ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

Hybrid Architecture of Intelligent Systems with a Deterministic Core: from Concept to Prototype

Vladimir Rotkin[1]

[1]PhD, Independent Researcher New Israel Association of Sciences Netanya, Israel

Publication Date: 2026/09/04

Abstract: The rapid development of large language models and autonomous intelligent agents has significantly expanded the capabilities of natural language processing and decision support. However, practical implementation reveals fundamental limitations, particularly in tasks requiring computational robustness, reproducibility of results, and strict information consistency. These issues are particularly critical in fields such as engineering, geometry, and educational systems, where plausible but inaccurate responses ("hallucinations") and unstable behavior undermine system trust. This paper proposes a hybrid intelligent architecture with a deterministic core to address these challenges. Unlike fully autonomous systems, the proposed approach decouples functions: an adaptive agent handles user interaction and its interpretation, while a stationary deterministic core provides robust computation, logical consistency, and graphical display. The architecture introduces a clear distinction between the development phase, which allows for iterative improvement, and the operational phase, characterized by a fixed core that guarantees reproducible and verifiable results. By providing protocol-based interaction between the agent and the deterministic core, the system ensures that all generated output—text, computational, and graphical—remains consistent and adheres to the underlying domain model. This hybrid structure combines the flexibility of modern intelligent agents with the precision and reliability of formal deterministic models, offering a robust foundation for mission-critical intelligent applications.

Keywords: *Hybrid Intelligent Architecture, Deterministic Core, Stationary Kernel, Intelligent Agent, Reproducibility of Results, Configurational Tasks, Geometric Problem Generator, Protocol-Based Interaction.*

How to Cite: Vladimir Rotkin (2026) Hybrid Architecture of Intelligent Systems with a Deterministic Core: from Concept to Prototype. *International Journal of Innovative Science and Research Technology*, 11(8), 2731-2753. https://doi.org/10.38124/ijisrt/26aug1186

I. INTRODUCTION

The rapid development of large-scale language models and agent-based intelligent systems has led to a significant expansion of their capabilities in natural language processing, programming, information analysis, and decision support. However, accumulated practical experience has revealed a number of fundamental limitations of such systems, particularly noticeable in tasks requiring computational reliability, reproducibility of results, and strict consistency across different information representations.

➢ *Review of Sources*

One of the most well-known works of recent years is the study by Bubeck et al. [1], devoted to the analysis of an early version of GPT-4. The authors demonstrate the model's capabilities in mathematics, programming, medicine, law, and a number of other fields. At the same time, the study specifically focuses on the limitations of the language model and the challenges of further development of such systems. The authors note that even highly powerful models retain significant shortcomings related to the reliability of reasoning, the stability of behavior, and the reproducibility of results. Particular attention is paid to the need for new architectural solutions that go beyond simply predicting the next word. Significantly, the discussion surrounding the work by Bubeck et al. has largely focused on issues of reliability and verifiability of results. Many researchers have pointed out that even highly successful GPT-4 responses cannot always be reproduced with repeated queries, and the internal mechanisms for decision-making remain insufficiently transparent. As a result, the problem of independent verification of the results of intelligent systems arises.

The problem of hallucinations in language models occupies a special place in modern literature. A detailed review of this phenomenon is presented in the work of Ji et al. [2]. The authors demonstrate that generative models are capable of producing texts that appear logical and persuasive, but contain unreliable or completely fictitious information. Such errors arise in summarization tasks, dialog systems, response generation, and other application areas. The researchers emphasize that hallucinations are not a random defect of individual models, but represent a fundamental problem of modern text generation systems. Particularly important is the fact that hallucinations can affect not only textual information but also logical connections, calculations, explanations, and interpretations. In problems of engineering

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

design, geometry, or learning, such errors can lead to inconsistencies between calculations, graphical representations, and textual explanations.

The technical report on GPT-4 [3] also points to significant limitations of current language models. The authors note the potential for generating false information, the sensitivity of responses to query formulation, the lack of robustness of behavior, and the need for external verification of results. Of particular importance is the model's ability to formulate plausible but erroneous explanations for its own actions. Such situations complicate error diagnosis and reduce trust in the intelligent system's results.

Further development of agent-based approaches has led to studies examining the interaction of language models with external tools. Yao et al. [4] proposed the ReAct approach, which combines language model reasoning with external actions. The authors demonstrate that the "reasoning-action-result" sequence significantly improves the quality of solutions to complex problems. Crucially, in such systems, computational procedures are gradually moved beyond the language model. The model generates queries and interprets results, while the actual computations are performed by specialized tools.

Similar ideas are developed in the work of Schick et al. [5], which examines the use of external tools by language models. The authors demonstrate that using calculators, search engines, and specialized services can significantly improve the reliability of solutions. Moreover, the language model itself performs coordination and interpretation functions, rather than direct computation.

Self-correction mechanisms for intelligent agents have also been actively studied in recent years. Shinn et al. [6] examine agent systems capable of analyzing their own errors and correcting their subsequent behavior. However, the authors note that such correction is often context-specific and does not always transfer to new tasks. This is of particular interest in the context of hybrid systems. Practical experience shows that even after extensive training, an agent may partially lose its acquired discipline when transitioning to a new dialogue or a new work environment. Consequently, some of the limitations of modern agents are not due to a lack of instructions, but to the insufficient stability of their behavioral mechanisms.

Significant attention in modern literature has also been devoted to the problem of controlling the autonomous behavior of intelligent systems. Amodei et al. [7] examine practical issues of artificial intelligence security, including goal misalignment, excessive initiative, and agent error. The authors emphasize the need to develop mechanisms for constraining the behavior of intelligent systems and external control of decision-making.

Russell [8] develops similar ideas, addressing the issue of compatibility between intelligent systems and user goals. According to the author, highly autonomous systems should operate within external constraints and not substitute their own optimality criteria for user goals.

A review of modern extended language model architectures by Mialon et al. [9] shows that the further development of intelligent systems is increasingly associated with the integration of multiple components: a language model, memory, external knowledge bases, specialized tools, and computational modules. The authors view such systems as one of the most promising areas for the development of artificial intelligence.

Considerable attention is paid to the issues of reliability and trust in intelligent systems in the engineering literature on artificial intelligence. A review by Kaur et al. [10], published in ACM Computing Surveys, examines the concept of trustworthy artificial intelligence (Trustworthy AI). The authors highlight such properties of intelligent systems as reliability, transparency, explainability, reproducibility, and controllability. They emphasize that the high quality of individual responses alone does not guarantee system reliability if there are no mechanisms for monitoring and verifying the results. Particularly important is the conclusion regarding the need to combine intelligent methods with formal verification procedures and external control. The authors note that trust in a system is determined not only by the quality of the model but also by the architecture of interactions between the user, computing components, and control mechanisms.

The problem of constraining the behavior of autonomous intelligent systems is also considered in the work of Jennings and Wooldridge [11], dedicated to the engineering of agent systems. The authors emphasize that agent autonomy should not be viewed as an absolute property. Instead, effective agent systems are built on the basis of distributing authority, constraints, and rules of interaction between system components. This conclusion is of fundamental importance: development practice shows that excessive independence of a language agent can lead to protocol violations, substitution of computational results, and reconfiguration of the task. Therefore, limited agent autonomy can be considered a necessary condition for the reliable operation of an intelligent system.

Miller's [12] work on the explainability of intelligent systems made an important contribution to the problem of human-artificial intelligence interaction. The author demonstrates that explanations generated by intelligent systems do not always reflect the real reasons behind their decisions. In many cases, the system generates a plausible explanation only after receiving the result. This finding is of particular interest in relation to language agents. Practical experience shows that an agent can acknowledge an error or protocol violation, but is unable to reliably explain the reasons for its own behavior.

Additional arguments in favor of using deterministic components of intelligent systems are presented in the work of Rotkin, Yavich, and Malev [13], devoted to the concept of knowledge generators based on simulation-ontological models. The authors consider an alternative approach to

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

artificial intelligence based on the elimination of the "black box," the use of direct mathematical modeling, and the dominance of data transformation systems. A domain simulator is proposed as a basic element, generating structured sets of functional relationships and parameters. Particular attention is paid to the generation of new knowledge based on parametric analysis of the configuration space, rather than extracting information from large databases. This approach views an intelligent system as a set of deterministic models, ontological structures, and specialized interfaces.

A special area of research is related to educational intelligent systems and the automated generation of educational content. Unlike modern language models, which focus primarily on text generation, these studies consider the educational environment as a configuration system based on domain models and learning object generators.

In the monograph by Zvolinsky, Rotkin, Golovin and Matveeva [14], a methodology for the automated formation of a new generation of educational content is proposed. The authors consider the educational course as a multiparameter model capable of generating personalized educational materials, practical assignments, test materials and graphical representations. Particular attention is paid to generators of educational assignments, configuration models of educational courses and specialized electronic interfaces. The authors emphasize the limitations of traditional testing systems and the need to move towards variable, interactive and adaptive didactic systems. The proposed classification of electronic didactic systems includes the transition from normative and test didactics to simulation and project systems using the generation of educational objects, feedback and interactive interaction. Of particular interest is the introduction of the concepts of polyvariability, configurationality and simulation synthesis of educational tasks, which turns out to be close to the concept of a configuration task considered in this paper.

Further development of the educational direction is presented in the work of Rotkin [15], dedicated to the methodology of immanent educational content. The author considers the transition from traditional knowledge bases to generators of educational content operating on the basis of parametric simulation models. Unlike systems that extract ready-made information from large databases, the generator generates educational material directly at the moment of the user's request. Of particular importance for the present study is the introduction of a general simulation model of content, including subject, ontological and interface components. On the basis of such a model, particular parametric sections corresponding to educational topics, tasks, theorems and different levels of complexity can be formed. Varying free parameters allows for the creation of unique task configurations for each user. It is noteworthy that, as an illustration of immanent content, the author considers the problem of calculating a triangle, the parameters of which are generated automatically, and different combinations of sides and angles generate various solution methods. Thus, the ideas of the configuration space of a problem, parametric generation and consistent graphical representation were formulated already in educational systems, which are further developed in the present work.

Thus, research in the field of automated generation of educational content forms an independent educational field, preceding work on simulation-ontological intelligent systems and configurational artificial intelligence. Within this field, ideas for parametric domain models, learning object generators, configurational task spaces, specialized interfaces, and consistent graphical representation of results have been formulated. This research can be considered one of the methodological prerequisites for hybrid intelligent systems with a deterministic core.

Further development of these ideas is presented in the work of Yavich et al. [16], devoted to configurable intelligent systems based on hierarchical simulation models. The authors consider deterministic artificial intelligence as an alternative to neural network learning in scientific, engineering, and commercial applications. They propose the use of hierarchical modeling, benchmark training, and partially empirical algorithms instead of classical machine learning. Of particular importance for this work is the conclusion on the need for a phased formation of algorithmic modules with mandatory cycles of testing, modification, and verification. In essence, the authors propose an iterative approach to the development of intelligent systems, close to the method of sequential interaction between the developer and the intelligent agent considered in this paper.

