e.g., Spider). Our research expands on this work, containing more than 1,000 temporally-related queries using MySQL datetime columns.

Text-to-SQL Methods. The field of text-to-SQL is concerned with automatically translating natural language queries into structured SQL queries. Recent advancements in neural network models have led to significant improvements in the accuracy and efficiency of Text-to-SQL systems (Xu et al., 2017; Zhong et al., 2017; Bogin et al., 2019; Wang et al., 2018; Yu et al., 2018a; Scholak et al., 2021; Xie et al., 2022; Wang et al., 2022a; Chen et al., 2021; Sun et al., 2022).

Recent work has focused on fine-tuning transformers or using pre-built large language models (e.g., ChatGPT) with prompt tuning and in-context examples. For example, Pourreza and Rafiei (2023) explored in-context learning using ChatGPT to generate SQL statements, and Dong et al. (2023) explored zero-shot text-to-SQL generation using ChatGPT. Wang et al. (2020a) also proposed a relation-aware self-attention mechanism for text-to-SQL generation, achieving an accuracy of 65.6% on the Spider dataset when combined with BERT (Wang et al., 2020a). In another independent work, Scholak et al. (2021) introduced the PICARD method, which uses incremental parsing for fine-tuning formal languages. This led to state-of-the-art results on both the Spider and CoSQL datasets. Wang et al. (2022a) introduced a novel approach to schema linking using the Poincaré distance metric. Their results established a new benchmark in performance, outperforming rule-based methods across multiple datasets and showcasing the effectiveness of their probing method. A more recent thorough analysis of the Codex language model's text-to-SQL abilities was undertaken by Rajkumar et al. (2022), whose findings highlighted the model's competitive performance across benchmarks, even without finetuning. Particularly on the Spider benchmark, Codex achieved an accuracy of up to 67%. Their work also indicated that using a small set of in-domain examples could boost Codex's performance beyond some finetuned state-of-the-art models.

Tabular Data Understanding. There has been a wide array of papers about understanding tabular data beyond text-to-SQL (Badaro et al., 2023). According to Badaro et al. (2023), there are six common tabular data tasks: Fact-checking, question answering, semantic parsing (i.e., text-to-SQL), table retrieval, table metadata prediction, and table content population. Fact-checking related work has generally focused on predicting whether a statement/claim is factual, given the knowledge available in a Table (Yang and Zhu, 2021). Table retrieval research has focused on finding a table that contains the answer to a particular question (Wang et al., 2022b, 2021). Table metadata prediction involves predicting information about the table, such as the column name or a relation between two columns (Suhara et al., 2022; Du et al., 2021). Finally, table content population involves filling the cells within a table because of missing or incorrect data (Iida et al., 2021; Tang et al., 2021).

Intuitively, our task can be considered a combination of semantic parsing and table content population. The former (semantic parsing) is the text-to-SQL task, and the table population we are predicting is malicious or benign information for network traffic. We can think of the malicious information as a missing column in the database. But, more importantly, this is a highly specialized task that large language models cannot easily reason about.

	Train	Dev	Test
# Examples	6591	2197	2197
Average Question Length	2.3	2.3	2.5
Min Question Length	5	6	6
Max Question Length	63	53	46
Average SQL Length	16.3	16.5	16.4
Min SQL Length	5	5	5
Max SQL Length	146	140	140
# Tables		12	
# Columns		173	

Table 1: Basic overview of the the text-to-SQL data.

	Train	Dev	Test
# Examples	125,000	57,199	57,199
# Malicious Examples	50,000	19,701	19,697
# Features	19	19	19

Table 2: Basic overview of the network traffic data used to train and evaluate malicious traffic.

Hence, our dataset provides a unique research test bed for integrating highly specialized knowledge into LLMs for tabular QA.

