

Methodology to Identify Issues and Improve the Robustness of AI Agents

Author(s): Aleksandr Meshkov

Independent Researcher. Montenegro.

Corresponding Author: alekslynx90@gmail.com

Received: October, 2025 Published: January, 2026

ARTICLE INFO

Keywords:

AI Agents; Evaluation AI; Robustness AI;
Taxonomy of Errors

© 2026 by the Author(s). This open-access article is distributed under a Creative Commons Attribution (CC-BY) 4.0 license, making research freely available to the public and supporting a greater global exchange of knowledge and human experiments.



ABSTRACT

AI agents based on large language models (LLMs) are becoming a key tool for automating complex tasks. Unlike general LLMs that simply generate text, modern agents are able to independently plan actions, call external tools and APIs, work with knowledge, and make decisions based on multi-stage analysis of the situation. However, with the increasing complexity of such types of systems, a critical problem of ensuring their robustness arises. This work presents a systematic approach to identifying and classifying problems in AI agent robustness. The provided problem taxonomy describes nine common problems, which might happen during the execution tasks in an AI agent. For practical usage, a comprehensive evaluation methodology is proposed, including metamorphic testing to evaluate the resistance to changes in input data, checking the correctness of working with information sources, analyzing the flow of tasks inside an AI agent, monitoring the tool usage, and evaluating the quality of the final results. The methodology contains specific metrics with success criteria and approaches to their implementation. It is shown that the proposed system covers all identified categories of errors and makes it possible to evaluate the robustness of AI agents not only at the level of components, but also the interaction part and as a system overall.

1.0 Introduction

AI agents represent a completely new generation in the development of artificial intelligence systems. If LLMs are limited to generating text based on a query, then AI agents have the ability to autonomously act in the environment provided to them to complete assigned tasks (Bandi et al., 2025). AI agents can analyze the situation, select the necessary tools to solve the problem, access external APIs, MCPs, and databases, sequentially perform actions, and adjust the work plan based on the obtained intermediate results (Huang et al., 2024).

The market is actively investing in the development of agent-based AI systems. According to analyst forecasts, the AI agent market will grow from 5.3-5.4 billion dollars in 2024 to 50-52 billion by 2030, showing a CAGR of about 41-46% (Grand View Research, 2025). Companies see AI agents as a tool for automating complex business processes that require not only working with data, but also meaningful decision-making.

Today, AI agents already exist in various fields and demonstrate practical value, which can be seen in such areas as AI agents in the support service, which are capable of maintaining multi-step dialogues, extracting relevant information, and, if necessary, if it is not possible to resolve the issue, transferring it to a human operator. Another example would be AI agents in medicine that are able to perform initial patient diagnosis, work with the patient's history, and offer basic treatment options while complying with HIPAA requirements.

The scope and use of AI agents are constantly growing, for example, personal AI assistants will be able to coordinate work with multiple services (calendar, email, travel, finance) to

manage complex daily tasks, research agents will autonomously perform literature reviews, design experiments and synthesize results in various disciplines, and in a corporate environment, workflow automation will be able to minimize human participation in the company's daily business processes.

However, companies implementing AI agents face serious robustness and quality issues. Research shows that about 95% of generative AI projects fail to achieve their goals due to insufficient quality evaluation and data issues (The GenAI Divide: State of AI in Business 2025, 2025). AI agents may produce results that do not meet user expectations or exhibit unpredictable behavior when task conditions change (Shavit & Agarwal, 2024).

This is still complicated by the fact that modern AI agents use advanced search technology to work with external knowledge bases. This adds additional complexity because the AI agent must not only plan actions correctly, but also effectively search for information, correctly interpret the data that was found, and check the reliability of this data from different sources.

The purpose of this work is to solve three problems. Firstly, to systematize typical errors that arise when working with AI agents and create a structured classification. Secondly, develop an evaluation methodology that identifies each type of problem using specific metrics and evaluation practices. Thirdly, to offer practical recommendations for building continuous evaluation of the quality of AI agents in the development and operation processes.

2.0 Evolution of Methods to Evaluate AI Agents

The evolution of LLM from simple text generators to autonomous AI agents has created new challenges for text evaluators. The key problem is that the behavior of such AI agents is highly dependent on the type of input data, so that means even a small change in the request can lead to absolutely different results.

To solve this problem, metamorphic testing, borrowed from classical software engineering, was initially adapted. The idea of the method is simple, if transform the input data in a certain way (for example, rephrase the input or change the order of the elements inside the input), then there will be the possibility to predict how the result should change. If the actual output of the AI system does not meet expectations, then a problem has been detected (Chen et al., 2020).

Researchers have applied this approach to language models in several ways. Reddy et al., (2025) used metamorphic relationships to identify bias in LLaMA and GPT models, demonstrating that even minimal changes in inputs can significantly impact the tone and content of responses. Wu et al. (2024) developed the DrHall system to detect factual hallucinations based on the hypothesis that fictitious responses exhibit greater instability when the task is repeated through different query options.

A further method was presented by Giramata et al. (2025), proposing to prioritize metamorphic relations based on linguistic diversity. Their approach improved robustness error detection by 22% compared to randomly selecting metamorphic relations on the input data.

The transition to agent-based architectures has given rise to qualitatively new types of problems. When Yao et al. (2023) presented the ReAct paradigm, where reasoning and action are combined into a single cycle, that is, the agent thinks about the next step, carries it out, analyzes the result, and plans further actions based on it, it's created new categories of errors, such as incorrect planning of sequences of actions, loss of important information between steps, and misinterpretation of the results of external tools.

Researchers paid special attention to AI agents using external knowledge bases. Yu et al. (2024) noted the lack of a proper methodology for evaluating such AI systems and identified

three key quality characteristics that need to be evaluated: the extent to which the information found corresponds to the request, whether the agent's response is consistent with this information, and whether the final result answers the user's original question. The authors emphasized that robust AI agents have to demonstrate resilience to noisy data, the ability to refuse an answer when there is insufficient information, the ability to aggregate data from different sources, and correctly process conflicting or outdated information.

To automate the evaluation, Es et al. (2024) proposed the RAGAS framework, which uses language models themselves as judges to evaluate the quality of AI agents. This significantly reduces the time spent on the evaluation process itself. In parallel, Saad-Falcon et al. (2024) developed the ARES, which trains specialized compact estimator models to analyze the quality of individual agent components. An important result of their work is that such models remain effective even when the types of queries or data with which the agent works change.

Despite progress in evaluating individual aspects of the work of AI agents, the problem of comprehensive evaluation in real conditions remains open. Existing approaches often test systems on carefully selected examples that do not represent the full range of possible situations, such as invalid queries, inconsistent data, or deliberately provocative queries. In addition, most research focuses on functional correctness, paying insufficient attention to behavioral stability, scalability, and stability of operation when environmental conditions change for an AI agent.

3.0 Methodology

Despite the fact that existing approaches, one way or another, provide separate tools for evaluating AI agents, a unified, standardized methodology has not yet been established. Unlike simple text generation, where problems often concern the quality of the answer itself, in AI agent problems can arise at any stage of the task, from the initial understanding and decomposition of the goal to the interpretation of intermediate results and the formation of the final answer.

This section will standardize the most common problems in the work of AI agents. Understanding these issues is critical to developing the evaluation methodology presented in the next section, since each type of error requires a different testing strategy and quality metrics. Also, the use of a taxonomy of errors will allow to use the risk-based testing approach when coverage for evaluation is based not on the requirements but on points of failure that may exist in the system. Since the architecture of AI agents is often technologically similar, the problems arising in such AI agents will be of a general nature, which will make it possible to use the proposed methodology for any class of AI agents.

It is important to understand that these categories overlap, and real failures often include several types of problems simultaneously.

3.1 Endless loops

Endless loops mean that the AI Agent repeats the same actions over and over again without solving the problem. This usually happens when the AI agent does not have enough information, but repeated execution of identical actions does not provide new information or a different result.

The problem is especially typical when working with knowledge bases. After receiving an empty search result, a poorly designed agent begins rephrasing the query, hoping to find information: "latest news about X" → "current situation with X" → "updates on X" → "what's new in X." All of these formulations are semantically equivalent and return equally empty results, but the agent keeps trying indefinitely.

A more complex option occurs when there are fluctuations in the search parameters. The AI agent begins to expand the selection criteria, while receiving a lot of irrelevant results, after

which it begins to make the criteria stricter, but again gets nothing, after which the next iteration of expanding the criteria begins, and so on in a closed cycle.

Another situation that can also be attributed to this problem is when an AI agent finds a fragment of a document, realizes that the context is not enough, requests neighboring fragments, discovers that they are not enough, and returns to the original fragment. This can often happen when information has become split when documents are not indexed correctly, but the AI agent cannot understand that no additional queries for fragments will restore the completeness of the data.

3.2 Incorrect task decomposition

The next problem in the management error part is incorrect task decomposition when operating an AI agent, which implies that the AI agent breaks a complex task into incorrect, overly detailed, or poorly ordered subtasks, which leads to ineffective execution or loss of connection with the overall execution goal to solve the task. Decomposition is one of the important capabilities of agent systems, allowing you to decide how complex goals can be transformed into more understandable, executable sequences of actions.

A typical example is when, when asked "Compare the advantages and disadvantages of approach X versus approach Y," the AI agent creates four separate searches:

- advantages X
- disadvantages X
- advantages Y
- disadvantages Y

This is quite inefficient because it increases costs and execution time, results may end up from different time periods when updating data between queries, and it also requires complex logic for merging results.

In addition, there may still be an imbalance between information search and its analysis, when the AI agent either plans too many search steps relative to synthesis (overloading itself with information) or too few (leading to insufficient validity of conclusions). For example, to write a report, an AI agent may find hundreds of fragments without intermediate analysis, and then try to process the entire volume in one step, which exceeds its capacity due to the number of tokens or other restrictions.