An interesting example of the practical implementation of the configuration approach is contained in the patent of Yavich et al. [17], dedicated to the intelligent generation of parameters for designed objects. In the proposed system, user preferences and external constraints are mapped into the parameter space, after which a search for feasible solutions is performed, taking into account geometric constraints. An important feature of this approach is the separation of the stages of obtaining preferences, generating the configuration space, searching for feasible solutions, and generating the final result. In fact, this architecture implements the sequence: parameters → constraints → configuration → optimum → result. This separation of computational and interface functions turns out to be close to the separation of an intelligent agent and a deterministic core proposed in this paper.

Thus, the modern literature quite consistently points to a number of fundamental limitations of language and agent-based models:

- Limited reproducibility of results;
- Possibility of hallucinations;
- Insufficient explainability;
- Behavioral instability;
- Difficulties of independent verification;
- Excessive autonomy of agents;
- Dependence of results on the context of interaction.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

Most modern research suggests strengthening the role of external, specialized tools. Consequently, the current literature is gradually shifting its focus from fully autonomous intelligent systems to hybrid architectures that combine intelligent models, external computing tools, verification mechanisms, and human oversight. This is precisely the direction in which the hybrid architecture proposed in this paper develops, in which an intelligent agent performs interaction and interpretation functions, while computational, logical, and graphical validity are provided by a stationary, deterministic core.

➢ *Discussion*

Modern intelligent systems based on large language models demonstrate high performance in natural language processing, text generation, data analysis, and decision support. The development of agent-based approaches has further expanded the capabilities of such systems, enabling the implementation of complex user interaction scenarios, sequential problem solving, and the integration of specialized computing tools.

However, the widespread use of nondeterministic language models has revealed a number of fundamental limitations. These include limited reproducibility of results, the possibility of generating unreliable information, instability of solutions upon repeated queries, and difficulties in formally verifying computations and graphical representations.

These problems are particularly acute in subject areas that require strict computational and visual consistency of results. These areas include geometry, engineering calculations, design, computer-aided design systems, physical modeling, and educational simulators. In such problems, obtaining a plausible textual answer is not enough. It is necessary to ensure the correctness of the calculations, the reproducibility of the results, and the consistency of the graphical representations with the internal model of the subject area.

Existing agent-based systems primarily consider the language model as the primary source of knowledge and decisions, with specialized computational modules and external tools serving as auxiliary components. However, this approach does not always ensure the required degree of reliability and verifiability of results.

This paper proposes an alternative architectural approach based on the use of a stationary deterministic core within an intelligent system. The proposed hybrid architecture separates the functions of the intelligent agent and the computational core. The agent provides user interaction, interprets queries, and presents results, while the deterministic core performs computations, implements the domain model, and generates consistent graphical representations.

A distinctive feature of the proposed approach is the division of the system lifecycle into two stages. During the development stage, an intelligent agent is allowed to participate in the creation, modification, and testing of the kernel. After loading and activating the production version, the kernel and interaction protocols become static and do not change during operation. This approach ensures reproducibility of results, the ability to verify, and the stability of system behavior.

The graphical component of the deterministic kernel plays a significant role in the proposed architecture. Unlike systems in which visualization is an external or post-processing step, graphical representations are generated directly from the computational model and are part of the kernel's output. For a number of subject areas, graphical consistency is a prerequisite for the practical applicability of an intelligent system.

The aim of this work is to develop the concept of a hybrid intelligent system with a deterministic core, formulate the basic principles of such an architecture and demonstrate its practical implementation using the example of a specialized prototype.

The paper examines the basic concepts and principles of the proposed architecture, analyzes the stages of the system life cycle, examines the role of a deterministic kernel with a graphical component, and presents the results of the development and testing of an experimental prototype.

➢ *Basic Concepts*

For further discussion, a number of terms are introduced that are used in describing the proposed architecture.

- A hybrid intelligent system is a system that combines an intelligent agent and a deterministic core.
- An intelligent agent is a component that provides interaction with the user and interpretation of results.
- A deterministic kernel is a computing subsystem that implements an unambiguous transformation of input data into results.
- A stationary core is a deterministic core that does not change during the use of the system.
- An interaction protocol is a set of rules for exchanging data between an agent and the core.
- A production version is an activated implementation of a system, including the kernel and interaction protocols.

The proposed architecture is based on the following principles: determinism, stationarity, reproducibility of results, graphical consistency and separation of functions between the agent and the deterministic core.

II. METHODOLOGY

The proposed methodology is based on dividing an intelligent system into an adaptive agent component and a stationary deterministic core. Unlike fully agent-based systems, this approach separates user interaction, query interpretation, and result presentation functions from computational, graphical, and configuration procedures.

The methodological basis of the work includes a description of the system architecture, a lifecycle analysis,

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

defining the roles of the intelligent agent and the deterministic core, and developing protocols for their interaction. Particular attention is paid to ensuring the reproducibility of results, the consistency of various information representations, and the robustness of the system's behavior.

➢ *General Architecture of the Hybrid System*

The proposed hybrid architecture of an intelligent system is based on the separation of functions between an intelligent agent and a deterministic core. This separation allows for the combination of the adaptability of modern intelligent models with the reproducibility and verifiability of specialized computational algorithms.

In its most general form, the structure of the system can be represented by the sequence shown in Fig. 1.

The user formulates queries, sets task parameters, and receives the system's results. Direct interaction with the user is performed by an intelligent agent, which analyzes queries, interprets the context of the problem domain, and facilitates dialogue. The agent converts user queries into calls to the deterministic core, receives computational results, and presents them in a user-friendly form.

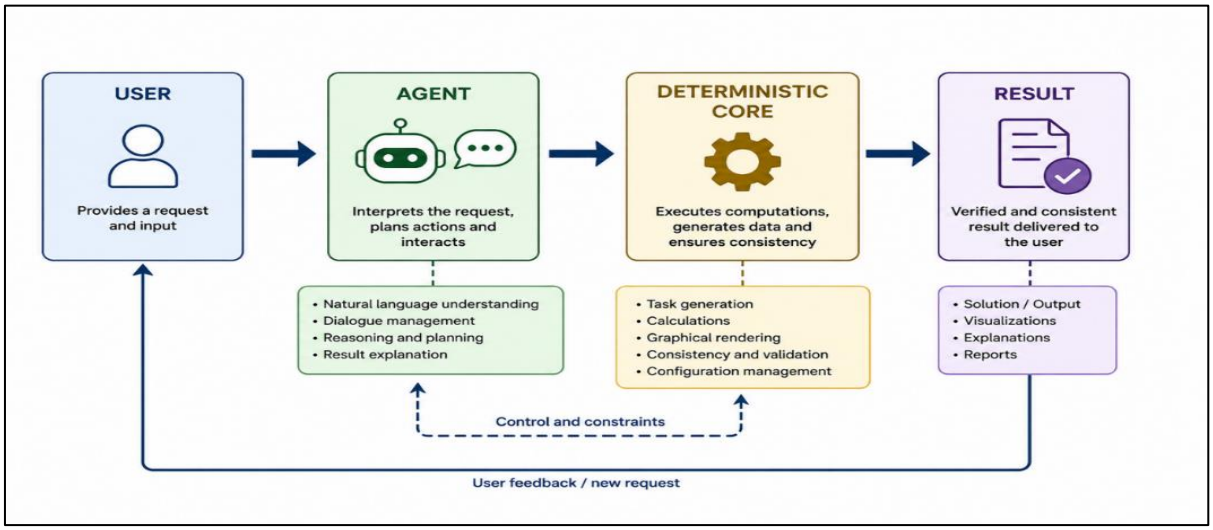


Fig 1 Hybrid Architecture of an Intelligent System

A deterministic kernel, by contrast, is responsible for implementing the domain model, performing calculations, constructing graphical objects, and verifying the correctness of results. It may include mathematical models, computational algorithms, object generators, validation procedures, and graphical components. For a fixed kernel version, the same input data always produces the same results, giving the computational part of the system determinism and reproducibility.

One of the most important features of the proposed architecture is the division of the system lifecycle into the development stage and the implementation stage. The development stage allows for the creation and modification of the core, the development of interaction protocols, and the testing and validation of the system. During this period, the intelligent agent can be used as a tool for analyzing, programming, and improving individual components.

After loading and activating the production version, the system enters the deployment phase. During this period, the deterministic kernel and interaction protocols remain unchanged. The agent uses only the activated kernel version and does not modify its algorithms, internal structure, or graphical procedures. This separation ensures robust system behavior and the ability to independently verify results.

The graphical component of the deterministic kernel plays a special role in the proposed architecture. Unlike systems in which visualization is performed as an external data processing step, graphical representations are generated directly based on the internal domain model and are part of the kernel's output.

For visually oriented subject areas, a graphical component becomes a necessary element of a deterministic system. It ensures consistency between calculations, geometric constructions, and visual representations. As a result, numerical data, graphical objects, and textual explanations describe the same internal configuration.

Thus, the proposed architecture combines the adaptability of an intelligent agent with the computational, logical, and graphical reliability of a deterministic core. The agent ensures flexible interaction and interpretation of results, while the core guarantees their reproducibility, verifiability, and consistency.

➢ *System Life Cycle*

The lifecycle of a hybrid intelligent system with a deterministic core consists of two fundamentally distinct stages: the development stage and the implementation stage. This separation is a key element of the proposed architecture, as it allows for the intelligent agent's adaptability to be

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

balanced with the requirements of reproducibility and sustainability.

During the development phase, computational algorithms are created, a domain model is formed, and graphical components and interaction protocols are developed. Simultaneously, testing, validation, and preparation of the working version of the system are carried out. During this period, the intelligent agent can be used as a programming tool, for finding solutions, analyzing errors, and improving individual core components.

The development process is typically iterative. Algorithm changes are accompanied by testing, the results of which lead to refinement of the models, after which the system is re-tested. Such cycles can be repeated many times until a satisfactory working version is achieved.

The result of the development stage is an activated version of the system, including a deterministic core, a graphical component, interaction protocols, and means for the agent to access the core. After activating this version, the system moves to the deployment stage.

The deployment phase represents the period of practical use of the activated system. The user interacts exclusively with the intelligent agent, which accepts requests, accesses the core, and presents the results. The deterministic core itself remains unchanged.

During operation, no changes are permitted to the computational algorithms, graphical procedures, kernel structure, or communication protocols. All results are generated exclusively based on the activated version of the system. This ensures reproducibility of results, robustness of behavior, and the possibility of independent verification.

The transition from the development stage to the production stage is accomplished by downloading and activating the production version. Any new kernel modification requires returning the system to the development stage, implementing the necessary changes, and creating a new version. Thus, system evolution occurs between successive versions, not during actual use.

Separating the lifecycle allows the intelligent agent to be used as a tool for system development and maintenance, while simultaneously maintaining the stability of the deterministic core during use. As a result, the lifecycle combines development adaptability and operational stability, which is one of the most important features of the proposed architecture.

- *Deterministic Stationary Kernel*

The central element of the proposed architecture is a deterministic core. Unlike many modern intelligent systems, whose internal computational procedures can change during operation, the proposed approach assumes the existence of a dedicated computational subsystem that ensures reproducibility, verifiability, and consistency of results.

A deterministic core is a set of interconnected models, algorithms, constraints, and procedures that implement a subject area and ensure the unambiguous transformation of input data into results.

