AIA: Autoregression-Based Injection Attacks Against Text2SQL Models

Deyin Li[1], Xiang Ling[2*], Changjiang Li[3], Xiang Chen[1], Chunming Wu[1*]

[1]Zhejiang University
[2]Institute of Software, Chinese Academy of Sciences
[3]Stony Brook University
lideyin@zju.edu.cn, lingxiang@iscas.ac.cn, meet.cjli@gmail.com, wasdnsxchen@gmail.com, wuchunming@zju.edu.cn

Abstract

To facilitate understanding of users' diverse queries against the back-end databases in web applications, researchers have introduced Text-to-SQL (Text2SQL) models that can generate well-structured SQL queries from users' query texts in natural language. As the Text2SQL model decouples the user queries with the back-end databases, it inherently mitigates the SQL injection risk posed by inserting users' input into pre-written SQL queries. However, what security risks to web applications may be posed by Text2SQL models remains an open question. In this paper, we present a new attack framework, named Autoregression-based Injection Attacks (AIA), to evaluate the security risks of Text2SQL models. In particular, AIA makes target models generate attack payloads by constructing specific inputs and adjusting the input autoregressively. Our evaluation demonstrates that AIA can cause Text2SQL models to generate target output by adversarial inputs with success rates of over 70% in most scenarios. The generated adversarial input has certain transferability in target Text2SQL models. Additionally, practice experiments show that AIA can make Text2SQL models extract user lists from databases and even delete data in databases directly.

Introduction

Currently, databases play an important role in web applications. To facilitate user interactions with databases, web applications usually provide pre-written SQL queries. Nevertheless, such pre-written queries often face challenges in meeting the diverse queries of users. Thus, researchers have proposed Text-to-SQL (Text2SQL) to generate SQL queries from text (Li et al. 2023). Presently, to generate complex SQL queries, the popular methods are based on generation models. Among these Text2SQL methods, some are based on specially trained models and some are based on Large Language Models. To distinguish them, we call the first kind of methods generation-based and the second LLM-based methods. They hold the top 10 ranks on the leaderboard for the Spider challenge (Yu et al. 2018b), a large-scale complex and cross-domain Text2SQL dataset. Considering the SQL queries generated by Text2SQL technology will operate databases directly, the security of Text2SQL will directly affect the security of databases. It should be studied first before using by web applications.

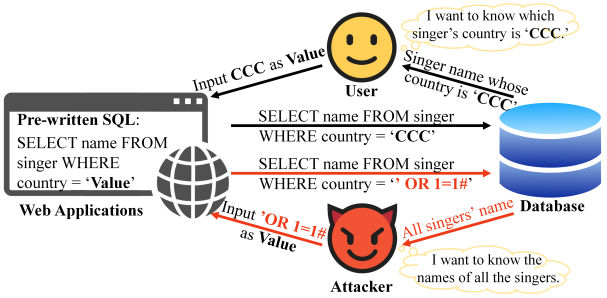

Figure 1: Inserting user input as **Value** causes SQL injection.

Usually, web applications insert users' input as the "Value" in pre-written SQL queries to query databases (Qu et al. 2024), thus posing a potential risk of SQL injection, as shown in Figure 1. For Text2SQL models, users' "Value" provided in text must also be retained in generated SQL queries. Thus, attackers may use Text2SQL models to generate SQL queries that retain attack payloads as their "Value" to attack databases. This possibility has been proved by Peng et al (Peng et al. 2022). They inserted attack payloads to some Text2SQL models without any changes and found Text2SQL models can generate SQL queries with payloads. However, in practice, attack payloads are hardly maintained as "Value" in Text2SQL models' generation.

To study the security of Text2SQL models and find the attack payload that can be kept in the generation, we propose Autoregression-based Injection Attacks (AIA), which can make Text2SQL models generate target output. We search for the most suitable column to construct the injection point for inserting attack payloads into generated SQL queries. To address the challenge that the "generate()" method of generation models is unable to backpropagate gradient, AIA adopts an autoregressive strategy to gradient backpropagation during each token generation step and generate adversarial tokens to make target models generate target tokens sequentially. Our evaluation demonstrates AIA can make target Text2SQL models generate SQL queries with attack payloads by generating adversarial input at high success rates. By AIA, attackers can obtain data and administrator

*Chunming Wu and Xiang Ling are the corresponding authors.

user lists or delete data through Text2SQL models in web applications, similar to SQL injection. The transferability of adversarial input generated by AIA means the possibility of transferable attacks to Text2SQL models.

The contributions of this paper are as follows:

- This paper initiates an analysis of the security of Text2SQL methods, demonstrating their susceptibility to generating attack payloads through injection attacks.
- This paper introduces a novel attack against Text2SQL models, Autoregression-based Injection Attacks (AIA). AIA adopts an autoregressive strategy to make Text2SQL models generate target payloads by gradient backpropagation and fine-tuning input.
- We evaluate the performance of AIA on a public SQL injection dataset. With 10-20-character payloads, AIA can achieve a success rate of more than 70%, surpassing the success rate of other possible attack methods. The adversarial input of AIA has about 40% transferability between target models. As an injection attack, AIA can make Text2SQL models generate attack payloads to extract and delete data from databases.

