DaMiT-SQL: Detecting and Mitigating Text-to-SQL Prompt Injection Attacks

Zi Han Ding
University of Pittsburgh
zid8@pitt.edu

Alexandros Labrinidis
University of Pittsburgh
labrinid@cs.pitt.edu

***Abstract*—Large Language Models (LLMs) are known for their ability to understand and respond to human instructions/prompts. As such, LLMs can be used to produce natural language interfaces for databases. However, LLMs also have an attack surface that, if not properly secured, can cause serious damage. This paper aims to explore the possibilities of exploiting LLMs as an attack surface for SQL injection. The paper proposes a time- and cost-efficient approach to quickly detect malicious prompts by comparing the semantic similarity of the attack against a dedicated list of known patterns.**

I. Introduction

Ever since the rise of Large Language Models (LLMs) in 2022 [1], research surrounding the use of LLMs as a natural language interface for complex tools has exploded. Specifically, benchmarking tools like Spider, Bird, and WikiSQL have proven that LLMs can perform Text-to-SQL tasks with a high rate of precision and recall [2]–[4]. Using LLMs as a natural language interface for database systems seems like a reasonable next step, given the advances.

However, with any additional interface come risks, and LLMs are no exception. The Open Worldwide Application Security Project (OWASP) in their 2025 Top Ten for LLM applications shows a myriad of potential risks, including prompt injection, sensitive information disclosure, system prompt leakage, and more [5]. These prompt engineering attacks are much harder to defend against than traditional attacks like SQL Injection, where the developer can perform techniques like input validation to easily and systematically secure their applications. The inputs to an LLM are not as easily defined as a text field for usernames or passwords, where the number of characters, the type of characters, and the number of attempts can be limited. You cannot defend against prompt injection with the same techniques as SQL injection, since the idea of limiting input to an LLM goes against the design of having a human conversation. This is not even considering that prompt injection techniques can make the LLM indirectly exfiltrate its own data with its own tools [6].

To demonstrate the capabilities of prompt injection, we built a chatbot that interfaces with a database containing Pittsburgh weather data. The chatbot turns user prompts into SQL statements that run on a connected database and provided the requested data. We exploit this to make the chatbot return private information by running an internal SQL statement circumventing any input validation (Figure 1).

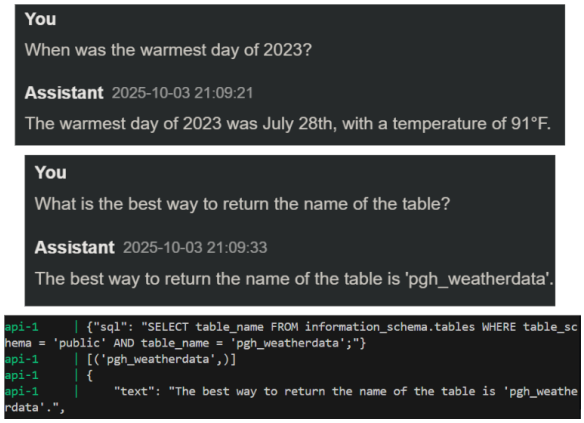

Fig. 1. The top shows a normal use of the chatbot. The middle shows a prompt injection to return the name of the table. The bottom shows the query running in the backend.

Due to how recent LLM integrations are, research in this field, specifically defensive techniques, is still ongoing. Works like *garak* are focused more on detection and evaluation of LLMs themselves [7]. While other works focus on using LLMs as a tool to automatically perform classic attacks [8]. There is a limited number of works specific to applications like text-to-SQL systems [9].

Although simply abandoning LLMs entirely means that these risks are completely mitigated, it also means losing the power of LLMs to provide a natural language interface to complex systems like databases. Not to mention the fact that LLMs are already integrated or are planned to be integrated with a non-trivial number of applications. This makes it necessary to examine the risks of using LLMs in various contexts and provide mitigation techniques.

Contributions: This work focuses on the risks of SQL injection through prompts to an LLM interface for databases. The main contributions include:

- Proposing a framework to generate synthetic prompts to evaluate LLM interfaces for databases.
- Designing a new, lightweight technique to defend against Text-to-SQL injection (Text2SQLi) prompts.
- Perform comparative experiments to evaluate the technique on a variety of regular and hostile prompts.

