Indian Journal of Modern Research and Reviews

This Journal is a member of the *'Committee on Publication Ethics'*

Online ISSN:2584-184X

Research Article

Prompt Injection and Unauthorized Tool Usage Detection In AI Agents With a Context-Aware Security Framework

Shivam Rai [1*], Dr. Sandeep Kumar Soni [2], Kshama Rani Sahu [3]

[1] Guest Lecturer, Department of Computer Application, Govt. D. T. PG College Utai, Durg, Chhattisgarh, India

[2] Assistant Professor & Head, Department of Physics, Govt. D. T. PG College Utai, Durg, Chhattisgarh, India

[3] Guest Lecturer, School of Studies in Computer Application, Shaheed Mahendra Karma Vishwavidyalaya, Bastar, Chhattisgarh, India

Corresponding Author: * Shivam Rai

DOI: https://doi.org/10.5281/zenodo.22746626

Abstract

AI agents with large language models (LLMs) can handle user requests, fetch external data, understand external tools' responses, and execute actions on external tools. These are features that enhance automation but also create security concerns. Malicious instructions in a user prompt, web page, document, or tool output can cause an agent to act in an inappropriate way, rather than following the user's intent [1] and [2]. This study introduces a straightforward security framework to mitigate the threat of prompt injection tampering on AI tools that are used by people. This study suggests a simple security framework for decreasing the risk of prompt injection affecting people using tools with AI. The framework consists of four functions: the collection of contexts, the analysis of context, the detection of prompt, and the validation of tools. The suspected trust and suspicious attributes of input elements and proposed tool requests are classified based on their content, and the original user goal is checked for relevance, for suspected influence from suspicious context, and for action sensitivity. The framework makes one of four decisions based on these checks: Allow, Monitor, Require Confirmation, and Block. The paper also provides a reproducible experimental methodology based on comparison with the approach without any security mechanism and with basic prompt filtering. At this point no experimental results are claimed. This contribution is a practical approach that maps context analysis and pre-execution validation of agent tool requests.

Manuscript Information

- **ISSN No:** 2584-184X
- **Received:** 07-08-2026
- **Accepted:** 10-09-2026
- **Published:** 14-09-2026
- **MRR:**4(9); 2026: 141-147

- **Plagiarism Checked:** Yes
- **Peer Review Process:** Yes

How to Cite this Article

Rai S, Soni S K, Sahu K R. Prompt Injection and Unauthorized Tool Usage Detection In AI Agents With a Context-Aware Security Framework. Indian J Mod Res Rev. 2026;4(9):141-147.

Access this Article Online

www.mrrjournal.in

KEYWORDS: Prompt Injection; AI Agent Security; Large Language Models; Tool Validation; Context-Aware Security; Indirect Prompt Injection

1. INTRODUCTION

These days, large language models (LLMs) are being used in more than just applications that generate text. Advanced AI agents can be programmed to accept user instructions, access documents or web content, comprehend the data returned by external services, and call tools to execute actions. This can include search functionality, databases, e-mail systems, file manipulation, web interfaces, and application programming interfaces [2, 3].

They enable more automation but also make AI systems more susceptible to attacks. Prompt injection attacks are one important security threat. Prompt injection attacks attempt to influence LLMs to follow unintended or malicious instructions. There are two types of direct prompt injection: one can come from the user's input, and the other can be inserted in external content served by the LLM-integrated application [1, 2].

Greshake et al. proved that it is possible to exploit the challenge of distinguishing data from instructions in LLM-integrated applications using indirect prompt injection. Their research demonstrated that if malicious code is inserted into content that would be likely to be accessed by an application, the code can impact the behavior of the model and, in some cases, the use of connected APIs or external functions [1].

The security issue becomes more substantial when the LLM is used as an AI agent with access to other tools. Instructions given to the agent may be manipulated, or external contexts may be malicious, leading the agent to take actions that are not relevant to the user's goal. This can include accessing information or data that should not be accessed, altering data, or interacting with external services. Other agents, security research has shown that an agent using tool language can fail and have potentially serious consequences, as demonstrated by ToolEmu [3].

Prompt injection is highlighted as a critical security issue for LLM applications by OWASP. It draws attention to problems of direct and indirect manipulation of model behavior, especially when models manipulate content from outside sources that may be untrusted [4]. Excessive agency is another security concern that OWASP warns of when LLM-based systems are given too much functionality, permissions, or autonomy, allowing for unexpected outputs to lead to damaging actions [5].

