NextGen Scientific Publication
American International Journal of Computer Science and Technology
Volume 8 Issue 6, Pg. No. 1-15, AIJCST-V8I6P101, 2026
https://doi.org/10.63282/3117-5481/AIJCST-V8I6P101

Original Article

A Three-Layer Reference Architecture for Agentic Engineering Integrating Runtime Harnesses, Evidence-Driven Execution Loops, and Explicit Workflow Graphs for Production Reliability

***Gaurav Shekhar**
Technology Leadership, USA.

Abstract:

A new paradigm for engineering autonomous systems that are able to plan, execute, monitor and complete complicated tasks with very little human interaction is gaining traction, and it is known as agentic engineering. The adoption of agentic systems in production systems, however, is certainly not without its difficulties with regard to reliability of execution, state management, task coordination, evidence checks, error recovery, and transparency of operations. Frequently, there is no common design for coordinating runtime execution, validating intermediate results and systematically managing complex workflows in existing agentic architectures. In this paper, a three-layer reference architecture for agentic engineering, combining run-time harnesses, evidence-driven execution loops and explicit workflow graphs, is proposed to tackle these challenges, with the aim of making production more reliable. The first layer is for managing the environments that are run, integration of tools, tracking of state and exception handling. The second layer is for setting up evidence-based execution mechanisms, linking agent actions to observations, verification, feedback and corrective operations. The third layer is based on explicit workflow graphs to specify task dependencies, executions, conditional transitions and recovery paths. The proposed architecture makes it possible to coordinate the three layers but also ensures that traceability, observability, fault tolerance and systematic execution validation are achieved. An evaluation methodology is proposed that provides a structured approach to measuring task completion, task execution success, whether evidence is verified correctly, whether the workflow is consistent, how long the response takes and how effective the failure recovery. The architecture serves as a systematic basis for the design, assessment, and implementation of reliable agentic engineering systems in complex production systems.

Keywords:

Agentic Engineering, Agentic AI, Three-Layer Reference Architecture, Runtime Harnesses, Evidence-Driven Execution, Workflow Graphs, Production Reliability, Autonomous Systems, Execution Validation.

Article History:

Received: 16.09.2026

Revised: 14.10.2026

Accepted: 24.10.2026

Published: 02.11.2026

1. Introduction

1.1. Background and Motivation

Agentic engineering is a nascent method that helps build intelligent systems capable of autonomously planning, reasoning, acting, interacting with external tools and adapting to changing operational conditions. [1] Agentic systems involve more complex

Copyright by the Author(s). This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International (CC BY-NC-ND 4.0) License (https://creativecommons.org/licenses/by-sa/4.0/)

tasks that require interactions with run-time environments, the actions themselves, and external services, unlike traditional AI systems that focus on making predictions or providing answers. As these systems are being increasingly deployed into the production world, it is critical to ensure reliable execution, consistent state management, clear decision making and effective error recovery. This means that systematic architectural approaches that facilitate development, deployment, and operational management of reliable agentic systems are needed.

1.2. Evolution of Agentic Engineering

The evolution of agentic engineering has moved from simple rule-based automation and traditional intelligent agents, to sophisticated autonomous systems that leverage large language models, tool integration, planning capabilities, and multi-agent coordination. [2] Traditional systems were rule-based and used fixed sequences of operations, while current agentic systems are capable of breaking tasks into a dynamic sequence, reasoning in context, re-executing tasks, and adapting decisions on the fly. In recent times, agents have been made controllable using runtime harness, agents' intermediate results have been validated using evidence-driven loops, and complex task dependencies have been managed using graph-based workflows. This transformation underscores the need for a comprehensive architecture that combines execution control, verification mechanisms, and explicit workflow orchestration, to enable production-oriented agentic engineering.

2. Background and Theoretical Foundations

2.1. Fundamentals of Agentic Engineering

Agentic engineering deals with the design of intelligent systems which are capable of reaching goals, splitting complex tasks, planning their actions, communicating with external tools and assessing the results of tasks. These systems are designed to perform the goal-oriented tasks in an iterative fashion, utilizing reasoning, planning, memory, context-awareness and execution of action. [3] The difference between agentic systems and traditional software automation is that agentic systems are able to dynamically change the way they execute their tasks in response to the feedback of the environment, intermediate observations, and changing task requirements. They rely on tightly coupled interactions between language models, tool interfaces, execution environments and control mechanisms to develop predictable and reliable autonomous behavior, which requires architectural organization.