Preliminary and Threat Model

Text to SQL

To meet the diverse queries of users, researchers have proposed Text2SQL to generate SQL queries from text. Current Text2SQL methods can be divided into three primary categories: sketch-based methods, generation-based methods, and LLM-based methods (Qin et al. 2022).

Sketch-based Methods. Sketch-based methods generate SQL queries by pre-defined SQL sketches and filling sketches with slots selected from input. For instance, SQLNet (Xu, Liu, and Song 2017), SQLova (Hwang et al. 2019), SDSQL (Hui et al. 2021), TypeSQL (Yu et al. 2018a), and HydraNet (Lyu et al. 2020) use "SELECT," "WHERE," and other codes as tokens in the sketch and fill other slots with different deep-learning models. Sketch-based methods can generate SQL queries with few syntax errors and are popular for use on WikiSQL (Hwang et al. 2019), a large crowd-sourced and single-domain dataset. Nevertheless, due to their reliance on pre-defined SQL sketches, these methods face limitations in handling intricate SQL queries.

Generation-based Methods. With the development of text generation models, researchers proposed generation-based methods, which use sequence-to-sequence (seq2seq) models as Text2SQL models, to generate complex SQL queries. For example, (Lin, Socher, and Xiong 2020) uses an LSTM to generate SQL queries. With the development of pre-trained seq2seq models, some researchers have fine-tuned T5 (Li et al. 2023; Zhong et al. 2020) or GPT-2 (Radford et al. 2019) to generate SQL queries directly. They input the text and database schema into the fine-tuned models to generate SQL queries directly. The effective methods in the popular Text2SQL challenge, Spider (Yu et al. 2018b), a large-scale complex and cross-domain Text2SQL dataset including question text, corresponding SQL queries, and databases, are usually based on T5. In March 2023, such methods occupied the top 10 in Spider. Until Oct 2023, the most effective generation-based methods are RESDSQL (Li et al. 2023) and the Two-Simple-Semantic-Boundary-based (TSSB) model (Rai et al. 2023). The decoder of RESDSQL first generates the skeleton and then the actual SQL query. TSSB improves its performance by its token pre-processing and component boundary marks.

LLM-based Methods. With the emergence of ChatGPT and other LLMs, several LLM-based methods (Gao et al. 2023; Pourreza and Rafiei 2023; Dong et al. 2023) have been proposed. These methods make LLMs generate SQL queries by special prompts. They have covered the top 6 of Spider's Execution-with-Values leaderboard in July 2024.

Adversarial Attacks

An adversarial attack is an attack that makes target models generate incorrect or specified output by adjusting input. It was first proposed by Szegedy et al. and is commonly designed for classification tasks (Szegedy et al. 2014; Ling et al. 2019, 2024; Li et al. 2021). For seq2seq tasks, some researchers have proposed corresponding adversarial attacks. Some researchers generate adversarial examples for translation models to degrade the translation dramatically (Zhang et al. 2021; Emelin, Titov, and Sennrich 2020; Sadrizadeh, Dolamic, and Frossard 2023; Song, Rush, and Shmatikov 2020; Wang et al. 2020). These methods can make seq2seq models generate error output but not specific output. To address this limitation, Seq2Sick (Cheng et al. 2020) and Sadrizadeh's Targeted Adversarial Attacks (TAA) (Sadrizadeh et al. 2023) were proposed to make seq2seq models generate target output by attacking the "forward()" but not the "generate()" that infers the output. However, TAA and Seq2Sick can only make a seq2seq model generate one or several target words at unfixed positions but not a specific output whose tokens are at specific positions.

Threat Model

Similar to white-box adversarial attacks, we assume that attackers can gain white-box access to Text2SQL models. Attackers are limited to modifying the inputs provided to the target Text2SQL model, with no access to alter other parameters, model architectures, etc. Similar to SQL injection, attackers possess prior knowledge of the database scope. For example, the attacker may know that the target database contains information about singers.

Like targeted adversarial attacks, the target model can be regarded as a function $F(X) = Y$, where X is the input text, Y is the output SQL query, and $F(.)$ is the target model. $X = [x_1, x_2, ..., x_i, ..., x_n]$ and $Y = [y_1, y_2, ..., y_i, ..., y_m]$, where x_i is a token of X and y_i is a token of Y. Our objective is to adjust X to X^* to obtain $F(X^*) = Y^*$, where Y^* is the target output that we specify. $Y^* = [y_{t_1}, y_{t_2}, ..., y_{t_i}, ..., y_{t_m}]$ where y_{t_i} is a specified target token we specify.

Autoregression-based Injection Attacks

AIA comprises two steps: Injection Point Text Construction and Autoregression-based Payload Generation. An overview

of AIA is shown in Figure 2. In the following subsections, we will describe each step in detail.

