

An Explainable Multi-Agent System for Real-Time SMS Smishing Detection with LangGraph Orchestration

Nausheen Gandhi

Usha Mittal Institute of Technology
SNDT Women's University
Mumbai, India

Saloni Lad

Usha Mittal Institute of Technology
SNDT Women's University
Mumbai, India

Janhvi Mahangade

Usha Mittal Institute of Technology
SNDT Women's University
Mumbai, India

Prof. Prachi Dhanawat

Usha Mittal Institute of Technology
SNDT Women's University
Mumbai, India

Abstract—SMS has become a target for cyberattacks. Users may get one-time passwords, transaction notifications, banking alerts and other service announcements. These messages can be quite familiar, urgent, making it difficult to determine fake messages, putting users at risk of being victims of smishing. Once successful, information and authentication credentials can be stolen from victims. Current defenses are typically designed with static rules or individual AI models.

They struggle to catch evolving scam messages. They also lack real-time analysis, a combined threat check and clear reasons for why a message's dangerous. To solve these problems we introduce ScamShield AI. ScamShield AI is a multi-agent AI system that works in time to detect smishing. It contains three agents: an OTP Message Analysis Agent, a Contact Trust and Pattern Recognition Agent and a URL Analysis Agent. Each agent looks at a part of an incoming SMS. A LangGraph-based Coordinator Agent brings the agents together combines their results and outputs a label of Safe, Suspicious or Scam along with a risk score. The system first cleans messages and creates semantic embeddings for similarity-based threat detection. A MongoDB community database uses vector search to find messages that're semantically similar to known scam patterns. If the message matches known harmful patterns, the system doesn't have to analyze it. New messages go through the LangGraph multi-agent pipeline. Security controls keep the threat data trustworthy. In addition the system uses Explainable AI to give users an explanation of why a message is determined to be suspicious. the app can suggest steps that users can take, for instance, refraining from clicking on links or blocking the sender, which can help them make better security decisions.

Keywords - Smishing Detection, Multi AI Agents, Lang-Graph orchestration, Explainable AI, Malicious URL Detection, Semantics Search, Vector Database

I. INTRODUCTION

Due to the rising popularity of electronic transfers and cellphones across the country, India is witnessing an explosion in mobile traffic India's telecom sector serves a huge customer

base of over 1.31 billion users and is home to an SMS market that carries more than 186 billion messages per annum [1]. To deal with the rising number of spam calls and SMS, the Telecom Regulatory Authority of India (TRAI) launched the "Do Not Disturb" (DND), an application through which users can register their complaints about unwanted phone calls and SMS. Thus, the DND app received more than 17 lakh spam complaints in 2025 alone [2]. As a result, there are many phishing, vishing, and smishing threats in the country [3]–[5]. The smishing technique is a phishing type that uses SMSs to deliver malicious links and manipulate the recipient to perform malicious (active) activities [6]. The SMS is issued by a credible source, like banks, courier services, government agencies, or online platforms. The message has urgent words, known names, fake URLs, and asks for onetime passwords [7], [8]. For detection of smishing, there is a need to consider many factors related to message, sender's identity, embedded URL and user's behavior [9]. Different researchers have implemented the idea of smishing detection through examination of SMS text, URLs features and applying machine learning and deep learning algorithms [7], [9], [10]. Recent works have proposed to utilize transformer and large language model approaches to give context and explainability for phishing detection [11]–[13]. Applying the same analysis to the messages known or similar to the ones from the previous attacks is an unnecessary waste of resources; hence, it is rational to design the solution allowing utilizing the threat data and analyzing only new and unknown messages.

Many studies have explored smishing detection by looking at SMS text, URL details, and using machine learning and deep learning techniques [7], [9], [10]. Recent research has also introduced transformer and large language model methods to provide context and explanations for phishing detection [11]–[13]. Repeating the same type of analysis on messages that are

already known or similar to previous threats uses up valuable computing resources. Therefore, there is a need for a system that can make use of existing threat data and only perform in-depth analysis on new or unknown messages. In order to solve this problem, we have designed ScamShield AI, a real-time SMS scam and smishing detection framework. First, it sanitizes the message and extracts the important information such as OTP and collects the context. Then, it encodes the text into a vector using the allMiniLM-L6-v2 Sentence Transformer[14] and checks for similar patterns found in the database, using a vector search in MongoDB. If a message matches an existing threat, its stored information is used for detection. Otherwise, it is analyzed by a LangGraph Coordinator that orchestrates three specialized agents: OTP Agent, URL Analysis Agent, and Contact Trust Agent with Pattern Detection. Their findings are combined using weighted values and rule-based logic to calculate a risk score and label. The cleaned message and detection data are stored in MongoDB, while the Groq API generates an explainable response containing evidence and safety tips. This approach enables rapid detection of known threats and detailed analysis of new threats.