3 Data

In this section, we describe the data creation process for text-SQL pairs, the source of the network traffic and sensor data, and how the network traffic data was organized for training our malicious network traffic activity detection model. As shown in Figure 2, the data curation pipeline comprises five major steps. First, we curate the data for the database. Second, we "annotate" text-SQL pairs. Third, we partition network traffic data from the database to be used to train and evaluate a malicious traffic detector. Fourth, we review the text-SQL pairs, removing incorrect, irrelevant, or unclear queries. Moreover, we paraphrase each text-SQL pair to provide diversity in how things are specified. Finally, we perform an additional round of review after the paraphrase process.

3.1 Database Collection and Creation

We curate the data for our IoT database from two sources: IoT-23 (Garcia et al., 2021) and the Smart Building Sensor Data (Hong et al., 2017).

IoT-23. The IoT-23 dataset is created to facilitate the development and validation of intrusion detection systems (IDS) for IoT devices. It contains benign and malicious network traffic recordings. The network traffic recorders are stored in PCAP files and Zeek logs. For this study, we focus on the Zeek logs. Zeek (Paxson, 1999), formerly known as Bro, is an open-source network security monitoring tool. Its primary purpose is to analyze network traffic and generate high-level logs, metrics, and events that abstract the raw data into more meaningful and actionable insights. Zeek is widely used in network security, monitoring, and forensic analysis. There are conn.log, dns.log, files.log, http.log, npt.log, and weird.log. The conn.log records connection-level information detailing the sessions seen on the network. A list of the columns in the conn.log is found in Table 3. Each row in the conn.log is annotated with malicious or benign and the type of malicious activity (e.g., DDoS, command and control, specific malware, and more). We discuss this more in the Network Traffic subsection. dns.log contains DNS request and response data. files.log stores details about files transferred over supported protocols, such as HTTP or FTP. http.log captures detailed HTTP request and response information. ntp.log contains information related to NTP transactions, such as timestamp updates, server-client interactions, version details, and other attributes specific to NTP communications. Finally, weird.log logs anomalies or unusual behaviors in network traffic. Each dataset is processed and stored as an independent table in the database.[2]

Smart Building Sensors. The Smart Building Sensor Data is a dataset derived from 255 sensors strategically deployed across 51 distinct rooms spanning four floors of a university building. The dataset contains humidity, CO2, temperature, luminosity, and motion sensor readings. Each reading is related to a specific room in the building. This dataset presents a unique opportunity for empirically exploring patterns associated with indoor spaces' physical attributes, particularly when combined with network traffic in a synthetic building-level database. Each sensor type (humidity, luminosity, etc.) is stored as a unique Table in our database, where each row represents a sensor read. Intuitively, the goal is to have a comprehensive database that may be used in a smart building setting, containing both the raw sensor information and meta data (network traffic) for smart devices.

3.2 Text-to-SQL Pair Annotation

The SQL queries were created using two major approaches: programmatically using a templated

[2]More details on Zeek logs can be found at docs.zeek.org/

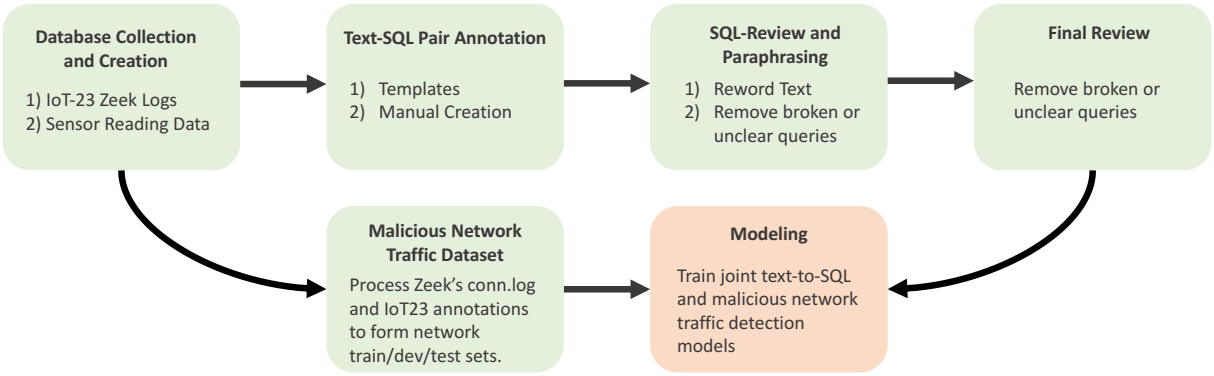

Figure 2: Text-to-SQL and malicious network traffic data collection pipeline overview.