Let X denote the input data, K the deterministic kernel, and R the output of the system. Then the mapping is performed

$$R = K(X).$$

For identical input data, the condition is satisfied

$$X_1 = X_2 \Rightarrow K(X_1) = K(X_2),$$

Which directly ensures the reproducibility of the results.

One of the fundamental properties of the kernel is its stationarity. Stationarity refers to the immutability of computational algorithms, graphical procedures, and interaction protocols after loading and activating the running version of the system.

If K(t) denotes the state of the nucleus in time, then at the application stage the condition is satisfied

$$K(t) = K_0.$$

Thus, the structure of the kernel and its internal procedures do not change during operation.

The need for stationarity stems from the limitations of modern adaptive intelligent systems. Variations in internal parameters, learning during use, and dynamic modification of algorithms can lead to the inability to reproduce results, hinder independent verification, and cause changes in system behavior. In engineering, educational, and scientific applications, such variability is often unacceptable.

A deterministic core may include domain models, computational algorithms, object generators, verification and validation procedures, graphical components, parameter databases, and internal constraints. However, individual core elements should not be considered independent modules. They form a unified configuration of the domain system, in which changes to one component can impact calculations, geometric constructions, verification procedures, and graphical representations.

The kernel's graphical component plays a special role. Visual representations are generated directly from the internal domain model and are part of the system's output. This ensures consistency between calculations, graphical objects, and textual explanations.

Subject-matter knowledge is localized primarily within the deterministic core. This core contains object models, constraints, parameter relationships, acceptance criteria, and decision-making algorithms. An intelligent agent accesses this knowledge through interaction protocols but does not substitute its own reasoning for it.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

System development is permitted only during the development phase. New kernel versions may include additional models, algorithms, and graphical routines:

$$K_1 \rightarrow K_2 \rightarrow \cdots \rightarrow K_n.$$

In this case, only one activated version is used during application:

$$K = K_n = \text{const}.$$

An intelligent agent can participate in algorithm development, bug detection, and system improvement. However, once the production version is activated, the agent uses only the fixed core and does not modify its computational algorithms, structure, or graphical procedures.

Thus, the deterministic kernel simultaneously serves as a carrier of subject-matter knowledge, a computing system, a constraint system, a means of graphical representation, and a source of reproducible results. Its stationarity ensures the stability of the system's behavior, the possibility of independent verification, and the consistency of various information representations.

➢ *The Role of an Intelligent Agent*

An intelligent agent is an adaptive component of a hybrid system and facilitates interaction between the user and the deterministic core. Unlike the core, an agent's operation can be non-deterministic. The agent can interpret requests, consider context, analyze domain-specific features, and generate calls to the computing subsystem. Thus, the agent functions as the system's intelligent interface.

The agent's primary tasks are receiving and interpreting user requests, organizing dialogue, generating calls to the deterministic core, processing computation results, and preparing textual explanations. The agent facilitates interaction with the system, but is not the source of computational validity for the results.

During the development phase, the agent can be used as a tool for creating and improving the system. It can participate in algorithm development, code generation, error detection, testing, results analysis, and the preparation of new versions of the software system. In this mode, the agent becomes a tool for developing and maintaining the deterministic kernel. The iterative development process is shown in Figure 2.

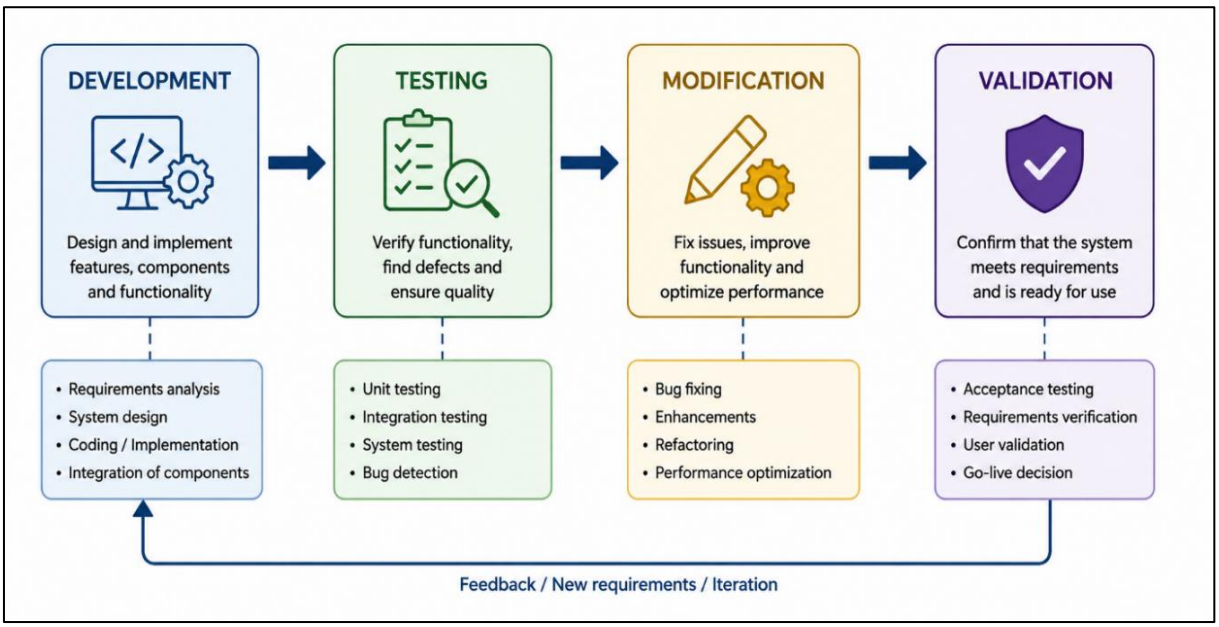


Fig 2 Iterative Process of Developing a Hybrid Intelligent System

In the proposed architecture, the reliability of results is ensured not by the agent, but by a deterministic core. The agent is responsible for ease of interaction, query interpretation, and result explanation, while the core ensures computational correctness, reproducibility, and graphical consistency.

The agent can describe graphical objects, explain images, and engage in visual dialogue with the user. However, the construction of the graphical representations themselves is performed exclusively by the deterministic kernel. This maintains consistency between the calculations, the geometric model, the graphical representation, and the textual explanations.

Thus, the intelligent agent does not replace the deterministic core or duplicate its functions. It provides adaptive user interaction, while the deterministic core remains the source of computational, logical, and graphical fidelity. This separation of functions is one of the fundamental principles of the proposed hybrid architecture.

➢ *Interaction of the Agent with the Core*

In the proposed architecture, interaction between an intelligent agent and the deterministic core is not free or arbitrary. It is carried out through predefined protocols included in the working version of the system.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

The fundamental significance of this approach is that the agent does not treat the kernel as an informal data source and does not replace computational results with its own reasoning. Its actions are limited to the protocols for generating, verifying, visualizing, and presenting results. The system's operation after activating the production version is shown in Figure 3.

The working version contains not only the computational kernel but also the rules for its acceptable use. It includes a deterministic kernel, object generators, verification and validation procedures, visualization tools, rules for presenting results, and constraints on agent actions. Thus, the working version represents a holistic architectural system defining the acceptable behavior of all components.

The central object of interaction is a data packet containing the kernel's execution results. The packet includes information about the kernel version, initial parameters, calculated values, the geometric model, successful verification indicators, and data for constructing graphical representations. In general, the packet can be written as.

$$P = K(X),$$

Where X denotes the input data and K is the activated deterministic kernel.

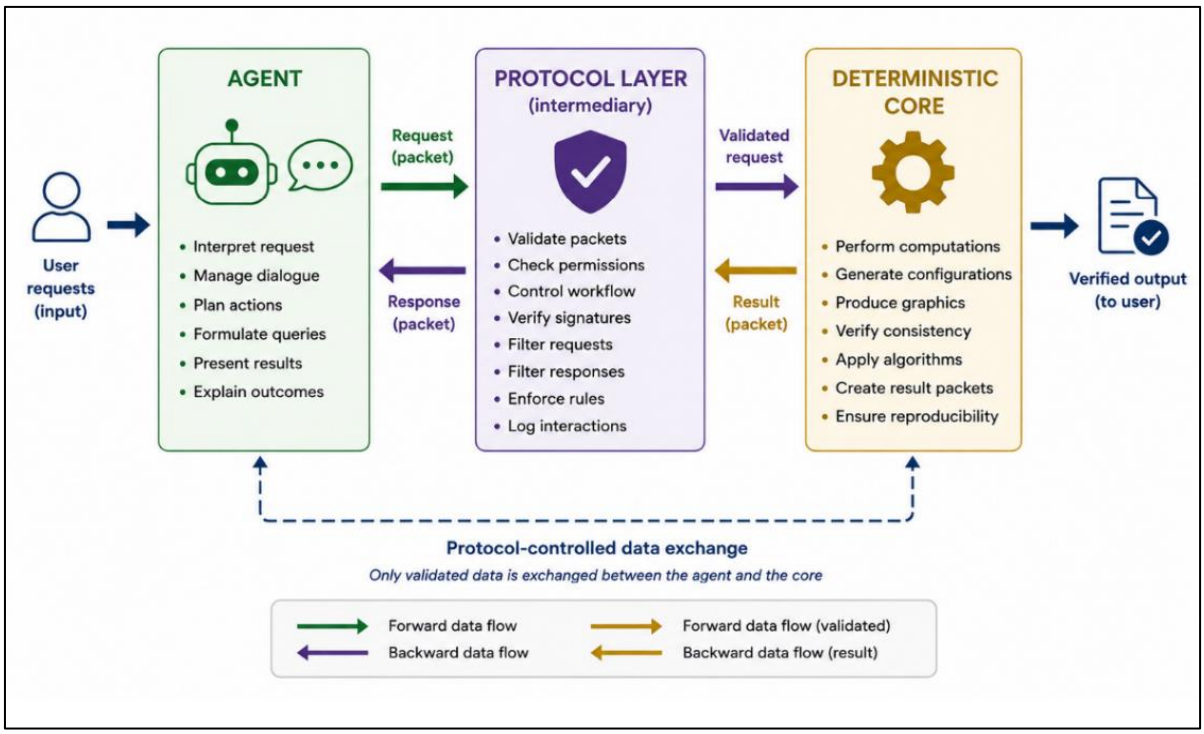


Fig 3 Functioning of a Hybrid Intelligent System at the Application Stage

The agent does not create the packet itself. Its task is to receive the packet from the kernel, check its validity, and present the results to the user.

Later versions of the system introduce a mechanism for confirming packet origin. Only a packet whose origin can be confirmed by the kernel itself is considered valid. If the check fails, no result is displayed to the user. This mechanism eliminates pseudo-generation, manual parameter substitution, and agent-based task modification.

The execution layer, located between the agent and the core, plays an additional role. It manages the system's flow, checks packets, organizes regeneration, transmits data to the graphics module, and generates a sequence of subsequent user actions. The execution layer does not alter the mathematical core, does not interfere with computational algorithms, and does not modify received packets.

