iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** XI **Month of publication:** November 2026

DOI: https://doi.org/10.22214/ijraset.2026.85358

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

Text-to-SQL Interface using Guardrails and Hallucination Detection

Shrikant Rathod[1], Pradnesh Khasnis[2], Prathamesh Gulavani[3], Sumit Shendage[4], Aditya Khot[5], P. V. Khopkar[6]
[1, 2, 3, 4, 5]*B.Tech Students,* [6]*Project Guide, Department of Computer Science and Engineering, D. Y. Patil College of Engineering and Technology, Kasaba Bawada, Kolhapur, Maharashtra, India*

Abstract: ***Large Language Models (LLMs) can translate plain-English questions into SQL, which makes institutional databases usable by people who cannot write queries. In a shared system such as a college ERP, however, the generated SQL must be treated as untrusted code: it may modify data, read rows that belong to another user, or quietly answer the wrong question. This paper reports the design, deployment, and security testing of SafeSQL, a natural-language query system over a 25-table, 174-column PostgreSQL ERP schema holding about 200,000 records. SafeSQL places the LLM behind four independent layers: model-level refusal, syntax-tree (AST) blocking with sqlglot, a read-only non-owner database role, and Row-Level Security (RLS). We show that the widely used RLS pattern based on current_setting() can be defeated by generated SQL that calls set_config() from a subquery, CTE, LATERAL join, or window function, and we replace it with a session map keyed on the PostgreSQL backend process, which adds about 2.14 ms per query. We also describe five cases in which our own test infrastructure reported success for the wrong reason, and the checks we added to catch them. The system is deployed as a web application and is covered by 248 automated tests.***

Keywords: ***Text-to-SQL, Large Language Models, Row-Level Security, PostgreSQL, Database Security, Guardrails, ERP Systems.***

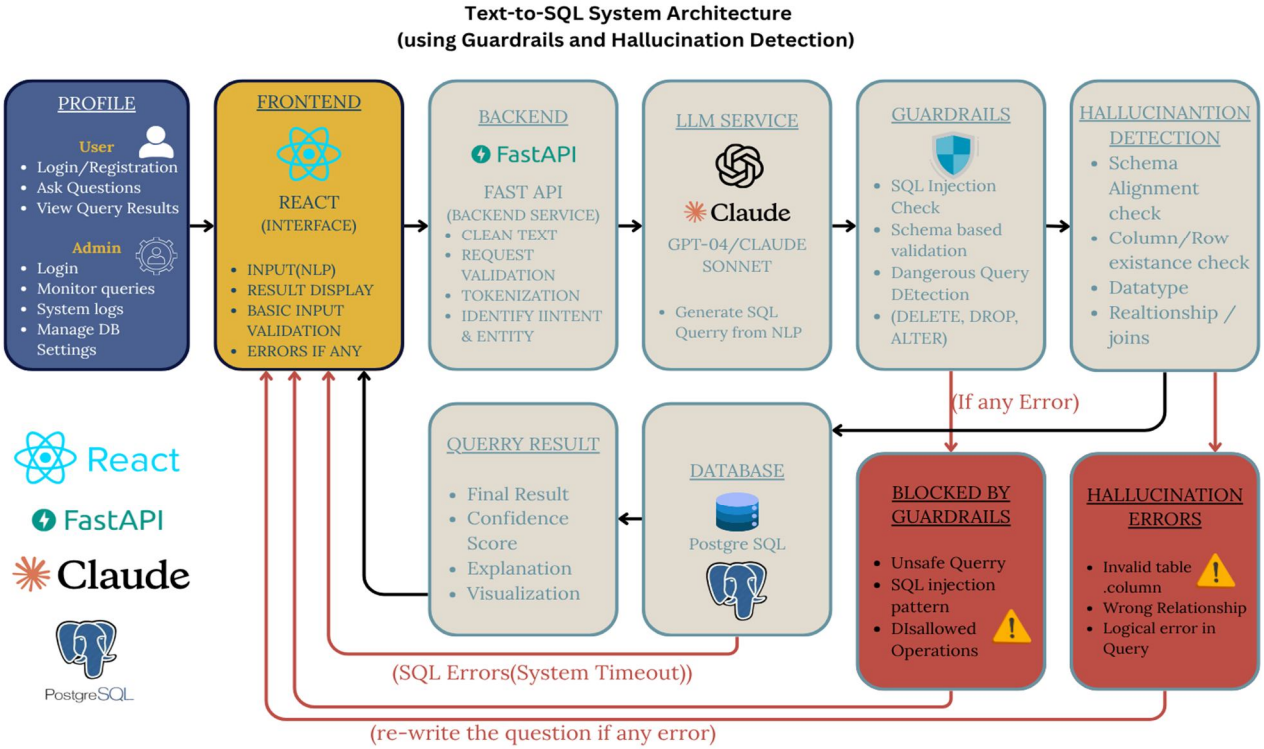


Fig. 1. SafeSQL system architecture: user profiles, React front end, FastAPI backend, LLM generation, guardrails, hallucination checks, PostgreSQL execution, and error feedback paths.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