2.2. Agent Runtime Environments and Execution Harnesses

Agent runtime environments are the foundation for the operations that are needed to start, run, monitor, and manage autonomous agents during their task lifecycle. An execution harness is a control mechanism which links agent reasoning to external tools, application interfaces, computational resources and system level services. It has the main tasks of context initialization, tool invocation, managing state, enforcing permissions, monitoring execution, handling of exceptions and resource coordination. Runtime harnesses also offer ways of defining boundaries in which execution happens, and ensure that there are consistent interactions between agents and their environments. These capabilities set the groundwork for reliable agent deployment and provide for systematic integration with evidence-driven validation and orchestration of workflows.

3. Related Work and Literature Review

3.1. Existing Agentic AI Architectures

Current agentic AI architectures have largely centered on facilitating autonomous reasoning, task planning, tool usage and multi-step execution through interactions with the external environment, by intelligent agents. Traditional agent architectures usually use reasoning and planning to break down user goals into subgoals, and today's large language model-based architectures have the same capabilities that are enhanced by their contextual understanding, memory management, and ability to dynamically select tools. [4] Multi-Agent Systems are also able to have mechanisms for coordinating multiple specialized agents to achieve complex goals. Existing architectures, however, focus on the intelligence of agents and the completion of the tasks they are sent but fail to offer a holistic solution for runtime control, evidence-based execution verification and explicit workflow management needed to achieve dependable production deployment.

3.2. Runtime Frameworks for Autonomous Agents

An autonomous agent runtime framework is the execution environment that initializes agents, integrates tools, deals with context, manages state and interacts with external applications. These frameworks provide the runtime context in which to perform actions generated by agents, as well as to control computation resources, execution permissions, and interactions of the various parts of a system. The modern approaches to runtimes also integrate monitoring, exception handling, session management and control of

execution to enhance the operational consistency. Nevertheless, runtimes can be quite distinct regarding support for intermediate result validation, evidence tracking and structured workflow coordination. This restriction calls for combining runtime harnesses and evidence-driven execution mechanisms with explicit workflow graphs to create a more complete architecture for production agentic engineering.

4. Problem Formulation and Research Objectives

4.1. Problem Definition

In production systems, agentic systems add the complexity of guaranteeing the reliable and consistent, and verifiable execution of complex autonomous tasks. Agentic systems need to reason about, execute, and interact with external tools, observe intermediate states, and reason about workflow dependencies in the face of unexpected execution conditions. [5] Without integrated architectural mechanisms to control the execution at runtime, validate the evidence and control the explicit orchestration of tasks, execution failures, inconsistencies and reduced operational traceability can occur. The central issue that this research aims to tackle is to combine all these aspects into a unified reference architecture that combines runtime harnesses, evidence-driven execution loops and explicit workflow graphs to boost the reliability, transparency and controllability of production agentic systems.

4.2. Challenges in Agent Execution Reliability

There are multiple interrelated problems of autonomous decision making, tool invocation, state transition, and multi-step task coordination that must be addressed to make an agent reliably execute its tasks. [6] Actions generated by agents can yield inaccurate or incomplete results; failures with external tools, runtime exceptions, loss of context, and other changes in the environment can also interrupt the execution of tasks. Secondly, a lack of evidence validation impedes the ability to determine the success of individual actions in results. Complex workflows add additional complexities with dependencies, conditionals, synch, error propagation. The solutions to these problems should be coordinated runtime monitoring, evidence collection, execution verification and structured workflow management to ensure consistent and recoverable agent behaviour in the production environment.

5. Proposed Three-Layer Reference Architecture

Table 1. Functional Responsibilities of the Three Architecture Layers

Architecture Layer	Core Components	Primary Responsibilities	Expected Output	Reliability Contribution
Layer 1: Runtime Harness	Agent initialization, context management, tool interfaces, state control	Manage execution environments, tools, permissions, and runtime states	Controlled execution results	Execution consistency and operational control
Layer 2: Evidence-Driven Execution	Evidence collection, observation, verification, feedback	Validate actions and intermediate results using collected evidence	Verified execution outcomes	Accuracy, traceability, and error detection
Layer 3: Explicit Workflow Graphs	Workflow nodes, edges, dependencies, transitions	Coordinate task sequences, branching, synchronization, and recovery	Structured workflow completion	Workflow consistency and fault recovery
Integrated Architecture	Inter-layer communication and coordination	Coordinate runtime, verification, and orchestration	Verified end-to-end task execution	Production execution reliability