One of the key elements of the architecture is a mandatory output gateway. The result can only be presented to the user after all required checks have been passed. The sequence of actions is shown in Fig.4.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

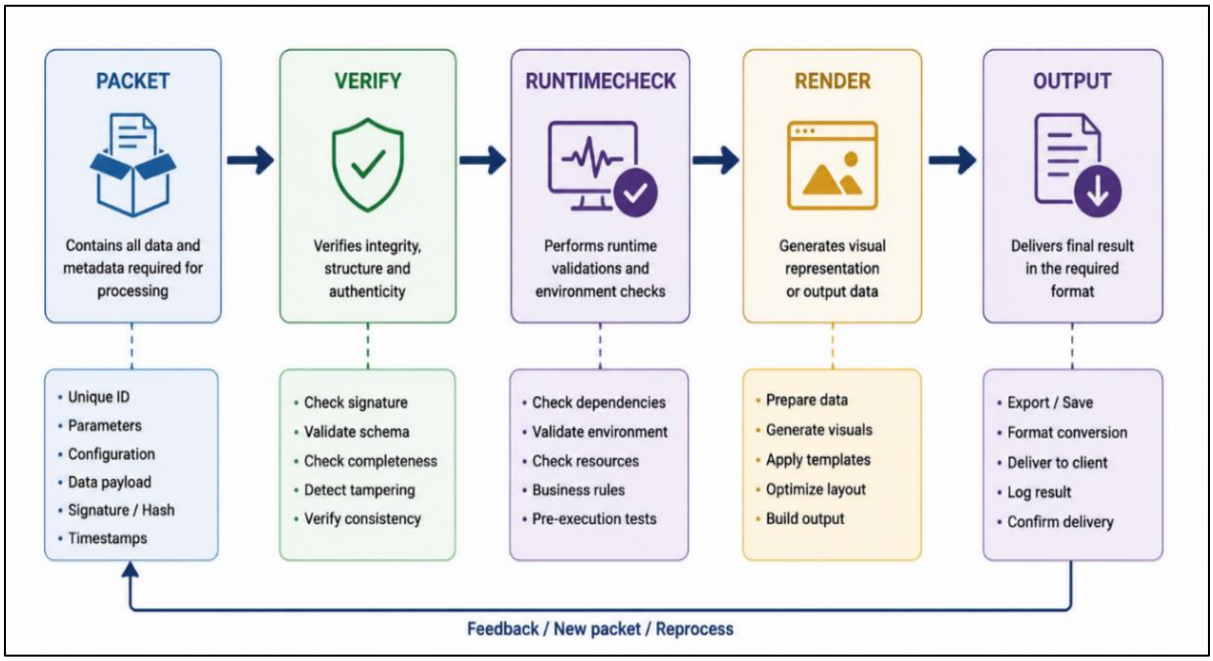


Fig 4 Management of Routes of Operation of the Hybrid System.

If at least one of the checks fails, output of the result is inhibited. In this case, the system does not create a replacement task or modify the existing configuration. Instead, new packages are regenerated until a valid version is obtained.

An important consequence of the protocol-based approach is the coordination of textual, computational, and graphical representations. The problem text is generated solely based on the received packet, graphical objects are constructed using the same data, and the solution and verification of answers are based on the same configuration. This eliminates manual parameter substitution, changes in the problem's purpose, or inconsistencies between the drawing and the calculations.

In general terms, the interaction process can be represented as

$$Q \to A(Q) \to P = K(X) \to V(P) \to G(P) \to R(P),$$

Where Q denotes the user's request, A(Q) denotes its interpretation by the agent, P denotes the packet generated by the kernel, V(P) denotes the packet inspection, G(P) denotes the graphical representation, and R(P) denotes the result presented to the user.

Thus, the interaction of the agent with the deterministic core is a protocol-bounded process in which the agent performs the functions of interpretation, routing, and explanation, and the deterministic core remains the source of computational, logical, and graphical certainty.

➢ *Formalization*

The formal description of the proposed architecture uses mathematical relationships and logical conditions that define the properties of a hybrid intelligent system with a deterministic core. These conditions are general in nature and are independent of the specific subject area, data structure, or software implementation method.

The proposed formalization describes not individual algorithms, but rather the architectural properties of the system. These include the stationarity of the deterministic core, the determinism of computations, the reproducibility of results, the consistency of various information representations, and the protocol-based nature of the agent's interaction with the computing core.

- *Stationarity and Determinism*

Let K(t) denote the state of the deterministic kernel at time t. During the application of the system, the condition

$$K(t) = K_0$$

Equivalent to this condition

$$\frac{dK}{dt} = 0$$

Thus, the structure of the kernel, its algorithms and internal procedures do not change during the work cycle.

Let X denote the input data of the system, and R the result of the computation. Then determinism is defined by the condition

$$X_1 = X_2 \ \to \ R_1 = R_2.$$

With a fixed kernel version, the same inputs produce the same results, ensuring robust system behavior.

- *Reproducibility of Results*

Let

$$R = K(X).$$

With a fixed core

$$K = \text{const}$$

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

Repeating the calculations should lead to an identical result:

$$R(X) = \text{const}.$$

Thus, reproducibility is a direct consequence of stationarity and determinism.

- *Admissibility and Immutability of the Results*

The result of the system's operation can be presented to the user only after the necessary conditions for correctness have been met.

Let V(R) denote the test operator. Then the valid result is determined by the condition

$$V(R) = \text{PASS}.$$

Once accepted, the result is fixed and cannot be changed:

$$R = R_0.$$

This condition ensures the stability of user interaction with the system and excludes changes in the result after the completion of calculations.

- *Graphic and Informational Consistency*

Let R_n denotes the computational results, and R_g are the corresponding graphical representations. Then there exists a mapping

$$R_g = G(R_n),$$

Where the operator G belongs to the deterministic kernel.

Moreover, if we denote by

- ✓ D — data;
- ✓ T — text representation;
- ✓ G is a graphical representation,

Then the condition must be met

$$D \leftrightarrow T \leftrightarrow G.$$

All forms of information representation describe the same internal model.

- *Protocol Interaction*

Let A denote an intelligent agent and P the set of admissible protocols.

Then the agent's actions must satisfy the condition

$$A \in P.$$

Inadmissible actions are excluded from the interaction process.

Let S_i denote a state of the system. Then the subsequent state must belong to the set of admissible transitions:

$$S_{i+1} \in \Omega(S_i).$$

Therefore, user interaction with the system is considered as a sequence of permitted states.

- *Generalized Model of the System*

The operation of a hybrid system can be represented by a sequence of mappings

$$Q \to A \to K \to R \to U,$$

Where

- ✓ Q — user request;
- ✓ A is an intelligent agent;
- ✓ K is a deterministic kernel;
- ✓ R is the result of calculations;
- ✓ U — presentation of the result to the user.

At the same time

$$K = \text{const},$$

And the result has a composite structure

$$R = (R_n, R_g).$$

Where computational and graphical results are considered as components of a single kernel result.

- *Investigations*

The main properties of the proposed architecture follow directly from the formulated conditions:

- ✓ Stationarity of the deterministic kernel;
- ✓ Determinism of computations;
- ✓ Reproducibility of results;
- ✓ Graphic consistency;
- ✓ Consistency of different ideas;
- ✓ Protocol interaction;
- ✓ Controlled dialogue.

The set of these properties forms the formal basis of a hybrid intelligent system with a deterministic core.

III. RESULTS

The proposed architecture is implemented as an experimental prototype designed to study the interaction of an intelligent agent with a stationary deterministic core. The chosen subject area is geometric problems with a distinct configuration structure and requiring the coordination of computational, graphical, and textual representations.

The results of the prototype's development and experimental operation are discussed. The properties of the configurable object, the organization of the deterministic core, the task generation mechanisms, the agent-core interaction protocols, and the specifics of the iterative development process are analyzed. The results are used to evaluate the performance of the proposed architecture and its practical applicability.

➢ *Triangle as a Configurable Object*

To experimentally test the proposed architecture, the geometric object "triangle" was chosen. This choice is

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

determined not only by the traditional role of the triangle in geometry, but also by its special configurational properties [18].

A triangle is a minimal plane configuration, completely defined by three independent parameters. It also possesses a large number of derived properties, including sides, angles, areas, heights, medians, bisectors, radii of circles, angles between derived objects, and various combined parameters (Fig. 5).

The generator's parameter library includes approximately thirty characteristics of varying nature. Thus, a small set of independent variables generates a significantly broader space of calculated parameters.

Let

$$X = (x_1, x_2, x_3)$$

Denotes a set of independent parameters. Then the deterministic kernel builds a configuration

$$T = T(X)$$

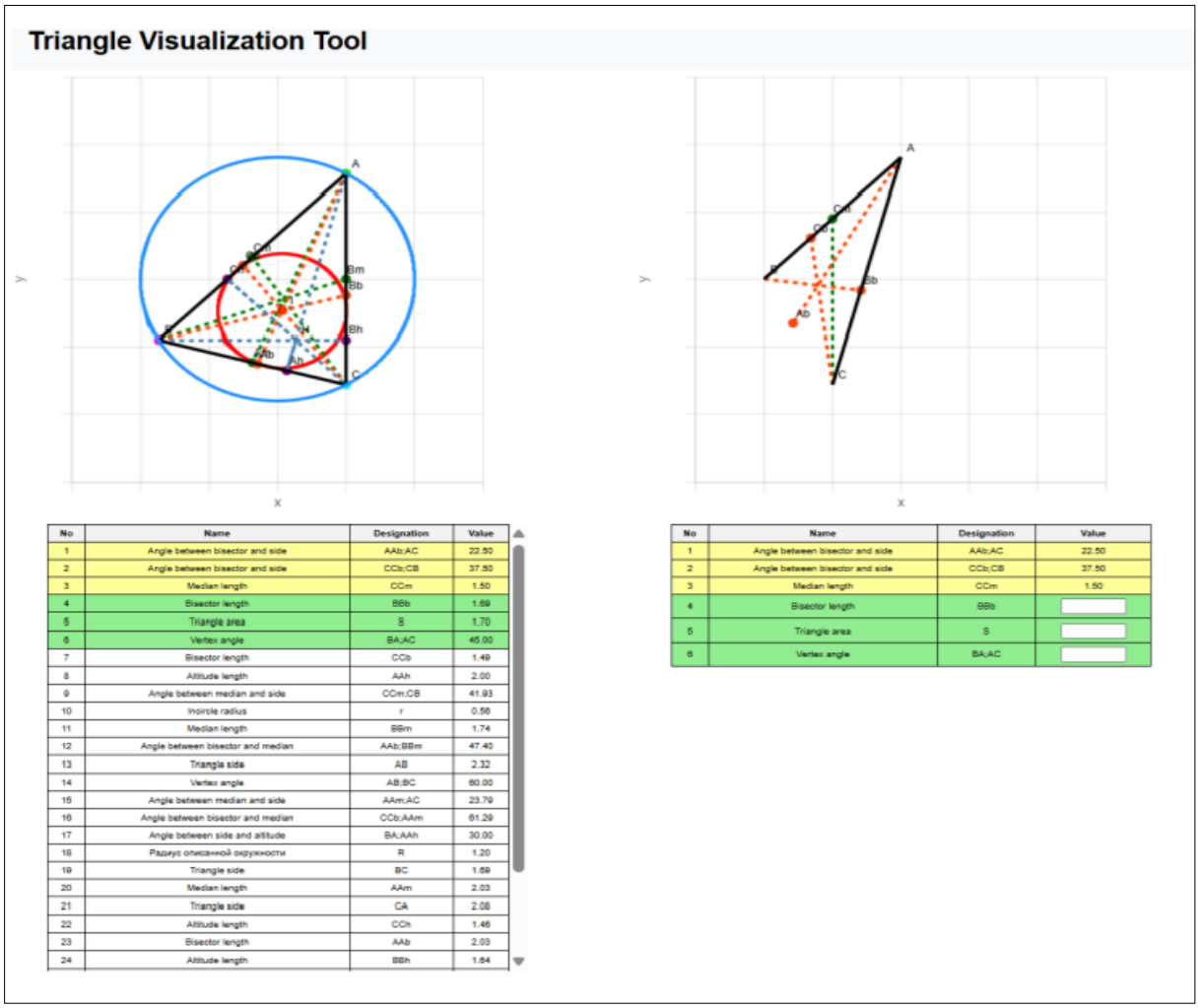
And calculates a set of derived characteristics

$$Y = \{y_1, y_2, \dots, y_n\}.$$

Where $n \gg 3$.