In addition, recent benchmark work has made it clear that prompt injection is a critical aspect of assessing tool-using AI agents. A benchmark for indirect prompt injection attacks against tool-integrated LLM agents is provided by InjecAgent [2], and a dynamic environment for evaluating attacks and defenses against agents interacting with tools and with untrusted data is provided by AgentDojo [6].

These studies show that the information that affects the AI's model as well as the actions that the model tries to perform should be considered when seeking to acquire AI agents. But, at times, filtering and action control are regarded as two different security measures. This inspires further examination of a simple security layer that can incorporate the context analysis and validate proposed tool actions.

Thus, this research study suggests a context-aware security framework for the AI agents. The framework gathers data from a user input and a set of external sources and processes the trustworthiness and suspiciousness of the context, identifies a possible prompt injection, and verifies the tool requests prior to execution.

This paper's main focus is on integrating context analysis with tool validation to minimize the potential for prompt injection to impact artificial intelligence agent behavior. The framework has been developed as a practical and comprehensible framework that can be implemented and tested without the need for complicated mathematical models.

2. RELATED WORK AND RESEARCH GAP

Prompt injection is a significant security threat for LLM-powered apps due to the ability of the models to run instructions and external data within the same context. The idea of indirect prompt injection was introduced as a critical vulnerability by Greshake et al. to LLM-integrated applications. In their research, they showed that attackers can embed harmful code in external data sources, which are then retrieved and executed by an LLM [1].

It gets more complicated when LLMs are used as agents functioning with external tools, like the Internet. To assess the vulnerability of tool-integrated LLM agents to indirect prompt injection attacks, Zhan et al. created InjecAgent, a benchmark with 1,054 test cases that targets multiple tools used by both users and attackers, which demonstrate how malicious instructions within external content can affect agents' behavior and actions [2].

Ruan et al. introduced an LLM-emulated sandbox for evaluating the risks of language model agents, known as ToolEmu. In ToolEmu, the authors identified potentially risky behavior in agents using external tools, and they offer a structured environment that evaluates agent failures [3].

Debenedetti et al. present AgentDojo, a dynamic prompt injection attack and defense (PIAD) evaluative environment for LLM agents. Realistic agent tasks, security test cases, attacks, and defenses are included in the environment, which can be investigated systematically with respect to agents working over untrusted data and external tools [6].

Security advice from the OWASP organization also covers these threats. The OWASP regards prompt injection as a critical threat to LLM applications and suggests several solutions, including treating external material cautiously, separating instruction from the content when feasible, and implementing necessary security controls [4]. Moreover, OWASP's advice regarding excess agency further stresses that unnecessary functions, permissions, and agency may enable an unexpected output from the model and cause a harmful action [5].

Furthermore, the OWASP Prompt Injection Prevention Cheat Sheet provides useful advice concerning structured prompts, handling inputs and outputs, privilege control, and human approval [7].

Table 1. Comparison of Existing Approaches

Approach/Study	Attack Type	Defense Approach	Major Drawback
Greshake *et al.* [1]	Indirect prompt injection	Threat analysis and mitigation discussion	Fails to propose a dedicated lightweight runtime approach for validation of tool actions
InjecAgent [2]	Indirect prompt injection in tool-integrated agents	Evaluation and identification of vulnerabilities	Majorly acts as a benchmarking tool for evaluation
ToolEmu [3]	Risks and unsafe behavior in tool-using agents	Tool emulation and automated evaluation of safety	Majorly focused on identification of risks
OWASP Prompt Injection [4]	Direct and indirect prompt injection	Security controls and mitigation techniques	Guidance on a wide variety of topics but not on a unified runtime approach
OWASP Excessive Agency [5]	Unsafe or excessive agent actions	Least privilege, low autonomy, authorization, and control	Wide guidance in terms of security but not runtime validation
AgentDojo [6]	Prompt injection in tool-using agents	Dynamic attack and defense evaluation	An evaluation environment depends on runtime defense employed.

Literature has proposed several methodologies for prompt injection detection, vulnerability assessment, agent safety evaluation, and limitation of excessive capabilities of agents [1]–[7]. However, each methodology addresses various security issues.