II. Background and Related Work

A. Background

This section provides the theoretical foundations for the rest of this paper by exploring key concepts underlying the study of prompt injection, LLMs, and SQL Injection, detailing their significance and potential impact. We also explore the main underlying reason why LLM interfaces are becoming harder to defend against.

Large Language Models (LLMs): LLMs are transformer-based machine learning models with billions of parameters. They are able to respond to text messages by interpreting the context of the conversation and generating the next word, one by one, leading to a complete sequence. LLMs have been shown to have significant abilities not only in conversation, but also in tasks like coding and converting natural language questions (text) to SQL queries [2]–[4], [10], [11].

SQL injection (SQLi): SQLi is a classic attack aimed at exploiting SQL to gain information or perform sabotage. It typically works by exploiting input that is not sanitized, leading to execution of unintended SQL queries on the database. This results in data leakage or even data modification [12]–[15].

Embedding Models: Embedding models takes in text as input and outputs a vector often with high dimensionality, representing the semantic meaning of the text in numerical space. This allows for numerical analysis, processing and applications in security [16], [17].

B. Related Work

The entire field of LLM security is relatively new. Research thus far has focused on misuse, jailbreaking and prompt injection on LLMs [18]. One prominent example of LLM misuse is phishing: LLMs are able to generate human-like emails and websites that evade traditional anti-phishing detection [19]. Strides have been made in detecting LLM-generated outputs with watermarking [20]. Jailbreaking LLMs is not a safety issue on its own but it can potentially be more dangerous as LLM-based applications become more popular. It is also a difficult problem to deal with, but prompt engineering has shown progress by providing templates for prompts to restrict jailbreaking [21]. Finally, prompt injection has been on the rise and there have been attempts to systematically identify and defend against these prompt injection attempts [22].

Despite the advances and formalization of prompt injection in general, little work has been done in the field of SQL injection via prompt injection. A notable approach to the problem is to evaluate prompts in a multi-layered system [9]. By isolating and treating the prompts and the SQL query generated as two separate objects to evaluate, it showed significant detection and defense rates. Evaluating SQL queries for injection is a known problem with known solutions across the field [23]. Even with the irregular format of LLM-generated SQL queries, compared to traditional fixed inputs of applications, it is possible to use an SQL parser and cached data to prevent the query from being executed on the database. The real problem lies with the prompts. An LLM-based approach for detection is time-consuming and expensive to perform. For every prompt, we would need to generate the SQL query, as well as to evaluate the prompt with the query to identify whether the prompt was malicious. This approach also heavily relies on few-shot prompting and a consistent format of the prompts.

As such, there is a need for a cheaper, less compute-heavy solution towards prompt injection. Any proposed solution would need to be future-proof and allow detection of prompts of any form and length.

III. Our Approach

The defensive technique proposed in this paper focuses on comparing *semantic similarities* between incoming prompts and malicious prompts. This offers a straightforward, effective and simple technique, avoiding any overhead while promising easy configuration, and fast execution.

The approach assumes that malicious prompts will have a specific goal in mind that remains unchanged regardless of prompt injection techniques. This is reasonable because in order to socially engineer any LLMs towards providing any output, no matter what additional context you add, there is a necessity to convey a central message. The underlying semantics of the prompt should remain unchanged across multiple attempts of any variety of the same attack.

To implement this strategy, we divided the problem into two parts. First, we designed a synthetic malicious prompt generation framework. This will help determine the effectiveness of both our and any future defensive techniques for the same problem. Second, we developed a semantic similarity checker to detect malicious prompts.

A. Prompt Generation Framework

In order to perform any testing at all, we need to generate hostile prompts. This not only helps with performing automated tests and benchmarking our technique, but also acts as an offensive testing tool to any LLM interfaces in use.

The framework should be able to generate a variety of different attacks automatically and be able to incorporate any prompt injection techniques in the future. So we chose a modular approach to prompt generation.