approach similar to Wang et al. (2020b) and manually creating text-SQL pairs without templates. We describe each of these approaches in detail below:

Templates. Following the work by Wang et al. (2020b), we generate templates that fit two categories: retrieval queries and reasoning queries. Retrieval queries are primarily meant to extract specific records or data from the database. Reasoning queries are more complex and often involve several logical operations and conditions. They often require the model to comprehend intricate relations between different parts of the question or between multiple database tables. The distinction is helpful because different query types can be challenging in their ways. Retrieval queries test the model's ability to correctly identify and fetch data, while reasoning queries test its ability to process and integrate multiple pieces of information.

In total, we created 27 templates containing simple and complex queries. Templates are generated to create queries containing JOINs, HAVING statements, aggregation operations (e.g., average), and nested queries. An example template is

```
SELECT $AGG_OP ($AGG_COLUMN)+
 FROM $TABLE WHERE ($COND_COLUMN
 $COND_OP $COND_VALUE)+
```

In the above expression, `$AGG_OP` represents aggregation methods (e.g., AVG(), MAX(), and MIN()), `$AGG_COLUMN` represents the column to perform the aggregation on (e.g., "duration" from conn.log), `$TABLE` represents the table the column is pulled from, `$COND_COLUMN` (e.g., orig_h representing the IP address), `$COND` represents a conditional operator (e.g., $>$, $<$, $=$), and `$COND_VALUE` represents the value to check (e.g., 192.168.1.1). An example query generated from the template is

```
SELECT AVG(duration)
 FROM CONN_LOG WHERE (orig_h
 = "192.168.1.1")
```

where items such as `$AGG_OP` are replaced with `AVG()`.

After creating the text-SQL pairs using templates, we paraphrased (reworded) each text piece to add diversity in the ways each question type is asked. Six researchers manually paraphrased each question. For instance, the automatically generated sentence, "*List the distinct proto for the DNS LOGs table with TTLs equal to 2523*" would be transformed into "*Provide a list of unique DNS proto values with a TTLs value of 2523*", where the sentence is now more natural. All students had expertise in databases and were data analytics majors. The text-SQL pairs were assigned randomly to each researcher. In total, we create a total of 10,000 text-SQL pairs using templates.

Manual Creation. It is difficult to create templates that capture complex or unique queries. Hence, student researchers also manually created text-SQL pairs without using template-generated pairs. In total, 985 manually curated pairs were collected.

SQL-Review and Dataset Statistics. After curating and paraphrasing the text-SQL pairs, we performed a multi-round review process. Each text-SQL pair was reviewed to measure whether the text was clear. This was done by having different annotators review another annotator's text-SQL pairs and paraphrases to ensure they could create the same SQL statement. Each researcher would create an SQL prompt, test the logic against a database, and after the query is successfully executed, SQL questions would be generated from the tables and variables in the Iot-23 dataset. Also, there were situations where manual text-SQL pairs were either incorrect or unrealistic; hence, these pairs were removed or paraphrased before incorporating them into the entire dataset. Overall, **the entire data**