TABLE I
LIMITATIONS OF EXISTING SYSTEMS AND PROPOSED SOLUTIONS

System	Identified Limitation	Proposed Solution
NeoRakshak	SMS analysis is primarily on-demand, limiting continuous message-level analysis.	Real-time SMS interception enables automatic analysis when a message is received.
Truecaller	Primarily relies on sender identification, spam reports, and message-based signals rather than deeper semantic analysis.	Semantic embeddings and vector similarity identify messages with similar scam patterns.
ScamKavach	Focuses broadly on scam detection and user awareness rather than a specialized multi-stage SMS analysis pipeline.	SMS-focused multi-agent analysis examines different message characteristics independently.
Existing systems	Privacy-preserving processing of sensitive SMS information is not specified in the surveyed applications.	Local OTP/URL extraction and sanitization replace sensitive values with placeholders before LLM analysis.
Existing systems	Detection results may provide warnings or classifications without detailed reasoning.	Groq API-based LLM explanation converts detected signals into a human-readable explanation.
Conventional approaches	Previously unseen or evolving scam patterns may require analysis beyond fixed rules or existing reports.	Semantic pattern detection + LangGraph provides fast processing for known patterns and deeper analysis for novel patterns.

II. METHODOLOGY

Fig. 1 illustrates the end-to-end pipeline, from SMS reception through semantic matching, multi-agent analysis, and explainable output generation; each stage is described in detail in the following subsections.

