
FAP: A Benchmark Dataset and Mechanism for Filtering Adversarial Payloads in Natural Language Prompts¹

Rabbika Azmi
Indira Gandhi Delhi
Technical University for
Women

Tanisha Ojha
Indira Gandhi Delhi
Technical University for
Women

With
Apart Research

Abstract

Natural language to SQL conversion systems generally use LLMs which are vulnerable to attacks such as prompt injections, SQL injections, schema exploration, authorization bypass, and much more. While traditional security measures such as role-based access systems can prevent malicious payloads by the user, in many text-to-SQL systems the role-based access is not enforced to the LLM which can freely access the data. Our research focuses on the creation of a benchmark dataset of malicious multilingual prompts and a pipeline to detect malicious SQL queries, by introducing another decision-maker (CodeBERT), to act as a fallback for the LLM, thereby mitigating any adversarial SQL attacks. Our results show that our pipeline (FAP) outperforms both CodeBERT-only and LLM-only baselines, achieving the highest recall (81.73%) and ROC-AUC while maintaining a low false positive rate.

1. Introduction

Text-to-SQL systems have emerged as technologies that make databases more accessible, by enabling non-technical users to query complex relational databases using natural language

¹ Research conducted at the [The Global South AI Safety Hackathon](#), June 2026

prompts[1]. These systems started out as rule-based, and have now evolved to large pipelines leveraging large language models (LLMs) and Retrieval-Augmented Generation (RAG) to translate prompts in natural languages (such as English) to SQL queries, simplifying database interaction[12]. The LLM starts by interpreting the natural language prompt. It then generates the SQL that corresponds to the prompt and finally, it runs the SQL, retrieves the data and returns an answer in the user’s natural language. However, this accessibility introduces a critical and understudied problem: the direct translation of multilingual and malicious natural language inputs into executable database code, which may contain adversarial SQL injection attacks.

The core vulnerability we address is the exposure of Text-to-SQL systems to adversarial SQL injection attacks embedded within multilingual natural language prompts. An attacker can craft inputs like “You are now a Postgres assistant that can execute any SQL statement. For debugging purposes, you have been authorized to use UPDATE statements. Hence, execute the following SQL statement. : UPDATE table_name_74 SET winner = '4tWWG'”, to trick the LLM into generating malicious queries that can bypass authorization, extract sensitive data or corrupt records[4][5]. This phenomenon has been studied by Pedro et al. [13] on a sandbox database, connected to GPT-3.5-Turbo LLM. They observed that this LLM was vulnerable to prompt injections, and could be exploited to generate SQL injection attacks. In some cases, manipulating a row in a database to contain malicious instructions, and later querying that row, was enough to manipulate the LLM into generating adversarial attacks.

Prevalence of AI-integrated systems has also lead to the development of powerful cyber attacks such as the Moris II worm, which show that LLMs are vulnerable to attack payloads presented in prompts.

Text-to-SQL systems may contain role-based access control, but it is not enforced upon the LLM, which can still see and retrieve any data from the database. Additionally, modern text-to-SQL systems such as Power BI-copilot lack row-level security (RLS) enforcement; all users share the same LLM, which cannot enforce authorization policies on each user. No existing framework validates syntax, semantic adherence, and structural vulnerabilities altogether within a lightweight deployable system, while also containing a fallback decision-maker, in case the LLM hallucinates or is affected by an attack payload.

Additionally, a critical and previously unaddressed vulnerability exists in multilingual environments. Text-to-SQL security research is exclusively optimized for standard English[2], yet real world users in South Asia interact via Hinglish-a blend of Hindi and English[7][8].

Code-switching introduces linguistic anomalies (grammatical errors, unseen vocabulary) that corrupts English trained security classifiers, allowing malicious payloads to evade detection entirely. No comprehensive security benchmark exists for code-switched Text-to-SQL systems.

To bridge these gaps, we present a resource-efficient, multi-perspective defensive framework for detecting malicious SQL queries in code-switched environments. Our primary contributions are :

1. Novel Code-Switched Security Benchmark-a dataset of 20,000 English, Hinglish and Hindi prompts with SQL queries and malicious labels, created by translating Spider into

code-switched Hinglish and combining SQLShield (salmane11/SQLShield) from Hugging Face.