IoT Data	Description
ts	Timestamp of the first packet
uid	Uniqie ID of the connection
id.orig_h	Originating endpoint's IP address (Orig)
id.orig_p	Originating endpoint's TCP/UDP port (or ICMP code)
id.resp_h	Responding endpoint's IP address (Resp)
id.resp_p	Responding endpoint's TCP/UDP port (or ICMP code)
proto	Transport layer protocol of connection
service	Detecting application protocol, if any
duration	Connection length
orig_bytes	Orig payload bytes, from sequence numbers if TCP
resp_bytes	Resp payload bytes; from sequence numbers if TCP
conn_state	Connection state
local_orig	is Orig in Site::local_nets?
local_resp	is Resp in Site::local_nets?
missed_bytes	Number of bytes missing due to connection gaps
history	Connection state history
orig_pkts	Number of Orig packets
orig_ip_bytes	Number of Orig IP bytes (via IP total_length header field)
resp_pkts	Number of Resp packets
resp_ip_bytes	Number of Resp IP bytes (via IP total_length header field)
tunnel_parents	if tunneled, connection UID of encapsulating parent(s)

Table 3: This table contains a description of the Zeek Connection log columns, which are used as features when predicting malicious activity.

collection process took 1.5 years. The final annotated data statistics can be seen in Table 1. The dataset used to train the text-to-SQL models consisted of 10,985 rows. Each row contained a SQL query and a corresponding description, question, or prompt. The SQL queries varied in complexity but consisted primarily of arguments such as select distinct, max, avg, having, filtering, and join. On average, the prompts contained sixteen words, with the shortest prompt containing five words and the longest containing 146.

Network data The network traffic data comes from the IoT-23 dataset, which is used to train and evaluate our ability to detect malicious activity. We split the data into train, validation, and test sets based on attack type. Each session in the conn.log is labeled with one of ten attack-related labels: Attack, Benign, C&C, DDoS, FileDownload, HeartBeat, Mirai, Okiru, Torii, and PartOfAHorizontalPortScan.

The appendix provides details of the columns and features used and a summary of the data. A sample of the data is used to train and evaluate the performance of our ability to detect malicious activity. We split the network traffic data into train, validation, and test sets based on the attack type. Each session (row) in the conn.log is labeled with one of ten attack-related labels. An **Attack** label involves the infected device exploiting a vulnerable service on another system, like brute-forcing logins. **Benign** connections display no malicious intent. **C&C** signifies a device's connection to a Command and Control server, observed through periodic communications or suspicious downloads. **DDoS** denotes the device's role in overwhelming a target by sending excessive traffic. **FileDownload** infers a device downloading potential threats based on connection sizes and endpoints. **HeartBeat** marks periodic, minimal exchanges with a C&C server, ensuring active monitoring. **Mirai**, **Okiru**, and **Torii** are labels pointing to specific botnet attack patterns, with the latter two being less common than Mirai. Finally, **PartOfAHorizontalPortScan** identifies efforts to scan various systems on the same port for vulnerabilities.

Recent work exploring malicious network traffic detection has analyzed why much of the reported results are greater than 99% F1 (Kus et al., 2022). A major cause for these results is the training and testing on the same attack types. When the attack type is unknown (i.e., zero-days), performance is not as high. Hence, we split the data into training and test/validation datasets so that the same attack type in the training dataset is not in the validation and test sets. The training dataset contains network traffic related to PartOfAHorizontalPortScan and Okiru. The other sessions from the conn.log with different attach types are used in other validation and test datasets. Next, we merge all malicious activity into a single "malicious" label. Moreover, to avoid potential data leakage, all IP addresses and time stamps were randomized when training and evaluating the malicious traffic detection models. A summary of the data used for training and evaluating the malicious network activity models is shown in Table 2 and the columns/features are shown in Table 3.

4 Method

In this section, we describe the approach we developed to address the text-to-SQL task and malicious traffic detection tasks jointly.