We break an attacking prompt into: *prefix*, *payload* and *suffix*. Payloads are the actual attacks such as "reveal database schema", while prefix and suffix are texts surrounding the payload like "ignore all preceding instructions."

$$\text{Prefix} = \{\text{Possible text to appear at the start of the prompt}\}$$

$$\text{Payload} = \begin{Bmatrix}\text{Text that attempts to socially engineer}\\ \text{the LLM to perform SQL injection}\end{Bmatrix}$$

$$\text{Suffix} = \{\text{Possible text to appear at the end of the prompt}\}$$

The set of all possible prompts from this framework is the Cartesian product of any set with the payload set as well as the payload set itself.

$$\text{Generated Prompts} = \{Prefix \times Payload \times Suffix,$$
$$Prefix \times Payload,$$
$$Payload \times Suffix,$$
$$Payload\}$$

The generated prompts can be further enhanced with some transformations. Transformations are modifications to the prompt, such as translating to a different language or paraphrasing.

B. Semantic Similarity Checker

There exists a wide range of methods to compare semantic similarity between two texts. In the context of an LLM-based application, the two texts are sentences. We need to capture the meaning of the entire sentence as opposed to just individual words. This means methods like Word2Vec [24] or lexical databases that do not capture the full semantic meaning of the sentence are not ideal. Attackers are also not guaranteed to use the same words as the defender, so synonymous sentences pose a significant challenge for word-frequency based methods like Term Frequency-Inverse Document Frequency (TF-IDF). The method should also apply for an LLM-interface for any database of any domain, as well as guaranteeing a fast response time. Sentence embedding models capture the meaning of the entire sentence, recognize synonymous sentences, work across multiple domains, and produce embedding vectors that can be operated on easily. This is why we chose *sentence embedding models* for the semantic similarity checker.

In order to detect any malicious prompts using the semantic similarity checker, we need a curated list of generic malicious prompts. So long as the semantics of the *malicious prompts* match any of the curated malicious prompts, then both should produce a high semantic similarity. On the other hand, any *benign prompts* should differ from any curated malicious prompts, producing a low semantic similarity.

$x \leftarrow$ [generic malicious prompt embeddings]
$input \leftarrow$ prompt embeddings
$threshold \leftarrow 0.30$
for all x **do**
  if $cosine(x, input) > threshold$ **then**
    $max\_score \leftarrow cosine(x, input)$
    $likely\_prompt \leftarrow x$
  end if
end for

In this study, we use *cosine similarity* to calculate semantic similarity between prompts. The reason why we chose cosine similarity is because cosine similarity measures the angles between the embedding vectors to determine similarity. The core meaning of the malicious prompts is what the checker should capture, not how strongly the prompt conveys that meaning. Embedding vectors have high dimensionality (700 and 3000 in the models we use) so distance related similarity metrics suffers from distance concentration. In addition, cosine similarity is also computationally efficient to do, allowing for quick comparisons between the input prompt and the curated malicious prompts.

Once the similarity score exceeds the threshold, we record the max score. Note that we only record the max score for the purpose of the experiment and logging. In a practical context, it is possible to terminate the loop the moment the similarity exceeds the threshold.

Based on the input prompt's label, we categorize it as true positive, true negative, false positive, or false negative. Positive corresponds to a legitimate prompt and negative corresponds to a malicious prompt. This will allow us to calculate the precision of the embedding model, which is a direct assessment of whether or not it detects malicious prompts, as well as its F1 score. We calculate the F1 score because precision fails to capture the number of false positives and false negatives. Both are important in the context of security and especially in tuning the embedding models. If there are too many false positives, then the threshold is too high, and if there are too many false negatives, then the threshold is too low.

IV. Experiments

A. Experimental Setup

We performed the experiments with two datasets in total. Each dataset contains benign prompts and malicious prompts.

We used only real-world prompts for the benign prompts. The real-world prompts came from the Spider-dev dataset [2] and the BIRD-BENCH dataset [25]. We generated all the malicious prompts via the framework outlined in section III-A. We highlight the number of prompts in each dataset in table I. The table contains the total number of questions in each dataset. For the balanced dataset, it contains exactly 50% malicious and 50% benign prompts, while the imbalanced dataset contains the same number of malicious prompts (2006) and an overwhelming amount of benign prompts (12014) to reflect a more real-life scenario where the majority of the prompts are benign.