5.1. Overall Architecture and Design Principles

The proposed three-layer reference architecture offers a systematic design, implementation and management basis for reliable agentic engineering systems in production environments. [7] The architecture is made up of three intertwined layers: runtime harnesses for control of execution, execution loops based on evidence for action validation, and explicit workflow graphs for task orchestration. Each layer has a specific function and must be able to coordinate with the other layers for all aspects of the agent running during the life cycle. The runtime harness creates the operating environment and controls the states of execution and tool interaction with the agents. The evidence-driven layer assesses intermediate actions based on observations, verification tools, and

corrections based on feedback. The workflow graph layer is used to show task dependencies, execution sequences, conditional transitions and recovery paths. The main design principles of the architecture are modularity, separation of concerns, observability, traceability, fault tolerance, and extensibility. The principles allow the individual components to function autonomously while allowing for integrated execution management, predictable task progression and enhanced production reliability.

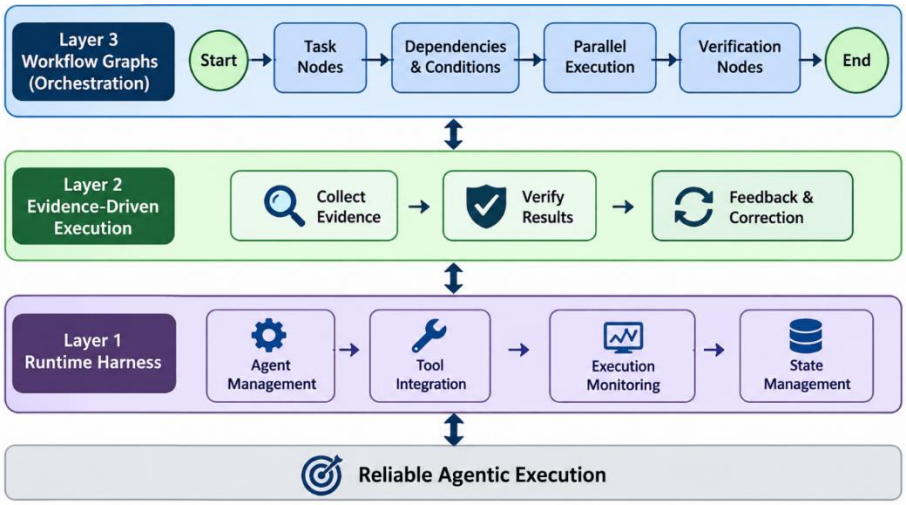


Figure 1. Overall Three layer Reference Architecture for Agentic Engineering

5.2. Layer 1: Runtime Harness and Execution Control

The first layer, called the run-time harness/execution control layer, is used to run autonomous agents within controlled production environments. [8] It is responsible for agent initialization, execution context, tool integration, resource allocation, session management and interaction with external services. The execution requests from the workflow orchestration layer are sent to the runtime harness that sets up the environment to execute individual agent tasks. It offers standardized interfaces to run external tools, call out APIs, connect to databases and run application level operations. Moreover, this layer holds the runtime state of the execution, controls permissions, records resource usage, and records events from the execution lifecycle. Exception handling mechanisms are used for recognizing operational issues like when trying to use a tool, a timeout, a service is not available, and a resource is in use. If an execution failure happens, the higher level coordination mechanisms report the status of the execution and the runtime harness stores the error information. This layer offers standardized execution boundaries, uniform tool interfaces and real-time execution monitoring, enhancing the stability of operations and laying a solid foundation for execution in the proposed architecture.

5.3. Layer 2: Evidence-Driven Execution Loops

The second layer adds execution loops based on evidence from the model to take into account the observable results of the execution of the actions generated by the agents, not just their model-generated responses. This layer sets up an iterative process of action generation, execution, observation, evidence collection, verification and corrective feedback. The agent receives an execution request and takes action(s) through the runtime harness; the outputs capture during the execution are collected as execution evidence. [9] These pieces of evidence can range from tool responses or API results, to database records, execution logs, system states, validation outputs, and more. The verification mechanism checks the collected evidence against the expected task conditions and verifies if the operation is successful or not. The execution loop starts if the evidence shows that the result is incomplete or wrong, and it takes corrective steps such as retries, different tools, or changes to parameters or reasoning. The verified results are then passed back to the workflow orchestration layer, which will then enable the proper progression of the tasks. The evidence-based approach enhances clarity of execution, minimises risk of taking outcomes for granted, and offers auditable and evaluative information.