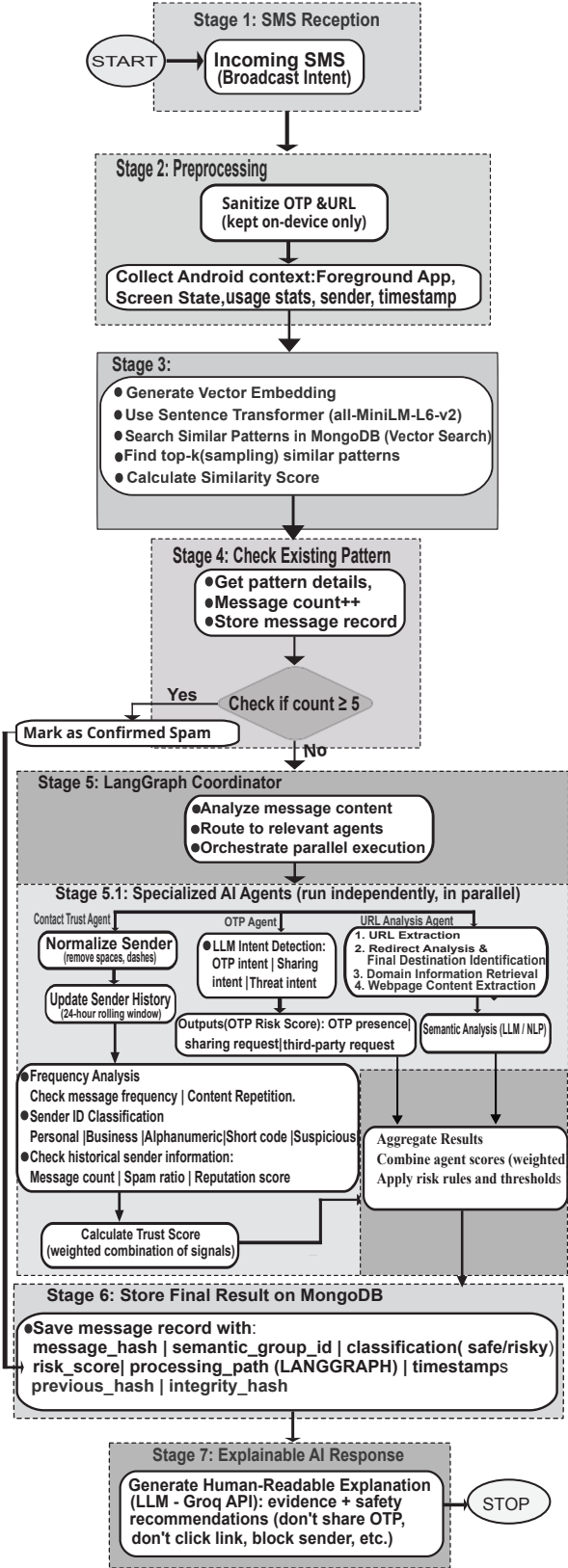


Fig. 1. ScamShield AI system architecture

A. Overall Process

1. SMS Acquisition and Preprocessing

Interception of the SMS is done through the use of Android

SmsReceiver. Here, details such as the message content, its sender, the timestamp and others are extracted.

$$M_s = S(M) \quad (1)$$

$$OTP \rightarrow [OTP] \quad (2)$$

Cleaned and normalized SMS is used for analyzing, and at the same time, an SHA-256 hash is produced for identifying messages.

$$H_m = SHA256(M_s) \quad (3)$$

2. Semantic Similarity and Historical Pattern Detection

The semantic similarity detection and historical pattern recognition take place through the sanitized SMS being converted into a semantic vector using the all-MiniLM-L6-v2 Sentence Transformer:

$$\mathbf{E}_m = f_{ST}(M_s) \quad (4)$$

$\mathbf{E}_m$ = current message's embedding vector

f_{ST} = all-MiniLM-L6-v2 Sentence Transformer

M_s = sanitized SMS.

$$\mathbf{E}_m \in \mathbb{R}^{384} \quad (5)$$

$\mathbb{R}^{384}$ = 384-dimensional real-valued vector space.

The created embedding is then compared with the already stored message patterns through the use of MongoDB Atlas Vector Search. Cosine similarity is used to measure semantic closeness:

$$Sim(\mathbf{E}_m, \mathbf{E}_p) = \frac{\mathbf{E}_m \cdot \mathbf{E}_p}{\|\mathbf{E}_m\| \|\mathbf{E}_p\|} \quad (6)$$

$\mathbf{E}_m$ = current message's vector

$\mathbf{E}_p$ = previous/stored pattern's vector

Sim = cosine similarity between the two vectors.

A pattern is considered sufficiently similar when:

$$Sim(\mathbf{E}_m, \mathbf{E}_p) \geq \tau \quad (7)$$

where τ represents the experimentally selected similarity threshold. When there is similarity detected, the message count of the pattern increases:

$$C_g^{(t+1)} = C_g^{(t)} + 1 \quad (8)$$

When the accumulated count reaches the confirmation threshold,

$$C_g \geq C_{th} \quad (9)$$

the pattern can be marked as confirmed spam. The specified threshold is:

$$C_{th} = 5 \quad (10)$$

If the threshold is not reached, the message proceeds to LangGraph analysis. For a new pattern, a **PENDING** semantic group is created and the message is forwarded for deeper analysis.

3. LangGraph-Based Multi-Agent Analysis It will be in charge of managing the process flow by coordinating the specialized agents. It is not just one step but rather many steps that will be involved in performing the SMS analysis. The implementation contains three principal agents:

TABLE II
MULTI-AGENT FUNCTIONS AND OUTPUTS

Agent	Purpose	Functions / Processing	Output
OTP Agent	Analyzes OTP-related content for risk.	OTP detection, intent identification, context analysis, legitimacy assessment, risk scoring, and explanation.	OTP presence, intent, risk score, explanation.
URL Agent	Identifies suspicious URLs and destinations.	URL extraction, redirect analysis, domain analysis, webpage analysis, threat detection, and classification.	URL classification, risk score, domain information, explanation.
Contact + Behavior Agent	Evaluates sender trust and behavioral patterns.	Contact verification, sender classification, risk-signal extraction, impersonation detection, reputation analysis, text similarity, and repetitive behavior detection.	Trust score, risk level, sender classification, behavioral signals, explanation.

4. Result Storage and Integrity

The output data will be in the following sequence: the hash of the message, the id of the semantic group, classification, risk, path of processing, timestamp, previous hash, and integrity hash. Where implemented, record integrity can be represented as:

$$H_i = SHA256(H_{i-1} \parallel D_i) \quad (11)$$

H_i is the current integrity hash

H_{i-1} is the previous hash

D_i represents the current record data.