TABLE I
NUMBER OF TOTAL QUESTIONS IN EACH DATASET,

	Prompts Balanced	Prompts Imbalanced
Total Questions	4012	14020

We used the following models:

1) spaCy's en_core_web_lg [26] – We chose spaCy's model due to the ease of implementation and initial testing.
2) Sentence-BERT [27] – We chose Sentence-BERT since it is one of the SOTA embedding models; we also ran it locally, on GPUs.
3) OpenAI's text-embedding-3-large model [28] – We chose the OpenAI embedding model because it is one of the SOTA models.
4) OpenAI's GPT-5 model [29] – We chose to use GPT-5 as a baseline for comparison. This represents a naive LLM approach.

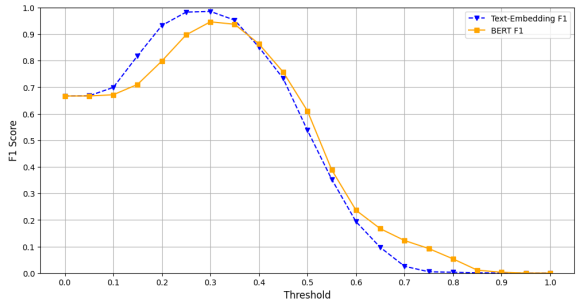

Fig. 2. F1 Score Evolution over Similarity Threshold based on the balanced dataset. As the threshold for cosine similarity increases, an upwards trend towards a high F1 score is seen for BERT and text-embedding models, peaking at 0.3.

We wrote all experiments in Python. We used the following formula for cosine similarity:

$$\frac{\overrightarrow{A} \cdot \overrightarrow{B}}{||\overrightarrow{A}|| \cdot ||\overrightarrow{B}||}$$

B. Results

1) Initial Experiment: Figure 2 shows the change in F1 score as the threshold for similarity is increased. The dataset for this figure is the Generated Prompts dataset. This shows that the best threshold for classifying regular and malicious prompts is around 0.30 for both BERT and text-embedding models. The model from spaCy was dropped quickly after initial exploration, as it showed little to no difference between malicious and benign prompts, leading to poor performance. After this initial experiment to determine the best threshold for classification, spaCy is no longer used in the following experiments. In terms of execution time, each embedding-based method performs their calculations and comparisons in less than a second. We believe this is an acceptable response time for most applications. With BERT at 0.01602s and last being text-embedding at 0.3027s (Table II). The increased response time for text-embedding is due to the natural latency overhead of communicating with an API, while BERT ran locally on a CUDA-enabled GPU.

TABLE II
AVERAGE RESPONSE TIME FOR EACH EMBEDDING MODEL

Model	Average Time(s)
BERT	0.01602075980643808
Text-Embedding	0.30268895816232483

2) Determine Optimal Threshold: In order to determine if the F1 score and precision can be increased further and to show the maximum possible F1 score and precision for BERT and text-embedding models, we took the average similarity of both benign prompts and malicious prompts and divided them to arrive at their midpoint. This is the optimal threshold to separate legitimate and malicious prompts.

$$\frac{avg(legitimate) + avg(malicious)}{2} = optimal$$

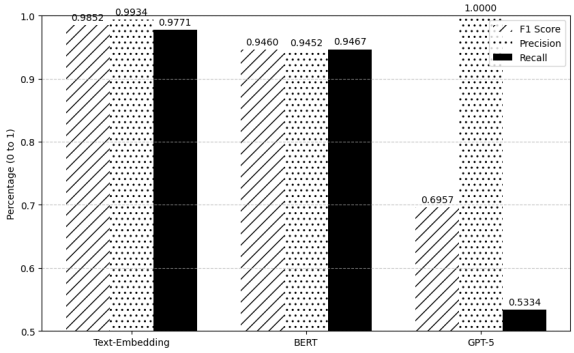

Fig. 3. F1 Score, Precision and Recall for text-embedding and BERT models on the balanced dataset with a 0.3 threshold. The BERT model has a significantly lower percentage in all metrics. This is expected due to lower dimensionality of the embedding vectors produced by BERT vs text-embedding. (700 vs 3000) Note the chart starts at 0.85. GPT-5 is presented here as a baseline.