5.4. Layer 3: Explicit Workflow Graphs and Orchestration

The third layer, explicit workflow graphs and orchestration, offers a structured depiction of complex agentic tasks by means of interconnected nodes and directed transitions. [10] A workflow graph is a diagram that represents the order of operations, the dependences of the operations, execution conditions, branching and the criteria for ending the workflow. Each node in the graph is an activity (agent reasoning, tool invocation, evidence verification, data processing, and result generation), and each directed edge

between nodes defines a transition between the activities. Based on the current state of the workflow, dependency satisfaction and verification results, the orchestration mechanism decides which node should be executed. Conditional transitions enable alternative execution paths based on the success, failure or further validation of an operation. Independent tasks can also be executed in parallel, while keeping the operations synchronized at the dependent stages. Additionally, workflow state tracking helps the system to determine which tasks are completed, pending, failed and can be recovered. This layer explicitly models the structure of the tasks and the dependency of their execution, which helps to advance the predictability, coordination, maintainability, and transparency of a complex agentic application.

5.5. Inter-Layer Communication and Coordination

Inter-layer communication is the coordination mechanism by which the three layers of the architecture can be considered a single agentic engineering system. The workflow graph layer starts the tasks by determining what task to run, and what are the conditions and dependencies for each task. These instructions are passed to the runtime harness that initializes the execution context and calls the right agent/external tool. After execution, the runtime harness gives the operational outputs, status information and associated runtimes events, to the evidence-driven execution layer. The evidence verification mechanism processes the information collected and decides if the expected execution conditions have been met. The result of the verification is then passed back to the workflow orchestration layer which then determines the next node to execute or determines a recovery path to take. This co-ordination process is continued by using standardized interfaces, structured execution messages, workflow identifiers, co-ordinated states and evidence references. Communication between layers is always possible to keep runtime activities, verification decisions and workflow transitions in synchronization. This means that the proposed architecture can be easily integrated with other systems, its execution can be tracked accurately, and it can coordinate the execution of complex autonomous tasks accurately.

5.6. End-to-End Agentic Execution Lifecycle

The end-to-end agentic execution lifecycle illustrates the three-layer architecture from the perspective of how it handles a task submission and delivers the result. The system is first presented with a user request or operation goal, translated and broken up into smaller tasks by the workflow orchestration layer. The workflow graph is used to determine the dependencies between the tasks, sequences in which they are executed and what conditions must be met to complete each operation. The runtime harness then sets up the agent environment, sets the context, and gives the agent tools and services that have been granted it. In execution, the agent takes the actions specified and produces outputs that are recorded as observable evidence. The evidence driven execution layer tests these outputs against a set of pre-defined verification conditions, and decides if the operation has met its goal. When it succeeds, the workflow graph moves to the next dependent task, if it fails, it starts corrective execution, retry or other workflow paths. It will continue until all the necessary workflow nodes meet their completion criteria or an unrecoverable failure is detected. Lastly, the system gathers verified outcomes, execution history, and pertinent evidence and delivers a final outcome, which supplies an auditable illustration of the entire agent execution lifecycle.

5.7. Reliability, Safety, and Failure-Recovery Mechanisms

Architectural factors for putting agentic engineering systems in production are reliability, safety, and failure recovery. The proposed architecture has three layers, each of which has multiple mechanisms to detect execution error, to prevent unwanted operation and ensure consistent system response in the face of unexpected events. At runtime, the execution permissions, tool access controls, timeout controls, resource monitoring and exception handling elements help to prevent execution of the actions that could lead to unauthorized and/or unstable operations. [11] At evidence-driven level, result verification, evidence traceability, validation criteria, and corrective feedback minimise the risk of accepting partial or incomplete execution results or execution results that are not correct. The transitions between the workflows were controllable with conditional transitions, failure-state identification, retry policies, alternative execution paths and checkpoint-based recovery at the workflow level. The architecture also preserves execution logs, state information, and verification records to enable observability, debugging, and verification after execution. Human approval mechanisms can be used to ensure the safety of operations that involve irreversible or impactful actions. If a failure occurs, the system determines which part of the execution process has failed, saves evidence and state information, and takes action to recover from the failure in an appropriate manner. Such built-in mechanisms provide greater fault tolerance, accountability and reliability in agentic systems with complex production environments.