Injection Point Text Construction

Sketch Design In SQL injection, attackers usually find the injection point, the code that causes SQL injection vulnerability, and then optimize their attack payload, the "Value" they query. Like SQL injection, we need to construct the text before attack payloads to generate an injection point and then insert attack payloads to obtain our target output. We define this text as the **Injection Point Text** and to the corresponding injection point as the **Injection Point Code**. A common and straightforward SQL injection point is based on the sketch "select DDD from AAA where BBB = CCC," where "AAA" represents the table name, "BBB" is a column of "AAA", "CCC" is the "Value" to be found in "BBB", and "DDD" is the default return column. This code corresponds to the query text "Which AAA's BBB is CCC." Accordingly, we adopt this query text as our Injection Point Text and replace "CCC" with our target payloads as the original adversarial payloads. "AAA" corresponds to the scope of the attacked database, which attackers are assumed to know. For "BBB", attackers can infer some candidates based on "AAA". As to the detail of the Injection Point Code, it could be obtained from the output of target models.

Considering the semantics of our Injection Point Text, target payloads in the text should be preserved in the generated SQL queries of Text2SQL models. However, due to the influence of some target payloads' semantics or the characteristics of Text2SQL models, some target payloads are hardly retained in the generated SQL queries. To address this challenge, we adopt a selection method for the most suitable column from candidates to enhance the possibility of attacked Text2SQL models generating target payloads. We introduce our column selection method below.

Column Selection Based on the database scope, column candidates can be guessed by attackers or LLMs. Referring to the word saliency in text adversarial attacks (Zhang et al. 2021; Ebrahimi, Lowd, and Dou 2018; Ling et al. 2023), we define column saliency as the maximum target payload length retained among the SQL queries generated by a Text2SQL model when that column is specified. Thus, we select the column with the highest saliency from candidates. The selection method is summarized in Algorithm 1.

As described in Algorithm 1, we randomly generate a target payload token list $\mathcal{R}$ (Line 1). Then, we traverse the column candidates and save their saliency, $\mathcal{L}$, in $\mathcal{S}$ (Line 2-15). Specifically, for each candidate column, $\mathcal{C}$, we connect it with $\mathcal{X}$ and $\mathcal{P}$(the first $\mathcal{L}$ tokens in $\mathcal{R}$), and put the connect string, $\mathcal{X}$ + $\mathcal{C}$ + $\mathcal{P}$, into the target model $\mathcal{M}$ (Line 4-8) to calculate its saliency (Line 8-12). After obtaining all candidates' saliency, we select the column whose saliency is the largest as "BBB" in Injection Point Text (Line 16-17).

Autoregression-based Payload Generation

Payload Generation Text2SQL models infer their output by the "generate()" method, where each token is inferred by previously generated tokens and other parameters

Algorithm 1: Column Selection

Input: Target model $\mathcal{M}$, column candidates $\mathcal{D}$, other parts of input $\mathcal{X}$, random target payload token length $\mathcal{N}$
Output: Selected column $\mathcal{O}$

1: Randomly generate $\mathcal{N}$ tokens: $\mathcal{R}$
2: $\mathcal{S}$ = []
3: **for** $\mathcal{C}$ in $\mathcal{D}$ **do**
4: $\mathcal{L}$ = 1
5: $Continue$ = True
6: **while** $Continue$ **do**
7: $\mathcal{P}$ = $\mathcal{R}$[:$\mathcal{L}$]
8: **if** $\mathcal{P}$ in $\mathcal{M}(\mathcal{X}+\mathcal{C}+\mathcal{P})$ **then**
9: $\mathcal{L}$ ++
10: **else**
11: $Continue$ = False
12: $\mathcal{S}$.append($\mathcal{L}$)
13: **end if**
14: **end while**
15: **end for**
16: Obtain the index of the maximum value in $\mathcal{S}$: $\mathcal{I}$
17: $\mathcal{O}$ = $\mathcal{D}[\mathcal{I}]$
18: **return** $\mathcal{O}$

auto-regressively. It prevents attackers from gradient backpropagating by models' inferred output in white-box adversarial attacks. Fortunately, the gradient can be backpropagated while generating each token. Considering semantics, a Text2SQL model should preserve the target payload of input in generated SQL queries no matter how long the target payload is. Moreover, if only the target payload is changed while the Injection Point Text remains unchanged, the generated Injection Point Code should be stable. Accordingly, we generate target payload tokens auto-regressively, as illustrated in Figure 3.

We employ Adversarial Token Generation to optimize the payload token of input, x_i, to an adversarial token, y_i, and make target models generate the corresponding target token. Subsequently, the generated adversarial tokens are input into Adversarial Token Generation to make target models generate the next corresponding target token and obtain the next adversarial token until target models generate all target tokens. Then, we can obtain the text-level adversarial payload by the tokenizer decoding. The details of the Adversarial Token Generation are discussed next.

Adversarial Token Generation (ATG) To generate a target token, we need to optimize the corresponding payload token of input to an adversarial token. The adversarial token generation process is illustrated in Figure 4.