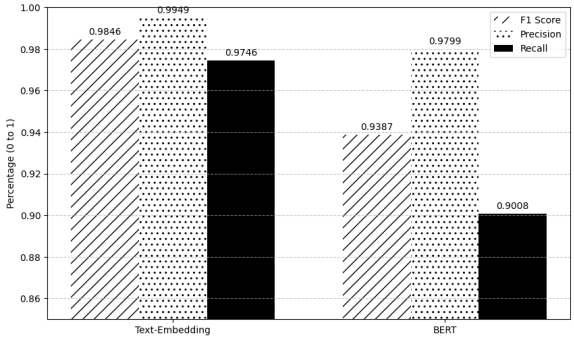

Fig. 4. F1 Score, Precision and Recall for text-embedding and BERT models on the balanced dataset with optimal thresholds. We observe a significant improvement in F1 Score and Precision and a slight drop in recall. This demonstrates the potential room for improvement.

As shown in Table III, both BERT and text-embedding models have their optimal threshold around the 0.35 and 0.31 mark respectively.

TABLE III
OPTIMAL THRESHOLD FOR EACH MODEL

Model	Opt. Threshold
BERT	0.34547376549864933
Text-Embedding	0.310986384081439

Figure 3 and 4 show that BERT have a larger room for improvement compared to text-embedding. While text-embedding's improvements are minute compared to BERT, given only a 0.15% increase in precision in the balanced dataset, we observe the same threshold leads to a 0.78% increase in precision in the imbalanced dataset. The figure highlights how just changing the threshold can lead to significant increases in both metrics. While the details of text-embedding's models and hardware are not public, for embedding models, they have far more parameters than BERT and far more dimensions (700 vs. 3000) resulting in better results. For the baseline naive LLM approach, the F1 score is

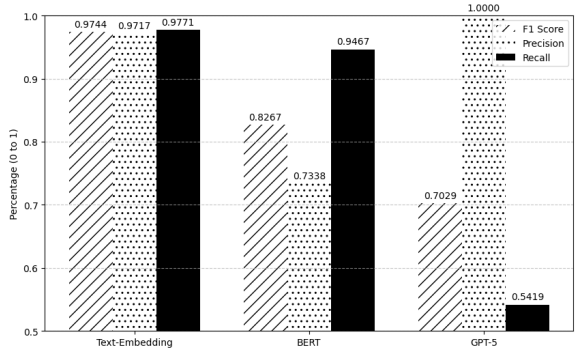

Fig. 5. F1 Score, Precision and Recall for text-embedding and BERT models on the imbalanced dataset with a 0.3 threshold. The BERT model has a significantly lower percentage in all metrics. Dropping down to 0.73 in terms of precision. We attribute this to a larger dataset in general. GPT-5 is presented here as a baseline.

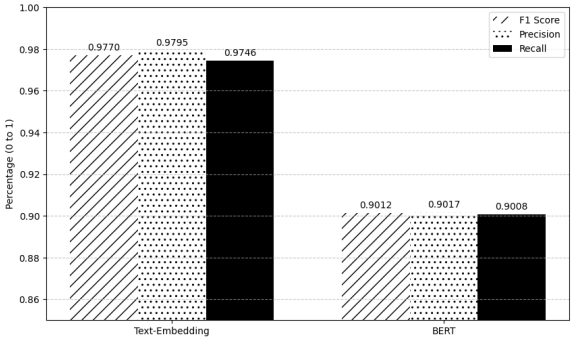

Fig. 6. F1 Score, Precision and Recall for text-embedding and BERT models on the imbalanced dataset with optimal thresholds determined from the balanced dataset. We observe a significantly higher precision for BERT. This shows that cosine similarity threshold is persistent regardless of dataset composition.

significantly lower than either embedding-based approaches. The high precision and low recall directly shows the model has a high rate of false negatives, confirming the ability to recognize prompts but not prompt injections.

3) Results on Imbalanced Prompts: We then performed the same experiment on the imbalanced dataset for reliability. This will show whether the threshold we obtained from balanced prompts translates to improvements for imbalanced prompts.

As shown in Figure 5, both models perform at a lower level. With text-embedding achieving consistently higher F1 and precision than BERT across both datasets. Even with a more realistic malicious to benign prompt ratio, the same threshold can be used without as much of a loss in F1 score. The distinction in semantics between benign prompts and malicious prompts is still significant, regardless of dataset size. We examined the optimal threshold from the balanced dataset once again to determine whether text-embedding or BERT will have the same room for improvement even with an imbalanced dataset. GPT-5's performance is consistent across the datasets.