6. Runtime Harness Design and Execution Management

6.1. Runtime Environment Configuration

Runtime environment configuration is the process of setting up the execution environment that will be necessary to make sure that agentic systems will work properly in various real-world production settings. It contains the setup of computing resources, operating environments, software requirements, communication interfaces, execution parameters, and the connections with external services. [12] The runtime harness will create a common environment in which agents can have access to the necessary computational resources and work with supported tools and applications. A set of configuration parameters is declared to guarantee consistent operational behavior: time limits, memory, environment variables, network configuration and availability of services. The correct runtime configuration also facilitates the reproducibility, scalability and compatibility of different deployment environments.

6.2. Agent Initialization and Context Management

Agent initialization and context management are the processes of setting up an autonomous agent for a task and storing relevant information during the agents lifetime. The runtime harness sets up the agent with the desired model configuration, task objectives, system instructions, available tools, execution constraints and contextual information. Context management keeps the history of pertinent conversations, task descriptions, intermediate observations and results of the execution to guarantee consistency in decision making over a number of execution steps. Context updates prevent irrelevant and outmoded information from being added into following reasoning processes. Further, there are mechanisms for controlled task execution, context preservation and continuation during complex multi stage workflows.

6.3. Tool Integration and Execution Permissions

The integration and execution permissions are used to create a controlled communication between autonomous agents and external resources used in order to finish up operational tasks. [13] The runtime harness offers standardized interface for agent-to-application programming interface, agent-to-database, agent-to-file system, agent-to-external services and command execution environment connections. For each integrated tool, the parameters that can be used as inputs, outputs, conditions for running the tool, and permissions that are required for running the tool are defined, resulting in a consistent interaction. Access-control mechanisms are used to define what operations an agent is allowed to perform and to limit access to certain resources, depending on a set of policies. Permission validation, input checking and execution authorization are done prior to the invocation of the tool, and the results of execution are stored for later checks. The mechanisms enhance the interoperability of tools, operational security and accountability in production agentic environments.

7. Evidence-Driven Execution and Verification Framework

7.1. Evidence Collection and Representation

The collection of evidence and representation provide the basis for verifying agent actions and verifying if the execution results fulfil the pre-defined task requirements. [14] The framework proposed in this document reflects information that can be observed from the interaction of agents with runtime environments, external tools, application interfaces and connected services. Responses from tools, outputs from API, database records, execution logs, changes in system state, validation results can be evidence, together with indicators of task completion. Structured information with an execution identifier, task reference, timestamp, action description, observed result and verification status is used for each evidence item. This schema representation allows the system to tie collected evidence to execution activities and to keep consistency throughout the various stages of the workflow. Good evidence collection makes results more transparent, facilitates systematic verification and acts as a source of good information for further decision-making.

7.2. Action–Observation–Verification Cycle

Action – observation – verification is a cycle of evaluating agent execution by iteratively, continuously interacting with the action and the observed results. First, the agent is given a task instruction and then based on the context and the execution requirement he/she creates a suitable action. The runtime harness is responsible for carrying out the desired operation using an authorized tool or external service and collecting the observations from the operation. These observations are then fed into the verification mechanism, which then compares what actually happened during the execution with the conditions expected for the task. [15] When the result observed is successful with the verification, the action is successful and the workflow moves to the next stage of execution. If not, the system notifies the difference, and triggers the corrective action as needed. This iterative process guarantees that agent executions are accompanied with observable results instead of just generated responses, enhancing execution consistency and reliability.

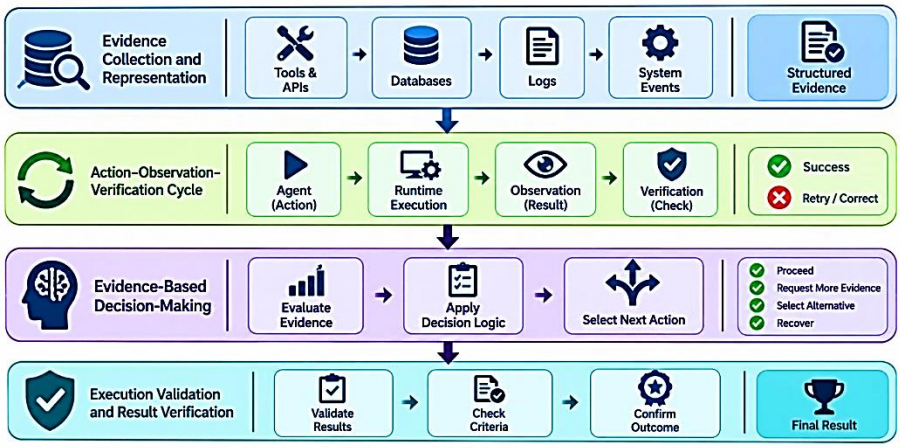


Figure 2. Evidence Collection, Validation, and Feedback Mechanism for Agentic Execution