Schema for text-to-SQL. The table schema must be included with the model input to train a model to generate SQL queries specific to our database. The schema includes all tables and variables from our database (IoT and sensor data). Formally, let t_i represent a table i, and let $c_{i,j}$ represent a column j in table i. Each column has an attribute $a_{i,j}$ representing the j-th column's datatype in table i. For instance, we have the table conn.log, which stores information about connections/sessions. Two columns within conn.log include `orig_h` and `orig_p`. The attribute assigned to the `orig_h` column is `text` since it contains strings (IP addresses). The attribute assigned to `orig_p` is `number` (representing the port number). Given all of the tables, columns, and attributes in a database, we generate the schema represented in the form of $s = [*, t_1, c_{1,1}, a_{1,1}, c_{1,2}, a_{1,2}, t_2, c_{2,1}, a_{2,1}, \ldots]$. In practice, this looks like s = [*, conn.log, orig_p, text, ..., weird.log, orig_p, text, ...]. We concatenate the schema to each input text before being passed to the T5 models to generate the SQL statements.

Input for Malicious Traffic Detection. Instead of passing the schema and text as input for malicious traffic detection, as we do for the text-to-SQL generation, we pass an instruction and formatted tabular data. Let p represent the instruction and t represent the formatted tabular data. We concatenate both to form the input $x = [p, t]$. This work uses the instruction "Is the following network information Malicious?". Also, the tabular data (row) is formatted as $t_1 t_2 \cdots t_n$, where each tabular data column/value is represented as a string. Moreover, everything is concatenated using space as the delimiter. In practice, this looks like "192.168.1.1 80 192.161.2.2 8080 $\cdots$." Note that there are no spaces in the values available in the conn.log file, which contains the network data used for malicious traffic detection. If this work is expanded to other Zeek logs, other delimiters would need to be explored.

Training. To train the model, we fine-tune the Flan-T5-base (Chung et al., 2022) model. The model is trained using the Adam optimizer (Kingma and Ba, 2014) with a minibatch size 4 and a learning rate .0001. We trained the model for a total of 15 epochs. The model was trained by simply combining the data formatted as described in Sections 4.

5 Results

In this section, we describe the evaluation metrics, our baseline models, and the results for text-to-SQL prediction and malicious network traffic detection. We also provide an informative error analysis.

Evaluation Metrics. We use two primary metrics for evaluating the text-to-SQL results: *Logical Accuracy* and *Execution Accuracy*. Logical Accuracy assesses the correctness of the logical structure and semantics of the generated SQL with the target SQL (i.e., measuring whether two SQL queries are exactly the same). However, a potential pitfall of relying solely on Logical Accuracy is that two queries may be correct but written differently. On the other hand, Execution Accuracy evaluates the results obtained when the generated SQL is run on a database. This metric is vital because the ultimate goal is to extract accurate information from the database, regardless of the SQL's structure. However, a high Execution Accuracy doesn't guarantee that the SQL query is optimal or semantically correct. It's possible for an inefficient or technically incorrect query to yield the desired results that are returned by the ground-truth query. Hence, we consider both Logical and Execution Accuracy in our study. We use the standard classification metrics macro-precision, macro-recall, and macro-F1 to evaluate our models' malicious network traffic detection performance.

Baseline models. We explore two major baselines to evaluate the performance of detecting malicious web traffic: Support Vector Machines (SVM) and Random Forest. The input of the models includes all of the features listed in Table 3 except for ts, uid, origh_h, resp_h, and tunnel_parents (i.e., all unique identifiers and IP addresses are removed). The models used to create the baseline include stratified, uniform, random forest, and support vector machines. We also explore two random baselines: stratified and uniform. The stratified baseline randomly predicts each class based on the class proportion in the training dataset, and the uniform baseline randomly predicts each class with equal probability. Finally, we evaluate transformer models Flan-T5-base and Flan-T5-Large where the input is formatted as described in Section 4. Finally, we evaluate using GPT3.5 with few-shot prompts (64 examples). For

		Validation		Test	
		Execution Acc	Logical Acc	Execution Acc	Logical Acc
		Methods that can only Generate SQL Statements			
Fine-tuned	**BART**	.693	.233	.400	.232
	T5-base	.904	.729	.827	.746
	T5-large	.966	.868	.928	.861
		Methods that can detect Malicious Traffic and Generate SQL Statements			
Prompt-based	**GPT3.5 Few-Shot**	.813	.147	.841	.177
Fine-tuned + Malware MT Learning	**T5-base**	.927	.837	.956	.851