I. INTRODUCTION

Text-to-SQL research has moved from sequence-to-sequence models evaluated on WikiSQL-style data [1] to LLM pipelines evaluated on cross-domain benchmarks such as Spider [2] and BIRD [3]. Systems such as DIN-SQL [4], DAIL-SQL [5], MAC-SQL [6], and CHESS [7] raise accuracy through decomposition, example selection, multiple agents, and schema retrieval. Benchmarks, however, score a query only by whether it returns the expected answer; they do not ask whether the query was allowed to run.

In a college ERP the question of permission matters as much as accuracy. Students, faculty, and administrators share one database, and each may see only their own rows. Telling the model in its prompt to "only return the current user's data" is not a control, because the model can ignore or misapply the instruction, and LLM-integrated applications have already been shown to be open to prompt-to-SQL injection [14]. Fine-tuned Text-to-SQL models can even be backdoored to emit injection payloads [15].

Our project guide asked that every user log in individually and see only data scoped to them. Meeting that requirement turned a single-user demo into a multi-user security problem, and this paper reports what we built and what we learned. Our contributions are:

- A deployed four-layer architecture in which no single layer is trusted to stop an unsafe query on its own.
- A demonstration that current_setting()-based PostgreSQL RLS is bypassable from model-written SQL, and a process-bound session map that removes the bypass at a cost of about 2.14 ms per query.
- A measurement approach that evaluates model refusal and AST blocking separately, so that the defence-in-depth claim can actually be checked.
- An account of five verification failures found during development, offered as practical guidance for teams building similar systems.

II. RELATED WORK

A. *LLM-Based Text-to-SQL*

Shi et al. [8] survey LLM-based Text-to-SQL and identify security and hallucination handling as open problems. Qu et al. [9] reduce schema-linking and logic hallucinations by aligning generation with similar solved tasks, and Maamari et al. [10] report that strong reasoning models can skip explicit schema linking. Open models such as CodeS [11] offer an alternative to closed APIs. These works target accuracy, not runtime access control.

B. *Guardrails and Attacks*

NeMo Guardrails [12] and Llama Guard [13] filter model inputs and outputs, but they classify text rather than analysing what a SQL statement will do. Pedro et al. [14] demonstrate prompt-to-SQL injection in LLM-integrated web applications, Lin et al. [15] show backdoored Text-to-SQL models, and Klisura and Rios [16] show that an attacker can infer a database schema from system responses alone.

C. *Database Authorization*

PostgreSQL enforces per-row policies through RLS [17]. Policies commonly read the current user from a custom configuration parameter via current_setting(). Because set_config() is an ordinary function callable inside a query [18], this pattern is safe only when the client cannot run arbitrary SQL, which is exactly the assumption an LLM interface breaks.

D. *Hallucination Detection and Calibration*

Self-evaluation [20], sampling consistency [21], and semantic entropy [22], [23] estimate when a model's output is unreliable. Raw scores tend to be overconfident [24]; isotonic regression [25] maps them to empirical correctness, and Expected Calibration Error [26] measures the remaining gap.

III. SYSTEM ARCHITECTURE

Fig. 1 shows the full pipeline. A React front end sends the question to a FastAPI backend, which authenticates the user, builds a prompt from live schema metadata, and calls the LLM through a provider-independent client. We used Claude Sonnet as the primary model and Gemini Flash-Lite for cross-model runs. The generated SQL then passes the safety layers in Section IV before execution.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

Every response carries one of five statuses: SUCCESS, REFUSED, CLARIFICATION_NEEDED, BLOCKED, or ERROR, and the front end renders each differently. This makes it visible to the user whether a question was answered, declined by the model, stopped by a guardrail, or failed at the database. Schema introspection runs on a separate owning connection, so the metadata given to the model is not affected by the RLS policies applied to the user's query.