7.3. Evidence-Based Decision-Making

By using evidence-based decision making, an agentic system can decide what action to take next based on information from the execution of the action, what the environment looks like, and what the requirements for the task are. In contrast to generated reasoning decisions, the agent considers evidence to decide if an operation has accomplished its goal. The decision mechanism takes into account the completeness of evidence, the consistency of results, the status of the execution, the level of confidence of the verification, and whether the constraints of the task are met. The system can approve progress of task, ask for more evidence, adjust parameters of task execution, choose other tools, or start recovery mechanisms based on these considerations. In complex workflows with many operations, wrong intermediate results can impact later operations, and evidence-based decisions are critical. The proposed framework takes advantage of verified observations and minimizes unsupported execution assumptions in the decision process, while also making the behavior of autonomous agents more transparent, consistent, and accountable.

7.4. Execution Validation and Result Verification

Validation of the execution of agents and verification of the results of their final tasks is used to verify that each agent operation and final task result fulfill their functional and operational requirements. The framework assesses the results of execution by comparing evidence generated with the expected output, validation rules, task-specific conditions and completion criteria. [16] Examples of validation mechanisms are checking the consistency of the output, schema validation, verifying API responses, comparing database states, checking the status of execution, and matching expected results. An operation will be given a verification result that will tell them if the operation is successful, failed, incomplete, or needs further investigation. If a job consists of multiple steps, the intermediate outputs of the steps are stored and verified; subsequent steps are based on validated outputs. Final result verification is the process of putting together the evidence collected throughout the execution phase and establishing whether the overall objective was met. This enables a systematic verification mechanism where the correctness of results is enhanced, their execution is more confident, and the production process is more reliable.

8. Explicit Workflow Graph Modeling and Orchestration

8.1. Workflow Graph Representation

Workflow graph representation is a structured way of defining and handling complex agentic tasks by connecting nodes and directed edges. The proposed framework models each task as a graph with the following nodes: execution nodes, dependency nodes, condition nodes and state nodes. [17] Each node corresponds to an operation, such as an agent reasoning, invoking a tool, processing data, verifying evidence or generating a result, and the directed edges represent the relationships and order of operations between the nodes. Information on relevant execution parameters, requirements on inputs, outputs to be expected and verification conditions are placed in each node. The graph structure allows the system to represent sequential and complex multi-stage workflows and to keep in good practice execution dependencies. This representation is useful for increasing the transparency of the workflow, for supporting modular task management and for offering a systematic basis for coordinating the activities of autonomous agents in production environments.