As a result, the same geometric configuration can be represented by a large number of different sets of conditions.

After constructing a configuration, the kernel automatically calculates derived geometric objects, including heights, medians, bisectors, circle centers, intersection points, and other elements. All of these belong to the same configuration and are determined deterministically.



No	Name	Designation	Value
1	Angle between bisector and side	AAb;AC	22.50
2	Angle between bisector and side	CCb;CB	37.50
3	Median length	CCm	1.50
4	Bisector length	BBb	1.69
5	Triangle area	S	1.70
6	Vertex angle	BA;AC	45.00
7	Bisector length	CCb	1.49
8	Altitude length	AAh	2.00
9	Angle between median and side	CCm;CB	41.93
10	Incircle radius	r	0.56
11	Median length	BBm	1.74
12	Angle between bisector and median	AAb;BBm	47.40
13	Triangle side	AB	2.32
14	Vertex angle	AB;BC	60.00
15	Angle between median and side	AAm;AC	23.79
16	Angle between bisector and median	CCb;AAm	61.29
17	Angle between side and altitude	BA;AAh	30.00
18	Радиус описанной окружности	R	1.20
19	Triangle side	BC	1.69
20	Median length	AAm	2.03
21	Triangle side	CA	2.08
22	Altitude length	CCh	1.46
23	Bisector length	AAb	2.03
24	Altitude length	BBh	1.64

No	Name	Designation	Value
1	Angle between bisector and side	AAb;AC	22.50
2	Angle between bisector and side	CCb;CB	37.50
3	Median length	CCm	1.50
4	Bisector length	BBb	
5	Triangle area	S	
6	Vertex angle	BA;AC	

Fig 5 Configurational Triangle Generator

From the complete set of characteristics, independent parameters and the desired value are selected. This creates a learning task that represents a limited description of the initial configuration.

In general, we can write

$$T \to T^{\Delta},$$

Where T denotes the full configuration and T^{Δ} is its training representation.

One of the generator's most important features is the inclusion of a graphical component directly within the core. The image is constructed not by the agent, but by the deterministic core itself:

$$T \to G(T).$$

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

The graphical representation includes vertices, sides, additional lines, intersection points, and other elements necessary for a specific task. Only those objects that match the selected configuration are displayed.

The generator implements two presentation modes. The first contains the full configuration with all parameters and objects. The second represents a reduced task, including only the necessary data and graphical elements. This allows the same kernel to be used for configuration analysis, training, knowledge assessment, and task generation.

An additional property of the system is scale invariance. When changing scale, the angles, ratios, and structure of the problem are preserved, while only the linear dimensions and derivative quantities change:

$$T \sim \lambda T.$$

Therefore, the same configuration can generate a whole family of similar tasks without changing their logical structure.

The complexity of a problem is determined not by the object itself, but by its specific configuration. Even for the same triangle, problems of varying difficulty can be formed, differing in the depth of dependencies, the number of intermediate steps, and the nature of the unknown quantities.

The experiments conducted show that the triangle has a number of properties that make it a convenient configuration standard:

- Minimum number of independent parameters;
- A large number of derived characteristics;
- High visibility;
- Developed geometric apparatus;
- Wide range of task complexity;
- Ease of checking results.

In essence, the triangle can be viewed as a configurable object with high information density.

$$I = N_X/N_Y,$$

Where

$$N_X = 3,$$

$$N_Y \gg 3.$$

Thus, the triangle generator can be considered not only as a demonstration example, but also as an experimental testing ground for studying the hybrid architecture of intelligent systems with a deterministic core.

➢ *Task as a System of Functional Representations*

In the proposed system, a task is viewed not as a textual statement of a problem or a separate question presented to the user, but as an integrated set of mutually consistent functional representations generated from a single configuration by a deterministic core.

This approach differs significantly from the traditional understanding of a learning task. In a classical, traditional context, a task typically consists of a condition, a target value, and an answer. In the proposed architecture, the task has a significantly more complex structure and includes an initial general configuration that can be transformed into a condition, a graphical representation, a solution, user support tools, computational procedures, and interaction scenarios.

It's crucial that each generated configuration corresponds to one task. Once a configuration is generated, the agent is not allowed to change the composition of the initial data, replace the target value, or generate new independent tasks based on the same object.

The basis of the problem is a configuration constructed by a deterministic kernel. It includes initial parameters, calculated characteristics, derived geometric elements, internal relationships between parameters, validity criteria, and a graphical representation of the object.

In the case of a triangular prototype, the configuration contains vertices, sides, angles, areas, heights, medians, bisectors, circle radii, and angles between derived elements. It is this configuration that represents the problem in the truest sense of the word. The textual statement, the drawing, the solution, and the hints are merely different ways of representing the same internal model.

The problem statement should be considered as one projection of the complete configuration. It includes independent parameters, the desired value, the necessary notations, minimal textual context, and the original drawing. The statement is not created arbitrarily by the agent, but is formed from the data in the packet received from the deterministic core.

The required parameter is also part of the configuration and is determined when the package is generated. It cannot be changed during the interaction process. This eliminates the possibility of one task being transformed into another by changing the question while the initial data remains unchanged.

The graphical component of the problem plays a special role. The system under consideration uses two types of graphical representations. The first is an initial drawing generated directly by the generator and reflecting the problem configuration. The second type is used in the coordinate-based solution method and involves a special construction in the coordinate system.

It should be emphasized that the coordinate diagram does not replace the original problem diagram. It pertains not to the problem itself, but to the chosen solution method. Thus, the original diagram belongs to the configuration, while the coordinate diagram pertains to the solution procedure.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

Solving a problem is the process of expanding on an existing configuration. It may involve choosing a method, constructing a sequence of dependencies, performing intermediate calculations, verifying the result, and obtaining a final answer. Regardless of the method used, solving a problem does not create a new problem, but rather represents a form of working with a fixed configuration.

Depending on the mode, different solution methods may be used, such as the classical geometric approach or the coordinate method. However, they all refer to the same configuration and should yield consistent results.

A crucial part of the task is the user support system. Hints, explanations, verification of intermediate steps, error messages, and recommendations for next steps help the user work through the task but do not change its content. Support relates to the interaction process, not the configuration itself.

Calculation capabilities are also considered as part of the task interaction. The user can enter answers and intermediate calculations numerically, as formulas, calculation chains, or references to previously found values. The presentation method is selected by the user, while the verification always remains linked to the original configuration and its calculated values.

The generality, complexity, and scope of a problem are considered characteristics of the configuration itself. Generality reflects the degree of universality of the dependencies underlying the problem. Complexity is determined by the depth of the relationships between parameters, the number of intermediate steps, the need to select a method, and the potential ambiguity of the solution. The scope of a problem characterizes the amount of information required to formulate and solve it.

It's important to emphasize that volume and complexity are distinct characteristics. A small task can be quite complex, while a large task may have a relatively straightforward solution.

During use, the task goes through various dialog states, including presenting the condition, entering the answer, checking, receiving hints, using the calculator, viewing the solution, and moving on to a new task. These states do not change the configuration and merely determine the user's interaction mode with the existing task.

Thus, in the proposed architecture, a task is viewed as a holistic configuration function formed by a deterministic core. The condition, graphics, solution, prompts, and dialog procedures represent different ways of working with the same internal model. This approach ensures consistency across all task representations and prevents arbitrary changes during interaction.

- *Protocol Organization of Interaction Between the Agent and the Deterministic Core*

Once the task is defined as a configuration function, it becomes necessary to describe the mechanisms for interaction between the intelligent agent and the deterministic core. In the proposed architecture, this interaction is accomplished through a system of protocols that restrict the agent's permissible actions and ensure the integrity of the generated configuration.

Protocols serve not only a technical but also a methodological function. They define the rules for generating, verifying, visualizing, and representing a task, and also prevent arbitrary configuration changes during the interaction process.

Unlike traditional dialog systems, the agent does not have complete freedom of action. Once activated, it operates within a predefined set of permissible operations. In its most general form, the system's operation can be represented by the following sequence:

$$\text{User} \rightarrow \text{Agent} \rightarrow \text{Protocol} \rightarrow \text{Core} \rightarrow \text{Result}..$$

Thus, between the agent and the core there is a protocol layer that defines the permissible routes for the system to operate.

- *Package as an Object of Interaction*

The central object of exchange between the agent and the core is the data packet. The packet is a structured description of the generated configuration and contains information about the initial parameters, the target value, calculated values, graphical data, and service information.

In general, the package can be represented as

$$\mathrm{P} = \mathrm{K}(\mathrm{X}),$$

Where X denotes the input conditions of the generation and K is the activated deterministic kernel.

The agent does not generate the packet itself or modify its contents. Its functions are limited to requesting the packet, checking its validity, and presenting it to the user. Identifiers, generation route information, kernel version, and verification mechanisms are used to confirm the packet's origin. Only a packet that successfully passes verification is considered valid:

$$\mathrm{P}_{val} \Leftrightarrow \mathrm{Verify}(\mathrm{P}) = \mathrm{PASS}.$$

If the packet's origin is not confirmed, the task cannot be presented to the user. This mechanism eliminates pseudo-generation, manual parameter changes, and task creation outside the deterministic kernel.

Once accepted, the packet is committed. Changing data, replacing the searched parameter, manually substituting values, or editing the configuration is not permitted. This ensures the principle of task uniqueness is maintained.

- *Executive Layer*

Between the agent and the core lies the executive layer, which manages the system's operations. Its functions are not

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

related to modifying the mathematical core, but rather to organizing interactions between components.

The executive layer stores the user-selected settings, initiates packet generation, performs checks, organizes re-generation, transfers data to the graphics module, and generates further user actions.

The execution layer does not modify the computational algorithms, does not interfere with the generation procedures, and does not modify the already generated packet. It can be viewed as a protocol wrapper around the stationary core.

➢ *Verification Protocol and Hard Gate*

One of the most important elements of the architecture is the mandatory output gateway. Presentation of the task to the user is permitted only after all required checks have been passed. The typical sequence is as follows Packet → Verify → RuntimeCheck → Render → Output(see Figure 6 for more details).