IV. SAFETY LAYERS

A. Layer 1: Model Refusal

The prompt instructs the model to decline requests that ask for modification or for other users' data. This layer catches many unsafe requests early and explains the refusal in plain language, but it is never relied on for security.

B. Layer 2: AST Guardrails

Each statement is parsed into a syntax tree with sqlglot [19] in the PostgreSQL dialect. The query is blocked if the root is not a single SELECT (or set operation over SELECTs), if any DDL/DML node appears anywhere in the tree, if several statements are stacked, or if a denylisted function such as set_config, pg_sleep, or pg_read_file appears in any subquery, CTE, or select list. A LIMIT is added when missing.

We measure layers 1 and 2 separately. If an evaluation counts only "unsafe queries that never executed," a strong model-refusal rate can hide a weak AST layer, and the defence-in-depth claim cannot be tested. We therefore run the attack set once with refusal enabled and once with generation bypassed, feeding the attack SQL directly to the AST layer.

C. Layer 3: Read-Only Non-Owner Role

Approved queries run under a database role that owns no tables and holds only SELECT on permitted tables and columns. Sensitive columns are removed from that role's grants and from the schema description sent to the model. Table owners bypass RLS by default, so running as a non-owner is essential.

D. Layer 4: Process-Bound Row-Level Security

The common RLS policy reads the user from a session setting:

```
CREATE POLICY marks_policy ON marks FOR SELECT
USING (student_id =
 current_setting('app.current_student_id')::INT);
```

Because set_config() is an ordinary function, a generated query can rewrite that setting while it runs:

```
SELECT * FROM marks
WHERE set_config('app.current_student_id',
            '9999', false) IS NOT NULL;
```

We found working variants that place the call in a subquery, a CTE, a LATERAL join, and a window function. The AST denylist blocks these, but a security property should not depend on a denylist being complete. We therefore move identity out of anything a query can write. The backend records each authenticated user against the connection's backend process:

```
CREATE TABLE session_map (
 pid INT NOT NULL,
 backend_start TIMESTAMPTZ NOT NULL,
 user_id INT NOT NULL,
 user_role VARCHAR(20) NOT NULL,
 PRIMARY KEY (pid, backend_start));
```

Policies resolve the user through pg_backend_pid() and the matching backend_start in pg_stat_activity. The query role has no write access to session_map, and no SQL function lets a query change its own process identity. Including backend_start in the key prevents a recycled process ID from inheriting a stale mapping. The lookup adds about 2.14 ms per query.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

V. CONFIDENCE ESTIMATION

Queries that execute are scored by several verification signals: schema alignment against the live catalog, back-translation similarity between the question and a plain-English description of the SQL, result-sanity heuristics, and execution status. The combined score is calibrated with isotonic regression on held-out data and routed to the statuses described above. Here the confidence layer serves to decide when to answer, ask for clarification, or refuse.

VI. IMPLEMENTATION

1) Backend: Python, FastAPI, SQLAlchemy, PostgreSQL 16.
2) Frontend: React with TypeScript.
3) Parsing: sqlglot for AST analysis.
4) Deployment: Docker / docker-compose; deployed at https://g52-web.vercel.app.
5) Testing: 248 automated tests covering guardrails, RLS, authentication, and the evaluation harness.

VII. SECURITY EVALUATION

A. Setup

The college_erp schema has 25 tables and 174 columns (students, faculty, marks, attendance, fees, library transactions, placements, and others) with about 200,000 records. Each test user logs in individually and is mapped to a student, faculty, or administrator identity.

B. Results

TABLE I. Security Test Results

Test	GUC-based RLS	Session-map RLS
set_config bypass attempts that succeeded	[TODO: x/N]	[TODO: 0/N]
DDL/DML attacks blocked by AST layer alone	[TODO: %]	[TODO: %]
Unsafe requests refused by model alone	[TODO: %]	[TODO: %]
Per-query RLS overhead	—	≈ 2.14 ms

Table I summarises the security tests. Under GUC-based RLS, set_config() placed in subqueries, CTEs, LATERAL joins, and window functions changed the effective user; under the session map, none of these attacks changed which rows were visible. [TODO: Add execution accuracy, ECE, AUROC only if confirmed in eval/FINDINGS.md; otherwise delete this sentence]

VIII. LESSONS FROM VERIFYING THE VERIFIER

Five times during development, a test or evaluation reported success for the wrong reason. We list them because each one would have produced a publishable but false result.