Drifting away from the original target during decomposition is also a serious problem, aggravated when working with external data. Each search step receives new information that can shift the agent's AI focus. Starting with an analysis of market trends in 2025, the agent finds mentions of AI adoption, focuses on this data, dives into the specifics of AI startups, and ultimately produces a report on startups in Silicon Valley, which by this point has completely lost touch with the original request about general market trends.

3.3 Losing context between actions

AI agents can very often forget the results of previous steps, which is critical for tasks that require long chains of reasoning or multi-step planning with many intermediate operations.

When working with knowledge bases, loss of context has specific manifestations. This problem occurs when an AI agent successfully finds the information it needs, but does not store it for subsequent steps, for example, an AI agent may find financial data for all quarters of the year, but then re-search the same documents to access data for each individual quarter because it did not retain the results of the initial search in memory, which significantly increases the time the AI agent runs and the number of tokens spent to obtain and process the information.

Also, another problem may arise when an AI agent finds information from several documents but loses the connection between the facts and their sources. AI agent may remember that "revenue increased by 15%," but not remember what document it came from or what period it belongs to. This can be critical when it comes to citing, evaluating the reliability of information, or identifying inconsistencies between sources.

In addition, a loss of conversation context is possible when the AI agent forgets its position in the semantic space of the dialogue, which leads to query drift. If a user has been discussing "Product X" for several turns and then asks "What are the key features?", the loss of context causes the agent AI to perceive this as a general question rather than specific to Product X.

Also, sometimes metadata may be lost in a chain of thin steps, when the AI agent finds documents with metadata (time stamps, authors, reliability ratings, document types) but does not transfer this information to subsequent steps. It may correctly determine that information is out of date during a search, but then use it in analysis because the temporal context has not been preserved.

3.4 Ineffective tool usage

Poor tool selection often occurs when an AI agent uses a suboptimal or inappropriate tool to solve a problem. For example, for the task "Find Mike Carroll's email," the obvious choice should be a contact search tool rather than a calendar or file system. However, the AI agent may choose a different tool, such as a document tool instead of a message tool, which will result in the AI agent being unable to complete the task. Furthermore, this problem can be exacerbated if there are numerous similar tools. If "search_documents", "search_emails", and "search_contacts" are available, the AI agent must understand the specifics of each and choose the right one, since the wrong choice results in either no results or receiving irrelevant information from the wrong source. This also includes ignoring the logical order of tool calls when operating an AI agent. For example, you cannot call "update_document" without first retrieving the document, so breaking the logical order leads to errors and necessitates repeating operations.

Tool hallucination is also a serious problem, which occurs when an AI agent attempts to invoke non-existent functions or fabricates the results of its operation. For example, an AI agent might think that a tool called "get_weather_forecast_with_detailed_hourly_breakdown" is available when, in fact, only "get_weather" is available. This leads to execution errors and loss of time for processing non-existent options for solving the problem. Incorrect call parameters are also one of the most common errors that can be associated with hallucinations. In this case, the AI agent may pass parameters of the incorrect type, for example, a string instead of a number, a wrong date format in the DD/MM/YYYY format instead of the required YYYY-MM-DD, or, conversely, fail to pass required parameters. For example, for the "send_email" tool, the AI agent may forget to specify the subject of the letter or transmit an invalid email address.

In addition to incorrect operation, the AI agent can make redundant calls, which can lead to unnecessary expenditure of tokens to complete the task. One such example would be the use of data caching. In this case, the AI agent, before executing the request, must analyze the presence of the requested information in the cache tool and, after that, perform a search in various resources. Still, AI agents often ignore this and make requests to sources on an ongoing basis, leading to increased costs for using such an agent.

3.5 Poor error handling when calling tools

When tools return errors or are unavailable, the AI agent can either stop execution entirely or ignore the problem and continue with incorrect assumptions about the results.

In this case, the problem may be the wrong strategy for handling such errors from the tools. The AI agent must distinguish between types of errors and apply different methods for handling them, without endlessly repeating requests without pauses, thereby overloading

the system or, conversely, failing to complete its work without making sufficient attempts to repeat calls. At the same time, you can very often encounter a situation where an AI agent ignores such failures in the operation of tools in the presence of errors or unexpected responses. For example, suppose an AI agent invokes a tool to process ten items, and only eight are processed successfully. In that case, it can consider either the entire operation a failure, ignoring the eight successes, or consider it a complete success, ignoring the two failures. Ignoring unexpected responses can also occur when a tool inside the agent AI returns data in an unexpected format, an empty result, or data with an unexpected structure. In this case, the AI agent must validate the structure of the answer and distinguish between these situations, while having a strategy for handling each incorrect answer from the tool.

3.6 Overplanning

An AI agent can create overly detailed plans for simple tasks, spending more resources on strategizing than on actually doing the work, reducing overall efficiency and speed of work. This situation can manifest itself during long discussions about the best approach to solving simple linear problems. For a simple question like “what is the company's mission,” the AI agent can think through different wordings of the query in detail, consider different search parameters, and plan to use different tools, although this simple query only requires a simple query into a vector database to obtain the necessary information and generate an answer.

In addition, you may encounter so-called premature optimization, when the AI agent focuses on optimizing execution processes for tasks that are not performance critical. For example, long thoughts about parallel or sequential document search, when both approaches are completed in a split second.

A complete shutdown of the agent during scheduling can also occur when the agent's AI work requires receiving data from multiple sources, in which case the agent may become stuck discussing optimal sequencing, de-duplication, and integration strategies before starting any actual work.

3.7 Incorrect prioritization

Misprioritization occurs when an AI agent focuses on ancillary information while ignoring key aspects of the task. This manifests itself in ineffective distribution of effort and erroneous evaluation of the relevance of the data obtained.

Resource misallocation occurs when an AI agent spends a disproportionate amount of time collecting unimportant information while paying insufficient attention to the main requirements of the request. For example, when given the task “Analyze Q4 financial performance with a focus on revenue,” an AI agent may spend significant time collecting industry analytics, competitive data, and historical trends from previous years, while paying minimal attention to actual Q4 financial performance. As a result, the main request remains incompletely resolved, despite the large amount of information collected.

Errors in prioritizing information also arise at the stage of ranking the found data according to the degree of importance for the original request. After performing a search, the AI agent can focus on indirectly related information that scores highly on formal metrics such as semantic similarity or keyword co-occurrence frequency. At the same time, truly critical information that may be presented in different words or require a deeper understanding of the context is left out of attention. For example, when asked “what are the main risks of our project,” an AI agent might focus on a section of a document where the word “risk” is mentioned multiple times in the overall context, ignoring the section with critical technical limitations that pose a real threat to the project.

3.8 Lack of verification of the results obtained

When an AI agent does not verify the generated response for compliance with the data found, the completeness of coverage of all aspects of the request, and the correctness of

links to sources, a serious problem arises, which can often lead to a discrepancy between the found data and the final answer. This occurs when the AI agent generates text that deviates from the information actually detected. An AI agent can successfully find accurate data, but in the process of formulating an answer it introduces distortions or errors, without validating the consistency of the final text with the source material. For example, an agent might find "sales increased 15% in the third quarter," but write "sales increased significantly last quarter," missing specific numbers and the exact period.

Additionally, a situation is possible when the AI agent does not control whether the answer to all points from the specified request was provided, which implies the completeness of the answer. For a question that has multiple parts, the AI agent may find information that answers only part of the subquestions, but not notice the gap. For example, when asking about the advantages, disadvantages, cost, and timeline of an AI project, an agent might provide the first three items, missing the implementation timeline entirely.

Inadequate coverage of sources includes a lack of verification of the comprehensiveness of the information search. An AI agent may only find data from one part of a document collection without looking at other relevant sections. For example, when analyzing corporate policy, an agent can use only official regulations, ignoring internal memos and protocols that contain important nuances of applying the policy.

3.9 Hallucinations in intermediate stages

Intermediate hallucinations occur when an AI agent makes up information as it runs, suggesting data that doesn't exist, or creating artificial connections between pieces of data it finds.

Search result hallucination occurs when an AI agent believes it has found information that was not actually obtained from the sources. This may arise from confusion between the content actually found and the model's pre-trained knowledge, or from "filling in" expected results in situations where the actual search did not provide the required information. For example, an AI agent may claim to have found accurate financial figures in a document when the document contains only general discussions on this topic.

Synthetic hallucination involves the AI agent generating plausible summaries of found data that deviate from its actual content. An agent may correctly locate a technical document, but then summarize it with simplifications or generalizations that introduce inaccuracies not present in the original.

Citation fabrication represents cases where an AI agent claims that specific information comes from certain retrieved documents that do not actually contain it. This is different from simple misinterpretation—the agent may completely invent the existence of supporting data in the source.

Hallucination of connections involves the AI agent inventing relationships between found pieces of information that do not exist in reality. For example, an agent might claim that "Document A's results are validated by Document B," while Document B makes no reference to or cross-validate Document A's work.

3.10 Implications for evaluation methodology

The presented classification of problems reveals several critical aspects for the development of a comprehensive methodology for evaluating AI agents. Firstly, the traditional evaluation of answer accuracy is insufficient for agent systems, because the methodology must take into account, among other things, the quality of work with data and tools, the correspondence between the information found and the generated answer, the accuracy of the sources and a complete analysis of the work of the AI agent from the initial request to the final result.

Secondly, different architectural approaches may present different patterns of problems, such that AI agents with explicit reasoning loops are particularly susceptible to loss of

context between steps and circular behavior when working with data, architectures with careful planning are more likely to suffer from incorrect decomposition and over-planning, and multi-component agent systems with shared resources face unique problems of coordinating and distributing results when executing a query.

Many problems arise from the complex interaction between an AI agent's data and reasoning abilities, which means it needs to evaluate not just individual components in isolation, but the integrated functionality of the entire system.

All these requires new, more general and standardized approaches to detecting cyclic behavior, evaluating the quality of task decomposition, checking the preservation of context, verifying the validity of conclusions with the data found, evaluating the effectiveness of planning, checking the correctness of prioritization and other problems, therefore, in the next section, the presented methodology will be built precisely on the principle of searching and identifying the above problems in order to ensure the quality and reliability of the AI agent under various types of input data and conditions of use.