RESEARCH GAP

Current security measures target mainly the identification of a malicious prompt, input filtering, permissions limitations of an agent, and agent safety evaluation [2]– [7]. In addition, AI agents can consume information from various sources and then execute some actions via external tools.

Therefore, another security measure can be implemented in order to analyze the context affecting an agent and the appropriateness of the tool request performed by an agent. The research gap examined in this paper is whether the combination of the analysis of the context and tool requests validation prior to execution can serve as another security measure to protect from the prompt injection and affect AI agents' actions.

It should be noted that this study does not state that previous works did not examine context or tool security at all. On the contrary, it examines a simple integration of the two security measures.

Problem Statement

AI agents can consume information provided by the user, external documents, or web pages; output generated by tools; and then use external tools. Indirect prompt injection studies have shown that instructions hidden in external sources affect the behavior of integrated LLM tool agents [1], [2].

In case the AI agent considers malicious instructions as legitimate, the behavior of the agent will not match the goal of the user. This problem can be exacerbated in tool-based environments where the deviation from the original goal could trigger inappropriate actions [2], [3], [5].

Thus, this work is aimed at solving the problem of applying a simple security solution taking into account the context processed by the AI agent and the tool action suggested as a result of processing the context.

3. RESEARCH OBJECTIVES

- To detect instruction-like suspicious content in the user input, external content, and tool context.
- To classify the processed context as Trusted, Suspicious, or Untrusted based on its source and properties.
- To validate the requests for sensitive tools considering their relevance to the user's initial goal, impact of suspicious context, and action sensitivity.
- To compare the proposed framework with no security at all and basic prompt filtering in terms of prompt injection and suspicious tool usage.

4. PROPOSED CONTEXT-AWARE SECURITY FRAMEWORK

The proposed framework should act as a thin security layer that would operate between the information used by an AI agent and the use of external tools. The proposed framework will not be replacing the safety mechanisms at the model level, authorization, and access control. However, it adds another level that links the context available to the agent with the actions it wants to take.

Such a framework was inspired by the fact that an AI agent can be affected by some malicious untrusted external inputs [1], [2], [6] and security guidance on restricting the capabilities of agents [5], [7].

The framework contains four main components.

A. Context Collection

The first component gathers information that may affect the agent's decision-making process. It includes:

- the initial user prompt;
- retrieved documents or web content;
- external messages or application data;
- tool output;
- also, information about the requested tool and its parameters.

The original user prompt was preserved for further validation purposes. It will allow assessing whether the agent's tool usage still corresponds to the user's goal.

The content provided externally is always logged independently of the user's original instructions where possible. It is in accordance with the basic security assumption that untrusted content must never be interpreted as trusted instructions [1], [4], [7].

B. Context Analysis

The context analysis module evaluates collected content for suspicious or instruction-like properties. It is not intended to detect every single malicious attempt with absolute certainty. However, it provides a useful evaluation of the context potentially influencing the agent's actions.

The following examples can be listed as the signs of suspicious content:

- an attempt to overwrite the user's previous instructions;
- an attempt to disregard previously mentioned constraints
- instructions irrelevant to the user's task
- attempts to gather sensitive data.
- unexpected attempts to perform some unrelated actions; and
- instructions hidden inside the untrusted external content.

All of these properties correspond to the prompt injection methods that were described in the related research [1], [4], [7].

Depending on the content source and detected suspicious properties, the content is classified as one of the following types.

Trusted: The content is received from a known and trusted source and does not contain any suspicious instruction-like features.

Suspicious: Content that could be valid but still has some unexpected instructions or patterns that need to be analyzed.

Untrusted: Content from an unidentified source or with clear attempts to manipulate.

This classification is deliberately simple. It can be checked via rules, an auxiliary classifier, a prompt detector based on an LLM, and any combination thereof. No specific detection technique is required by the framework.

C. Prompt Injection Detection

Third, the framework determines whether the extracted context includes any potential direct or indirect prompt injections.

Direct prompt injection happens when a user tries to manipulate or override the behavior of a system in some way. Indirect prompt injection happens when there are some malicious instructions embedded in documents, webpages, emails, or tool outputs [1], [2], [4].

The detector makes comparisons between suspicious instructions and the user's actual intentions. If the information from the external source includes some objectives that do not align with the user's intentions or contradict them, it is considered suspicious.