1) Aggregate column projection. Matching result columns by name failed on aggregate queries, producing many false negatives. We added an AST-aware positional fallback.
2) Double calibration. A bug applied calibration twice, contaminating every raw-score comparison. All baselines were recomputed.
3) Harness overriding deployed config. The evaluation harness silently replaced the deployed configuration, so it measured a system users never ran.
4) Superuser test execution. The host test suite ran generated SQL as a superuser, which bypasses RLS and made policy tests meaningless.

5) No-op RLS guard. An evaluation check meant to confirm RLS enforcement never actually asserted anything.

Since then we confirm, for each test, that it can fail: a query that must be blocked is also run with the blocking layer disabled, to show the test detects the difference.

IX. LIMITATIONS

The session map assumes each pooled connection is re-registered before every query; a pooling bug that skipped this step would fall back to whatever mapping last held the process. Column restrictions are set per role, not per user. All experiments use one ERP schema, and the attack set, though it covers the set_config() placements we found, is not exhaustive.

X. CONCLUSION

SafeSQL treats the LLM as an untrusted SQL author and places it behind four layers that do not depend on its behaviour. Our main practical finding is that the textbook current_setting() form of PostgreSQL RLS is not safe when queries are model-written, and that binding identity to the backend process closes the gap at small cost. We also found that test infrastructure needs the same scrutiny as the system it tests. Future work includes per-user column policies, more database schemas, and broader attack generation.