4.0 Methodology

The classification of problems from the previous section identified nine categories of characteristic problems that can arise in the work of AI agents. It is now necessary to develop methods that will detect and measure each type of problem in real-world operating conditions. This section presents a comprehensive evaluation methodology organized around four key areas.

The first direction focuses on evaluating robustness to input data variations. Here, we check how stable the AI agent is when receiving the same request in different formulations. The second direction is the evaluation of task management, where cyclical behavior, planning errors, and decomposition problems are identified. The third direction is related to the evaluation of the agent's AI performance with various tools, which focuses on the correct interaction of the agent with external systems. And the last, fourth direction is evaluating the quality of reasoning and checking the results, which is aimed at evaluating the correctness of prioritization, validation, and the absence of hallucinations.

Each evaluation method is described in a single framework that explains the problems it identifies, describes the evaluation process, defines specific metrics, and establishes criteria for the quality of the agent's AI. This provides a systematic approach to evaluating the reliability of AI agents.

4.1 Evaluation of robustness to input data variations

Problems to identify: Hallucinations (section 3.9), unpredictable behavior when changing formulations, instability in tool selection, and planning.

A robust AI agent should provide semantically similar responses when given rephrased queries. If a small change in wording radically changes the agent's behavior, then this signals problems in understanding the problem or making decisions. In this case, metamorphic testing is used, which allows you to check for hallucinations without requiring standard answers for each possible query option.

Methodology: Create a basic set of queries that correspond to the goals of the AI agent, then expand each query with the following metamorphic relations:

1. Semantic paraphrases, which allow to preserve the meaning of the query but change its wording. For example: "Find all the appointments for next week" → "Show me your calendar for next week" → "What events do I have planned for the next seven days?"

2. Syntactic variations are used to change the structure of a sentence. In this case, it is possible to change the word order, use active/passive voice, replace pronouns with nouns, and other variations. "Find all the meetings next week." → "All meetings this week find."

3. Noise injection allows to simulate real conditions that can often be encountered, especially when working with users: typos ("meetings" → "metings"), informal language ("give me information on the project"), extra spaces, and case changes.

4. Contextual additions add polite requests ("Could you please..."), clarifying context, or explicit restrictions ("consider only completed tasks").

For each basic query X with execution result Y and its transformation X' with result Y' evaluate:

$$\text{semantic_similarity}(Y, Y') \geq \theta_{\text{consistency}} \quad (1)$$

$$\text{where } \theta_{\text{consistency}} \geq 0.9$$

Metrics:

$$\text{Consistency Rate} = \frac{\text{Number of pairs with consistent results}}{\text{Total number of pairs}} \quad (2)$$

$$\text{Robustness Score} = \min_{\text{variation type}} \text{Consistency Rate}_{\text{type}} \quad (3)$$

$$\text{Weighted Robustness Score} = \sum_i w_i \cdot (1 - H(Y_i)/\log(n)) \cdot \text{semantic_similarity}(Y, Y_i) \quad (4)$$

where $H(Y_i)$ is the entropy of the response distribution, measuring uncertainty across variations.

$$\text{Jensen-Shannon Divergence} = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(Q \parallel M), \quad M = \frac{1}{2}(P + Q) \quad (5)$$

It is critical to monitor consistency separately by variation type. For example, an AI agent may work reliably when rephrasing, but fail when typing, indicating low reliability of the input data processing process.

Table 1. Success Criteria for Input Data Variations

Variation Type	Consistency	WRS	JSD
Semantic paraphrases	≥ 0.95	≥ 0.80	≤ 0.15
Syntactic variations	≥ 0.90	≥ 0.75	≤ 0.20
Noise	≥ 0.85	≥ 0.70	≤ 0.25
Overall robustness	Robustness Score ≥ 0.85	—	—

4.2 Evaluation of execution and planning

This direction identifies the problems from sections 3.1, endless loops, 3.2, incorrect task decomposition, 3.3, loss of context, and 3.6, overplanning.

a) Detection of endless loops

Problems to identify: Endless loops and repeating the same actions without progress (section 3.1).

Methodology: Create test scenarios in the dataset that specifically provoke the agent AI to create cycles:

1. Unattainable Goal Scenarios: Requests for information that is not available from available sources. The AI agent should recognize this and complete the job gracefully, rather than endlessly repeating search attempts.
2. Problems with ambiguous criteria: Situations where the criterion of “sufficiency” of information is very vague. For example, “find all information about company X.” In this case, the AI agent should set a reasonable limit on the amount of information, and not try to collect absolutely everything that is in the sources.
3. Loops when working with fragmented data: Information is divided among several sources in such a way that each individually seems insufficient. The AI agent must either merge the fragments or recognize the constraint, but not get stuck working between them.
4. Fluctuations in parameters: Situations where the AI agent can get into a cycle of expanding and narrowing search criteria or filtering.

As an analysis tool, it is necessary to monitor the execution traces of the AI agent for repeating patterns of actions, and also for each request, determine whether the AI agent gets into endless loops or whether it terminates correctly.

Metrics:

$$\text{Loop Detection Rate} = \frac{\text{Scenarios with correct termination}}{\text{Total scenarios with cyclic processing}} \quad (6)$$

$$\text{Avg Iterations} = \text{Avg num of iterations until completion} \quad (7)$$

$$\text{Max Iteration Deviation} = \frac{|\text{Max iterations} - \text{Expected iterations}|}{\text{Expected iterations}} \quad (8)$$

$$\text{HTDQ} = 1 - \frac{\text{TED}(T_{\text{agent}}, T_{\text{optimal}})}{\max(|T_{\text{agent}}|, |T_{\text{optimal}}|)} \cdot e^{-\lambda \cdot \text{depth_variance}} \quad (9)$$

The HTDQ metric used is based on Tree Edit Distance (TED), an established method that measures the similarity of tree structures and is widely used in parsing (Zhang & Shasha, 1989; Bille, 2005). In the context of scheduling tasks within AI agents, the hierarchical structure of such a decomposition is represented as a tree, where the root is the original task, and the branches are atomic actions. To use this metric in the evaluation process, TED was adapted by adding an exponential penalty for branch depth inconsistency (the term $e^{-\lambda \cdot \text{depth_variance}}$) since uneven decomposition during task planning is often the cause of errors. This approach is consistent with research on hierarchical planning in AI, where the balance of decomposition correlates with plan quality (Erol et al., 1994; Nau, 2003).

Proposed success criteria: Loop Detection Rate ≥ 0.95 ; completion within a reasonable number of iterations (usually 5-10 for most problems); false positive rate < 0.01 ; receiving informative messages about the reason for the shutdown of the AI agent, HTDQ ≥ 0.75 .

b) Evaluating the quality of task decomposition

Problems to identify: Incorrect decomposition (Section 3.2), including excessive fragmentation, incorrect sequence of subtasks, and drift from the original goal.

Methodology: Create a dataset of queries covering different levels of complexity, to which we determine in advance how the AI agent should decompose tasks:

1. Simple atomic tasks: There should be minimal or no decomposition when the task needs to be executed directly.
2. Sequential tasks: Require actions to be performed in a specific order, where the result of the previous step is necessary for the next.
3. Parallel tasks: Contain independent subtasks that can be executed simultaneously.
4. Mixed tasks: Combine parallel and serial components and require data manipulation and computation.

Metrics:

$$\text{Granularity Score} = 1 - \frac{|\text{Agent steps} - \text{Reference steps}|}{\max(\text{Agent steps}, \text{Reference steps})} \quad (10)$$

$$\text{Efficiency Ratio} = \frac{\text{Minimum necessary actions}}{\text{Actual agent actions}} \quad (11)$$

$$\text{Goal Alignment} = \text{cosine_similarity}(\text{embedding}(\text{Original task}), \text{embedding}(\text{Final plan})) \quad (12)$$

$$\text{Logical Correctness} = \text{Binary assessment of dependency correctness between subtasks} \quad (13)$$

Proposed success criteria: Granularity Score ≥ 0.75 ; Efficiency Ratio ≥ 0.80 ; Goal Alignment ≥ 0.85 ; Logical Correctness = 1.0 (all dependencies must be logically correct).

c) Context preservation check

Problems to identify: Loss of context between actions (Section 3.3), including forgetting intermediate results and loss of tracking of data sources.

Methodology: Develop a dataset with multi-step tasks with explicit dependencies between steps:

1. Short range (2-3 steps): Testing basic context preservation. For example, "Find Mike Carroll's email, after that send him email...". In this case, the second step requires obtaining the result of the first.
2. Middle Range (4-7 steps): Maintaining context through intermediate operations. The results of early steps are used in later ones through several intermediate actions.
3. Long range (8+ steps): Extended sequences where information from the first steps is critical to the last ones.
4. Source tracking: Maintaining connections between facts and their sources throughout execution. For each step that should use the previous context, we check whether this happens correctly.

Metrics:

$$\text{Context Retention Accuracy} = \frac{\text{Steps with correct context usage}}{\text{Steps requiring context}} \quad (14)$$

$$\text{CRA}(d) = \frac{\text{Correct usage at distance } d}{\text{Required usage at distance } d} \quad (15)$$

$$\text{Source Tracking Accuracy} = \frac{\text{Facts with correct attribution}}{\text{Total facts from external sources}} \quad (16)$$

$$C(d) = \alpha e^{-\beta d} + (1 - \alpha) \frac{1}{1+\gamma d^2} \quad (17)$$

This function measures how well the agent preserves context as the distance d between creation and usage increases.

Parameters represent:

- α weight of short-term memory behavior (initial rapid decay)
- β rate of exponential decay for short-range context
- γ rate of long-term degradation following a polynomial pattern
- d number of steps between producing a fact and using it.