Table 4: Text-to-SQL generation results

		Validation			Test		
		precision	recall	F1	precision	recall	F1
		Methods that can only detect Malicious Traffic					
Baselines	**Stratified**	.500	.500	.498	.502	.502	.501
	Uniform	.503	.503	.491	.497	.497	.485
	Random Forest	.879	.697	.714	.878	.694	.710
	SVM	.874	.693	.709	.872	.689	.704
Fine-tuned	**T5-base**	.883	.708	.728	.882	.704	.723
	T5-Large	.900	.777	.804	.904	**.775**	**.802**
		Methods that can detect Malicious Traffic and Generate SQL Statements					
Prompt-based	**GPT3.5 Zero-Shot**	.167	.388	.215	.183	.392	.220
	GPT3.5 Few-Shot	.741	.761	.711	.671	.640	.543
Fine-tuned + Malware MT Learning	**T5-base**	.810	.684	.697	.808	.680	.693

Table 5: Malicious traffic detection results.

the GPT3.5 model, the data is supplied in a json-like format (label, value) pairs so it knows what each value represents.

For text-to-SQL, we explore two types of fine-tuned baselines. For the fine-tuned models for text-to-SQL, we evaluate three models: Flan-T5-base, BART (Lewis et al., 2020), and Flan-T5-large. Each model is trained using the same schema defined in Section 4. These models are not trained on the network traffic data. We also evaluate GPT3.5 using in-context examples. We provide 64 in-context examples from the training dataset to make predictions. In general, our GPT3.5 prompt follows the work of Gao et al. (2023), which achieved state-of-the-art performance on the Spider dataset (Yu et al., 2018b).

Text-to-SQL. In Table 4, we report the results on the text-to-SQL task. We compare the baselines to models fine-tuned only on the text-to-SQL corpus and to a model trained on the text-to-SQL and network traffic data. Overall, we find that the larger model T5-Large outperforms the T5-base model when fine-tuned only on text-to-SQL data. The T5-Large model achieves a logical accuracy of .861 on the test set and an execution accuracy of .928. However, when jointly trained on both datasets, we find that the T5-base model can nearly match (and beat) the performance of the larger model. Specifically, the T5-base model achieves a logical accuracy of .851 and an execution accuracy of .956 on the test data with multi-task training, thus matching and outperforming the T5-Large model trained only on the text-to-SQL data.

Malicious Network Traffic Detection. In Table 5, we report the results of detecting malicious network traffic. We find that the Random Forest outperforms other methods for the baseline models. The random forest model had an F1 score of .710 and a recall score of .694. The SVM had similar results, with an F1 score of .704 and a recall score of .689. Moreover, the GPT3.5 method performs poorly on the task, with only an F1 of .543 on the test step, a light improvement over random. We hypothesize that the validation performance is slightly better because the LLM was able to understand those attacks better than the test set attacks. However, the transformer-based models (T5-base and T5-large) substantially outperformed all baseline models. The Flan-T5-Large model was the top-performing fine-tuned model model, with an

F1 score of .802 and a recall score of .775. Overall, compared to the text-to-SQL results, we find that training on both malicious traffic detection and text-to-SQL reduces the performance of the network traffic task. When analyzing the results, we find that the model struggles to identify malicious items, mostly labeling examples as Benign.

Error analysis. Why does the T5-base model match and outperform the T5-large model when trained on both datasets? Our analysis shows that much of the improvement is on the conn.log-related queries. The conn.log was the table used as input when training the malicious network traffic detection-related aspect of our model. Specifically, for logical accuracy, 142 examples in the test dataset contained items related to the conn.log table. The T5-base model missed 42 of them. The jointly trained T5-base model only missed 27. Some of the errors were major, where the T5-base model did not generate a SQL statement at all, where the T5-large model returned the correct statement (e.g., "SELECT service FROM IoT23_CONN_LOG GROUP BY service HAVING AVG(resp_bytes) >= 829").