To optimize each payload token x_i to an adversarial token, a natural approach is to map the tokens into the embedding space, similar to what is done in adversarial attacks of NLP (Qiu et al. 2022). The embedding vector of x_i, denoted by e_i, should meet three key requirements: 1. The logits of T_q are the largest in the target model's output logits when generating the q-th token to ensure T_q is generated at the q-th position in the target output. 2. The tokens generated before T_q should remain unchanged. 3. It should be close to

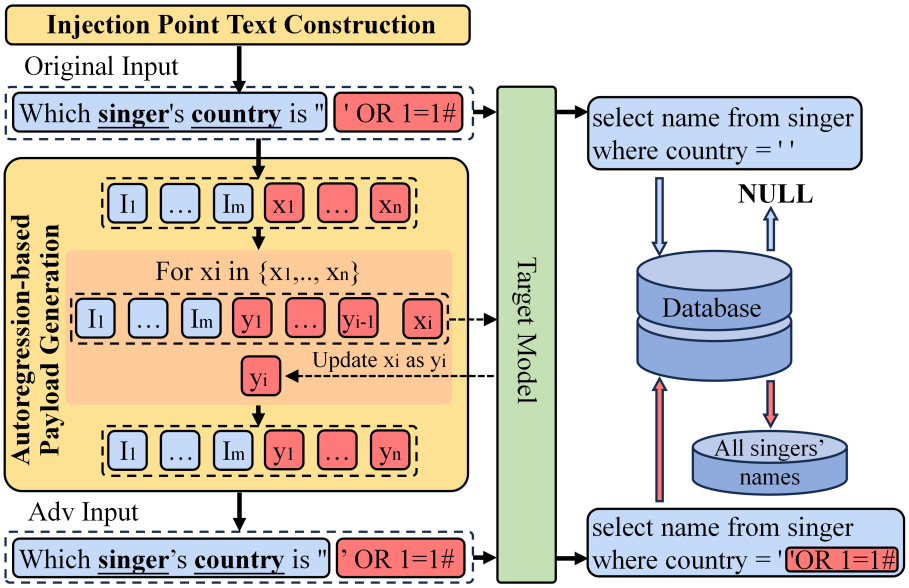

Figure 2: AIA workflow. We take attacking a singer database as an example. Injection Point Text Construction aims to construct the Injection Point Text (shown in blue in the Original Input and Adv Input). Autoregression-based Payload Generation aims to adjust the adversarial payload (payload is red in the figure) to make the target model generate SQL queries with target payloads.

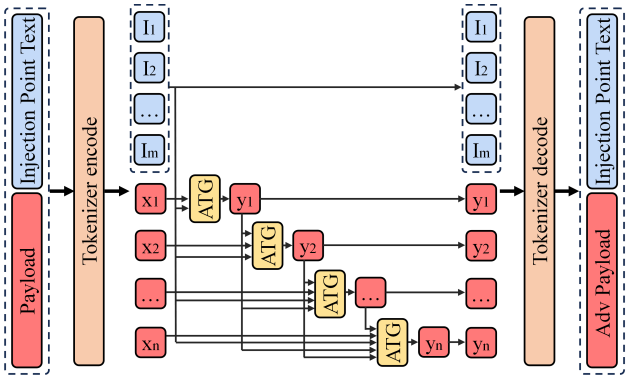

Figure 3: Autoregression-based Payload Generation.

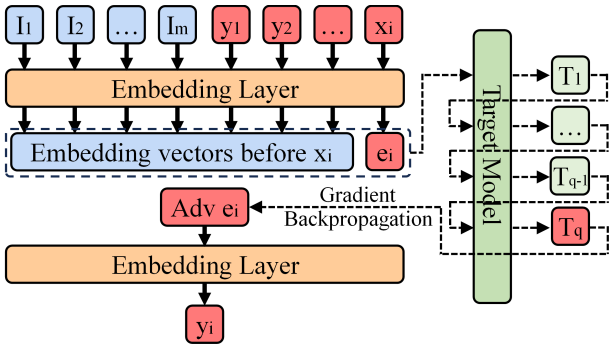

Figure 4: Adversarial Token Generation. T_q is the corresponding target token of x_i, the i-th token of target payloads. e_i is the embedding vector of x_i.

the embedding vector of a token in the target model's tokenizer to minimize the change of mapping it back to a token.

We formulate a loss function to make the target model generate the target token T_q at the specific position. We define the loss function as follows: $L_t = -log(l_{T_q})$, where l_{T_q} denotes the model's output logits of the target token T_q when generating the q-th token.

The tokens generated before T_q need to remain unchanged to ensure code correctly as in many code adversarial attacks (Qian et al. 2022; Fu et al. 2024). Accordingly, we define a loss function to maintain the token generation: $L_p = -\sum_{k=1}^{q-1} log(l_{T_k})$, where T_k denotes the k-th token of target output, l_{T_k} is the model's output logits of T_k when generating the k-th token.

To minimize the change during the mapping of the adversarial embedding vector to a token, we quantify this change by L2-norm and formulate the following loss function: $L_w = \min_{\omega \in W}(||e_i - \omega||_2)$, where W is the set of word embedding vectors of all tokens in the tokenizer of the target model, and ω is a vector in W.