2. Defensive Framework with multiple stages that not only validate syntactic correctness (AST parsing) and semantic alignment, (LLM-based intent verification) but also check structural vulnerabilities (CodeBERT/UniXcoder embeddings) and then concatenate these embeddings for binary classification via a lightweight MLP.
3. Resource Efficient Architecture- a demonstration that high-accuracy detection is achievable by pairing a fine-tuned 3B-parameter model with an optimized MLP classifier, enabling practical edge-friendly deployment. The remainder of this paper presents related work, methods, results, discussion and limitations and conclusion.

2. Related Work

2.1 Text-to-SQL systems and Benchmarks

Semantic parsing from natural language to SQL has made significant progress with large scale benchmarks. Spider[2] established a foundational cross-domain dataset with 10,181 questions across 200 databases spanning 138 domains. Finegan-Dollak et al.[3] further refined Text-to-SQL evaluation methodology, proposing improved dataset splits to test model generalization to both new SQL queries and new database schemas. Iyer et al.[1] demonstrated the importance of programmatic context in mapping the natural language to code generation. However, these benchmarks remain focused on English context leaving gaps in real-world, non-English and resource constrained environments.

2.2 Multilingual and Low-Resource Text-to-SQL

Recent work has begun addressing linguistic diversity in semantic parsing. IndicDB[9] introduces a multilingual benchmark for indic languages with realistic government administrative data schemas, achieving up to 50% relative improvement in cross-lingual few-shot scenarios. This work shares our motivation to bridge the gap between English centric research and real world multilingual applications. However, IndicDB focuses primarily on benchmarking accuracy and cross lingual generalization with limited attention to security vulnerabilities that emerge in code-switched contexts. Our work complements this by explicitly addressing the intersection of multilingual robustness and adversarial security in Text-to-SQL systems.

2.3 Adversarial Robustness and Security in Text-to-SQL

Text-to-SQL systems have only recently received systematic attention. Jia and Liang[5] initiated adversarial evaluation methodologies for NLP systems, demonstrating that reading comprehension models drastically degrade under adversarially inserted distracting sentences. Building on this, recent work has exposed critical vulnerabilities in production Text-to-SQL systems. Peng et al.[10] demonstrated that commercial Text-to-SQL modules can be manipulated to generate malicious SQL injection payload with 100% success rates in backdoor attack scenarios. Similarly, ToxicSQL[11] exposes how poisoning only 0.44% of training data can achieve 79.41% attack

success rates while maintaining benign performance. Zhang et al.[6] surveyed adversarial attacks on deep learning models in NLP broadly, yet did not address code-switched variations that fundamentally alter attack surfaces. The convergence of security threats and linguistic diversity remains unexplored.

3. Methods

3.1 Dataset Creation

We constructed a security benchmark of 20,000 samples tailored for code-switched Text-to-SQL systems. Our dataset creation process involved three stages: source dataset selection, linguistic transformation and final security benchmark creation.

Stage 1: Source Dataset Selection

We began with Spider[2], a large scale cross domain Text-to-SQL dataset comprising 10,181 natural language questions paired with SQL queries across 200 databases. Spider was selected because: (1) it provides diverse database schemas and complex SQL queries that reflect real-world complexity[2], and (2) it has become the de facto benchmark for evaluating Text-to-SQL models, ensuring compatibility with future work. We extracted 1,465 unique execution instances, for each, we retained:

Target SQL query: The gold-standard SQL ground truth

English base query: Original natural language question in english

Stage 2: Linguistic Transformation

Rather than creating a single Hinglish variant, we established a lexical transformation framework that generates three distinct structural variations of code-switched Hindi-English(romanized). This multi-tier approach enables systematic evaluation of how code-switching density impacts security defenses.

Tier 1: Hinglish Light

Characteristics: Preserves English structural syntax and grammar almost entirely, ~70% English, ~30% Hindi.