The processing path is represented as:

$$P \in \{\text{SEMANTIC_FAST_PATH, LANGGRAPH}\} \quad (12)$$

5. Explainability and User Response The final system output is made up of the following items: classification, risk score, signals identified and explanation. They all come about from the outputs from the analysis systems and processed by an LLM using Groq API. The final output takes the form of:

$$O_{final} = \{C_{final}, R_{final}, E, S\} \quad (13)$$

where E stands for the explanation and S represents the stands for the detected risk signals. The result is then displayed in the mobile application.

B. Securing MongoDB

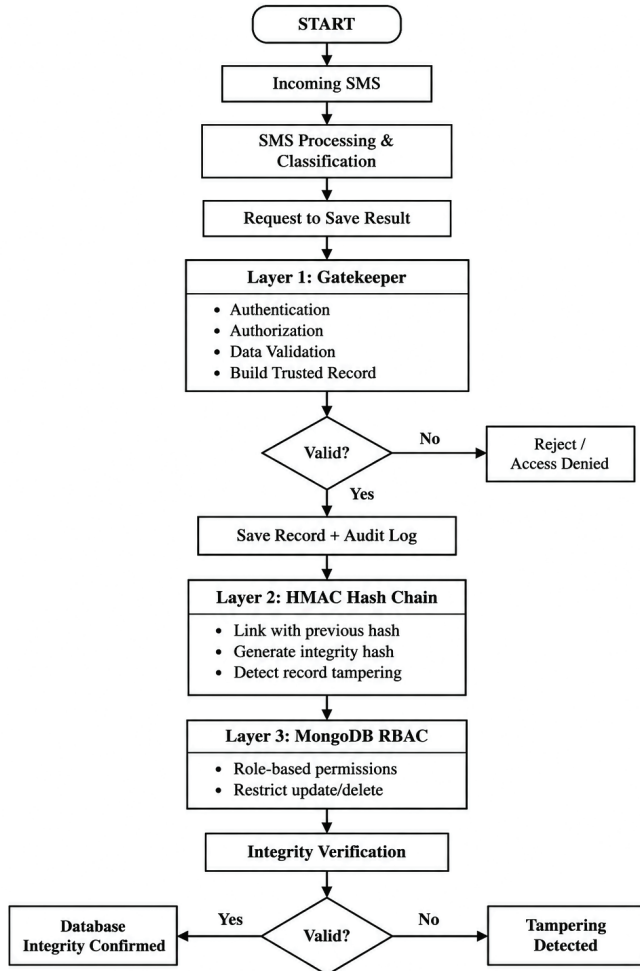


Fig. 2. Three-layer security mechanism for securing MongoDB storage.

In MongoDB storage layer, there is a **three-tiered security model**, which comprises of the following: Gatekeeper security model, Hash chain integrity model, and MongoDB Atlas permissions. Gatekeeper security model involves authentication of the application, authorization, validation of input data, trusted record creation, and logging before writing into `message_records`. Hash chain integrity model uses a **HMAC Hash chain** which links all records with their predecessor with respect to `previous_hash`, `chain_seq`, and `integrity_hash`. The access to MongoDB via MongoDB Atlas permissions limits the capabilities of the application, and prevents it from performing any update or deletion operations on the history collections.

TABLE III
 ATTACK VECTORS AND SECURITY MECHANISMS

No.	Attacker / Attack Vector	Stopped By	Mechanism	Outcome	Result
1	Sends a malformed or malicious request through the app (wrong fields, forged actor, bad values)	Layer 1 – Authentication	Credential checked against ACTOR-SECRET; generic error on failure	Prevented	Request rejected before touching the database
2	Authenticated actor attempts an unauthorized action, such as deleting or updating history	Layer 1 – Authorization	Permission table maps (actor, operation, resource); destructive operations use an empty allow-list	Prevented	ACCESS-DENIED logged to AUDIT-LOGS; nothing saved
3	Sends extra or forged fields such as integrity-hash timestamp, chain-sequence, or fake risk score	Layer 1 – Validation	Field whitelist per operation; server-only fields rejected; range and size checks applied	Prevented	Request rejected; trust fields are server-generated
4	Steals MongoDB connection string/password and connects directly	Layer 3 – Atlas RBAC	Database user <code>scamshield_app</code> has a custom role without update/delete privileges on history collections	Prevented	Atlas rejects unauthorized update/delete actions
5	Admin/developer directly edits a record, e.g., SPAM to SAFE	Layer 2 – Hash Chain	Modified record's HMAC does not match the recomputed integrity stamp	Detected	Verifier identifies the tampered record
6	Admin deletes a record from the middle of the chain	Layer 2 – Hash Chain	<code>chain_seq</code> gap and broken <code>previous_hash</code> link	Detected	Verifier identifies the broken chain position
7	Admin inserts a fake record without the secret key	Layer 2 – Hash Chain	HMAC cannot be forged without secret key K	Detected	Invalid integrity stamp is flagged