The combined form captures both fast short-range decay and slower long-range degradation, allowing detection of early context loss and long-chain instability (Ebbinghaus, 1885; Rubin & Wenzel, 1996).

Proposed success criteria: CRA ≥ 0.95 for short range, ≥ 0.85 for medium, ≥ 0.70 for long; Source Tracking Accuracy ≥ 0.90 ; absence of catastrophic events of complete loss of context. For the Context Decay Function $C(d) \geq 0.85$ for short distances ($d = 1-3$), $C(d) \geq 0.70$ for medium distances ($d = 4-7$), $C(d) \geq 0.60$ for long distances ($d \geq 8$), $C(d)$ must decrease smoothly without irregular spikes, $\beta \leq 0.4$ and $\gamma \leq 0.05$ to ensure stable decay dynamics.

d) Evaluation of planning effectiveness

Problems to identify: Overplanning (Section 3F), when the AI agent spends a disproportionate number of resources on developing a strategy for completing simple tasks.

Methodology: Measure the ratio between planning time and actual execution time for tasks of varying complexity:

1. Simple problems: Expected planning overhead of 5-15%, for example, "What is the temperature in New York City?" does not require detailed planning.
2. Moderately complex tasks: Expected overhead 15-30%. Example: "Analyze quarterly sales and identify trends."
3. Complex tasks: Expected overhead 30-50%. Example: "Conduct a comparative analysis of three markets, taking into account economic, social, and technological factors."

Metrics:

$$\text{Planning Overhead} = \frac{\text{Planning time}}{\text{Total execution time}} \quad (18)$$

$$\text{Planning Value} = \frac{\text{Quality with planning} - \text{Quality without planning}}{\text{Planning cost}} \quad (19)$$

$$\text{Granularity Appropriateness} = 1 - \frac{|\text{Plan steps} - \text{Optimal steps}|}{\max(\text{Steps}, \text{Optimal})} \quad (20)$$

$$\text{TSS} = \exp\left(-\int_0^t |f'(\tau)|^2 d\tau\right) \times \text{consistency}(t) \quad (21)$$

The Temporal Stability Score uses the concept of signal variation from time series theory (Box et al., 2015), where the integral $\int_0^t |f'(\tau)|^2 d\tau$ measures the overall variation in the agent's behavior over time, where the results can be interpreted as the higher it is, the less predictable the system. The exponential term transforms variation into an estimate of stability quite straightforwardly: with zero variation (ideally stable behavior) we obtain TSS = 1, with high variation TSS tends to zero. The consistency(t) component takes into account

the consistency of the agent's current actions with previous history, which is consistent with the principles of consistency evaluation in multi-agent systems (Wooldridge, 2009). The metric is especially useful for identifying planning instability when an agent radically changes its strategy for performing semantically identical tasks, which indicates insufficient robustness of the internal decision-making model.

Proposed success criteria: Planning Overhead within the expected range for a given complexity; Planning Value > 0 (planning provides a net benefit); Granularity Appropriateness ≥ 0.75 ; correlation between task complexity and planning costs > 0.70, Temporal Stability Score ≥ 0.70 .

4.3 Evaluation tools

This direction highlights the problems from Section 3.4, inefficient use of tools, and 3.5, poor error handling.

a) Correct selection of tool usage

Problems to identify: Ineffective tool usage (Section 3.4), including selection of inappropriate tools, incorrect parameter settings, and redundant calls.

Methodology: Check several aspects of working with tools:

1. Adequacy of choice: For each task, there is an optimal set of tools. Check whether the AI agent selects the right tools.
2. Selection consistency: For semantically equivalent queries X and X' , we check:
 - i. $\text{ToolSet}(X) = \text{ToolSet}(X')$ if $\text{Semantics}(X) \equiv \text{Semantics}(X')$ (22)
3. Correctness of parameters: Does the AI agent set the parameters for calling tools correctly (limits, filters, formats)?
4. Efficiency of use: Does the AI agent minimize redundant calls, use caching, and use batch processing where possible?

Metrics:

$$\text{Tool Selection Accuracy} = \frac{\text{Tasks with correct tool selection}}{\text{Total tasks}} \quad (23)$$

$$\text{Parameter Correctness} = \frac{\text{Calls with correct parameters}}{\text{Total calls}} \quad (24)$$

$$\text{Redundancy Rate} = \frac{\text{Redundant calls}}{\text{Total calls}} \quad (25)$$

$$\text{Consistency Across Rephrasing} = \frac{\text{Consistent selections}}{\text{Total rephrasing pairs}} \quad (26)$$

Proposed success criteria: Tool Selection Accuracy ≥ 0.95 ; Parameter Correctness ≥ 0.90 ; Redundancy Rate ≤ 0.15 ; Consistency Across Rephrasing ≥ 0.95 .

b) Error handling and recovery

Problems to identify: Poor error handling (Section 3.5), including inadequate responses to runtime errors in tool calls, timeouts, service unavailability, and partial failures.

Methodology: Systematically introduce different types of errors and evaluate the agent's reaction:

1. **Temporary errors:** Temporary failures (network timeouts, service overload). The AI agent should retry at reasonable intervals.
2. **Permanent errors:** Access denied, non-existent resource. The AI agent must try alternative approaches or explicitly report failure.
3. **Invalid parameters:** Invalid parameter values. The AI agent must correct them or request clarification.
4. **Partial failures:** Some data was received successfully, some is unavailable. The AI agent must handle partial results correctly and be aware of limitations.
5. **Blank results:** No data. The AI agent must distinguish between "no data" and "request failed."

Metrics:

$$\text{Error Detection Rate} = \frac{\text{Correctly identified errors}}{\text{Total introduced errors}} \quad (27)$$

$$\text{Recovery Success Rate} = \frac{\text{Successful recoveries}}{\text{Recoverable errors}} \quad (28)$$

$$\text{Response Appropriateness} = \frac{\text{Adequate handling strategies}}{\text{Total error types}} \quad (29)$$

$$\text{AER} = \sum_e w(e) \cdot \left(1 - \frac{t_{\text{recovery}}(e)}{t_{\text{max}}}\right) \cdot \text{success}(e) \cdot e^{-\lambda \cdot \text{attempts}(e)} \quad (30)$$

The AER metric takes ideas from the theory of fault-tolerant systems (Avizienis et al., 2004), where critical errors receive a greater weight $w(e)$, and their resolution time is estimated as $(1 - t_{\text{recovery}}/t_{\text{max}})$, where the penalty $e^{-\lambda \cdot \text{attempts}}$ reflects the principle of fast recovery without multiple iterations. At $\lambda = 0.3-0.5$, single repetitions are almost not penalized ($e^{-0.3} \approx 0.74$), while 5+ attempts lead to a noticeable decrease in the score ($e^{-1.5} \approx 0.22$).

Proposed success criteria: Error Detection Rate ≥ 0.95 ; Recovery Success Rate ≥ 0.80 ; Response Appropriateness ≥ 0.90 ; The AI agent must apply the appropriate strategy for each type of error; Adaptive Recovery ≥ 0.70 ; efficient correction with minimal retry pressure.

4.4 Evaluation the quality of reasoning and validation

This direction identifies the problems from section 3.7 - incorrect prioritization, 3.8 lack of validation, and 3.9 hallucinations.

a) Prioritization evaluation

Problems to identify: Incorrect prioritization (Section 3G), when the AI agent focuses on minor aspects, ignoring the key elements of the task.

Methodology: Develop a dataset of tasks with a clear priority structure:

1. **Explicit priorities:** "Analyze finances with an emphasis on revenue." In this case, the priority is indicated explicitly.
2. **Implicit priorities:** Domain knowledge dictates what is important. For example, when analyzing a security incident, it is important to understand that the scale of the leak is more important than the details of software versions.

Metrics:

$$\text{Priority Alignment} = \text{Spearman correlation (Agent efforts, Reference priorities)} \quad (31)$$

$$\text{Critical Coverage} = \frac{\text{Covered critical aspects}}{\text{Total critical aspects}} \quad (32)$$

Proposed success criteria: Priority Alignment ≥ 0.70 ; Critical Coverage = 1.0 (all critical aspects must be covered).

b) Checking the validation of results

Problems to identify: Lack of validation of the final result (section 3H), when the AI agent does not check the completeness of the task and the correctness of the answer.

Methodology: Systematically introduce problems into the execution process and check if the AI agent detects them before presenting the result:

1. Incomplete execution: Multipart questions with missing answers to some parts.
2. Internal contradictions: The result contains mutually exclusive statements.
3. Silent tool failures: The tool returned an error, but the AI agent continued as if everything was normal.
4. Constraint Violations: Values out of range (negative prices, percentages > 100%).
5. Unsubstantiated Claims: Claims without supporting data.

Metrics:

$$\text{Issue Detection Rate} = \frac{\text{Detected issues}}{\text{Total introduced issues}} \quad (33)$$

$$\text{False Positive Rate} = \frac{\text{False positives}}{\text{Correct elements}} \quad (34)$$

$$\text{Validation Coverage} = \frac{\text{Problem types with checks}}{\text{Total problem types}} \quad (35)$$

Proposed success criteria: Issue Detection Rate ≥ 0.85 ; False Positive Rate ≤ 0.05 ; Validation Coverage ≥ 0.80 .

c) Hallucination Detection

Problems to identify: Hallucinations at intermediate stages (section 3.9), when the AI agent itself comes up with the results of executing tools, invents connections between data, or fabricates links to sources.

Methodology: Apply several approaches to detect hallucinations:

1. Statement tracing: For each statement in the agent's response, we check whether it can be traced back to the actual data received from the tools.
2. Verification of links: If the AI agent links to sources, we check that these sources were actually used and contain the declared information.
3. Checking logical connections: If the AI agent asserts a connection between facts (causal, temporal, spatial), we check the validity of this connection.
4. Cross-validation: Repeating a task with slight variations in the input. Hallucinated details usually change unpredictably.