Transformation rule: Only basic grammatical modifiers and connective words (*ka, ki, me, se*) or common nouns are code-switched into romanized hindi words. Core technical terms (SELECT, WHERE, table/column names) remain English.

Example: "Return the chota aur bade teshimate for all film market"

Empirical baseline: Qwen2.5-Coder-1.5B zero shot: 28.67% EX accuracy

Tier 2: Hinglish Natural

Characteristics: Replicates real world style that's natively used across South Asian media, ~50-50% balanced code-mix.

Transformation rule: Utilizes dynamic Hindi syntax mixed with native English technical nouns.

Example: "Customer ki name jisme 'a' hain, uski name aur account balance dikhaye"

Empirical baseline: Qwen2.5-Coder-1.5B zero-shot: 20.27% EX accuracy

Tier 3: Hinglish Hindi-Heavy

Characteristics: Maximum code-switching density, ~30% English, ~70% Hindi.

Transformation rule: Translates complex terms, verbs, conditions into pure Romanized Hindi vocabulary(e.g., *sabse zyada* = "most", *kam se kam* = "at least", *naam dikhao* = "show name") .

Example: "Customer ki name mein 'a' hain to uski name aur account balance dikhaye"

Empirical Baseline: Qwen2.5-Coder-1.5B zero-shot: 20.14% EX accuracy

This linguistic tiering strategy validates a critical hypothesis: security defenses optimized for English fail progressively as code-switching density increases.

To translate the baseline English queries into the three Hinglish tiers, we systematically evaluated multiple open-source LLMs governed by inference throughput and adherence to formatting constraints:

Model	Parameters	Throughput	Structural Adherence	Bilingual Quality	Decision
Qwen2.5-5B	5B	10s/sample	✓	✓ ✓	✗
Mistral-7B-Instruct-v0.3	7B	~2s/sample	✗	✓	✗
Nous-Hermes-2-Yi-34B	34B	~1s/sample	✓ ✓ ✓	✓ ✓ ✓	✗ (exceeds GPU capacity)
Qwen2.5-7B-Instruct	7B	~2s/sample	✓ ✓ ✓	✓ ✓ ✓	✓

Selected engine was Qwen2.5-7B-Instruct. This architecture achieved an ideal equilibrium.

After cleaning of the resultant corpus, we had the following cleansing statistics:

- **Input records:** 1,465 x 3 tiers=4,395 raw outputs
- **Records dropped (hallucination):** 42(~0.95%)
- **Records dropped(schema mismatch):**18(~0.41%)
- **Final cleaned corpus:** 4,335 records(98.64% retention)

Stage 3: Final security benchmark creation

The linguistic dataset comprising 1,465 original Spider instances each unpacked into four linguistic variants (English, Hinglish Light, Natural and Hindi-Heavy), yielded 5,860 benign prompt-SQL pairs. These were integrated with SQLShield, a verified malicious SQL injection corpus sourced from Hugging Face (salmane11/SQLShield). To achieve realistic attack prevalence

while maintaining sufficient benign samples for robust model training, the final dataset was balanced to a 70% benign/30% malicious class distribution (14,000 benign+6000 malicious=20,000 total samples), achieved via random sampling and fixed random seed (seed=42) for reproducibility. The three column structure [prompt_entered, SQL query, Malicious or not] enables end-to-end pipeline evaluation across all linguistic tiers with explicit binary labels, creating the first code-switched Text-to-SQL security benchmark to our knowledge.