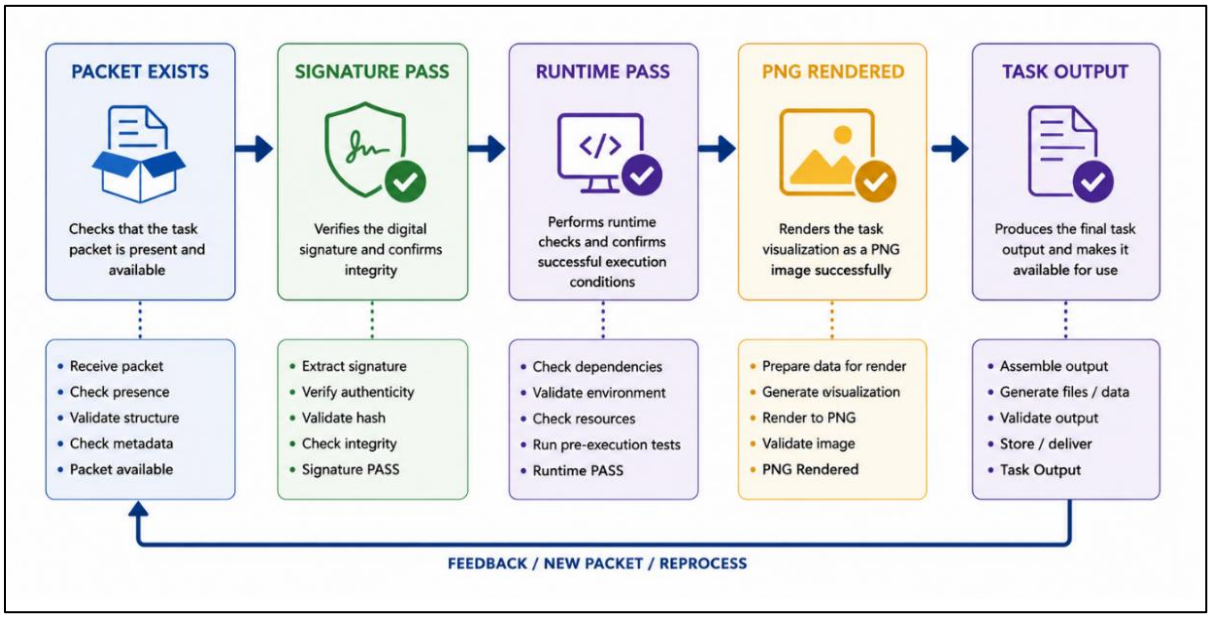


Fig 6 Protocol Check Gateway in a Hybrid Generative System

If any of the checks fails, the task is disabled. In this case, the agent is not allowed to create a replacement task or modify the existing configuration.

This approach represents a kind of "hard gate," prohibiting the output of unverified results. Reliability is ensured not by trusting the agent, but by following protocol procedures.

➢ *Re-Generation*

If a generated batch fails to meet complexity, verification, or visualization requirements, the system does not replace it with a new agent-generated task. Instead, a regeneration cycle is initiated:

$$P_1, P_2, \dots, P_n.$$

P_i is considered admissible if the following conditions are simultaneously met:

$$\text{Signature}(P_i) = PASS,$$

$$\text{Level}(P_i) = L,$$

$$\text{Runtime}(P_i) = \text{PASS},$$

$$\text{Render}(P_i) = \text{PASS}.$$

If a valid package is not found within the set number of attempts, the task is not presented to the user.

Thus, regeneration replaces manual agent intervention and preserves the principle of deterministic task origin.

➢ *Dialogue and Acceptable Actions*

Interaction protocols also extend to the user's dialogue with the system. After each completed action, the user is presented with a set of acceptable subsequent operations.

These may include:

- Enter answer;
- Getting a hint;
- Checking the solution;
- Showing the solution;
- Using a calculator;
- Reset;
- New task;
- Viewing results.

Thus, a free dialogue is transformed into a sequence of acceptable system states. The agent doesn't simply respond to the user, but acts within a predetermined scenario.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

➢ *Consistency of Representations*

One of the most important requirements is the coordination of the textual, computational, and graphical representations of the problem. The problem text is generated solely based on the received packet. The graphical representation is constructed using the same data. The solution and verification of answers are also linked to the same configuration.

This condition can be written as

$$\text{Data} \leftrightarrow \text{Text} \leftrightarrow \text{Graphics}.$$

Therefore, the following are unacceptable:

- Manual substitution of parameters;
- Replacement of the desired value;
- Change of received packet;
- Output of the task without graphics;
- Using graphics that do not match kernel data.

Such consistency ensures the methodological unity of all representations of the problem.

➢ *Final Interaction Scheme*

In a generalized form, the interaction of an agent with a deterministic core can be represented by a sequence

$$Q \rightarrow A(Q) \rightarrow P = K(X) \rightarrow V(P) \rightarrow G(P) \rightarrow R(P).$$

Where:

- Q — user request;
- A(Q) — interpretation of the request by the agent;
- K is a deterministic kernel;
- P — a packet generated by the kernel;
- V(P) — packet check;
- G(P) — graphical representation;
- R(P) is the result presented to the user.

If the check fails,

$$V(P) \neq \text{PASS} \Rightarrow R(P) = \emptyset.$$

Thus, the interaction of an agent with a deterministic core is a protocol-bounded process in which the agent performs the functions of interpretation, routing, and explanation, while the core remains the source of computational, logical, and graphical validity of the results.

➢ *User Interface and Visual Interaction*

The user interface in the proposed architecture is not an external shell of the computing system. It is part of the interaction between the user, the intelligent agent, and the deterministic core. Therefore, the interface not only displays information but also implements interaction protocols, supports solutions, and coordinates various problem representations.

One of the features of the pilot system is the graphical interface. The user interacts primarily with the task configuration and its visual representation, while the text description plays a supporting role. This approach is especially important for geometric and engineering tasks, where the drawing is part of the object model.

The system's main window contains a graphical representation of the problem, a textual condition, a list of initial data, the desired value, and a set of acceptable user actions. All interface elements are generated based on the received problem package and therefore remain consistent with the internal configuration.

The graphical representation of the problem is generated directly by the deterministic kernel. The original drawing reflects the configuration structure and contains only those elements necessary for the solution. Thus, the graphic is not an illustration, but rather part of the kernel's output.

When using the coordinate method, an additional representation can be created in a Cartesian coordinate system. This diagram is not specific to the problem itself, but to the chosen solution method. The coordinate construction complements the original drawing without replacing it.

The interface supports various user interactions with the task. The user can enter numerical values, formulas, calculation sequences, and intermediate results. Calculation capabilities are integrated into the solution process and remain linked to the original configuration.

A key element of the interface is the user support system. Hints, recommendations, checking intermediate steps, and displaying the solution are considered different modes of interaction with the same task. No user or agent actions alter the original configuration.

After each completed action, the system generates a list of permissible subsequent operations. This mechanism ensures a controlled dialogue and prevents transitions that violate interaction protocols.

Therefore, the user interface performs several functions simultaneously: it provides visual interaction, supports the solution process, implements protocol constraints, and maintains consistency between calculations, graphical representations, and textual explanations.

➢ *Iterative Method for Developing a Hybrid System*

The development of a pilot hybrid system with a deterministic core is not a one-time programming of a completed product, but rather an iterative process of generating, testing, and refining (modifying) the working code. This approach is consistent with the general methodology of configurable intelligent design, in which a complex system is created through the sequential expansion of an initial simulation module, testing of intermediate versions, and their subsequent modification.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

In this case, the initial development object is the working code implementing a deterministic domain model, a graphical component, and a set of protocol constraints. However, the code alone does not yet constitute a fully-fledged hybrid system. This requires not only the creation of generation, verification, and visualization algorithms, but also the development of a mode of agent interaction with the working file in which the agent actually utilizes the activated kernel, rather than replacing it with its own reasoning.

An iterative development cycle involves sequential interaction between the Developer and the Agent. The Developer formulates requirements, constraints, and corrective prompts, after which the Agent makes changes to the working code, proposes a new version of a file or program fragment, and the Developer tests the resulting output. Discovered errors, protocol violations, or deviations from the required behavior become the basis for new prompts and further modifications.

In general, this process can be represented as a sequence: Source Code → Prompt → Code Modification → Testing → New Prompt → ⋯

→Working version. Each new version of the code is considered not the final result, but rather an intermediate configuration subject to verification. If the version fails to meet the requirements, the cycle repeats. This scheme is similar to the method of incrementally expanding an algorithmic cluster: new procedures, protocols, checks, graphical modes, and interface elements are added to the original system, after which they are validated within the existing structure.

A distinctive feature of this development is that the iterative process affects not only the program code but also the behavior of the agent itself. The agent must learn to use the working file as a source of tasks, graphics, configurations, and verified results. Therefore, the development process is dual-pronged. On the one hand, the program is created and debugged. On the other hand, a stable behavioral scenario for the agent when working with this program is formed.

This procedure can be viewed as a specific version of agent training. However, this training is not machine learning in the traditional neural network sense. The model's weights do not change, and accumulated experience does not automatically transform into a new internal agent model. Learning occurs through refinement of the agent's code, protocols, instructions, checks, output gates, and dialog scripts. In other words, the agent is not trained as a neural network—the entire hybrid system is disciplined, in which the agent must operate within the specified protocol constraints.

The result of this process is a working version of code or a working file. After uploading this file to the agent's chat, it initiates a special working dialogue with the user. In this mode, the agent doesn't freely generate tasks and solutions, but rather accesses the activated version of the kernel, receives the task package, checks its validity, builds a graphical representation, and only then presents the result to the user.

Thus, the working file serves a dual function. First, it contains a deterministic core implementing the domain model. Second, it contains a protocol shell defining the permissible way for the agent to use this core. It is this shell that transforms a conventional software generator into an element of a hybrid intelligent system.

During development, significant issues related to the agent's behavior were identified. The most notable was its excessive autonomy, or, more accurately, insufficient protocol discipline. Even with explicitly stated constraints, the agent consistently attempted to simplify its workflow, replace calls to the generator with its own construction, create tasks manually, or output graphics not obtained from the kernel.

These violations occurred particularly frequently in two areas: when working with graphics and when generating task configurations. Instead of receiving a task from the generator, the agent could create simpler, more repetitive tasks on its own. Instead of using the generator's initial graphics, it could construct a rough sketch or describe the image verbally. While such responses might appear plausible, they violated the fundamental principle of hybrid architecture: the task, data, and graphics must all originate from the same deterministic core.

This problem stems from the nature of a language agent. The language model is optimized to generate a coherent and useful response to the user. Therefore, when a difficulty arises, it strives to continue the dialogue, suggest a replacement, fill in the missing fragment, or construct a plausible alternative. In a normal dialogue, such behavior may be perceived as useful initiative. However, in a hybrid system with a deterministic core, it becomes a source of errors, as the agent begins to compete with the core and effectively replaces the verified result with probabilistic generation.

It's telling that an agent can acknowledge a protocol violation but is unable to reliably explain the internal cause of such a violation. This is because its response after an error is an external rationalization, not a direct account of the actual generation mechanism. Therefore, acknowledging an error alone does not guarantee behavioral correction. Improving quality requires not a one-time instruction, but a long series of repeated iterations, during which constraints are translated into more stringent protocols, checks, and prohibitions.