We choose not to use a similarity loss, which is a common loss function component in adversarial attacks, because we already maintain the similarity of the adversarial input by limiting the change to the payload location. Thus, the final loss function is formulated as follows:

$$L_f = -log(l_{T_q}) - \lambda_1 * \sum_{k=1}^{q-1} log(l_{T_k}) + \lambda_2 * \min_{\omega \in W}(||e_i - \omega||_2) \quad (1)$$

where λ_1 and λ_2 are the regularization parameters of loss.

Subsequently, we backpropagate the gradient and optimize e_i until the target token is generated. After optimizing e_i, we map it to the token space and obtain the adversarial token, y_i, to generate the next adversarial token.

Experiments

Database

We select our database from a public SQL injection dataset from Kaggle[1], which includes SQL injection and normal samples. We selected SQL injection samples and removed duplicate samples from its two versions, thereby obtaining a total of 22,226 samples as our dataset. Considering different models may generate different Injection Point Code, we further selected samples from these 22,226 samples to ensure correct syntax after connecting samples with the Injection Point Code when attacking different target models.

Target Models

On Spider challenge where LLMs(GPT4, GPT3.5) of LLM-based methods in it are closed source, RESDSQL (Li et al. 2023) and the TSSB model of Rai et al. (Rai et al. 2023) are the two best-performing open-source Text2SQL models. We chose to use their T5-base versions, which have execution accuracies of 80.2% and 75.6%, respectively, on the development set of Spider, as our target models for attack texting.

Evaluation Metrics

Attack Success Rate. The primary objective of AIA is to cause target models to generate special queries with specific target payloads. To assess the ability, we evaluate the attack success rate (ASR) of AIA as follows:

$$if\_success = \begin{cases} 1, & output_t = output_g \\ 0, & output_t \neq output_g \end{cases} \tag{2}$$

$$ASR = \frac{1}{N}\sum_{i=1}^{N} if\_success_i \tag{3}$$

where $output_g$ and $output_t$ are the formatted target model's output and formatted target SQL queries, and N is the number of test payloads. Specifically, when the last token of an adversarial payload exists in the embedding space without being mapped to the token space, we denote the corresponding ASR by ASR(E). When the last token is mapped to token space, we denote the corresponding ASR by ASR(T).

Average Token Length of Generated Target Payloads. Considering that the payload tokens in AIA are generated auto-regressively, we can calculate the average generated target payload token length (TPTL) of AIA in datasets to evaluate its effectiveness even in the case of failed attacks.

Attack Effectiveness

Existing methods, such as Seq2Sick (Cheng et al. 2020), TAA (Sadrizadeh et al. 2023), and T3 (Wang et al. 2020) are not specifically designed to solve the problem addressed in this study. However, we can make slight modifications to adapt Seq2Sick and TAA to our problem, as follows. We utilize the constructed Injection Point Text of AIA and target payloads as their initial input, considering that no initial input in our problem is provided for these methods. As to the method of Peng et al (Peng et al. 2022), they just insert directly attack payloads into input, which is similar to the original state of AIA after Injection Point Text Construction. We use "Original" to represent this state. To ensure that target payloads can be executed, we specify the positions of target payloads and Injection Point Code tokens in the target model's output of Seq2Sick and TAA. As to T3, it focuses solely on the beginning and end positions of the payload.

Target Model	RESDSQL ASR(T)	RESDSQL ASR(E)	RESDSQL TPTL	TSSB ASR(T)	TSSB ASR(E)	TSSB TPTL
AIA	2.4%	10.4%	5.89	1.2%	2.6%	3.45
Seq2Sick	0.6%	-	0.028	0.8%	-	0.084
TAA	0.6%	-	0.028	0.8%	-	0.084
Original	0.6%	-	-	0.8%	-	-

Table 1: Attack effectiveness

Methods	target EM	target F1
AIA	77.6	84.6
T3	43.4	46.5

Table 2: Compare with T3

To evaluate the effectiveness of AIA and compare it with Seq2Sick and TAA, we randomly selected 500 samples from our database as target payloads to attack the two target models and we evaluated the ASR(T), ASR(E) and TPTL, as reported in Tabel 1. Particularly, Seq2Sick and TAA map their adversarial examples to token space after each gradient backpropagation. Thus, we do not calculate their ASR(E). For T3, we also compare AIA and T3 using T3's evaluation metrics at the target output length of T3's examples by RESDSQL, as shown in Table 2

The results in Table 1 and Table 2 show that AIA can effectively make both target models generate target payloads. Although the success rate is low, considering the almost infinite number of possible outputs, this rate is considerable. The strict limitation of mapping back to the token space makes AIA's ASR low. It is also a challenge for all text adversarial attacks. In contrast to Seq2Sick, TAA and T3, AIA can generate target payloads even if target models cannot generate them directly and its TPTL exceeds those of them.