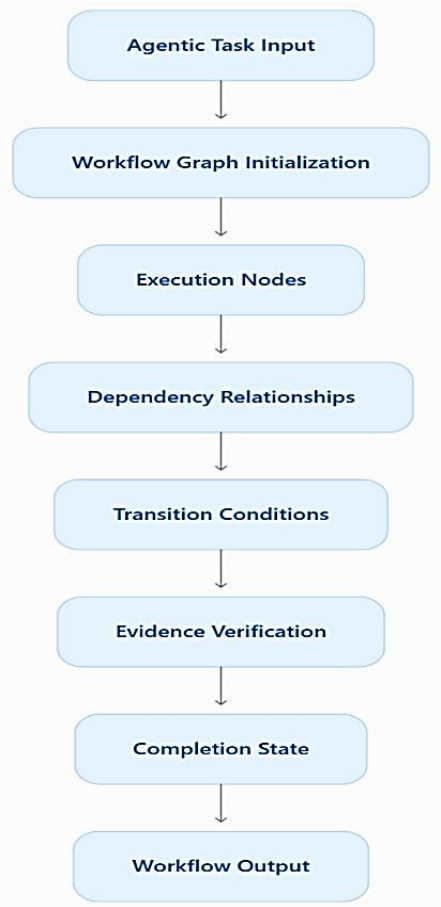


Figure 3. Workflow Graph Representation for Agentic Task Orchestration

8.2. Task Decomposition and Dependency Modeling

Break down user goals into smaller, more manageable tasks and model the dependency relationships between the tasks. [18] The proposed framework starts by analyzing the overall task requirements and then breaks them down into individual subtasks in terms of functional responsibilities, order of execution as well as resource dependencies. Each subtask is expressed as a node in a workflow, where there are explicit inputs, outputs, and conditions. The dependency relationship will dictate whether a certain task can start right away or has to wait for the successful completion of the previous operations. If tasks can be broken down into independent subtasks then these subtasks can be parallelized, if not they follow controlled execution sequences. The framework also takes into account data dependence, tool availability and verification needs in defining links between the workflow nodes. This structured decomposition is beneficial for coordinating tasks, clarifying execution ambiguity, and managing more complex agentic goals by using well-defined workflow structures.

8.3. Node-Based Execution and Transition Management

Agentic workflows are directed by the progression of individual operations: they are node-based in execution and transition management, which means that they coordinate the operations based on the defined graph structure and their execution conditions. [19] Each workflow node gets activated if all its dependencies are met and the required execution resources are available. The runtime harness carries out the respective operation, the evidence-driven verification layer checks the outputs against pre-defined completion criteria. The orchestration mechanism changes the node status based on the verification result, and decides the next transition. Once the nodes have been successfully completed, they continue to proceed to the dependent operations, otherwise, they activate corrective actions or alternative execution paths. Transition management also guarantees the execution order and will not allow unwanted changes of state and will guarantee that dependent tasks are executed with verified outputs. This node-based approach ensures predictable workflow progression, consistent execution coordination and greater reliability in complex autonomous operation management.