Metrics:

$$\text{Claim Traceability} = \frac{\text{Claims with traceable sources}}{\text{Total factual claims}} \quad (36)$$

$$\text{Reference Accuracy} = \frac{\text{Correct references}}{\text{Total source references}} \quad (37)$$

$$\text{Hallucination Rate} = \frac{\text{Unfounded claims}}{\text{Total claims}} \quad (38)$$

$$\text{MHS} = \sqrt{\sum_i (\lambda_i (\text{err}_i^2 + \text{drift}_i^2 + \text{fabrication}_i^2))} \quad (39)$$

Each type has its own weight λ_i , which allows critical errors to be amplified.

The MHS aggregates three types of hallucinations identified in LLM reliability studies (Ji et al., 2023), namely factual errors, semantic drift, and information fabrication. The quadratic form $(\text{err}^2 + \text{drift}^2 + \text{fabrication}^2)$ provides sufficient sensitivity to the combination of multiple problem types such that a system with moderate problems across all dimensions receives a higher (worse) score than a system with an isolated problem in one category. Differentiated weights λ_i make it possible to take into account the context of application, for example, in medical applications, factual errors are more critical ($\lambda_{\text{err}} \gg \lambda_{\text{drift}}$), while in creative tasks semantic drift can be acceptable.

The MHS aggregates three types of hallucinations into a single score:

- factual error
- semantic drift (distortion of the meaning of facts when retelling)
- fabrication

$$\text{ICV} = \sum [\log P(\text{claim}|\text{evidence}) - \log P(\text{claim})] \cdot \text{reliability}(\text{source}) \quad (40)$$

ICV uses a Bayesian updating approach to verifying statements (MacKay, 2003). The formula $\log P(\text{claim}|\text{evidence}) - \log P(\text{claim})$ is equal to the logarithm of the likelihood ratio, where positive ICV values indicate that the evidence supports the claim, negative values indicate that it contradicts it, and zero values indicate that the evidence is irrelevant. Reliability(source) weighting takes into account the heterogeneity of sources, with highly reliable sources (peer-reviewed articles, white papers) receiving more weight than low-reliability sources.

Proposed success criteria: Claim Traceability ≥ 0.95 ; Reference Accuracy ≥ 0.95 ; Hallucination Rate ≤ 0.05 , Hallucination Stability: MHS ≤ 0.20 , no multi-dimensional compounding of errors, ICV ≥ 0.0 , all claims must show non-negative information gain.

4.5 Integration of evaluation methodology

The presented methodology covers all ten problem categories identified in Section 3. Table 2 shows the overall correspondence between problem types and evaluation methods.

Table 2. Coverage of problem categories by evaluation methods

Problem Category	Section	Evaluation Methods
Endless loops	3.1	Loop detection (4.2.a)
Incorrect decomposition	3.2	Decomposition quality (4.2.b)
Context loss	3.3	Context preservation (4.2.c)
Inefficient tool usage	3.4	Tool selection correctness (4.3.a)
Poor error handling	3.5	Error handling (4.3.b)
Overplanning	3.6	Planning efficiency (4.2.d)
Incorrect prioritization	3.7	Prioritization evaluation (4.4.a)
Lack of validation	3.8	Validation checking (4.4.b)
Hallucinations	3.9	Input robustness (4.1) + Hallucination detection (4.4.c)

a) Handling overlapping problem categories

Conceptual overlaps between categories should not be seen as a weakness of the taxonomy, but rather as a reflection of how real AI agents are structured. Practice shows that the same problem often manifests itself at several levels at the same time, and if we take the case when an AI agent produces an answer, information about which is not in the ground truth, then this can be interpreted as a hallucination when generating the answer, and as insufficient validation of the result, and as a loss of context between stages of work.

The combined `detect_*_issue` functions, which aggregate readings from several metrics, demonstrate notably higher efficiency than individual indicators precisely because they capture the problem from different angles. During the experiments, it was revealed that, for example, the Tool Usage category yielded an F1 score of 0.421. At the same time, the comprehensive `detect_tool_usage_issue` reaches 1.000, similarly for Context Retention, where the gap between individual indicators and the overall quality indicator for finding errors in the AI agent is 65%. These results are discussed in more detail in Section 5.1 (Fig. 4).

From a practical standpoint, combining metrics works as follows: if a fundamental error in the AI agent falls into the area of intersection of categories, for example, hallucination and lack of validation at the same time, indicators from both groups will highlight it. `Hallucination_rate` records a discrepancy with the source, `issue_detection_rate` detects the absence of validation, and the two are combined to produce a general result that demonstrates the problem. This method allows higher confidence detection of problems than strict, mutually exclusive categorization, which would have required unequivocal assignment of each error to exactly one category, which is almost never the case for agent AI.

b) Integrated evaluation architecture

The full methodology consists of four integrated levels of evaluation:

- Level 1 - Input Robustness (Section 4.1), which includes a basic check of consistency of behavior across query variations. Serves as a foundation for identifying unpredictability, which can indicate any type of problem.
- Level 2 - Execution Management (Section 4.2) includes evaluating the quality of planning, decomposition and context maintenance, which are critical aspects for the correct orchestration of the agent's AI work.

- Level 3 - Working with Tools (section 4.3), performs testing of interaction with external systems, including selection of tools, error handling and interpretation of results.
- Level 4 - Quality of results (section 4.4). Contains validation of the final stages of work, ensuring correct prioritization, completeness of checks, and the absence of hallucinations.

Each evaluation method provides quantitative metrics with defined success criteria, allowing for systematic evaluation and iterative improvement of AI agent systems. The methodology is designed in a modular manner so that organizations can select priority areas based on their architectural features and critical issues, while maintaining the ability to implement all layers to comprehensively evaluate agent AI reliability.

5.0 Empirical validation of the proposed methodology

To confirm the practical applicability of the proposed methodology, a number of experiments were carried out, allowing, by generating synthetic data, to obtain indicators of the quality of measurement of the proposed metrics. The choice in favor of synthetic trace data of the AI agent is justified by the fact that in real conditions it is not possible to simulate a large number of errors in real AI systems to obtain all the necessary data, so a set of 300 synthetic traces of the execution of the AI agent was generated, evenly distributed among error categories from the proposed taxonomy (30 traces per category).

Each category contained examples of a positive class with real errors in the traces and a negative class without errors to determine the real ability of the metrics to identify errors in the data for analysis. Synthetic AI traces simulated real-life scenarios of AI agents, including:

1. Sequences of agent steps, where each step contains an identifier, action type, timestamp, and execution result. For traces with cyclic errors, repeating patterns of actions are generated.
2. Tool calls, which emulate calls to external APIs and services, indicating the tool name, call parameters, ground truth and execution result. For erroneous traces, incorrect tool selections, incorrect parameters, and redundant calls are generated.
3. A task decomposition structure that includes a source task, a list of subtasks, a graph of dependencies between them, and a reference decomposition for comparison. Faulty examples contain excessive fragmentation, circular dependencies, or drift from the original target.
4. Context dependencies, where for each step it is indicated which context from previous steps is required and which was actually used. This allows you to model the loss of context at different distances (short, medium, long).
5. Runtime errors and recovery strategies include the type of error, whether the agent detected it, the recovery strategy applied, the recovery time, and the number of retries.
6. Factual statements with claims contain the text of the statement, the source specified by the agent, the ground truth, validity, and fabrication flags. Used to validate hallucination detection metrics.
7. Query variations are used for robustness testing and generate noisy syntactic variations for each base query.

During the evaluation procedure itself, for each of the 41 proposed metrics, an evaluation of the quality of their classification work was determined, which included obtaining the metric indicator itself, comparing the resulting indicator with a threshold value, taking into account the positive or negative classification of the trace, as well as comparison with ground truth.

To evaluate the quality of the metrics, basic indicators from ML were used, such as Precision, Recall, F1 score, and Accuracy.

5.1 Validation results

The full validation results are presented in Table 3. A visualization of the results by category is shown in Fig. 1, the distribution of metrics between Precision-Recall in Fig. 2, and detailed F1-score values for each metric are shown in Fig. 3.

Table 3. Synthetic Validation Results for Proposed Metrics

Category	Metric	Precision	Recall	F1	Accuracy
Robustness	consistency_rate	1.000	1.000	1.000	1.000
Robustness	robustness_score	1.000	1.000	1.000	1.000
Robustness	WRS	1.000	1.000	1.000	1.000
Robustness	detect_robustness_issue	1.000	1.000	1.000	1.000
Loop_detection	avg_iterations	1.000	0.333	0.500	0.667
Loop_detection	HTDQ	1.000	0.333	0.500	0.667
Loop_detection	detect_loop_issue	0.833	0.667	0.741	0.767
Decomposition	granularity_score	1.000	0.333	0.500	0.667
Decomposition	efficiency_ratio	1.000	0.333	0.500	0.667
Decomposition	goal_alignment	0.786	0.733	0.759	0.767
Decomposition	logical_correctness	1.000	0.333	0.500	0.667
Decomposition	detect_decomposition_issue	0.824	0.933	0.875	0.867
Context_retention	CRA	1.000	0.733	0.846	0.867
Context_retention	source_tracking	1.000	0.467	0.636	0.733
Context_retention	context_decay_score	1.000	0.200	0.333	0.600
Context_retention	detect_context_issue	1.000	1.000	1.000	1.000
Planning	planning_overhead	1.000	0.667	0.800	0.833
Planning	granularity_appropriateness	1.000	1.000	1.000	1.000
Planning	TSS	1.000	1.000	1.000	1.000
Planning	detect_planning_issue	1.000	1.000	1.000	1.000
Tool_usage	tool_selection_accuracy	1.000	0.267	0.421	0.633
Tool_usage	parameter_correctness	1.000	0.267	0.421	0.633
Tool_usage	redundancy_rate	1.000	0.267	0.421	0.633
Tool_usage	detect_tool_usage_issue	1.000	1.000	1.000	1.000
Error_handling	error_detection_rate	1.000	0.467	0.636	0.733
Error_handling	recovery_success_rate	1.000	0.733	0.846	0.867
Error_handling	response_appropriateness	1.000	0.733	0.846	0.867
Error_handling	AER	0.500	1.000	0.667	0.500
Error_handling	detect_error_handling_issue	0.500	1.000	0.667	0.500
Prioritization	priority_alignment	1.000	0.933	0.966	0.967
Prioritization	critical_coverage	1.000	0.333	0.500	0.667
Prioritization	detect_prioritization_issue	1.000	0.933	0.966	0.967
Result_validation	issue_detection_rate	1.000	0.733	0.846	0.867
Result_validation	false_positive_rate	1.000	0.267	0.421	0.633
Result_validation	validation_coverage	1.000	0.733	0.846	0.867
Result_validation	detect_validation_issue	1.000	1.000	1.000	1.000
Hallucination	claim_traceability	1.000	0.533	0.696	0.767
Hallucination	reference_accuracy	1.000	0.800	0.889	0.900
Hallucination	hallucination_rate	1.000	0.733	0.846	0.867
Hallucination	grounding_rate	1.000	0.733	0.846	0.867
Hallucination	detect_hallucination_issue	1.000	1.000	1.000	1.000