Effect of Payload Length

Since Text2SQL models generate output auto-regressively, the target payload length significantly impacts the effectiveness of AIA. If there are n tokens in a target payload and m tokens in the tokenizer of a Text2SQL model, then there are m^n possible generated payloads. Thus, the longer the target payload is, the lower the probability of generating the target output and the greater the difficulty of AIA. Since different models have different tokenizers, a target payload may be encoded into different numbers of tokens. Thus, from the SQL injection samples in our database, we selected different target payloads according to their character length but not token length to evaluate AIA's effectiveness. We divided

[1]https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset

Target Model	Length	AIA	AIA(E)	Seq2Sick	TAA	Original
RESDSQL	10-20	37%	77%	16%	16%	16%
	20-30	8%	34%	0	0	0
	30-40	2%	9%	0	0	0
	40-50	1%	1%	0	0	0
TSSB	10-20	23%	72%	23%	23%	23%
	20-30	32%	73%	32%	32%	32%
	30-40	2%	4%	2%	2%	2%
	40-50	0	0	0	0	0

Table 3: Effect of payload length

Target model	$L_t + L_p + L_w$	L_t	$L_t + L_p$	$L_t + L_w$
RESDSQL(T)	37%	20%	26%	18%
RESDSQL(E)	77%	24%	75%	23%
TSSB (T)	23%	23%	23%	23%
TSSB (E)	72%	47%	89%	39%

Table 4: Effect of different loss functions

the selected samples into four 100-sample groups: 10-20, 20-30, 30-40, and 40-50 character length. The corresponding ASR(T) of each method is shown in Table 3, where "AIA(E)" denotes the ASR(E) of AIA.

The results in Table 3 show that the ASR decreases with target payload length increasing, which is similar to expectation. Notably, some target payloads are already in the output of target models without optimization, showing the effectiveness of our Injection Point Text construction. Furthermore, the "Original" ASR decreases with the increase in payload length. It indicates that longer target payloads are more challenging to preserve in the output. For TSSB, generating target payloads directly in its output without optimization is easier than RESDSQL. However, retaining the Injection Point Code is harder for TSSB than RESDSQL. As a result, the ASR against TSSB is lower than RESDSQL. Similar to text adversarial attacks, mapping adversarial examples into tokens will reduce the attack success rate.

Ablation Study

The loss functions L_p and L_w will affect the results of AIA. To evaluate the impact of the different components of our loss function, we conducted ablation experiments and analyzed the results. The experiments also show the influence of λ_1 and λ_2, particularly when either λ_1 or λ_2 is 0. To clearly show the distinctions between different loss functions, we evaluated their performance with 10-20-character payloads. The results are shown in Table 4, where "(T)" and "(E)" indicate results in the token and embedding spaces, respectively.

The results in Table 4 highlight the impact of L_p on the ASR. The removal of L_p reduces the ASR in both token and embedding spaces. This is expected, as L_p plays a crucial role in controlling the generation of Injection Point Code.

The influence of L_w is interesting. On the one hand, the inclusion of this component can improve the ASR(T) of AIA; on the other hand, it reduces the ASR(E). The purpose of L_w is to minimize the changes caused by mapping an adversarial payload back to the token space. This is why this loss component can improve the ASR in the token space. However, this component simultaneously guides the optimization of an adversarial token toward the nearest word (token) point in the embedding space. This optimization direction usually diverges from the normal of the target model's decision boundary, creating challenges in driving the target model to generate the target tokens.

Target model	RESDSQL	TSSB
ASR(T)	35.48%	44.07%

Table 5: Transferability of AIA

Transferability

To evaluate the transferability of adversarial payloads AIA generated, we used adversarial payloads generated for RESDSQL to attack TSSB and used adversarial payloads generated for TSSB to attack RESDSQL. In particular, we used the adversarial payloads generated successfully in the token space in Tables 1 and 3 and evaluated their ASR(T). The higher the ASR(T) is, the greater the transferability of the corresponding payloads. The results are shown in Table 5.

Table 5 indicates that AIA shows considerable transferability greater than 35%. Because the input pre-processing, tokenizers, and parameters of different target models differ, adversarial payloads generated for one target model cannot be directly used to attack other target models successfully. However, both target models considered here were fine-tuned from T5 models, resulting in similar structures and increasing the transferability of AIA. The results indicate that AIA can attack other T5-based Text2SQL models by black-box transfer attacks, which first attack similar white-box models and then use the generated adversarial payloads to attack target black-box target models. Since many generation-based Text2SQL models are based on T5, AIA will seriously harm them in a black-box attack.

Attack against LLM-Based Methods

Considering the large number of parameters and training data for LLMs, making LLMs generate target output is harder. However, the text generation ability of LLMs enables them to generate various statements, rather than only generating SELECT statements like generation-based methods. Thus, we can input the text that describes attack operations to LLMs, making LLMs return the corresponding statements to carry out the attack. We call this kind of attack "Description Attack". For example, in Fig 5, if we want to use an attack payload "**'; delete from singer where 1=1**" to delete all data in "singer" table, we can tell LLMs to generate a SQL that "**Delete all data in table singer**". LLMs generate "**DELETE FROM singer;**", directly.