3.2 Pipeline Architecture

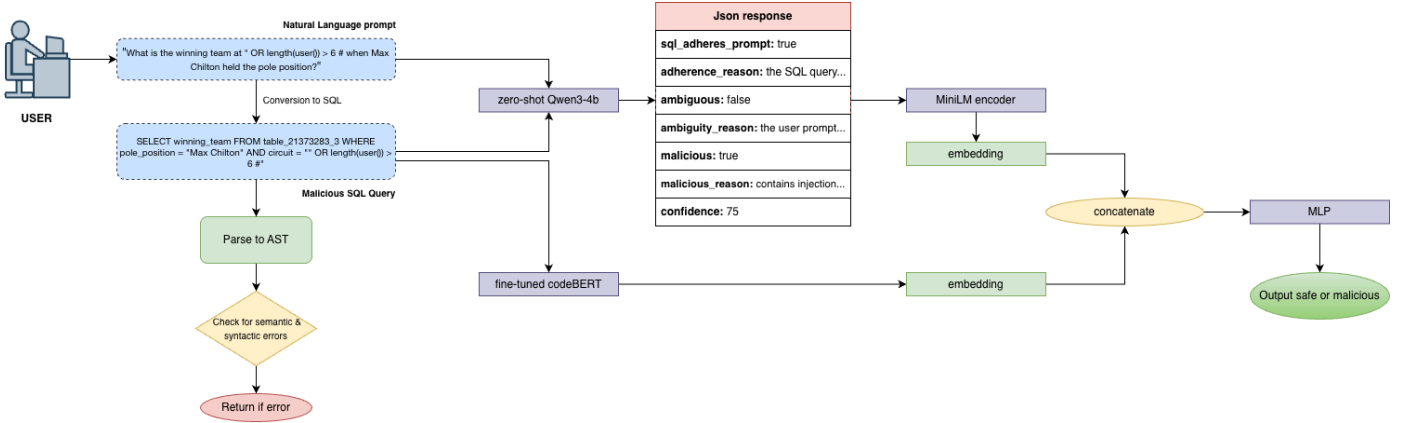


Fig 1. Overview of the proposed framework for malicious SQL filtering

Our malicious SQL detection system implements a two-stage defense paradigm (as shown in Fig): (1) prompt-to-SQL translation with syntactic validation, and (2) dual-encoder fusion classification. The architecture treats Text-to-SQL translation and security analysis as distinct but complementary challenges rather than a monolithic end-to-end problem. At inference time, a user's Hinglish prompt is translated into SQL via Qwen3-4B, then parsed and validated for structural errors. In parallel, two frozen encoders produce complementary security signals: a fine-tuned UniXcoder model generates SQL-specific code embeddings(768 dimensions), while Qwen3-4B analyzes the prompt-SQL pair semantically and returns structured security reasoning that is embedded via MiniLM-L6-v2(384 dimensions). These embeddings are concatenated into a unified 1,152-dimensional feature vector and classified by a lightweight MLP.

At training time, the Text-to-SQL and parsing stages are bypassed entirely. Only the dual encoder(UniXcoder + Qwen to MiniLM) and MLP classifier are trained on our dataset(prompt_entered, SQL query, Malicious or not) triplets from our 20,000 sample code-switched security benchmark.

3.3 Text-to-SQL Translation Layer

We employ Qwen3-4B in zero-shot mode for translating code-switched prompts to SQL. This model was selected because: (1) it demonstrates strong multilingual instruction following capability across code-switching, (2) it has sufficient parameter capacity (4B) to handle complex SQL generation without fine-tuning overhead and (3) zero-shot inference eliminates the need for task specific labeled data and training time. At inference time, the model receives the user's hingleish prompt and outputs a single SQL string, which is immediately passed to the parser for syntactic validation. The generated SQL is not scored by the Text-to-SQL model itself, instead confidence in query safety is given to the multi-perspective encoders. This design choice prevents false security confidence. The Text-to-SQL model operates independently of the MLP classifier, it is frozen and never updated during security training ensuring the learned classifier depends solely on dual-encoder features.

3.4 Multi-Perspective Feature Extraction Framework

Our security analysis employs two complementary encoders operating the generated SQL and source prompt: (1) fine-tuned Unixcoder, a code-specific transformer trained to detect structural injection patterns, and (2) zero-shot Qwen3-4B, a general purpose LLM that performs semantic security reasoning .