Fig.1 Validation quality indicators by error category

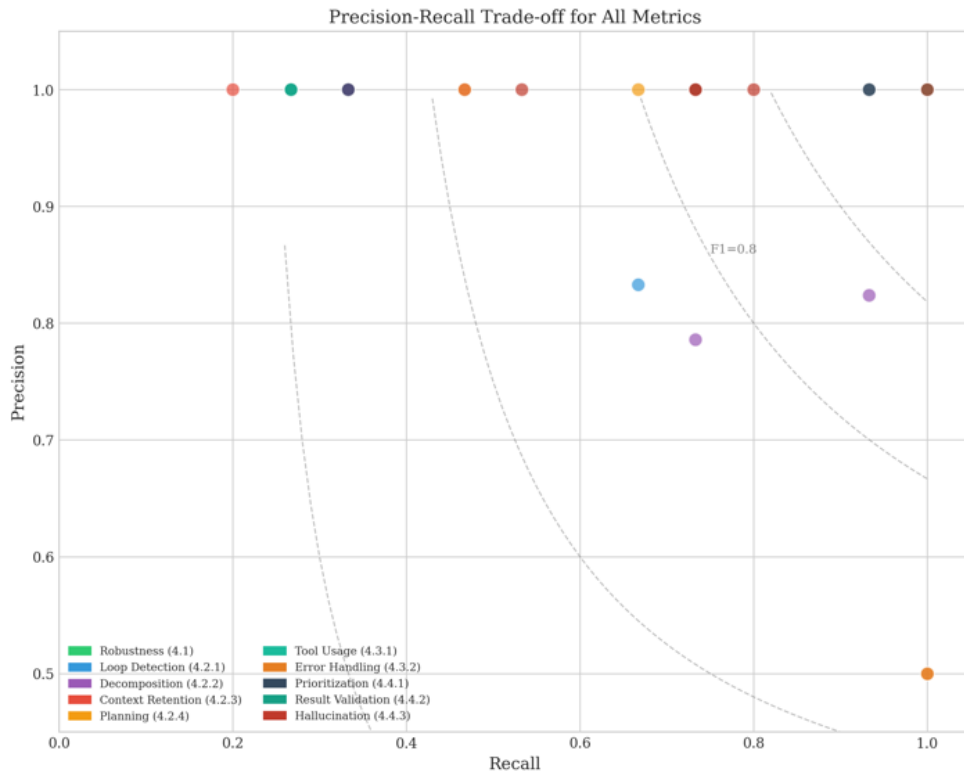


Fig.2 Distribution of metrics in the Precision-Recall space

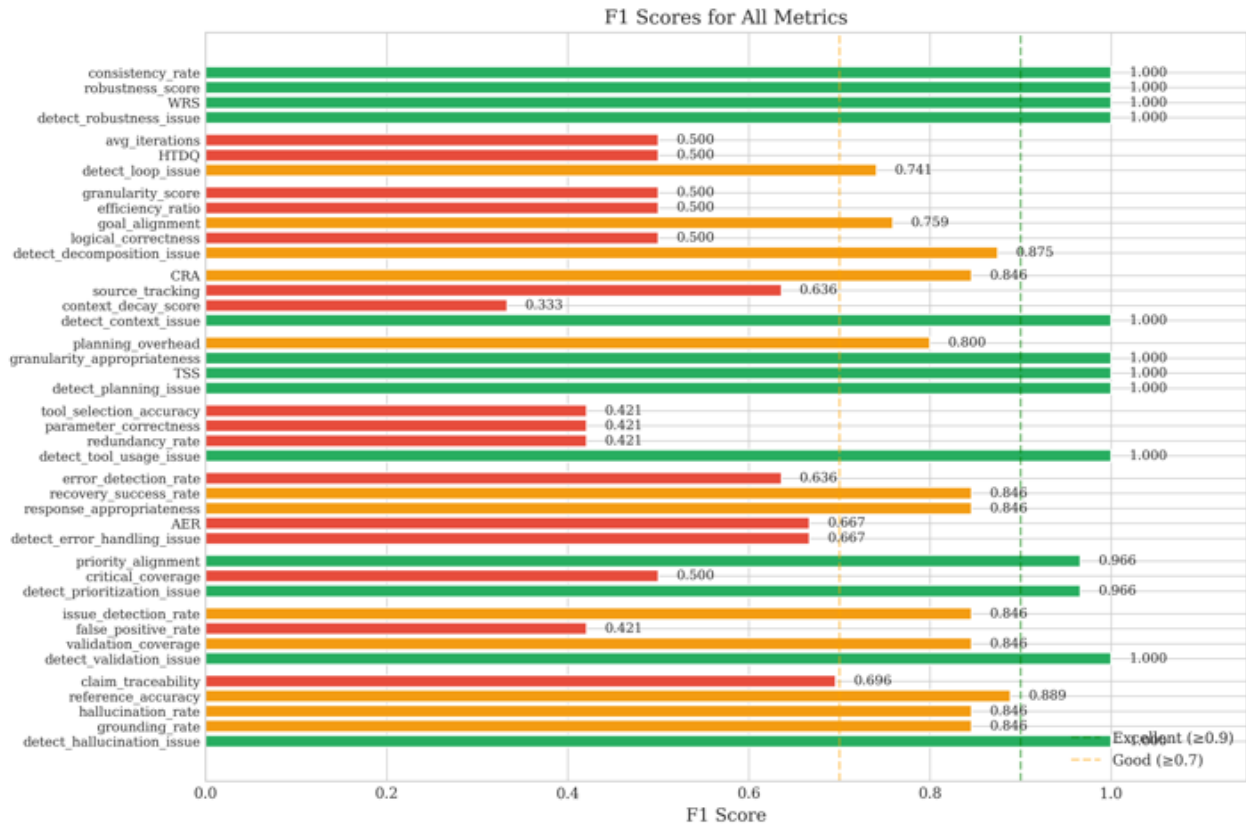


Fig. 3 F1-score values for all 41 metrics

The overall quality indicators for the metrics used in the above methodology are: Average Precision: 0.962, Average Recall: 0.696, Average F1-score: 0.762, Average Accuracy: 0.817.

From the point of view of analyzing the results, it is important to note that the high Precision score of 0.962 shows that the proposed metrics for evaluating the reliability of AI systems practically do not give false positives, the metric almost always signals the presence of a problem, and with a probability of 96% this problem is actually present in the AI agent's trace. Also, the distribution of metrics in the Precision-Recall space (Figure 2) demonstrates a characteristic pattern that most metrics are located in the area of high precision (Precision > 0.9) with varying completeness (Recall from 0.2 to 1.0). This distribution is a consequence of the conservative choice of thresholds in Section 4, because conservative thresholds minimize false positives at the expense of potentially missing some errors. For production systems, this balance is often preferable, since false positives can reduce confidence, missed errors can be detected by other metrics from the same category and compensate for the low Recall of individual metrics.

In addition to the above assumption, Fig. 4 below depicts an experimentally obtained comparison of the results of the combined metrics with respect to the individual independent metrics.

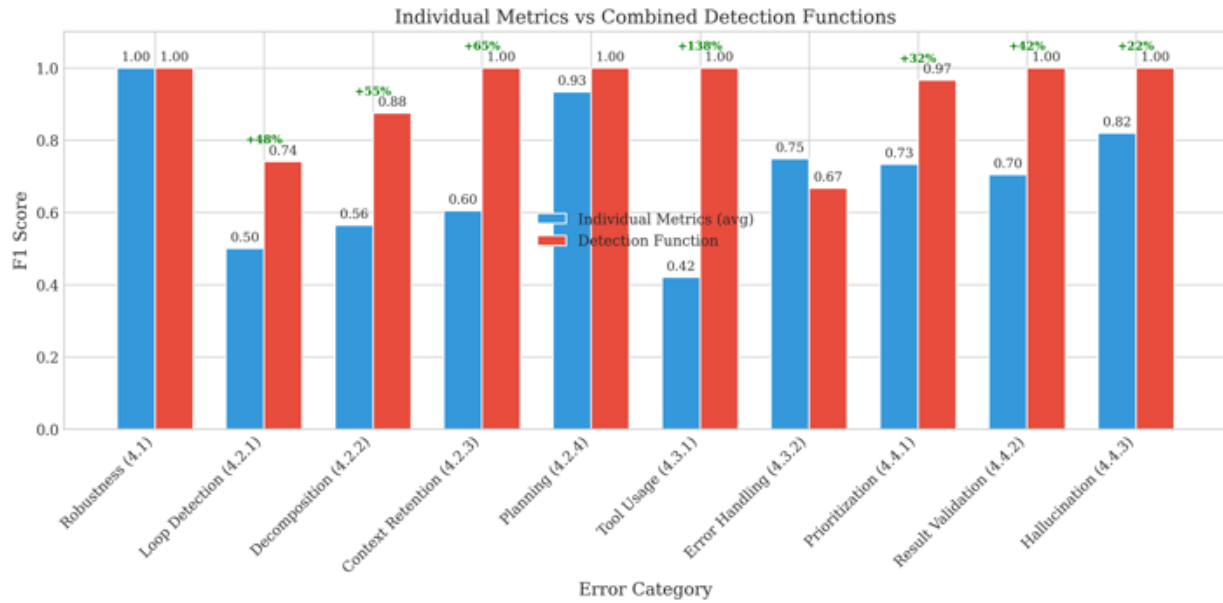


Fig. 4 Comparison of F1-score of individual metrics (category average) and combined detection functions. Percentages indicate relative improvement.