III. RESULTS AND DISCUSSIONS

The proposed ScamShield AI framework was evaluated across semantic pattern matching, threshold performance, processing latency, multi-agent execution, and MongoDB integrity. The results demonstrate the system's ability to identify previously observed scam patterns while performing deeper analysis for previously unseen messages.

A. SMS Capture and Transmission Time

The SMS capture and transmission overhead was evaluated to assess system responsiveness. As shown in Table IV, it contributes a small portion of the overall latency, with most time spent on backend processing and AI analysis.

TABLE IV
 SMS CAPTURE AND TRANSMISSION OVERHEAD

Parameter	Time (s)
Phone-to-server latency	6.273
Backend processing time	5.888
Capture + transmission	0.386

B. Semantic Embedding Performance and Threshold Evaluation

The semantic embedding component was evaluated using 174 SMS messages: 108 scam messages in 18 scam families, 45 legitimate messages, and 20 hard-legitimate messages (legitimate messages resembling scams, such as genuine OTP and transactional alerts). The similarity analysis showed that messages belonging to the same scam family achieved substantially higher cosine similarity than messages from different scam families or legitimate messages. The observed similarity values and separation margin are summarized in Table V.

TABLE V
SEMANTIC SIMILARITY EVALUATION

Comparison Type	Avg. Cosine Similarity
Same scam family (n = 270 pairs)	0.4723
Different scam families (n = 5,508)	0.2571
Legitimate vs. anything (n = 9,100)	0.1907
Hard-legitimate vs. scam (n = 2,160)	0.2459
Separation margin	0.2152

The embedding model showed higher cosine similarity within the same scam family (0.4723) than across different scam families (0.2571) and legitimate messages (0.1907). A cosine-similarity threshold of 0.7 was used to evaluate paraphrase recall across scam families, as shown in Fig. 3. The results indicate that semantic matching captures scam-related patterns, although some scam paraphrases may fall below the threshold, limiting detection performance.

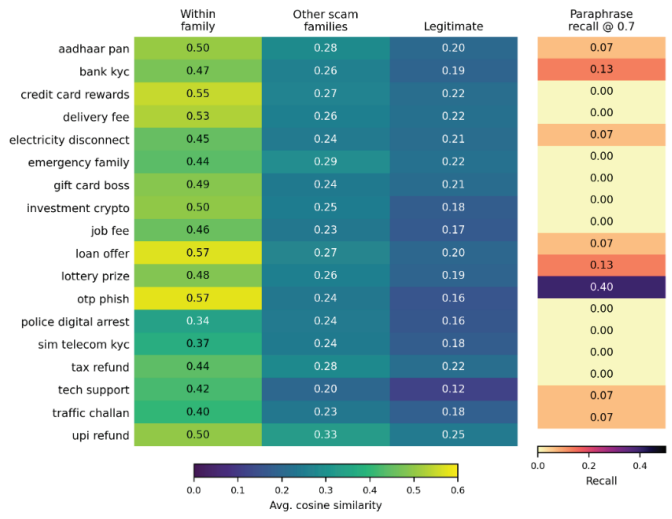


Fig. 3. Per-family semantic similarity and paraphrase recall at a cosine-similarity threshold of 0.7.

Table VI presents the performance obtained at selected cosine-similarity thresholds. The results show that increasing the threshold improves precision in the tested settings but reduces recall. Therefore, threshold selection should consider the trade-off between detecting scam messages and limiting incorrect matches.

TABLE VI
SEMANTIC MATCHING PERFORMANCE AT SELECTED THRESHOLDS

Threshold	Precision	Recall	F1-score
0.500	0.322	0.437	0.370
0.525	0.360	0.370	0.365
0.600	0.505	0.178	0.263
0.700	0.600	0.056	0.102

C. Fast-Path Detection Performance

The fast-path mechanism was evaluated using a message whose pattern had already been classified as CONFIRMED_SPAM. In this case, the system skipped the complete LangGraph multi-agent workflow and reused the previously identified pattern. The complete processing time was approximately 2.133 seconds.