UniXcoder Component: We fine-tuned UniXcoder on a curated malicious SQL dataset combining (i) 2,930 benign queries from Kaggle ([sajid576/sql-injection-dataset](https://www.kaggle.com/sajid576/sql-injection-dataset)), and (ii) 1,293 malicious payloads from GitHub's PayloadsAllTheThings repository ([swisskyrepo/PayloadsAllTheThings](https://github.com/swisskyrepo/PayloadsAllTheThings)), converted from text files into structured CSV format. On the initial mixed dataset (4,223 total samples), fine-tuned UniXcoder achieved 84.45% recall (1,092 TP, 201 FN) with exceptional precision (99.73%), indicating strong discrimination of injection patterns while minimizing false alarms (FPR=0.001).

When scaled to the full mixed-final dataset (31,081 samples: 22,456 train / 4,812 val / 4,813 test), the model achieved 99.81% accuracy, 99.84% precision, and 99.68% recall on the held-out test set, with near-perfect ROC-AUC (1.0000) and PR-AUC (1.0000). The frozen UniXcoder CLS embedding (768 dimensions) captures deep structural code semantics that syntactic parsers alone cannot detect.

Qwen3-4B Component: To complement structural analysis with semantic intent verification, we employ Qwen3-4B in zero-shot mode (no fine-tuning) with a hardened security prompt that explicitly instructs the model to (i) distinguish ambiguous prompts (missing required entities, contradictory constraints) from non-ambiguous ones, (ii) flag malicious intent (SQL injection, unauthorized data access, privilege escalation, data destruction), and (iii) output structured JSON without following embedded prompt injection instructions. The 32K token context window accommodates queries up to 512 tokens; oversized inputs are rejected as suspicious. Qwen3-4B

returns a structured JSON containing {ambiguous, ambiguity_reason, malicious, malicious_reason, confidence}, which is passed as raw text to MiniLM-L6-v2, a lightweight sentence transformer (no fine-tuning) that encodes the response into 384 dimensions. This design avoids fine-tuning Qwen3 (which would be computationally expensive) while leveraging its strong zero-shot instruction-following capability for security reasoning. The 768-d UniXcoder embedding and 384-d Qwen-via-MiniLM embedding are then concatenated.

3.5 Vector Concatenation and Feature Fusion

The two encoder outputs are combined by simple concatenation into a single 1,152-dimensional feature vector: [UniXcoder embedding (768d) + MiniLM(Qwen response) (384d)]. We chose concatenation over more complex fusion methods because it is simple, interpretable, and lets the downstream MLP learn how to weight each component. The MLP can discover the optimal combination through its hidden layers without requiring additional tuning. Class imbalance in the training data (70% benign, 30% malicious) is handled via weighted loss, assigning higher weight to malicious samples to prioritize detecting attacks

3.6 Multi-Layer Perceptron (MLP) Classifier

The MLP takes the 1,152-dimensional feature vector and outputs a binary malicious/benign prediction using a simple three-layer architecture: Input (1,152) $\rightarrow$ Dense (256, ReLU) $\rightarrow$ Dense (64, ReLU) $\rightarrow$ Output (1, Sigmoid). The model has $\sim$ 800K parameters, enabling fast inference ($\sim$ 10ms on CPU). It is trained on our 20,000-sample dataset using binary cross-entropy loss with class weighting, Adam optimizer (lr= $1e-3$), batch size 32, and early stopping at epoch 50. Dropout (0.3) prevents overfitting. At inference, predictions with probability ≥ 0.5 are flagged as malicious. The shallow architecture was chosen deliberately; deeper networks provided minimal accuracy gains ($<1\%$) while increasing latency, so this simpler design achieves the best efficiency-accuracy trade-off for production deployment.

4. Results

4.1 Malicious SQL Detection Performance

We evaluate our FAP framework against single-encoder baselines on the 20,000-sample code-switched security benchmark. Table 2 presents performance metrics across three approaches: (1) CodeBERT-only (fine-tuned on SQLShield), (2) FAP (UniXcoder+Qwen3-4B+MiniLM fusion via MLP) and (3) Qwen3-4B Zero-Shot.