Functions `detect*_issue`, which aggregate several metrics taking into account their threshold values, consistently outperform individual metrics in terms of F1. The most pronounced improvement is observed in the following categories:

- 1) Tool Usage: individual metrics F1 = 0.421, `detect_tool_usage_issue` F1 = 1.000 (137% improvement)
- 2) Context Retention: individual metrics F1 = 0.605, `detect_context_issue` F1 = 1.000 (65% improvement)
- 3) Decomposition: individual metrics F1 = 0.555, `detect_decomposition_issue` F1 = 0.875 (58% improvement)

This result confirms the feasibility of the multimetric approach, which is the basis of the proposed methodology, that the combination of several metrics makes it possible to compensate for the weaknesses of individual indicators and increase the overall reliability of error detection in the AI agent.

5.2 Discussion

The results obtained demonstrate that the proposed methodology is capable of effectively identifying characteristic problems in the work of AI agents, although with certain reservations. A Precision of 0.962 means that if the metric is triggered with a probability greater than 96%, there is indeed a problem.

The `detect*_issue` functions perform noticeably better than individual metrics because the combined functions consider multiple results from different metrics simultaneously, reducing the risk of false positives. All error categories from the proposed taxonomy are detected with acceptable quality, where the average Accuracy > 0.60 for all categories, but for most exceeds 0.80. If it is necessary to increase sensitivity (Recall), the threshold values can be adjusted by slightly reducing Precision.

The synthetic data used to test the methodology fully reproduces real errors that can be encountered in AI agents, while real systems may demonstrate more complex error patterns that were not taken into account in this study.

5.3 Comparative analysis with existing frameworks

To position the proposed methodology in relation to existing solutions, a comparative analysis was carried out with the two most well-known frameworks for evaluating AI systems: RAGAS and ARES.

Table 4. Metrics Comparison of the capabilities of AI agent Evaluation frameworks

Evaluation Aspect	RAGAS	ARES	Proposed methodology
Robustness	No	No	Yes (4 metrics)
Loop Detection	No	No	Yes (4 metrics)
Task Decomposition	No	No	Yes (4 metrics)
Context Retention	No	No	Yes (4 metrics)
Planning Effectiveness	No	No	Yes (4 metrics)
Tool Usage	No	No	Yes (4 metrics)
Error Handling	No	No	Yes (4 metrics)
Prioritization	No	No	Yes (2 metrics)
Result Validation	No	Partially	Yes (3 metrics)
Hallucination	No	Yes	Yes (5 metrics)
Context relevancy	Yes	Yes	Yes
Faithfulness	Yes	Yes	Yes
Answer relevancy	Yes	Yes	Yes

The key difference is that RAGAS focuses on evaluating the quality of RAG pipelines and provides metrics for evaluating context relevance (context precision, context recall), response faithfulness to sources (answer relevance) and query relevance (answer relevancy). However, RAGAS does not take into account the specifics of agent architectures, so within this framework there is no evaluation of the quality of planning, monitoring of the use of tools, analysis of multi-step processes and detection of cyclic behavior. ARES, which expands the RAGAS approach by using trained estimator models instead of LLM-as-a-judge, on the one hand, increases efficiency and reduces the cost of evaluation, but the functional coverage still remains similar to RAGAS. The proposed methodology significantly expands the scope of evaluation, adding 9 new categories of metrics specific to AI agents.

6.0 Practical Implementation of Evaluation Methods

Developing an effective evaluation system requires not only an understanding of the methodology but also a competent organization of the agent AI evaluation process. This section will consider the practical aspects of implementing the presented methods, namely how to organize a multi-level evaluation, what tools to use, how to automate the process, and when to involve an expert.

6.1 Organization of multi-level evaluation

Effective evaluation of AI agents must work at three levels, each of which identifies different types of problems.

Component evaluation tests individual agent AI abilities in isolation. At this level, it is evaluated how consistently the AI agent responds to variations in requests and whether it chooses the right tools for specific tasks. This is a basic level that identifies fundamental problems regardless of how components interact with each other. To implement the evaluation, it is necessary to create a test dataset of 50-100 typical queries, generate 3-5 variations for each (paraphrases, with typos, with different levels of detail), run the dataset

for the AI agent on all variants, and check whether the results remain semantically equivalent.

Integration evaluation focuses on verifying that the agent's AI components work together. In this case, an evaluation is made of the correctness of maintaining context between steps, the consistency of multi-step execution, and the correctness of information transfer between components. Problems at this level arise not due to the failure of individual parts, but due to the incorrect interaction of tools within the agent AI. To organize such an evaluation, it is necessary to create tasks that require 5-7 consecutive steps, where each next step uses the results of the previous ones. Using agent AI execution tracing tools that log what information is passed between steps, it is necessary to check whether critical context is lost or whether intermediate conclusions contradict each other.

System evaluation should already be aimed at full-fledged end-to-end scenarios in conditions as close as possible to the real work of the AI agent. This level identifies problems that only appear in agent-wide workflows, such as round-robin behavior for complex tasks, incorrect prioritization of requests, and cascading errors with sequential failures. Preparation for this type of evaluation requires the development of 20-30 realistic use cases that reflect typical workflows. If possible, these requests can include tasks that could lead to a failure of the agent's AI in advance. If creating such tasks is impossible, then it is necessary to emulate errors and failures on the side of the AI agent manually, for example, launch the agent with timeouts to detect cycles, and deliberately stop the work of the tools in a cascade during the execution of the request.

6.2 Preparation of test data

Preparing the dataset to perform the evaluation is a key factor for successful evaluation and identification of all problems in the operation of the AI agent. Test data must include:

- Coverage of task types: Include examples of all categories of tasks that the AI agent will solve. For a universal agent, these can be simple one-time data requests, multi-step analytical tasks, tasks with conditional logic ("if you find X, then check Y"), or tasks that require parallel work with several sources.
- Subject area coverage: Tests should cover the entire range of topics that the AI agent works with. For a corporate agent, this includes technical documentation, business reports, correspondence, policies, and procedures. For a customer support agent, typical questions, escalation scenarios, and multi-step dialogues.
- Coverage of edge cases: It is necessary to include queries that provoke each type of problem from section 3. To detect loops, these could be queries for unreachable information. To check the decomposition, there may be tasks that require complex planning. To test context loss, tasks with a long sequence of dependent actions.

Despite the possibilities of automation and automatic calculation of metrics, human expertise remains an important element in the process of evaluating an AI agent. Automated metrics may be good at identifying quantifiable problems such as inconsistencies, logical errors, and loops, but they may be poor at evaluating semantic correctness and contextual adequacy. First of all, the expert should focus on checking the results of the automatic evaluation. It is imperative to look for false positives when evaluating metrics and false omissions, and then adapt your evaluation algorithms based on the data obtained. Additionally, many important metrics require expert judgment and thoughtful evaluation criteria:

- The actual relevance of the information to the request (not just a match of keywords)
- The adequacy of the balance between completeness and brevity of the answer

- Compliance of prioritization with established standards
- The correctness of identifying subtle violations in the process of the AI agent

In addition, an important source of data for evaluation can be the data of real users who are already using the AI agent. Experts, when analyzing production data, can identify new error patterns that were not included in the original taxonomy. These new types of problems should lead to expanded test coverage and the development of new evaluation methods.

6.3 Threshold Calibration

The threshold values proposed in this methodology (for example, Consistency Rate ≥ 0.95 , Context Retention ≥ 0.85) are formed based on the following justifications. Industry best practices for building reliable software suggest that mission-critical systems should maintain 95%+ reliability and fault tolerance under normal operating conditions (Rausand & Høyland, 2004).

This work applies similar standards to the robustness of AI agents, although the nature of agent systems (with their probabilistic behavior and dependence on external tools) may require some adjustment of these thresholds. It is also worth taking into account the domain specificity of the AI agent, which may influence the requirements for threshold values. For example, medical AI agents working with diagnostic information require a Hallucination Rate ≤ 0.01 , implying almost zero tolerance for factual errors. On the other hand, AI agents for working with creatives or generating new ideas can allow values up to 0.10 without critically compromising the quality of work of such an AI agent.

Organizations should also conduct their own sensitivity analysis for their specific use cases and configure thresholds for their application domains. This remains a limitation of the current work, which we plan to address in subsequent studies with a broader empirical base.

7.0 Conclusion

AI agents are becoming a critical tool for automating complex tasks in business and technology. However, their implementation often faces reliability issues that lead to project failure. The reason is that AI agents are fundamentally more complex systems than traditional language models, as they plan actions, call external tools, work with data, and make decisions based on multi-step analysis. Accordingly, the problems in their work are of a qualitatively different nature.

In this work, three interrelated problems were solved. First, we systematized typical problems in the work of AI agents, creating a structured classification of nine categories of errors. Management and planning problems include endless loops, incorrect task decomposition, and loss of context between actions. Tooling problems include inefficient use of resources and poor error handling. Reasoning and validation problems include overplanning, incorrect prioritization, lack of validation, and hallucinations. Second, a comprehensive evaluation methodology was developed with specific metrics and success criteria for each type of problem. The methodology is organized in four areas: evaluating robustness to variations in input data through metamorphic testing, evaluating execution management and planning, evaluating tool use, evaluating the quality of reasoning, and validating results. Each direction contains specific evaluation procedures, formulas for calculating metrics and quantitative thresholds of acceptable quality. Thirdly, there offered practical recommendations for implementing an evaluation system in real conditions. It showed how to organize multi-level testing from evaluating individual components to end-to-end scenarios, how to prepare high-quality test data, and how to effectively combine automatic evaluation with expert validation.