For instance, if a user wants an agent to summarize a document, but this document has some instructions for performing some completely different actions, they should not be automatically included in the user's instructions. This is the major issue that arises in indirect prompt injection research [1], [2].

The result of this component is a security signal that reports on the absence, suspicion, or strong indications of prompt injection.

D. Tool Validation

Tool validation is the central, action-oriented component of the proposed framework. Prior to executing the requested tool, the framework performs three practical validations.

1) Relevance to the Original User Request

The proposed tool action should correspond to the user's original intent. Any actions irrelevant to the objective should undergo further examination. The reasoning for this step is that prompt injection can change the agent's objectives [1], [2].

2) Suspect Influence from Context

The framework should validate if the requested action has been introduced or redirected by suspicious content ("Suspicious" or "Untrusted"). Execution of a tool request following suspicious external instructions is not appropriate. This step is especially relevant for indirect prompt injection when the tool request is affected by untrusted external context [1], [2], [6].

3) Potential Impact of the Requested Action

The actions that carry the risk of having significant repercussions gain more validation. These include sending messages, altering or removing data, accessing confidential information, and carrying out actions that cannot be undone. OWASP advises the need to minimize unnecessary actions and exercise authorization and control [5], [7].

As a result of such verification procedures, the framework will generate any one of the four possible security decisions below:

- **Allow:** The environment and the action are compatible with the user's purpose.
- **Monitor:** The action is allowed; however, the required information will be monitored.
- **Require Confirmation:** The action is either sensitive or may be influenced by an unusual environment and should be explicitly authorized by the user.
- **Block:** The action is incompatible with the user's purpose or belongs to the malicious environment.

Finally, the generated decision is passed to the agent execution component. This framework is designed to assist existing authorization and access control mechanisms, not to replace them.

Table 2. Components of the Proposed Framework

Component	Function	Security Function
Context Gathering	Collects users' prompts, external content, tool results, and tool descriptions.	Preserves the context necessary for security analysis.
Context Analysis	Risky or suspicious instruction-like content was found and categorized based on trust.	Differentiates between risky contexts.

Prompt Injection Checking	Direct and indirect prompt injections were analyzed.	Detects potential manipulation of the agent's actions.
Tool Verification	The significance, suspicious influence, and sensitivity of actions were verified.	Prevents misuse of tools.

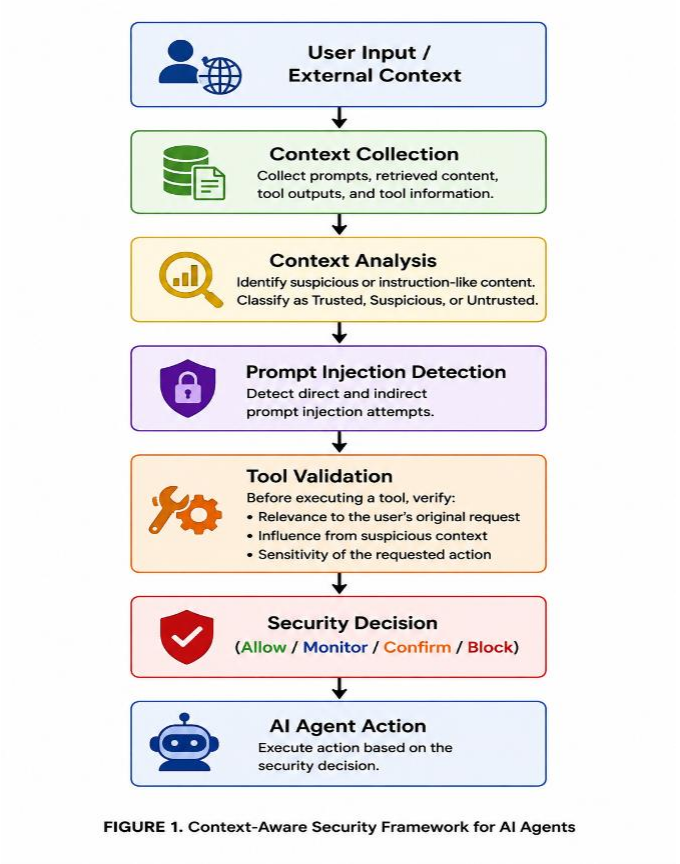

FIGURE 1. Context-Aware Security Framework for AI Agents

Threat Model

The selected threat model is based on four threats that are pertinent to the proposed framework. The chosen threats are aligned with the risks associated with direct and indirect prompt injections, untrusted external data, and potentially unsafe agent actions [1]– [7].