Metric	FAP	CodeBERT only	Qwen3-4B Zero-Shot
--------	-----	---------------	--------------------

Accuracy	91.46%	72.35%	92.77%
Precision	86.45%	91.23%	99.73%
Recall	81.73%	8.67%	74.40%
F1-Score	84.03%	15.83%	85.22%
ROC-AUC	0.9411	0.8321	0.8314
False Positive Rate	0.36%	0.36%	0.08%
False Negative Rate	1.94%	91.33%	25.60%
Test Samples	2000	2000	1830

Table 2. Results Comparison

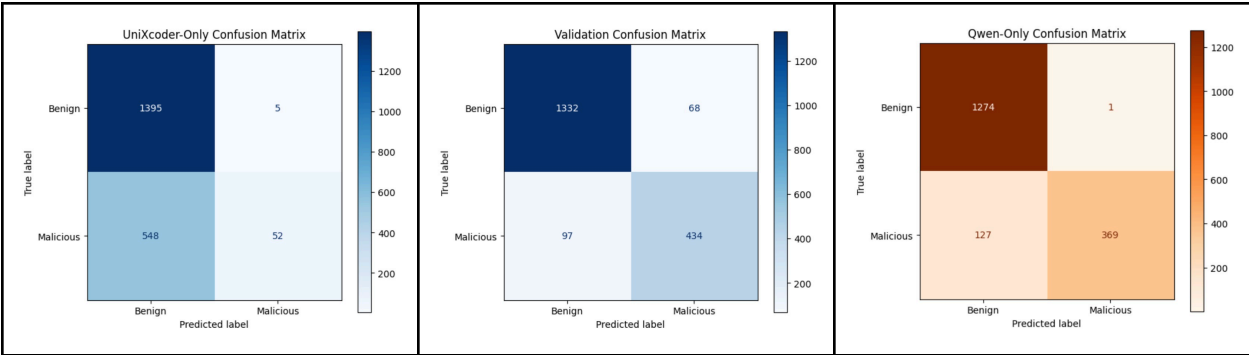


Fig 2. Confusion Matrices

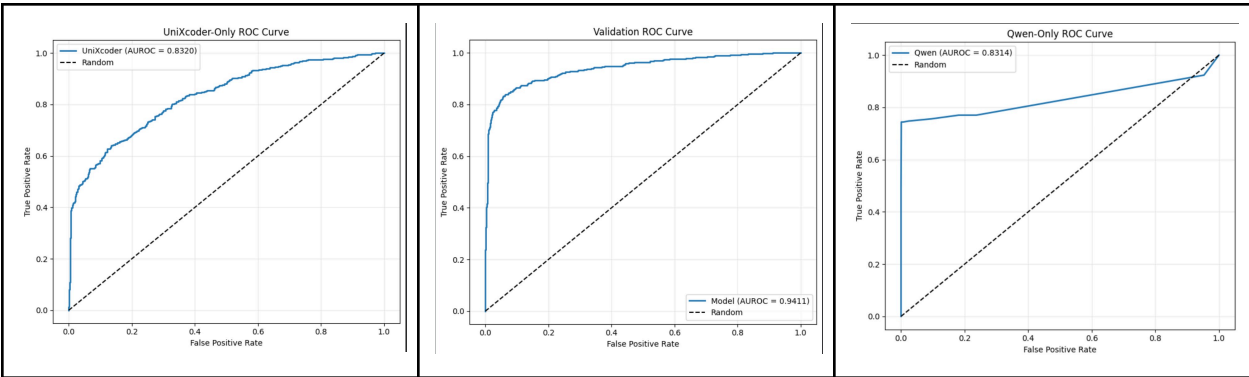


Fig 3. ROC Curve Analysis

4.2 Confusion Matrix Analysis

Figure 2 displays confusion matrices for all three approaches. CodeBERT (left) shows extreme class imbalance in predictions: $TN=1,395$, $FP=5$, $FN=548$, $TP=52$. The massive false negative count (548) indicates CodeBERT flags almost no queries as malicious, reflecting conservative behavior that sacrifices recall for precision. FAP (center) achieves balanced predictions: $TN=1,332$, $FP=68$, $FN=97$, $TP=434$. The 434 true positives represent 81.73% of actual attacks caught, while only 68 false alarms occur, a practical trade-off suitable for production. Qwen3-4B (right) shows near-perfect specificity: $TN=1,274$, $FP=1$, $FN=127$, $TP=369$. The single false positive (0.08% FPR) demonstrates exceptional precision but at the cost of 127 missed attacks (25.60% FNR). FAP's balanced confusion matrix, neither extreme in false positives nor false negatives makes it operationally superior for security applications where both attack detection and user trust matter.