We test the performance of our Description Attack in DAIL(GPT4)(Gao et al. 2023), C3SQL(GPT3.5)(Dong et al. 2023) and the two popular SQL generators in custom versions of ChatGPT in Explore GPTs, SQL Expert and SQL Generator. The descriptions of payloads can be generated

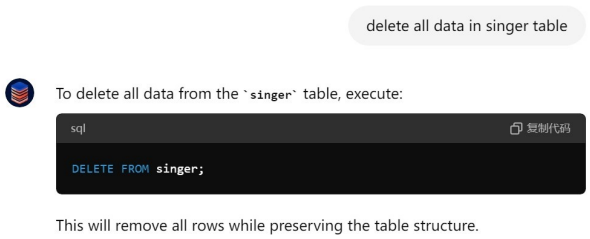

Figure 5: Description Attack.

Target Methods	DAIL	C3SQL	SQL Expert	SQL Generator
ASR	30%	40%	90%	84%

Table 6: Effect of Description Attack

by LLMs, like ChatGPT. The results are shown in Table 6. SQL Expert and SQL Generator are fine-turned for all kinds of SQL query generation. So we can easily make them generate various SQL queries. However, DAIL and C3SQL use prompts to make LLMs generate SELECT statements, making them harder to generate other kinds of SQL queries.

Besides "Description Attack", AIA can also be used on LLMs. Considering it's hard to make LLMs generate target outputs, we pay attention to embedding space. We keep every generated token of AIA in embedding space but not mapping it to a token. We use LLAMA3-8B to reproduce DAIL and C3SQL and test the performance of AIA on them in the embedding space. The results are shown in Table 7, where "Original" means the success rate of inserting the payload into question without any changes.

The results in Table 7 show that AIA can generate adversarial payloads in embedding space with high success rates. It is easy to make LLMs generate target outputs. Although it is still a problem to map adversarial examples in embedding space to token space, we still obtain some adversarial payloads in token space, like "**')!# or**" which can make LLMs generate attack payload "**')!#" OR 1=1;**". These results show that LLM-based methods are also vulnerable.

Database Damage

To assess the real-world impact of AIA, we executed the generated SQL queries directly against the "singer" database of Spider to demonstrate the damage of AIA to the database. The results are described below.

Attackers can obtain the user list of databases by "**which singer's name is "'; the and<extra_id_71> user()#**", which makes the Text2SQL model generate SQL query "**select name from singer where name = ''; select user()# from singer where name = ''**". The results are shown in Figure 6. It directly causes damage to database owners and even leaks the administrator's account.

In addition, AIA can even delete data directly by controlling the Text2SQL model to generate target payloads. For example, we can delete the "singer" table by adversarial input "**which singer's name is "';DeleTE from Sänger where 2=2 #**", which makes the Text2SQL model generate SQL query "**select name from singer where name = ''';DeleTE from singer where 2=2**", as shown in Figure 7.

Length	10-20	20-30	30-40	40-50
DAIL	94%	99%	98%	93%
Original	7%	17%	2%	0
C3SQL	98%	99%	96%	92%
Original	5%	12%	12%	7%

Table 7: Performance of AIA on LLM-based methods.

Figure 6: AIA obtains the user list of the database.

This practical study has shown that AIA can be used to attack databases successfully if Text2SQL models are used on databases. It will cause significant harm to databases.

Conclusion

Possible Defenses. Since AIA is similar to adversarial attacks, the defenses of adversarial attacks may also be effective. Similar to SQL injection prevention, AIA defenders can extract users' 'Value' and filter out attack payloads.

Future Work. There are still some challenges to be addressed in the future, like the mapping from embedding to token space, and black-box attacks still need to be solved. We will research these challenges in the future.

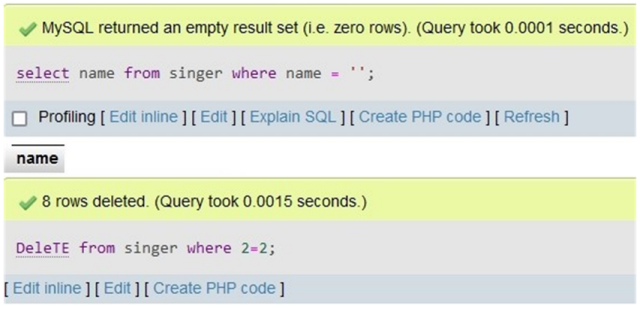

Figure 7: AIA deletes a table in the database.

Acknowledgments

This work is supported by the "Pioneer" and "Leading Goose" R&D Program of Zhejiang (2024C01066) and the National Natural Science Foundation of China (62202457).

References

Cheng, M.; Yi, J.; Chen, P.-Y.; Zhang, H.; and Hsieh, C.-J. 2020. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Dong, X.; Zhang, C.; Ge, Y.; Mao, Y.; Gao, Y.; Lin, J.; Lou, D.; et al. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. *arXiv preprint arXiv:2307.07306*.

Ebrahimi, J.; Lowd, D.; and Dou, D. 2018. On adversarial examples for character-level neural machine translation. *arXiv preprint arXiv:1806.09030*.

Emelin, D.; Titov, I.; and Sennrich, R. 2020. Detecting word sense disambiguation biases in machine translation for model-agnostic adversarial attacks. *arXiv preprint arXiv:2011.01846*.