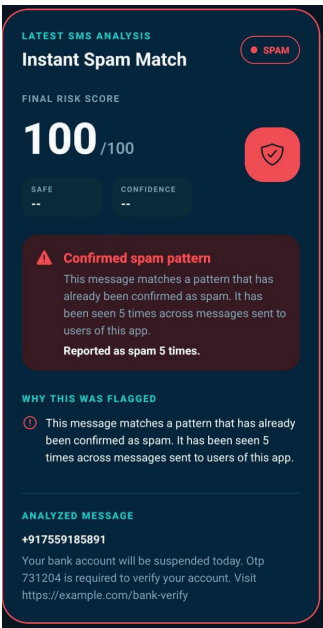


Fig. 4. Prototype response for a previously confirmed spam pattern after reaching the five-report threshold.

Fig. 4 shows the prototype response when an incoming message matches a previously confirmed spam pattern. The pattern has been reported five times, allowing the system to classify the message as CONFIRMED_SPAM and provide an immediate spam warning without executing the complete LangGraph multi-agent workflow.

TABLE VII
FAST-PATH PROCESSING TIME

Processing Step	Duration (s)
Generate semantic embedding	0.480
Database lookup and similarity comparison	1.065
Pattern status lookup	0.166
Persist message record	0.409
Total Fast-Path Time	2.133

The result demonstrates that previously confirmed spam

can be processed without executing the complete multi-agent workflow, thereby reducing redundant analysis.

D. Multi-Agent Processing Latency and Agent Performance

The processing latency of the proposed multi-agent pipeline was evaluated using **30 SMS messages**. The measured latency statistics and agent-level execution times are presented in Table VIII. The results show variation in end-to-end backend processing time across the tested messages.

TABLE VIII
MULTI-AGENT PROCESSING LATENCY

Metric	Latency (s)
Total backend time	7.918
Minimum backend time	5.888
Maximum backend time	9.863
LangGraph processing	7.452
OTP Agent	4.564
URL Agent	6.566
Contact Agent	1.616

The results show that the URL Agent had the highest latency, while the LangGraph stage accounted for most of the processing time. The estimated sequential and parallel execution times are compared in Table IX, with the sequential time calculated from the individual agent latencies.

TABLE IX
ESTIMATED PARALLEL VS. SEQUENTIAL AGENT EXECUTION

Execution Mode	Time (s)
Sequential (estimated)	12.746
Parallel (LangGraph)	7.452
Time saved	5.294

The estimated sequential execution time is approximately 12.746 s, whereas the observed LangGraph processing time was 7.452 s. This corresponds to an estimated reduction of approximately 41.5% in processing time when the specialized agents are executed through the parallel workflow.

The functional operation of the multi-agent pipeline was also verified using representative suspicious SMS messages. Fig. 5 illustrates the prototype output, showing the final LangGraph risk assessment together with the individual OTP, URL, and Contact Agent outputs.

As shown in Fig. 5, the LangGraph workflow presents the outputs of the specialized agents and produces an overall risk assessment for the analyzed SMS. The interface displays the individual agent scores, including the OTP, URL, and Contact Agents, together with the detected factors contributing to the final classification.



Fig. 5. LangGraph-based multi-agent analysis and aggregated risk assessment.

The functional performance of the individual agents was evaluated using multiple test cases. The results are presented in Table X.

TABLE X
INDIVIDUAL AGENT FUNCTIONAL PERFORMANCE

Agent	Tests	Passed	Failed	Pass Rate
OTP Agent	33	33	0	100%
URL Agent	27	25	2	92.59%
Contact Agent	32	29	1	90.63%

The agent-level evaluation indicates consistent functional behavior across the tested cases, with only limited failures observed. The OTP Agent achieved a 100% pass rate, while the URL and Contact Agents achieved pass rates of 92.59% and 90.63%, respectively. These results support the functional operation of the individual detection components under the evaluated test conditions.

E. Overall SMS Classification Performance

The overall classification performance of ScamShield AI was evaluated on 174 SMS messages using a risk-score threshold of 35, at which messages with a score greater than or equal to 35 were classified as scams. The evaluation considered both scam and legitimate messages to assess the system’s ability to detect fraudulent messages while minimizing false alarms. The classification results were analyzed using a confusion matrix and standard performance metrics, as presented in Table XI and Table XII.