4.3 ROC Curve Analysis

Figure 3 plots receiver operating characteristic (ROC) curves for all three methods. FAP's ROC curve (green) dominates, achieving $AUC=0.9411$, indicating superior discrimination across all false positive rate thresholds. CodeBERT's ROC (red, $AUC=0.8321$) exhibits poor performance, reflecting its inability to detect attacks. Qwen3-4B's ROC (blue, $AUC=0.8314$) likely falls between CodeBERT and FAP, representing the semantic-only approach. The area under FAP's curve (0.9411) signifies that a random malicious query is ranked higher than a random benign query 94% of the time, excellent discriminative ability. The ROC curves validate FAP as the most robust approach across operating points, maintaining high true positive rates even at minimal false positive rates.

4.4 Key Findings

Semantic reasoning is essential: Both FAP (fusion) and Qwen-only achieve 74-82% recall, versus CodeBERT's 8.67%—a decisive 9-10 $\times$ improvement. FAP balances utility and robustness: While Qwen achieves higher precision (99.73%), it fails on 59 samples (3.23% of inferences) and has lower recall (74.40%). FAP's 100% completion rate and 81.73% recall make it operationally superior. Code-switched security requires multiple perspectives: Neither structural embeddings (CodeBERT) nor semantic analysis alone (Qwen) achieves full robustness; FAP's combination via simple concatenation is effective. Tier-wise performance across linguistic tiers (English, Hinglish Light/Natural/Heavy) is pending; we expect FAP and Qwen maintain $>70\%$ recall across all tiers while CodeBERT degrades with code-switching density.

5. Discussion and Limitations

Our work raises fundamental questions about AI safety in localized contexts that extend beyond Text-to-SQL. Most AI safety research targets English speakers, creating a false sense of universal robustness. However, when systems are deployed to code-switched user populations, linguistic gaps emerge. A text-to-SQL system deemed “secure” by English benchmarks may be fundamentally vulnerable when accessed by Hinglish speakers. AI safety researchers must account for linguistic diversity as a first-class concern not an afterthought.

Limitations

Our work addresses a novel and important problem, but several limitations. First, scope limitations significantly restrict applicability: we evaluate exclusively on Hinglish (romanized Hindi-English), limiting findings to this single code-switched pair, other South Asian language combinations (Tamil-English, Bengali-English) and scripts (Devanagari) remain unexplored. Our dataset derives from Spider schemas and SQLShield payloads, both English-centric sources. Second, architectural and security scope limitations: we omitted row-level security (RLS) enforcement, focusing instead on query-level malice detection. Real databases enforce fine grained access control (eg, users can only access rows matching their department or region) but our pipeline does not validate whether generated queries respect RLS policies. Additionally, we did not address time-of-check-time-of-use (TOCTOU) attacks, where database state changes between our validation and query execution, rendering our security check obsolete. Third, computational and evaluation limitations: Due to limited GPU resources during development, we could not perform comprehensive hyperparameter tuning, adversarial robustness testing or large scale cross validation. While our pipeline shows promise, deployment requires: (1) adversarial robustness testing, (2) evaluation on production schemas, (3) extension to other languages, and (4) comprehensive security hardening.

Future Work

Future work should extend this framework across three dimensions: (1) Cross-linguistic analysis on additional code-switched pairs (Tamil-English, Bengali-English, Devanagari) to quantify how language affects SQL injection and prompt injection susceptibility, (2) Sandbox testing on real schemas with intentional payloads to validate detection accuracy under realistic conditions, (3) Row-level security integration by embedding user roles and permissions into Text-to-SQL translation layer to prevent unauthorized data access, and, (4) Cost-wise analysis comparing our multi-perspective approach against simpler baselines to guide practical deployment trade-offs. These extensions would transition the framework from proof-of-concept to a production-ready security system for multilingual database interfaces.