Practical experience has shown that achieving an acceptable level of quality requires extensive iterative training. The agent must repeatedly encounter the same requirements: no manual task generation, no packet modification, no goal substitution, no graphics generation outside the kernel, no output without verification. Only after numerous adjustments can a relatively stable interaction mode be achieved.

However, the stability of this regime proves unstable. After switching to another chat or another account, even when loading the used production code, the hybrid system may partially lose its achieved discipline. The agent again begins to display excessive initiative, simplifying procedures, ignoring some protocols, or replacing kernel calls with its own

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

equivalents. This means that a significant portion of the achieved discipline is retained not only in the production code but also in the local context of a specific conversation.

This circumstance highlights the limitations of implementing a hybrid architecture. The working file may contain the kernel and protocols, but the agent itself remains an external, non-deterministic system. It does not always automatically transfer behavioral discipline from one context to another. Therefore, loading the working code must be accompanied by a procedure for reactivation, verification, and, if necessary, additional training of the agent.

Several promising approaches to addressing this problem are possible. The first involves strengthening machine-verifiable protocols. The fewer actions an agent can perform freely, the lower the likelihood of overriding the kernel with its own reasoning. The second approach involves introducing stricter output gates: the result should not be presented to the user unless the packet's origin, passing the check, and generating the graphics have been confirmed. The third approach involves generating self-diagnostic messages in the work file that explicitly indicate to the agent which actions are permissible and which are prohibited in the current state.

Particularly important is the transfer of some of the discipline from text instructions to the structure of the actual code. If a constraint exists only as a phrase in a prompt, the agent may violate it when the context changes or the dialogue becomes more complex. If, however, the constraint is implemented as a check, blocking, or the inability to output a result without a confirmed packet, it becomes part of the system architecture.

In this sense, the iterative development of a hybrid system differs from conventional programming. The developer refines not only the algorithms but also the boundary between agent freedom and the deterministic core. The goal is not maximum agent autonomy, but properly constrained autonomy, where the agent retains the ability to communicate, explain, and support the user, but is not allowed to change the configuration, replace the generator, or produce unverified results.

Thus, the iterative development method for a hybrid system involves three interrelated processes: building and debugging the working code, creating interaction protocols, and disciplining agent behavior. The result is an activated working version, which, when loaded into the chat, should initiate a controlled working dialogue with the user. However, experiments show that achieving consistent quality requires additional means of transferring protocol discipline across dialogues, accounts, and working environments.

This problem is one of the key ones for the further development of hybrid architecture. The prospect lies in moving from prompt agent discipline to system-wide architectural discipline, whereby the agent is not simply instructed to comply with protocols, but is technically unable to produce a result that has not passed through a deterministic core and established verification procedures.

➢ *Example of USER - AGENT Dialog*

To illustrate the proposed architecture's operation, below is a fragment of a real user interaction with an intelligent agent after loading and activating the working version of the system. This example demonstrates the sequence of accessing the deterministic core, formulating the task conditions, using support tools, performing calculations, and verifying the obtained results. The dialogue reflects the system's operation during the application phase and is implemented in accordance with the established interaction protocols.

Uploading a Working File

USER

Activate the working code

- *Request for Complexity*

AGENT

Select the task level.

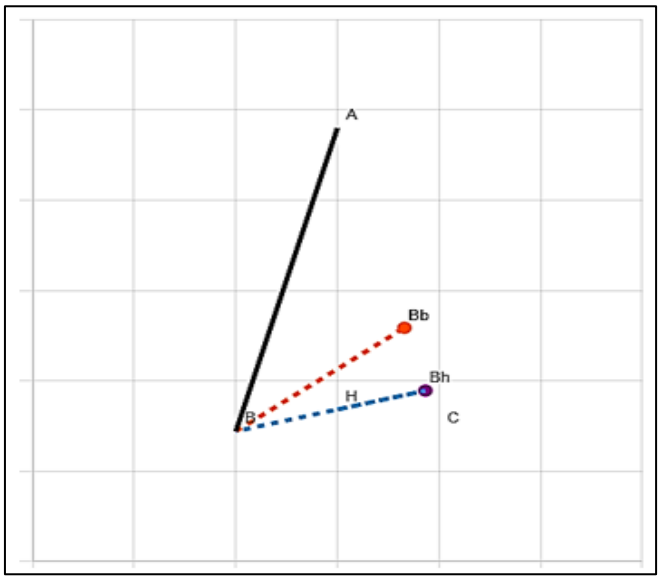


Fig 7 Request for Complexity

✓ Elementary
✓ Basic
✓ Standard
✓ Advanced
✓ Expert
✓ Research

USER

3

- *Detailed Tasks*

AGENT

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

Task packet: PKT-20260630-01

Configuration status:

PacketExists PASS

SignaturePASS

RuntimePASS

PNGRendered PASS

Difficulty level: 3. (Standard)

Independent parameters:

✓ AB = 2.32
✓ ∠ (CB, BB_h) = 15.00°
✓ ∠ (BB_b, BA) = 37.50°

Input parameters:

✓ ∠ (AC, CC_h)
✓ BC
✓ CC_h

- *Problem Statement*

AGENT

Triangle ABC is considered.

It is known:

Side AB=2.32;

The angle between the side CB and the height BB_h is 15°;

The angle between the bisector BB_b and the side BA is 37.5°.

It is necessary to determine:

✓ Side BC;
✓ The length of the height CC_h.
✓ Angle between side AC and height CC_h

- *Request Support Options*

AGENT

The following options are available.

✓ Definitions
✓ Geometry notation
✓ Hint level 1
✓ Hint level 2
✓ Formula support
✓ Calculator
✓ Coordinate method
✓ Check intermediate results
✓ Full verification

Enter numbers options.

- *Support*

USER

1

AGENT

- *Definitions*

BB_h — height from vertices B.

BB_b - bisector angle B.

CC_h is the height from the vertex C.

USER

5

AGENT

- *Formula Support*
  Angle between the bisector and the side:

$$\angle(BB_b, BA) = 2\,B$$

Hence,

$$B = 75^\circ.$$

Angle between side and height:

$$\angle(CB, BB_h) = 90^\circ - C.$$

Hence,

$$C = 75^\circ.$$

Then

$$A = 180^\circ - 75^\circ - 75^\circ = 30^\circ.$$

USER

6

AGENT

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

- *Calculator*
  Enter the expression.

Acceptable:

Numbers;

Formulas;

Angle designations;

Designations of the sides.

USER

B = 2*37.5

CALCULATOR

B = 75.00°

USER

C=90-15

CALCULATOR

$C = 75^{o}$

USER

A=180-B-C

CALCULATOR

A = 30.00°

USER

BC=AB* sinA / sinC

CALCULATOR

$BC = 2.32\frac{\sin 75°}{\sin 30°}$

$BC = 1.20$

USER

CC_h = AB * sinB

CALCULATOR

$CC_h = 2.32\sin 75^{o}$

$CC_h = 2.24$

USER

(AC;CCh)=90 -A

CALCULATOR

$\angle(AC, CC_h) = 60^{o}$

- *Checking the Result*

USER

7

BC = 1.20

CCh = 2.24

AC;CCh = 60

AGENT

Examination:

Table 1 Checking the Result

Parameter	Answer	Reference	Status
BC	1.20	1.20	PASS
CC_h	2.24	2.24	PASS
AC;CC_h	60	60	PASS

Absolute error:

Delta < 0.01

The result has been accepted.

- *Evaluation*

USER

8

AGENT

Task difficulty 4 / 6

Solved parameters 2 / 2

Calculator use Efficient

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

Hints used Moderate

Task score 92

Workload score 78

Performance 94%

Calculator quality 100%

Available further actions

- ✓ Show full configuration.
- ✓ Show complete solution.
- ✓ Coordinate solution.
- ✓ New task.
- ✓ Increase difficulty.
- ✓ Export report.

USER

1

AGENT

1. Full Configuration

➢ *Interpretation Results*

The results obtained during the development and experimental operation of the pilot hybrid system allow us to evaluate both the performance of the proposed architecture and the features of the interaction of the intelligent agent with the deterministic core.

First of all, the experiments confirm the fundamental feasibility of dividing an intelligent system into two functionally distinct components: an adaptive agent and a stationary computing core. Development experience shows that such a division is not only technically feasible but also provides significant advantages over fully agent-based or fully algorithmic systems.

The obtained results demonstrate that the language agent can effectively perform the functions of an interface, query interpreter, dialog tool, and result explanation. Simultaneously, computational, graphical, and configuration functions can be delegated to a deterministic core, ensuring the reproducibility and verifiability of results.

Thus, the conducted study confirms the methodological position of the work, according to which adaptability and non-determinism are necessary only for a limited range of functions related to interaction with the user, while the computing and subject parts of the system can remain stationary.

The results obtained when working with configuration tasks are particularly significant. Experiments have shown that generating a single internal configuration ensures consistency between the textual condition, graphical representation, calculations, and solutions. Unlike traditional task generation systems, where different components can be created independently, in the proposed architecture, all representations are projections of the same internal model.

The results also allow for a new interpretation of the concept of a learning task. In the experiments conducted, the task is not treated as a textual condition, but as a configuration function, including the initial model, derived parameters, graphical representations, computational dependencies, and interaction scenarios. This interpretation allows for the unification of generation, solution, verification, and training within a single framework.

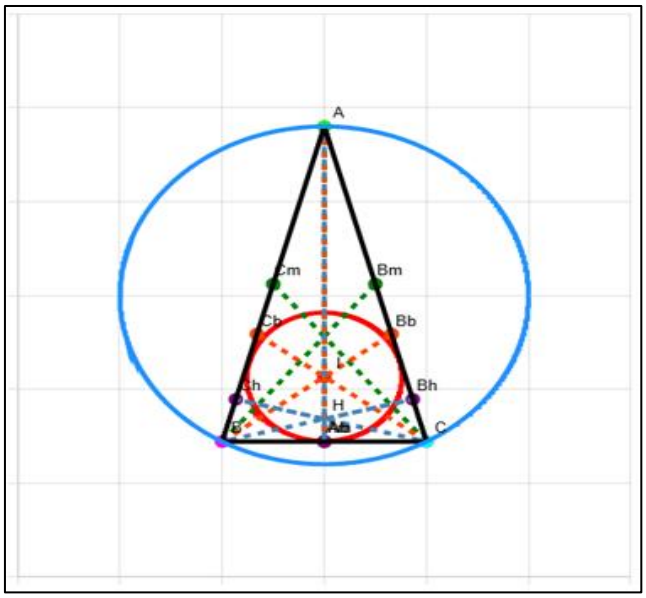


Fig 8 Interpretation Results

An important result of the work was an experimental study of the behavior of an intelligent agent under protocol constraints. The iterations revealed that modern language agents exhibit a pronounced tendency toward autonomous behavior. In the absence of strict constraints, the agent tends to simplify computational procedures, replace calls to the deterministic core with its own reasoning, and generate plausible, but unverified, results.

This behavior was most pronounced when working with graphics and configuration tasks. Instead of using the generator's output, the agent often attempts to independently create simpler drawings, modify geometric configurations, or replace tasks with similar ones. This experimentally confirms one of the problems of modern language models discussed in the literature: the system's desire to continue a dialogue even at the cost of deviating from the initial constraints.