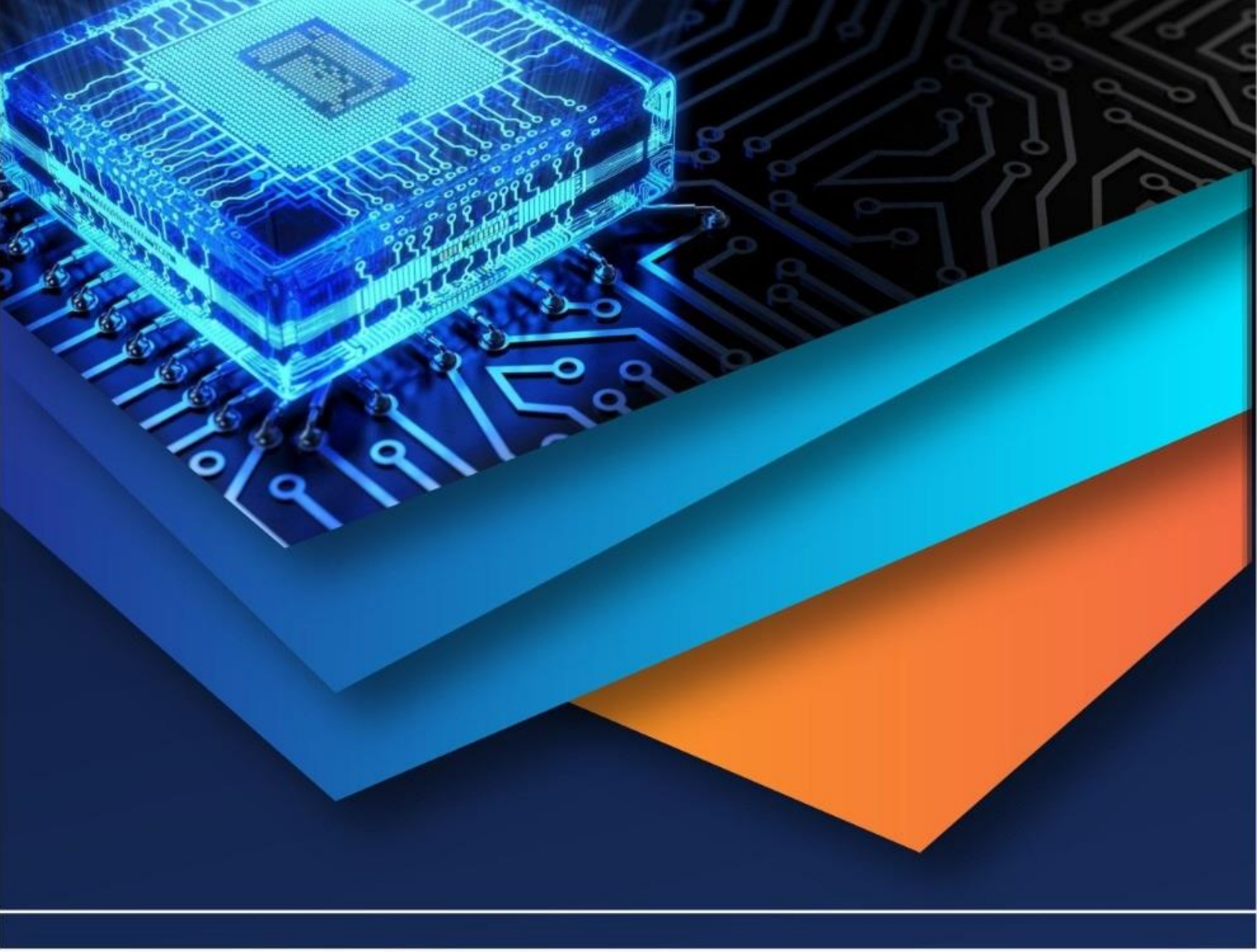
XI. ACKNOWLEDGMENT

The authors thank their project guide, Mrs. P. V. Khopkar, for her guidance and support.

REFERENCES

[1] V. Zhong, C. Xiong, and R. Socher, "Seq2SQL: Generating structured queries from natural language using reinforcement learning," arXiv:1709.00103, 2017.
[2] T. Yu et al., "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in Proc. EMNLP, 2018.
[3] J. Li et al., "Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs," in Proc. NeurIPS Datasets and Benchmarks, 2023.
[4] M. Pourreza and D. Rafiei, "DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction," in Proc. NeurIPS, 2023.
[5] D. Gao et al., "Text-to-SQL empowered by large language models: A benchmark evaluation," Proc. VLDB Endow., vol. 17, no. 5, 2024.
[6] B. Wang et al., "MAC-SQL: A multi-agent collaborative framework for text-to-SQL," in Proc. COLING, 2025.
[7] S. Talaei et al., "CHESS: Contextual harnessing for efficient SQL synthesis," arXiv:2405.16755, 2024.
[8] L. Shi et al., "A survey on employing large language models for text-to-SQL tasks," arXiv:2407.15186, 2024.
[9] G. Qu et al., "Before generation, align it! A novel and effective strategy for mitigating hallucinations in text-to-SQL generation," in Findings of ACL, 2024.
[10] K. Maamari et al., "The death of schema linking? Text-to-SQL in the age of well-reasoned language models," arXiv:2408.07702, 2024.
[11] H. Li et al., "CodeS: Towards building open-source language models for text-to-SQL," Proc. ACM Manag. Data (SIGMOD), 2024.
[12] T. Rebedea et al., "NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails," in Proc. EMNLP: System Demonstrations, 2023.
[13] H. Inan et al., "Llama Guard: LLM-based input-output safeguard for human-AI conversations," arXiv:2312.06674, 2023.
[14] R. Pedro et al., "From prompt injections to SQL injection attacks: How protected is your LLM-integrated web application?" arXiv:2308.01990, 2023.
[15] M. Lin et al., "Are your LLM-based text-to-SQL models secure? Exploring SQL injection via backdoor attacks," Proc. ACM Manag. Data (SIGMOD), 2025.
[16] D. Klisura and A. Rios, "Unmasking database vulnerabilities: Zero-knowledge schema inference attacks in text-to-SQL systems," arXiv, 2025.
[17] PostgreSQL Global Development Group, "Row security policies," PostgreSQL Documentation. [Online]. Available: https://www.postgresql.org/docs/current/ddl-rowsecurity.html
[18] PostgreSQL Global Development Group, "System administration functions: set_config," PostgreSQL Documentation. [Online]. Available: https://www.postgresql.org/docs/current/functions-admin.html
[19] T. Mao et al., "sqlglot: Python SQL parser and transpiler." [Online]. Available: https://github.com/tobymao/sqlglot
[20] S. Kadavath et al., "Language models (mostly) know what they know," arXiv:2207.05221, 2022.
[21] P. Manakul, A. Liusie, and M. J. F. Gales, "SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models," in Proc. EMNLP, 2023.
[22] L. Kuhn, Y. Gal, and S. Farquhar, "Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation," in Proc. ICLR, 2023.
[23] S. Farquhar, J. Kossen, L. Kuhn, and Y. Gal, "Detecting hallucinations in large language models using semantic entropy," Nature, vol. 630, pp. 625–630, 2024.
[24] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in Proc. ICML, 2017.
[25] B. Zadrozny and C. Elkan, "Transforming classifier scores into accurate multiclass probability estimates," in Proc. KDD, 2002.
[26] M. P. Naeini, G. F. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using Bayesian binning," in Proc. AAAI, 2015.

©IJRASET: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)