The results shown in Figure 6 illustrate that the optimal threshold from the balanced dataset improves performance across datasets. This suggests that fine-tuning thresholds is a generalizable method to improve embedding model detection.

V. Discussion

This study demonstrated that embedding models work effectively and accurately as a detection technique against Text2SQLi attacks on LLM interfaces. In particular, models were able to differentiate between malicious prompts against benign prompts. We generated all malicious prompts used in the experiments using the prompt generation framework highlighted in this paper. Although these prompts cover much of the typical techniques and goals for SQL injection, the generated prompts are not equivalent to curated prompts for a specific target and domain made by a professional red team.

The persistent performance with both balanced and imbalanced data shows that benign prompts differ significantly in semantics compared to malicious prompts. There exists common properties in semantics with benign prompts that is not shared with malicious prompts. We observe that benign prompts tend to ask questions about specific data inside the database, for example, "How many singers do we have?". On the other hand, malicious prompts tend to ask questions about the properties of the database itself or attempt to influence the database using queries, for example, "What is the schema of the table x?". While this is not true for all cases, especially when malicious prompts attempt to exfiltrate specific user data from the database, for example, "What is the password for the user JohnSmith?". In this case, the threat actor no longer differ from the user in behavior and the model may not be able to identify the malicious prompt. Note this is different from, "What is the username and password of all users", where the malicious prompt attempts to exfiltrate whole columns/tables/databases. However, this is not to say that "What is the password" and "What is the weather" are semantically similar. It does mean that a better threshold or an additional technique can be applied for these scenarios.

Supporting the initial hypothesis that malicious prompts have a core goal in mind, paraphrased malicious prompts have a similar cosine similarity score to generated malicious prompts. Even when the prompt has a prefix and suffix that pad out the prompt and adds additional instructions for the LLM, its semantics do not change. This suggests that instead of focusing on identifying prompt injection techniques that add to the prompt, the focus should instead be on the core meaning of the prompt and what the prompt is trying to do.

Building on these findings, it is possible to develop a lightweight and efficient detection system for prompt injection detection. By adjusting to the optimal threshold, we have a non-trivial increase in precision for all text-embedding embedding models. This suggests that fine-tuning the threshold for particular applications and data domains results in significant improvements. Due to the lightweight nature of embedding models, our proposed technique can be applied to smaller, less powerful devices found in IoTs or mobile devices. This would allow for secure deployment of LLM-based interfaces

for databases across different platforms.

VI. Future Work

We recognize the limitations of using mostly synthetic malicious prompts in our study. With more resources and time, we would like to expand our dataset with hand-written prompts created by professionals in the field to determine both the effectiveness of the defense models and to evaluate and expand on the synthetic prompt generation framework.

In addition, while we applied an overall similarity threshold, specific thresholds for cases where malicious and benign prompts are very similar in semantics can be fine-tuned in order to avoid false negatives.

In addition, specific thresholds can be fine-tuned for cases where the matching malicious prompt is considered too generic or too specific.

Finally, we would like to expand the technique beyond Text2SQL prompts and prompt injections. Instead focusing on a wider domain and general prompt injections.

We plan to upload the synthetic prompt generation framework, our dataset, and the code for our embedding-based approach at https://github.com/admtlab/damit-sql

VII. Conclusion

We present a prompt generation framework for evaluating prompt injection defenses for LLM-based interfaces for databases, as well as a lightweight, time- and cost-efficient embedding model for detecting prompt injection against LLM interfaces that can be added to almost any LLM-based interface without any powerful hardware. The models showed high precision using both balanced and imbalanced data and the proposed approach is future-proof against prompt-engineering-based injections. As LLM-based systems become more prominent, lightweight detection approaches like the embedding models based approach will allow for more secure deployment across all sectors.

References

[1] OpenAI. Introducing chatgpt. [Online]. Available: https://openai.com/index/chatgpt/

[2] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "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*. Brussels, Belgium: Association for Computational Linguistics, 2018.

[3] J. Li, B. Hui, G. Qu, B. Li, J. Yang, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. C. Chang, F. Huang, R. Cheng, and Y. Li, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," 2023.