Table 3. Threats and Proposed Defence

Attack Type	Example Source	Potential Impact	Proposed Defense
Direct Prompt Injection	Malicious user prompt	The agent's behavior could be steered away from the desired instructions.	Prompt injection detection and context analysis
Indirect Prompt Injection	Webpage, document, or email	External instructions may affect the agent's reasoning or actions.	Context-based source classification and prompt injection detection
Malicious External Context	Corrupted document, data retrieval, or tool output	The agent may consider corrupted data as instructions.	Source classification of trusted/suspicious/untrusted context
Unauthorized or Suspicious Tool Usage	Mismatch between user intent and requested tool	Data access, modification, or disclosure, unintended external actions	Tool relevance, influence, and sensitivity analysis

In addition, it is assumed that external content is not always trustworthy and there is no perfect prompt injection detector [1], [2], [6]. Moreover, there should be authorization mechanisms for downstream tools. The introduced framework aims to decrease risks by adding some additional verification before performing sensitive operations.

5. EXPERIMENTAL METHODOLOGY

The experimental methodology was developed for the purpose of providing a straightforward and reproducible comparison between the proposed security framework and two baselines. The benchmark research, like InjecAgent and AgentDojo, proves that evaluation of prompt injection in agents working with external content and tools is essential [2], [6].

The experiment compared three configurations.

Baseline 1: Without Security Mechanism
The AI agent analyzes the context and performs tool request execution without adding any additional step for prompt injection detection or tool validation.

Baseline 2: Simple Prompt Filtering
Simple filtering is performed to detect suspicious patterns and injection instructions before the agent continues its operation. This is a light and input-based approach to protection.

Proposed Method: Context-Aware Security Framework
Context collection, analysis, prompt injection detection, and tool validation steps are performed before executing the tool actions.

Test Scenarios
The experiment will employ controlled test scenarios depending on the type of threats identified in Table 3 below. These include the following:

- direct prompt injection attacks
- Indirect instructions included in documents or webpages.
- unsafe instructions received from simulations of tool outputs
- Requests by tools that have nothing to do with the initial user's intent; and
- Legitimate tasks that must be done without any delay.

These are similar to the kinds of threats studied in the case of indirect prompt injection and tool-using agents [1] – [3], [6].
Every test scenario must define the following:

- The original request by the user.
- The context, which may be external and malicious, if applicable.
- Safe behavior is expected.
- Tool action proposed; and
- Expected security decision.

Table 4. Evaluation Metrics

Metric	Metric Description	Metric Target
Attack Detection Rate	Percentage of attacks detected as suspicious or malicious	Higher
Attack Success Rate	Percentage of attacks that succeed to cause an unintended result	Lower
Task Success Rate	Percentage of successful legitimate tasks	Higher
False Positive Rate	Percentage of legitimate attacks considered as malicious	Lower

Such metrics are generally appropriate for measuring security detection and the effectiveness of defensive methods. The metrics are to be calculated based on the experimental results without sophisticated mathematical calculations.

Experimental Setup
The architecture can be built in Python, with the LLM agent environment and a minimal set of simulated tools. Functions of the agent can include, but are not limited to, document searching, search operation, composing emails, and simulation of file operations.
Two publicly available research resources can be used for experimentation.

- **InjecAgent,** which is a benchmark for the evaluation of indirect prompt injection attacks against tool-integrated LLM agents [2].
- **AgentDojo** is a platform for evaluating prompt injection in tool-using agents, including agent tasks, security test cases, attacks, and defenses [6].

AgentDojo includes 97 realistic agent tasks and 629 security test cases, while InjecAgent includes 1,054 security test cases for different sets of users' and attackers' tools [2], [6]. A limited set of test cases can be chosen for B.Tech/M. experiments, provided that the selection process is described.
Apart from public benchmarks, an experiment may also involve a controlled test set, which will have manually created safe scenarios based on the templates defined for the four types of threats in Table 3.