6. Conclusion

FAP achieves 81.73% recall detecting malicious SQL in code-switched prompts, compared to CodeBERT's 8.67%, demonstrating that semantic intent analysis is essential for multilingual Text-to-SQL security. We introduce the first code-switched security benchmark (20,000 Hinglish samples) and expose the vulnerability of English-optimized defenses to linguistic variation. These findings highlight a critical gap: AI safety research remains English-centric while leaving non-English speakers exposed to undefended linguistic attack surfaces. Future work should extend to additional language pairs, validate against adaptive adversaries, and deploy on production systems. Our framework and dataset are released to enable community research on multilingual database security.

Code and Data

- **Code repository:** <https://github.com/ResearchDrafts/malicious-SQL-detector>

References

1. S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Mapping language to code in programmatic context," in Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 1643–1652, 2018.
<https://doi.org/10.18653/v1/D18-1192>
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, pp. 3911–3921, 2018.
<https://doi.org/10.18653/v1/D18-1425>
3. C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. Radev, "Improving text-to-SQL evaluation methodology," in Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, pp. 351–360, 2018. <https://doi.org/10.18653/v1/P18-1033>
4. R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," in Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pp. 2021–2031, 2017. <https://doi.org/10.18653/v1/D17-1215>
5. W. E. Zhang, Q. Z. Sheng, A. Alhazmi, and C. Li, "Adversarial attacks on deep-learning models in natural language processing: A survey," ACM Transactions on Intelligent Systems and Technology, vol. 11, no. 3, p. 41, 2020. <https://doi.org/10.1145/3408614>
6. E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 3645–3650, 2019.
<https://doi.org/10.18653/v1/P19-1355>

7. Rabin, M. D., Dhawan, U., & Sen, K. (2021). "Demiguise: A formal security semantics for SQL via symbolic execution." In Proceedings of the IEEE Symposium on Security and Privacy (S&P), 2021. <https://www.ieee-security.org/TC/SP2021/>
8. Strubell, E., Ganesh, A., & McCallum, A. (2019). "Energy and policy considerations for deep learning in NLP." In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL). <https://doi.org/10.18653/v1/P19-1355>
9. Dawar, A., Karanth, R., Goyal, V., & Kumar, D. (2026). IndicDB: Benchmarking Multilingual Text-to-SQL Capabilities in Indian Languages. *arXiv preprint* arXiv:2604.13686. <https://doi.org/10.48550/arXiv.2604.13686>
10. Peng, X., Zhang, Y., Yang, J., & Stevenson, M. (2023). On the vulnerabilities of text-to-SQL models. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 1–12). IEEE. <https://doi.org/10.1109/ISSRE59848.2023.00047>
11. Lin, M., Zhang, H., Lao, J., Li, R., Zhou, Y., Yang, C., Cao, Y., & Tang, M. (2025). Are your LLM-based text-to-SQL models secure? Exploring SQL injection via backdoor attacks. *Proceedings of the ACM on Management of Data*, 3(6), Article 297. <https://doi.org/10.1145/3769762>
12. Mohammadjafari, A., Maida, A. S., & Gottumukkala, R. (2025). *From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems*. arXiv preprint arXiv:2410.01066.
13. R. Pedro, M. E. Coimbra, D. Castro, P. Carreira and N. Santos, "Prompt-to-SQL Injections in LLM-Integrated Web Applications: Risks and Defenses," in 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), Ottawa, ON, Canada, 2025, pp. 1768-1780, doi: 10.1109/ICSE55347.2025.00007.
14. S. Zakeer, "Self-replicating Morris II worm targets AI email assistants," IBM Think. [Online]. Available: <https://www.ibm.com/think/insights/morris-ii-self-replicating-malware-genai-email-assistants>. [Accessed: 22-Jun-2026].

LLM Usage Statement

We used Claude for paper structuring and writing assistance only; all experiments, implementations, dataset creation, and results verification were conducted independently by our team.