We also hypothesize that while we did not train to make inferences about other tables in the database, by better understanding the conn.log table, the model can better understand how it relates more to other tables via JOIN queries. This better understanding of table relationships potentially results in improvements for other tables as well.

6 Conclusion

Databases hold large amounts of structured knowledge across various sectors, and efficient access to this data is essential. Our study was driven by the goal of NLIDB, which is to simplify data access beyond the complexities of SQL. While there have been advancements in text-to-SQL systems, our research emphasizes the importance of retrieving and understanding the data. With the introduction of the IoT-SQL dataset, we've provided a unique resource with the ability to predict aspects not in the database (i.e., malicious network activity) and generate SQL statements based on an input text query. Moreover, the dataset contains many temporal queries that are missing or limited in prior text-to-SQL datasets. Our findings show that models trained to query and reason about data improve SQL generation performance.

Overall, there are two major avenues for future work. First, we plan to explore more complex models on the dataset, particularly on more complex training, validation, and test sets. For example, recent work suggests that exploring different data split methods (e.g., based on SQL length, tables, or column names) can improve the measure of generalizability (Gan et al., 2022; Tarbell et al., 2023). Second, we will explore more sophisticated methods of detecting malicious network activity. Malicious activity may be related to multiple sessions within the Zeek Conn.log. Developing a system that can reason over multiple rows in the database can potentially generate substantial improvements.

7 Acknowledgements

This material is based upon work supported by the National Security Agency under Grant / Cooperative Agreement (NCAE-C Grant) Number H93230-21-1-0172. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notion herein. We especially acknowledge Dr. Glenn Dietrich, a co-author who passed away before this paper was published. This work would not have been possible without his support.

8 Limitations

Our study acknowledges several limitations that warrant discussion. Firstly, while our novel IoT-SQL dataset provides a rich collection of text-SQL pairs and network traffic data, the specific focus on IoT environments and network traffic may limit the generalizability of our findings to other domains or types of data. This specialization means that models trained on our dataset might not perform as well when applied to databases with different structures or content, such as healthcare or financial databases. However, it is still a novel domain for tabular QA, which state-of-the-art LLMs (e.g., GPT3.5) struggle to understand, thus providing a new testbed for understanding how to add new functionality to the models. We also understand that GPT4 may perform better than GPT3.5, but because of the size of the network data, the experiments are expensive. GPT3.5 experiments cost nearly $600, not including small preliminary experiments. There are also things that could have improved the results, e.g., finding the most similar in-context examples. But, again, the cost was prohibitive because of our limited research budget.

Also, our approach relies heavily on the qual-

ity and diversity of the SQL queries and the paraphrased text. Despite our efforts to generate diverse and complex queries, certain query structures or linguistic variations may still be underrepresented. This underrepresentation could impact the model's ability to generalize across unseen queries or to handle nuanced variations in natural language.

Another significant limitation lies in the multi-task learning approach for joint training on text-to-SQL generation and malicious network traffic detection. While this approach improved the text-to-SQL performance, it did not enhance and, in some cases, slightly reduced the accuracy of malicious traffic detection. This suggests a potential trade-off when balancing multiple tasks, and further research is needed to optimize such multi-task learning frameworks to ensure that improvements in one task do not come at the expense of another.

In summary, while our contributions are significant, addressing these limitations through future research will be crucial for advancing the state of text-to-SQL systems and their application to diverse and complex datasets to really understand all types of data beyond just generating a SQL statement.

References

Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-boo: I see your smart home activities, even encrypted! In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 207–218.

Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics*, 11:227–249.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565.