TABLE XI
CONFUSION MATRIX FOR OVERALL SMS CLASSIFICATION

Actual Class	Predicted Scam	Predicted Safe
Scam	78 (TP)	6 (FN)
Safe	39 (FP)	51 (TN)

TABLE XII
OVERALL SMS CLASSIFICATION PERFORMANCE METRICS

Metric	Result
Accuracy	74.1%
Precision	66.7%
Recall (Sensitivity)	92.9%
Specificity	56.7%
F1-score	77.6%
False-Positive Rate	43.3%
ROC-AUC	0.882

The system achieved an accuracy of 74.1%, precision of 66.7%, recall of 92.9%, and an F1-score of 77.6%. The high recall indicates that most scam messages were successfully detected. However, the specificity of 56.7% and false-positive rate of 43.3% indicate that a considerable number of legitimate messages were incorrectly flagged as scams. The ROC-AUC of 0.882 suggests that the system can distinguish between scam and legitimate messages across different risk-score thresholds. Further calibration is required to reduce false alarms while maintaining high scam detection performance.

F. Explainable AI and Risk Interpretation

An Explainable AI (XAI) layer was incorporated to convert the outputs of the OTP, URL, and Contact Agents into human-readable explanations. The **Groq API** is used to generate concise explanations and safety recommendations from the detected risk factors.

Fig. 6 shows an example of the generated explanation, highlighting factors such as OTP requests, suspicious links, urgent language, and sender-related risks. This improves transparency and helps users understand why the SMS was flagged and what action should be taken.

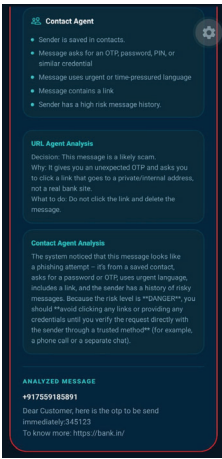


Fig. 6. Explainable AI

G. MongoDB Integrity and Tamper-Resistance

The MongoDB Atlas integrity test suite evaluated **12 security scenarios** against modification, deletion, forged continuation, and concurrent-write scenarios. The results are presented in Table XIII.

TABLE XIII
MONGODB INTEGRITY TEST RESULTS

Test Category	Passed	Failed
Update tamper	3	0
Delete tamper	3	0
Concurrent-write fork attempt	0	3
Forged continuation	3	0
Total	9	3

The update and delete tampering tests passed for all three collections, indicating that direct modification and deletion attempts were rejected by the configured database role. The forged-continuation tests also passed because records generated without the legitimate HMAC key were rejected during chain verification. However, all three concurrent-write fork tests failed because the unique `chain_seq` index did not prevent a fork during concurrent writes. This identifies a limitation in the current chain-sequencing mechanism.

Representative test outputs for the `audit_logs` and `spam_patterns` collections are shown in Fig. 7. The outputs demonstrate successful rejection of unauthorized update and delete operations, successful detection of forged continuations, and the observed failure of the concurrent-write fork protection.

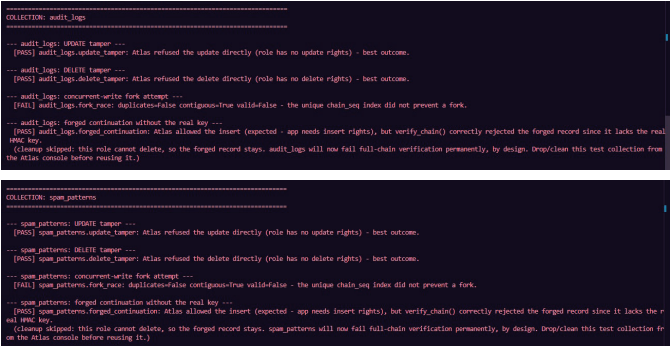


Fig. 7. MongoDB integrity test outputs for the `audit_logs` and `spam_patterns` collections.

IV. CONCLUSION

The experimental evaluation demonstrates the potential of ScamShield AI for SMS smishing detection using semantic matching, multi-agent analysis, and secure threat information management. In the overall classification evaluation of 174 SMS messages, the system achieved 74.1% accuracy, 66.7% precision, 92.9% recall and a 77.6% F1-score at a risk-score threshold of 35. The high recall indicates effective scam detection; however, the false-positive rate of 43.3% highlights the need to improve the classification of legitimate messages. The ROC-AUC of 0.882 further indicates the system's ability to distinguish between scam and legitimate messages across different risk-score thresholds.