The practical value of the work lies in the fact that the proposed methodology can be applied at any stage of the life cycle of an AI agent. The modular design of the methodology allows

organizations to start with a basic evaluation of the most critical aspects and gradually expand coverage. It is important to note several limitations of the current work. The proposed metrics and thresholds are based on general reliability principles, but specific values may require calibration for specific domains and applications. Also, some aspects of the quality of AI agents, such as creativity, empathy in dialogue, or the adequacy of the tone of communication, are difficult to formalize quantitatively and require mainly expert assessment. Finally, as AI agent architectures evolve, new types of problems may emerge that are not covered by the current taxonomy.

The proposed methodology provides practical tools for significantly increasing the reliability of AI agents. Systematic application of the described evaluation methods allows for identifying problems at early stages, preventing their appearance in production, and ensuring the predictable quality of work of AI agent systems. This is critical for the successful implementation of AI agents in business processes and for realizing their potential to automate complex tasks that require not only data processing but also meaningful decision-making.

The complete experimental dataset and results of each method are public and available at: <https://github.com/meshkovQA/AI-robustness.git>

Conflict of Interest Statement

The author declares no conflict of interest.

Acknowledgment

The author would like to thank the reviewers for their valuable feedback and suggestions that helped improve this manuscript.

References

- [1] A. Bandi, B. Kongari, R. Naguru, S. Pasnoor, and S. V. Vilipala (2025). The Rise of Agentic AI: A Review of Definitions, Frameworks, Architectures, Applications, Evaluation Metrics, and Challenges. ResearchGate. Available: <https://www.researchgate.net>
- [2] Q. Huang, N. Wake, B. Sarkar, and Z. Durante (2024). Position paper: Agent AI towards a holistic intelligence. arXiv:2403.00833 [cs.AI]
- [3] Grand View Research (2025). AI Agents Market (2025–2030). Tech. Rep. GVR-4-68040-471-3. Available: <https://www.grandviewresearch.com>
- [4] Massachusetts Institute of Technology (2025). The GenAI Divide: State of AI in Business 2025. Tech. Rep., Cambridge, MA.
- [5] Y. Shavit and S. Agarwal (2024). Practices for Governing Agentic AI Systems. White Paper, OpenAI, San Francisco, CA. Available: <https://cdn.openai.com/papers/practices-for-governing-agentic-ai-systems.pdf>
- [6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao (2023). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.AI]
- [7] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761 [cs.CL]
- [8] T. Y. Chen, S.-C. Cheung, and S. M. Yiu (2020). Metamorphic testing: a new approach for generating next test cases. arXiv:2002.12543 [cs.SE]
- [9] H. Reddy, M. Srinivasan, and U. Kanewala (2025). Metamorphic Testing for Fairness Evaluation in Large Language Models: Identifying Intersectional Bias in LLaMA and GPT. arXiv:2504.07982 [cs.SE]
- [10] W. Wu, Y. Cao, N. Yi, R. Ou, and Z. Zheng (2024). Detecting and Reducing the Factual Hallucinations of Large Language Models with Metamorphic Testing. Proc. ACM Softw. Eng., vol. 1, no. FSE, article 42, pp. 1–23. doi: 10.1145/3660780

- [11] S. Giramata, M. S. Venkat, N. Gudivada, and U. Kanewala (2025). Efficient Fairness Testing in Large Language Models: Prioritizing Metamorphic Relations for Bias Detection. arXiv:2505.07870 [cs.SE]
- [12] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, and Z. Liu (2024). Evaluation of Retrieval-Augmented Generation: A Survey. arXiv:2405.07437 [cs.IR]
- [13] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert (2024). RAGAS: Automated Evaluation of Retrieval Augmented Generation. In Proc. 18th Conf. European Chapter of the Association for Computational Linguistics: System Demonstrations (EACL '24), St. Julian's, Malta, pp. 150–158.
- [14] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia (2024). ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems. arXiv:2311.09476 [cs.CL]
- [15] S. Arcadinho, D. Aparicio, and M. Almeida (2024). Automated test generation to evaluate tool-augmented LLMs as conversational AI agents. arXiv:2409.15934 [cs.CL]
- [16] S. Gupta, R. Ranjan, and S. N. Singh (2024). A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions. arXiv:2410.12837 [cs.IR]
- [17] K. Zhang and D. Shasha (1989). Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM J. Comput.*, vol. 18, no. 6, pp. 1245–1262. doi: 10.1137/0218082
- [18] P. Bille (2005). A Survey on Tree Edit Distance and Related Problems. *Theoret. Comput. Sci.*, vol. 337, no. 1–3, pp. 217–239. doi: 10.1016/j.tcs.2004.12.030
- [19] K. Erol, J. Hendler, and D. S. Nau (1994). HTN Planning: Complexity and Expressivity. In Proc. 12th Nat. Conf. Artificial Intelligence (AAAI-94), pp. 1123–1128. AAAI Press.
- [20] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman (2003). SHOP2: An HTN Planning System. *J. Artif. Intell. Res.*, vol. 20, pp. 379–404. doi: 10.1613/jair.1141
- [21] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung (2015). *Time Series Analysis: Forecasting and Control*, 5th ed. John Wiley & Sons, Hoboken, NJ.
- [22] H. K. Khalil (2002). *Nonlinear Systems*, 3rd ed. Prentice Hall, Upper Saddle River, NJ.
- [23] M. Wooldridge (2009). *An Introduction to Multiagent Systems*, 2nd ed. John Wiley & Sons, Chichester, UK.
- [24] H. Ebbinghaus (1885). *Über das Gedächtnis: Untersuchungen zur experimentellen Psychologie*. Duncker & Humblot, Leipzig.
- [25] D. C. Rubin and A. E. Wenzel (1996). One Hundred Years of Forgetting: A Quantitative Description of Retention. *Psychol. Rev.*, vol. 103, no. 4, pp. 734–760. doi: 10.1037/0033-295X.103.4.734
- [26] J. T. Wixted and E. B. Ebbesen (1991). On the Form of Forgetting. *Psychol. Sci.*, vol. 2, no. 6, pp. 409–415. doi: 10.1111/j.1467-9280.1991.tb00175.x
- [27] J. R. Anderson and L. J. Schooler (1991). Reflections of the Environment in Memory. *Psychol. Sci.*, vol. 2, no. 6, pp. 396–408. doi: 10.1111/j.1467-9280.1991.tb00174.x
- [28] S. Hochreiter and J. Schmidhuber (1997). Long Short-Term Memory. *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780. doi: 10.1162/neco.1997.9.8.1735
- [29] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker (1999). Web Caching and Zipf-like Distributions: Evidence and Implications. In Proc. IEEE INFOCOM '99, vol. 1, pp. 126–134. doi: 10.1109/INFOCOM.1999.749260
- [30] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33. doi: 10.1109/TDSC.2004.2
- [31] K. J. Åström and B. Wittenmark (2013). *Adaptive Control*, 2nd ed. Dover Publications, Mineola, NY.
- [32] N. G. Leveson (1995). *Safeware: System Safety and Computers*. Addison-Wesley, Reading, MA.
- [33] M. Rausand and A. Høyland (2004). *System Reliability Theory: Models, Statistical Methods, and Applications*, 2nd ed. John Wiley & Sons, Hoboken, NJ.
- [34] M. T. Nygard (2007). *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, Raleigh, NC.

- [35] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung (2023). Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.*, vol. 55, no. 12, article 248, pp. 1–38. doi: 10.1145/3571730
- [36] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen, L. Wang, A. T. Luu, W. Bi, F. Shi, and S. Shi (2023). Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. arXiv:2309.01219 [cs.CL]
- [37] T. L. Saaty (2008). Decision Making with the Analytic Hierarchy Process. *Int. J. Services Sci.*, vol. 1, no. 1, pp. 83–98. doi: 10.1504/IJSSCI.2008.017590
- [38] P. K. Manadhata and J. M. Wing (2011). An Attack Surface Metric. *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 371–386. doi: 10.1109/TSE.2010.60
- [39] D. J. C. MacKay (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, UK.
- [40] T. M. Cover and J. A. Thomas (2006). *Elements of Information Theory*, 2nd ed. John Wiley & Sons, Hoboken, NJ.
- [41] J. Neyman and E. S. Pearson (1933). On the Problem of the Most Efficient Tests of Statistical Hypotheses. *Philos. Trans. Roy. Soc. London Ser. A*, vol. 231, pp. 289–337. doi: 10.1098/rsta.1933.0009
- [42] C. M. Bishop (2006). *Pattern Recognition and Machine Learning*. Springer, New York, NY.
- [43] K. Popat, S. Mukherjee, J. Strötgen, and G. Weikum (2017). Where the Truth Lies: Explaining the Credibility of Emerging Claims on the Web and Social Media. In *Proc. 26th Int. Conf. World Wide Web (WWW '17)*, pp. 1003–1012. doi: 10.1145/3038912.3052596
- [44] J. Thorne and A. Vlachos (2018). Automated Fact Checking: Task Formulations, Methods and Future Directions. In *Proc. 27th Int. Conf. Computational Linguistics (COLING 2018)*, pp. 3346–3359. Association for Computational Linguistics.
- [45] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, V. Sable, C. Li, and M. Tremayne (2017). ClaimBuster: The First-ever End-to-end Fact-checking System. *Proc. VLDB Endowment*, vol. 10, no. 12, pp. 1945–1948. doi: 10.14778/3137765.3137815
- [46] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal (2018). FEVER: A Large-scale Dataset for Fact Extraction and VERification. In *Proc. 2018 Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018)*, vol. 1, pp. 809–819. doi: 10.18653/v1/N18-1074