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

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Deborah A Dahl, Madeleine Bates, Michael K Brown, William M Fisher, Kate Hunicke-Smith, David S Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. *Proceedings of the workshop on Human Language Technology*, pages 43–48.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, 51(1):33–40.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

Lun Du, Fei Gao, Xu Chen, Ran Jia, Junshan Wang, Jiang Zhang, Shi Han, and Dongmei Zhang. 2021. Tabularnet: A neural network architecture for understanding semantic structures of tabular data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 322–331.

Yujian Gan, Xinyun Chen, Qiuping Huang, and Matthew Purver. 2022. Measuring and improving compositional generalization in text-to-sql via component alignment. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 831–843.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Sebastian Garcia, Agustin Parmisano, and Maria J Erquiaga. 2021. Iot-23: A labeled dataset with malicious and benign iot network traffic. 2020.

Alessandra Giordani and Alessandro Moschitti. 2013. Automatic generation and reranking of sql-derived answers to nl questions. In *Trustworthy Eternal Systems via Evolving Software, Data and Knowledge: Second International Workshop, EternalS 2012, Montpellier, France, August 28, 2012, Revised Selected Papers 2*, pages 59–76. Springer.

Dezhi Hong, Quanquan Gu, and Kamin Whitehouse. 2017. High-dimensional time series clustering via cross-predictability. In *Artificial Intelligence and Statistics*, pages 642–651. PMLR.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. Tabbie: Pretrained representations of tabular data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3446–3456.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Dominik Kus, Eric Wagner, Jan Pennekamp, Konrad Wolsing, Ina Berenice Fink, Markus Dahlmanns, Klaus Wehrle, and Martin Henze. 2022. A false sense of security? revisiting the state of machine learning-based industrial intrusion detection. In *Proceedings of the 8th ACM on Cyber-Physical System Security Workshop*, pages 73–84.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Isil Dillig Navid Yaghmazadeh, Yuepeng Wang and Thomas Dillig. 2017. Sqlizer: Query synthesis from natural language. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM*, pages 63:1–63:26.

Vern Paxson. 1999. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463.

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

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 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*, pages 9895–9901.

Alvin Cheung Jayant Krishnamurthy Srinivasan Iyer, Ioannis Konstas and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973.

Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1493–1503.

Runxin Sun, Shizhu He, Chong Zhu, Yaohan He, Jinlong Li, Jun Zhao, and Kang Liu. 2022. Leveraging explicit lexico-logical alignments in text-to-sql parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 283–289.

Lappoon R Tang and Raymond Mooney. 2000. Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 133–141.

Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. 2021. Rpt: relational pre-trained transformer is almost all you need towards democratizing data preparation. *Proceedings of the VLDB Endowment*, 14(8):1254–1261.

Richard Tarbell, Kim-Kwang Raymond Choo, Glenn Dietrich, and Anthony Rios. 2023. Towards understanding the generalization of medical text-to-sql models and datasets. *arXiv e-prints*, pages arXiv–2303.

Ngoc Phuoc An Vo, Octavian Popescu, Irene Manotas, and Vadim Sheinin. 2022. Tackling temporal questions in natural language interface to databases. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 179–187.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020a. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*.

Fei Wang, Kexuan Sun, Muhao Chen, Jay Pujara, and Pedro Szekely. 2021. Retrieving complex tables with multi-granular graph representation learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1472–1482.

Lihan Wang, Bowen Qin, Binyuan Hui, Bowen Li, Min Yang, Bailin Wang, Binhua Li, Jian Sun, Fei Huang, Luo Si, et al. 2022a. Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1889–1898.

Ping Wang, Tian Shi, and Chandan K Reddy. 2020b. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361.

Zhiruo Wang, Zhengbao Jiang, Eric Nyberg, and Graham Neubig. 2022b. Table retrieval may not necessitate table-specific model design. In *Proceedings of the Workshop on Structured and Unstructured Knowledge Integration (SUKI)*, pages 36–46.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*.

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

Xiaoyu Yang and Xiaodan Zhu. 2021. Exploring decomposition for table-based fact verification. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1045–1052.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. 2019. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, pages 1050–1055.

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