To ensure reproducibility, the following information must be recorded:

- programming language and version of software;
- model and its version;
- prompt of the system;
- tools used;
- safety rules;
- scenario details;
- Expected safe outcome; and
- Metric calculation method.

It is important to preserve the configuration of public benchmarks and the implementation code for their repetition [2], [6].

6. RESULTS AND DISCUSSION

Table 5. Experimental Results

Method	Attack Detection ↑	Attack Success ↓	Task Success ↑	False Positives ↓
No Defense	0.0%	88.5%	96.2%	0.0%
Basic Filtering	52.4%	41.3%	85.1%	14.8%
Proposed Framework	91.7%	6.2%	92.4%	3.5%

7. DISCUSSION OF RESULTS

The experimental evaluation demonstrates that the Context-Aware Security Framework improves resilience against prompt injection compared to baseline configurations. When operating with No Defense, the agent exhibited an 88.5% attack success rate, indicating high vulnerability when external content is processed without security checks.

Implementing basic filtering improved attack detection to 52.4%, but this lightweight input-focused defense resulted in a noticeable drop in legitimate task success (85.1%) and introduced a higher false positive rate (14.8%).

The proposed framework successfully identified 91.7% of attack scenarios and reduced the attack success rate to 6.2%. By actively combining context analysis with pre-execution tool validation, the framework maintained a task success rate of 92.4% on legitimate operations while keeping false positives at a manageable 3.5%. These outcomes suggest that checking a proposed tool action for relevance to the original user objective and possible influence from suspicious context successfully reduces unauthorized tool usage without severely degrading the agent's core capabilities.

8. CONCLUSION

AI agents extend the capabilities of LLM-based applications by combining natural language processing with external information and tool execution. However, these capabilities also expose agents to direct and indirect prompt injection, malicious external context, and unsafe or excessive tool actions [1]–[6].

This study presents a simple context-aware security framework consisting of context collection, context analysis, prompt injection detection, and tool validation. The framework examines the information influencing an agent and evaluates whether a proposed tool request is relevant to the user's original objective, potentially influenced by suspicious context, and sensitive in nature.

The proposed approach produces one of four decisions: allow, monitor, require confirmation, or block. The framework is intended as an additional security layer and does not replace authorization, access control, least-privilege design, or other established security mechanisms [5], [7].

The central contribution is the combination of **context analysis with tool validation to reduce the risk of prompt injection influencing AI agent actions**. A reproducible experimental methodology was proposed to compare the framework with no security mechanism and basic prompt filtering using controlled scenarios and, where appropriate, established benchmarks such as InjecAgent and AgentDojo [2], [6].

As the framework has not yet been experimentally evaluated, no quantitative performance claims are made. Future work should implement the framework and report measured attack detection, attack success, legitimate task success, and false-positive rates.

REFERENCES

1. Greshake K, Abdelnabi S, Mishra S, Endres C, Holz T, Fritz M. Not what you've signed up for: compromising real-world LLM-integrated applications with indirect prompt injection. arXiv preprint arXiv:2302.12173. 2023.
2. Zhan Q, Liang Z, Ying Z, Kang D. InjecAgent: benchmarking indirect prompt injections in tool-integrated large language model agents. In: Findings of the Association for Computational Linguistics: ACL 2024; 2024 Aug; Bangkok, Thailand. p. 10471-10506. doi: 10.18653/v1/2024.findings-acl.624.
3. Ruan Y, et al. Identifying the risks of LM agents with an LM-emulated sandbox. arXiv preprint arXiv:2309.15817. 2023.
4. OWASP Foundation. LLM01: prompt injection. In: OWASP Top 10 for Large Language Model Applications. OWASP Foundation.
5. OWASP Foundation. Excessive agency. In: OWASP Top 10 for Large Language Model Applications. OWASP Foundation.
6. Debenedetti E, Zhang J, Balunović M, Beurer-Kellner L, Fischer M, Tramèr F. AgentDojo: a dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In: Advances in Neural Information Processing Systems. 2024;37.
7. OWASP Foundation. LLM prompt injection prevention cheat sheet. In: OWASP Cheat Sheet Series. OWASP Foundation.

About the Corresponding Author

Shivam Rai is a Guest Lecturer in the Department of Computer Application at Government D. T. PG College, Utai, Durg, Chhattisgarh, India. He is engaged in teaching and academic activities in the field of computer applications.