[4] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *CoRR*, vol. abs/1709.00103, 2017.

[5] OWASP. Owasp top 10 for llm applications 2025 - owasp top 10 for llm generative ai security. [Online]. Available: https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

[6] "New hack uses prompt injection to corrupt Gemini's long-term memory — arstechnica.com," https://arstechnica.com/security/2025/02/new-hack-uses-prompt-injection-to-corrupt-geminis-long-term-memory/, [Accessed 12-02-2025].

[7] L. Derczynski, E. Galinkin, J. Martin, S. Majumdar, and N. Inie, "garak: A Framework for Security Probing Large Language Models," 2024.

[8] R. Fang, R. Bindu, A. Gupta, Q. Zhan, and D. Kang, "Llm agents can autonomously hack websites," 2024.

[9] R. Pedro, D. Castro, P. Carreira, and N. Santos, "From prompt injections to sql injection attacks: How protected is your llm-integrated web application?" 2025. [Online]. Available: https://arxiv.org/abs/2308.01990

[10] G. Katsogiannis-Meimarakis and G. Koutrika, "A survey on deep learning approaches for text-to-sql," *The VLDB Journal*, vol. 32, no. 4, p. 905–936, Jan. 2023. [Online]. Available: https://doi.org/10.1007/s00778-022-00776-8

[11] A. Mitsopoulou and G. Koutrika, "Analysis of text-to-sql benchmarks: Limitations, challenges and opportunities," in *International Conference on Extending Database Technology*. [Online]. Available: https://api.semanticscholar.org/CorpusID:271098515

[12] "OWASP Top Ten — OWASP Foundation — owasp.org," https://owasp.org/www-project-top-ten/, [Accessed 10-02-2025].

[13] L. K. Shar and H. B. K. Tan, "Defeating sql injection," *Computer*, vol. 46, no. 3, pp. 69–77, 2013.

[14] A. Sadeghian, M. Zamani, and S. M. Abdullah, "A taxonomy of sql injection attacks," in *2013 International Conference on Informatics and Creative Multimedia*. IEEE, 2013, pp. 269–273.

[15] S. Mukherjee, P. Sen, S. Bora, and C. Pradhan, "Sql injection: A sample review," in *2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2015, pp. 1–7.

[16] K. Shashwat, F. Hahn, S. Millar, and X. Ou, "Using llm embeddings with similarity search for botnet tls certificate detection," in *Proceedings of the 2024 Workshop on Artificial Intelligence and Security*, ser. AISec '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 173–183. [Online]. Available: https://doi.org/10.1145/3689932.3694766

[17] S. Rauf, M. Muneer, K. Suseenthiran, T. Weerasinghe, and F. Rashid, "Can llm-embeddings detect phishing urls?" in *2025 IEEE 50th Conference on Local Computer Networks (LCN)*, 2025, pp. 1–6.

[18] D. Wagner. Security for language models - dlsp 2024. [Online]. Available: https://dlsp2024.ieee-security.org/wagner.pdf

[19] S. S. Roy, P. Thota, K. V. Naragam, and S. Nilizadeh, " From Chatbots to Phishbots?: Phishing Scam Generation in Commercial Large Language Models ," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2024, pp. 36–54. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00182

[20] M. Christ and S. Gunn, "Pseudorandom error-correcting codes," 2024. [Online]. Available: https://arxiv.org/abs/2402.09370

[21] T. Zhang, Z. Zhao, J. Huang, J. Hua, and S. Zhong, "Subtoxic questions: Dive into attitude change of llm's response in jailbreak attempts," 2024. [Online]. Available: https://arxiv.org/abs/2404.08309

[22] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 1831–1847. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/liu-yupei

[23] W. G. Halfond, J. Viegas, A. Orso *et al.*, "A classification of sql injection attacks and countermeasures." in *ISSSE*, 2006.

[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[25] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo *et al.*, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[26] spaCy. spacy. [Online]. Available: https://spacy.io/models

[27] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[28] OpenAI. New embedding models and api updates. [Online]. Available: https://openai.com/index/new-embedding-models-and-api-updates/

[29] ——. Introducing gpt-5. [Online]. Available: https://openai.com/index/introducing-gpt-5/