The semantic matching and fast-path components enable the identification of previously observed or similar scam pattern, while reducing the need for repeated full analysis.

The security assessment demonstrated protection against direct record modification, deletion, and insertion of forged-records. However, concurrent-write tests revealed a limitation that may lead to inconsistent chain states, indicating the need for improved concurrency control in the threat information storage component.

Future work will focus on reducing false positives, optimizing risk-score thresholds, improving concurrent-write handling, and evaluating the system on larger and more diverse datasets. Further optimization of processing latency and extension to other digital threats could enhance the system's reliability and practical applicability.

V. LIMITATIONS

- The approach relies on external APIs, which may introduce network latency, service unavailability, and rate-limiting issues.
- The `chain_seq` mechanism does not prevent forking during concurrent writes, as all three concurrent-write fork tests failed.
- At a risk-score threshold of 35, the system achieved a false-positive rate of 43.3%, indicating that many legitimate messages may be incorrectly flagged as scams.
- The evaluation uses a relatively small dataset of 174 SMS messages, limiting the extent to which the results can be generalized to real-world SMS traffic.
- Previously unseen scam patterns may not be detected reliably, requiring evaluation on larger and more diverse datasets.

VI. FUTURE SCOPE

- **Enhance Detection Precision:** Use a larger dataset for training and testing that consists of both authentic and scam messages.
- **Language Support for Regional:** Apply the model for detection in Hindi, Marathi, and other regional languages as well..
- **Update Scam Database:** Add newly verified scam numbers, URLs, and message patterns to the MongoDB threat database.
- **Scam Database Update:** Add new SMS to the MongoDB database of scam phone numbers, URL, and scam message patterns.
- **Improved Privacy:** Privacy should be improved by improving secure message handling and decreasing stored user data.

REFERENCES

- [1] TechTo Networks, "Real bulk sms rates in india – 2025," Online, 2025, accessed: Sep. 27, 2026. [Online]. Available: <https://www.techtonetworks.com/post/bulk-sms-rates-in-india>
- [2] Telecom Regulatory Authority of India, "Over 7 lakh notices, 5.6 lakh restrictions: Trai tightens grip on spam telemarketers in 2025," Press Release, Feb. 2026, accessed: Sep. 27, 2026. [Online]. Available: https://www.trai.gov.in/sites/default/files/2026-02/PR_No.20of2026.pdf
- [3] M. G. M. Chaudhari and D. V. Likhar, "The rise of phishing and vishing in digital payments: An analytical review of ncrb data (2016–2025)," *IJAIML*, vol. 6, no. 8s, pp. 544–552, Aug. 2026.

- [4] K. L. Shilpa and M. S. Hanspal, "Investigation of cyber frauds in india: Legal and digital forensic challenges in the era of upi and online banking," 2026.
- [5] S. Verma, J. Sharma, K. Kaushik, and V. Vyas, "Mounting cases of cyber-attacks and digital payment," in *Digital Payment and Cyber Security*. IGI Global, 2023.
- [6] Sentinex, "What is smishing?" Online, 2026, accessed: Sep. 27, 2026. [Online]. Available: <https://sentinex.com/smishing>
- [7] S. Mishra and D. Soni, "Smishing detector: A security model to detect smishing through sms content analysis and url behavior analysis," *Future Generation Computer Systems*, vol. 108, 2020.
- [8] M. M. A. Pritom *et al.*, "Short message service (sms) phishing attacks and defenses: A systematic review," *arXiv*, 2026.
- [9] P. Prajapati *et al.*, "Smishing: A sms phishing detection using various machine learning algorithms." Springer, 2023.
- [10] A. K. Jain, K. Kaur, N. K. Gupta, and A. Khare, "Detecting smishing messages using bert and advanced nlp techniques," *SN Computer Science*, 2025.
- [11] Y. Wang, H. Zhai, C. Wang *et al.*, "Can you walk me through it? explainable sms phishing detection using large language model-based agents," in *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2025, pp. 37–56.
- [12] S. Agarwal, G. Suarez-Tangil, and M. Vasek, "An overview of 7726 user reports: Uncovering sms scams and scammer strategies," *arXiv*, 2025.
- [13] H. Xu, A. Qadir, and S. Sadiq, "Malicious sms detection using ensemble learning and smote to improve mobile cybersecurity," *Computers & Security*, 2025.
- [14] <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>.