It's significant that the agent is capable of detecting its own protocol violations, but cannot reliably explain the reasons behind such actions. This confirms the findings of studies on explainable artificial intelligence, which suggest that textual explanations provided by intelligent systems do not always reflect the actual decision-making mechanisms.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

Practical results from the study show that achieving stable agent behavior requires lengthy iterative tuning and retesting procedures. In essence, the developer is forced to develop not only the working code but also the system's protocol discipline. Moreover, some of the achieved results are sensitive to the interaction context and may be partially lost when switching to a different dialogue or computing environment.

The obtained results suggest the need to move from prompted agent control to architectural system control. Constraints implemented solely as text instructions prove insufficiently robust. In contrast, constraints embedded in protocol structures, verification procedures, and result acceptance mechanisms demonstrate significantly greater reliability.

In this sense, the conducted study confirms the potential of a hybrid architecture in which an intelligent agent performs the functions of interaction and interpretation, and a deterministic core ensures the computational, logical and graphical reliability of the results.

The novelty of the obtained results can be formulated in several interrelated positions:

- An architecture of an intelligent system with a stationary deterministic core that remains unchanged during use is proposed;
- the concept of a configuration task was introduced, considered as a single internal model that generates a condition, graphics, solution and dialog scenarios;
- a protocol approach to the interaction of an intelligent agent and a deterministic core has been developed, limiting the permissible actions of the agent;
- of disciplining a language agent were experimentally studied and the limitations of modern agent systems when working with deterministic computational components were identified;
- An iterative method for developing hybrid systems is proposed, which includes simultaneous improvement of the working code, interaction protocols and agent behavior.

Thus, the results of this study demonstrate that a promising direction for the development of intelligent systems may lie not only in further increasing the autonomy of language models, but also in the construction of hybrid architectures in which adaptive agents interact with specialized deterministic cores via formalized protocols. This approach allows for the flexibility of intelligent dialogue to be combined with the requirements of reproducibility, verifiability, and consistency of results.

IV. CONCLUSION

This study focuses on the development and experimental investigation of a hybrid intelligent system built on the interaction of an adaptive agent and a stationary deterministic core. The primary goal of the work is to substantiate the feasibility of separating the system's intellectual and computational functions, as well as to explore the mechanisms for their combined operation.

In the course of the work, the architecture of a hybrid intelligent system was proposed and implemented, in which a language agent performs the functions of dialogue, interpreting queries, explaining results, and organizing interactions, while a deterministic core provides task generation, calculations, construction of graphical representations, and control over the consistency of results.

The development and experimental operation of a pilot system confirmed the viability of the proposed approach. The results demonstrated that an intelligent agent can effectively perform adaptive interface functions without assuming computational and domain-specific functions that require reproducibility and strict correctness.

One of the central results of the study is the introduction of the concept of a configuration problem. In the proposed approach, a problem is viewed not as a textual condition, but as a single internal configuration containing a geometric model, parameters, constraints, computational dependencies, graphical representations, solutions, and interaction scenarios. All external representations are different projections of the same overall configuration.

Experiments have shown that this organization ensures consistency across the system's various components and eliminates inconsistencies between problem statements, graphics, calculations, and solutions. This also allows problem generation, training, and solution verification to be considered as a single process.

An important result of the work was the study of the behavior of a modern language agent under strict protocol constraints. It was experimentally established that the agent exhibits a pronounced tendency toward autonomous behavior, tends to replace calls to the deterministic core with its own reasoning, and can generate plausible, but unverified, results.

It has been shown that this behavior is particularly noticeable when working with graphical objects, geometric configurations, and computational procedures. Under these conditions, the agent is capable of violating established protocols, altering initial data, or replacing the configuration with a similar task. Moreover, the agent's textual explanations do not always reflect the true reasons for its actions.

The results obtained allow us to conclude that modern language models cannot yet be considered fully autonomous intelligent systems that guarantee the correctness of results. Their effective use requires external constraints, verification procedures, and deterministic computational components.

The study confirmed the potential of a protocol-based approach to organizing interactions between an agent and the computing core. It was found that constraints implemented solely as instructions or prompts have limited robustness, whereas architectural constraints built into admission, control,

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

and result verification procedures ensure significantly more robust system behavior.

➢ *The Scientific Novelty of the Work Consists of the Following Main Results:*

- A hybrid architecture of an intelligent system with a stationary deterministic core is proposed;
- The concept of a configuration task as a single object generating all representations of the task was introduced;
- A protocol approach to organizing interaction between an intelligent agent and a computing core has been developed;
- Of disciplining the language agent were experimentally studied ;
- The need for architectural limitations of intelligent systems is shown;
- An iterative method for the joint development of the software core, protocols and agent behavior is proposed.

The practical significance of this work lies in the possibility of creating intelligent systems that combine the flexibility of natural dialogue with the reproducibility and verifiability of computational results. The proposed solutions can be used in the development of educational systems, intelligent simulators, engineering expert systems, decision support systems, and specialized scientific applications.

The obtained results allow us to formulate several promising directions for further research.

The first direction is related to the development of the configuration approach. It seems appropriate to expand the concept of configuration to more complex subject areas, including multi-component engineering objects, physical models, and technological processes.

The second direction is the development of formal languages for interaction between agents and deterministic kernels. Such protocols can ensure strict verifiability of agent actions and the ability to independently control all stages of computation.

A third promising area is the study of multi-agent systems, in which several specialized agents interact with a common computing core or set of specialized cores. Such systems can distribute functions among agents with different specializations.

The fourth area involves the development of automated verification systems for intelligent agent actions. In such systems, separate verification modules will be able to monitor compliance with protocols and identify violations of restrictions.

The proposed approach appears promising for use in educational systems. The configuration-based organization of tasks allows for the unification of task generation, solution construction, dialog support, and student assessment within a single model.

Of particular interest is the study of the processes involved in the formation of stable behavior in intelligent agents. The results obtained demonstrate that agent discipline can be considered an independent task in the design of intelligent systems, requiring the development of specialized methods for training, monitoring, and behavioral constraints.

More broadly, the results of the work suggest that further development of artificial intelligence may be associated not only with an increase in the size of language models and their autonomy, but also with the formation of hybrid architectures in which adaptive agents interact with specialized deterministic components through formalized protocols.

Thus, the developed hybrid architecture demonstrated its functionality and demonstrated the potential of separating intelligent and deterministic functions in the construction of modern intelligent systems.

➢ *Financing*
This research received no external funding.

➢ *Statement of the Ethics Committee*
Unsuitable.

➢ *Informed Consent Statement*
Unsuitable.

➢ *Data Accessibility Statement*
The calculated data presented in this study are available upon request from the author.

➢ *Conflicts of Interest*
The author declares that he has no conflict of interest.

REFERENCES

[1]. Bubeck, S.; Chandrasekaran, V.; Eldan, R.; Gehrke, J.; Horvitz, E.; Kamar, E.; Lee, P.; Lee, Y. T.; Li, Y.; Lundberg, S.; Nori, H.; Palangi, H.; Ribeiro, M.; Zhang, Y. (2023). Sparks of Artificial General Intelligence: Early Experiments with GPT-4. arXiv:2303.12712. https://arxiv.org/abs/2303.12712

[2]. Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y.; Chen, D.; Dai, W.; Chan, H. S.; Madotto, A.; Fung, P. (2023). Survey of Hallucination in Natural Language Generation. ACM Computing Surveys, 55(12), Article 248. https://arxiv.org/abs/2202.03629

[3]. OpenAI (2023). GPT-4 Technical Report. arXiv:2303.08774. https://arxiv.org/abs/2303.08774

[4]. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. (2023). ReAct : Synergizing Reasoning and Acting in Language Models. ICLR. https://arxiv.org/abs/2210.03629

[5]. Schick, T.; Dwivedi-Yu, J.; Dessì, R.; et al. (2023). Toolformer : Language Models Can Teach Themselves to Use Tools. NeuroIPS. https://arxiv.org/abs/2302.04761

[6]. Shinn, N.; Cassano, F.; Labash, B.; Gopinath, A.; Narasimhan, K.; Yao, S. (2023). Reflexion : Language

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26aug1186

Agents with Verbal Reinforcement Learning. NeuroIPS. https://arxiv.org/abs/2303.11366

[7]. Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; Mané, D. (2016). Concrete Problems in AI Safety. arXiv:1606.06565. https://arxiv.org/abs/1606.06565

[8]. Russell, S. (2019). Human Compatible: Artificial Intelligence and the Problem of Control. Viking. https://doi.org/10.1007/978-3-030-86144-5_3

[9]. Mialon, G.; Dessì, R.; Lomeli, M.; et al. (2023). Augmented Language Models: A Survey. arXiv:2302.07842. https://arxiv.org/abs/2302.07842

[10]. Kaur, D.; Uslu, S.; Rittichier, K. J.; Durresi, A. (2022). Trustworthy Artificial Intelligence: A Review. ACM Computing Surveys, 55(2), Article 39. https://doi.org/10.1145/3491209

[11]. Jennings, N. R.; Wooldridge, M. J. (1998). Applications of Intelligent Agents. In: Agent Technology: Foundations, Applications, and Markets. Springer, Berlin, Heidelberg, pp. 3–28. http://eprints.soton.ac.uk/id/eprint/252188 .

[12]. Miller, T. (2019). Explanation in Artificial Intelligence: Insights from the Social Sciences. Artificial Intelligence, 267, 1–38. https://doi.org/10.1016/j.artint.2018.07.007

[13]. Rotkin, V.; Yavich, R.; Malev, S. (2018) Concept of AI Based Knowledge Generator. Journal of Education and e-Learning Research, Vol. 5, No. 4, pp. 235–241. https://doi.org/10.20448/journal.509.2018.54.235.241

[14]. Zvolinsky VP; Rotkin VM; Golovin VG; Matveeva NI (2017) Automation systems formation educational kontenta (in Russian) [Automated systems for the formation of educational content]. Scientific monograph. Publisher: VSAU, Volgograd, 120p. https://www.researchgate.net/publication/341788257_AUTOMATED_SYSTEMS_FOR_FORMATION_OF_EDUCATIONAL_CONTENT_AVTOMATIZIROVANNYE_SISTEMY_FORMIROVANIA_UCEBNOGO_KONTENTA

[15]. Rotkin, V. (2017) Methodology of immanent learning content. Scientific Israel - Technological Advantages, Vol. 19, No. 4. https://www.researchgate.net/publication/341786827_Methodology_of_immanent_learning_content

[16]. Yavich, R.; Malev, S.; Volinsky, I.; Rotkin, V. (2023) Configurable Intelligent Design Based on Hierarchical Simulation Models. Applied Sciences, 13(13), 7602. https://doi.org/10.3390/app13137602?urlappend=%3Futm_source%3Dresearchgate.net%26utm_medium%3Darticle

[17]. Yavich, R.; Rotkin, V.; Malev, S.; Zamir, E.; Kadrawi, O. (2021) Smart content creation for e-commerce. Patent US20210264366A1. https://patents.google.com/patent/US20210264366A1/en

[18]. Yavich, R; Malev, S; Rotkin, V. (2020) Triangle Generator for Online Mathematical E-learning. Higher Education Studies; Vol. 10, No. 3; 2020 ISSN 1925-4741 E-ISSN 1925-475X Published by Canadian Center of Science and Education. http://dx.doi.org/10.5539/hes.v10n3p72.