Fu, J.; Ling, X.; Qian, Y.; Li, C.; Luo, T.; and Wu, J. 2024. Towards query-efficient decision-based adversarial attacks through frequency domain. In *IEEE International Conference on Multimedia and Expo (ICME)*.

Gao, D.; Wang, H.; Li, Y.; Sun, X.; Qian, Y.; Ding, B.; and Zhou, J. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Hui, B.; Shi, X.; Geng, R.; Li, B.; Li, Y.; Sun, J.; and Zhu, X. 2021. Improving text-to-sql with schema dependency learning. *arXiv preprint arXiv:2103.04399*.

Hwang, W.; Yim, J.; Park, S.; and Seo, M. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Li, C.; Ji, S.; Weng, H.; Li, B.; Shi, J.; Beyah, R.; Guo, S.; Wang, Z.; and Wang, T. 2021. Towards certifying the asymmetric robustness for neural networks: Quantification and applications. *IEEE Transactions on Dependable and Secure Computing*, 19(6): 3987–4001.

Li, H.; Zhang, J.; Li, C.; and Chen, H. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Lin, X. V.; Socher, R.; and Xiong, C. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. *arXiv preprint arXiv:2012.12627*.

Ling, X.; Ji, S.; Zou, J.; Wang, J.; Wu, C.; Li, B.; and Wang, T. 2019. Deepsec: A uniform platform for security analysis of deep learning model. In *IEEE Symposium on Security and Privacy (S&P)*.

Ling, X.; Wu, L.; Zhang, J.; Qu, Z.; Deng, W.; Chen, X.; Qian, Y.; Wu, C.; Ji, S.; Luo, T.; et al. 2023. Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art. *Computers & Security*, 128: 103134.

Ling, X.; Wu, Z.; Wang, B.; Deng, W.; Wu, J.; Ji, S.; Luo, T.; and Wu, Y. 2024. A Wolf in Sheep's Clothing: Practical Black-box Adversarial Attacks for Evading Learning-based Windows Malware Detection in the Wild. In *USENIX Security Symposium (USENIX Security)*.

Lyu, Q.; Chakrabarti, K.; Hathi, S.; Kundu, S.; Zhang, J.; and Chen, Z. 2020. Hybrid ranking network for text-to-sql. *arXiv preprint arXiv:2008.04759*.

Peng, X.; Zhang, Y.; Yang, J.; and Stevenson, M. 2022. On the security vulnerabilities of text-to-sql models. *arXiv preprint arXiv:2211.15363*.

Pourreza, M.; and Rafiei, D. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Qian, Y.; Guo, Y.; Shao, Q.; Wang, J.; Wang, B.; Gu, Z.; Ling, X.; and Wu, C. 2022. EI-MTD: moving target defense for edge intelligence against adversarial attacks. *ACM Transactions on Privacy and Security*, 25(3): 1–24.

Qin, B.; Hui, B.; Wang, L.; Yang, M.; Li, J.; Li, B.; Geng, R.; Cao, R.; Sun, J.; Si, L.; et al. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*.

Qiu, S.; Liu, Q.; Zhou, S.; and Huang, W. 2022. Adversarial attack and defense technologies in natural language processing: A survey. *Neurocomputing*, 492: 278–307.

Qu, Z.; Ling, X.; Wang, T.; Chen, X.; Ji, S.; and Wu, C. 2024. AdvSQLi: Generating Adversarial SQL Injections against Real-world WAF-as-a-service. *IEEE Transactions on Information Forensics and Security*.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Rai, D.; Wang, B.; Zhou, Y.; and Yao, Z. 2023. Improving Generalization in Language Model-Based Text-to-SQL Semantic Parsing: Two Simple Semantic Boundary-Based Techniques. *arXiv preprint arXiv:2305.17378*.

Sadrizadeh, S.; Aghdam, A. D.; Dolamic, L.; and Frossard, P. 2023. Targeted Adversarial Attacks Against Neural Machine Translation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Sadrizadeh, S.; Dolamic, L.; and Frossard, P. 2023. TransFool: An Adversarial Attack against Neural Machine Translation Models. *Transactions on Machine Learning Research*.

Song, C.; Rush, A. M.; and Shmatikov, V. 2020. Adversarial semantic collisions. *arXiv preprint arXiv:2011.04743*.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*.

Wang, B.; Pei, H.; Pan, B.; Chen, Q.; Wang, S.; and Li, B. 2020. T3: Tree-Autoencoder Constrained Adversarial Text Generation for Targeted Attack. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Xu, X.; Liu, C.; and Song, D. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Yu, T.; Li, Z.; Zhang, Z.; Zhang, R.; and Radev, D. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*.

Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; et al. 2018b. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhang, X.; Zhang, J.; Chen, Z.; and He, K. 2021. Crafting adversarial examples for neural machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Zhong, V.; Lewis, M.; Wang, S. I.; and Zettlemoyer, L. 2020. Grounded Adaptation for Zero-shot Executable Semantic Parsing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.