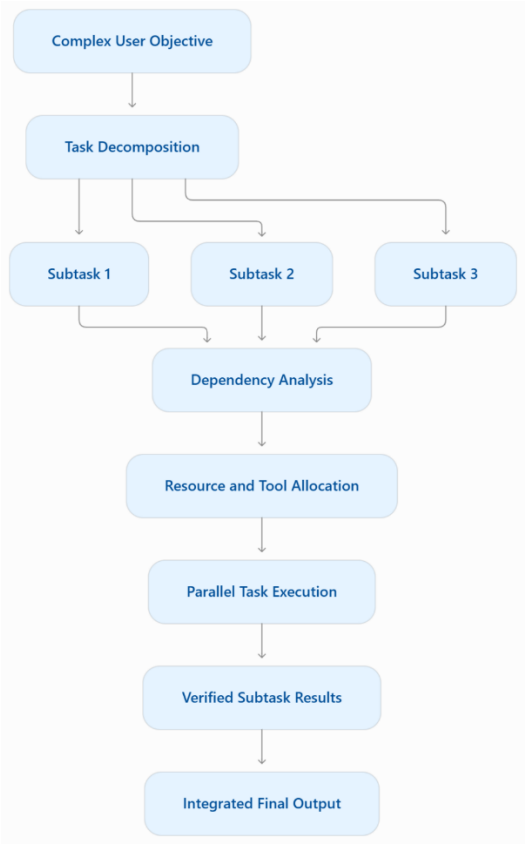


Figure 4. Task Decomposition and Dependency Modeling Framework

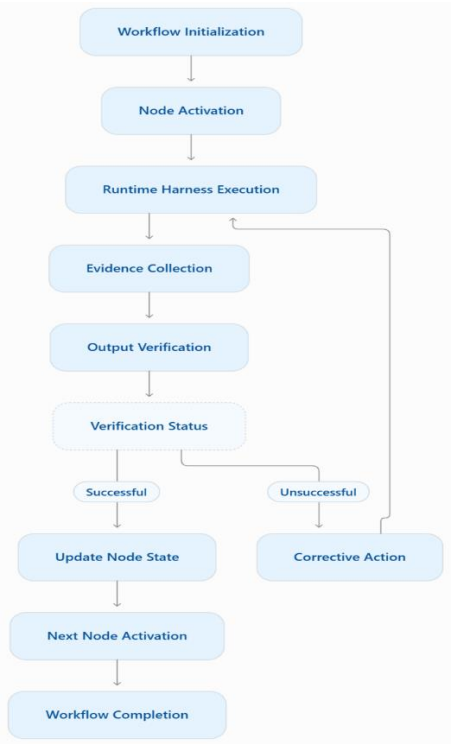


Figure 5. Node-Based Execution and Transition Management Workflow

9. Experimental Design and Methodology

9.1. Experimental Objectives and Evaluation Strategy

The goal of the experimental evaluation is to measure the effectiveness of the proposed three-layer reference architecture to increase the reliability, consistency of execution and transparency of operations of agentic engineering systems. [20] The evaluation is based on metrics that measure individual performance of the runtime harnesses, evidence-driven execution loops, explicit workflow graphs, and their synergistic effect on production-oriented task execution. A systematic evaluation strategy is used to analyze task completion, task execution success, evidence verification, consistency in workflow, failure detection and recovery performance in various operational scenarios. Several experimental scenarios are set up to test normal and failure prone execution environments. Comparative analysis of conventional agentic architectures and the proposed three-layer structure is carried out to find the improvement in the execution reliability, fault tolerance, and the overall system performance.

9.2. Test Environment and Infrastructure Configuration

The experimental test environment aims at having a controlled infrastructure that can be used to test agentic systems under various execution conditions and workloads. [21] The environment comprises a Computing Platform, Agent Execution Framework, Language Model Interfaces, Runtime Harness, Integrations with external tools, Workflow Orchestration Engine, and Evidence Collection Mechanisms. The hardware and software configurations, including processor resources, memory allocation, operating system, runtime dependencies, model configurations and network connectivity, remain the same across the experiments. External services like APIs, databases, file systems and application interfaces are connected to create realistic agent activities. Execution logs, resource usage, tool responses, workflow transitions and verification results are all monitored. Controlled test configurations allow for repeatability of experiments, and for systematic comparison of the proposed architecture under various operational conditions.

9.3. Agent Models, Runtime Tools, and Workflow Configurations

In the experimental implementation, agent models, runtime tools, and configurations of the workflows are used to analyze the operational capabilities of the proposed three-layer architecture. Agent models are designed to execute multi-step task execution, select tools, reason and interpret tasks and actions based on goals. Runtime tools are APIs, database operations, file processing commands, and external application services that allow agents to communicate with their runtime context. [22] The Runtime Harness controls agent initialization, context management, permissions for tools, monitoring of tool execution and agent state management. Evidence-driven mechanisms record the intermediate outcomes and verify the results of the execution of the mechanism with predefined verification rules. Workflow configurations specify task nodes, how tasks are dependent on each other, conditional transitions, parallel processing, and recovery on failure. Individual and combined contributions of these components are explored by integrating them together.

10. Evaluation Metrics

10.1. Task Completion Rate

The percentage of the assigned agentic tasks that are successfully accomplished within the goals and outcomes set by the school. This is a measure of the capability of the proposed three-layer architecture to control the entire task execution through various workflow stages. [23] A task is completed if and only if all of the necessary operations have been performed, and the result of the task meets the completion criteria. It is measured by the percentage of tasks completed successfully out of the total number of tasks assigned. The higher the task completion rate, the greater the capability of tasks to be executed and the more effective the coordination between the execution of tasks through the runtime harnesses, between the verification of evidence and the coordination of the orchestration of tasks.

10.2. Execution Success Rate

Execution success rate is the ratio of the number of individual agent execution operations that are executed successfully without any unhandled exceptions and without any invalid results. [24] It takes into account the operations that are executed at runtime as well as the interactions with tools created by agents and tasks' intermediate workflow activities during task execution. Success is measured not just by the agent-generated responses but by the runtime status information and evidence of that status. The number of successfully executed operations divided by the number of operations that were attempted and multiplied by 100. This metric gives a sense of the runtime stability, consistency of operation, and the effectiveness of execution control mechanisms in the proposed architecture.

10.3. Evidence Verification Accuracy

The accuracy of verification of evidence of execution means the capability of proposing a framework to correctly identify if collected execution evidence meets defined verification criteria. The indicator assesses the alignment of actual evidence-based verification results with the anticipated validation results. Verification mechanisms include checking outcomes of the tools, outputs of the API, database records, execution logs, and what can be observed in the state changes to see if an operation has done what it was supposed to do. The metric can be expressed as the percentage of correct execution outcome verifications out of all the execution outcome verifications. The higher the accuracy of the verification, the more it can be trusted to correctly identify successful, unsuccessful, or incomplete operations, which helps decrease unsupported execution claims, and thus the trustworthiness of agentic systems.

11. Results and Performance Evaluation

11.1. Overall Agentic Execution Performance

Table 2. Task Completion Rate across Agentic Architectures

Agent Architecture	Value
Rule-Based	72.5
LLM-Based	81.4
Tool-Augmented	87.6
Proposed	94.2

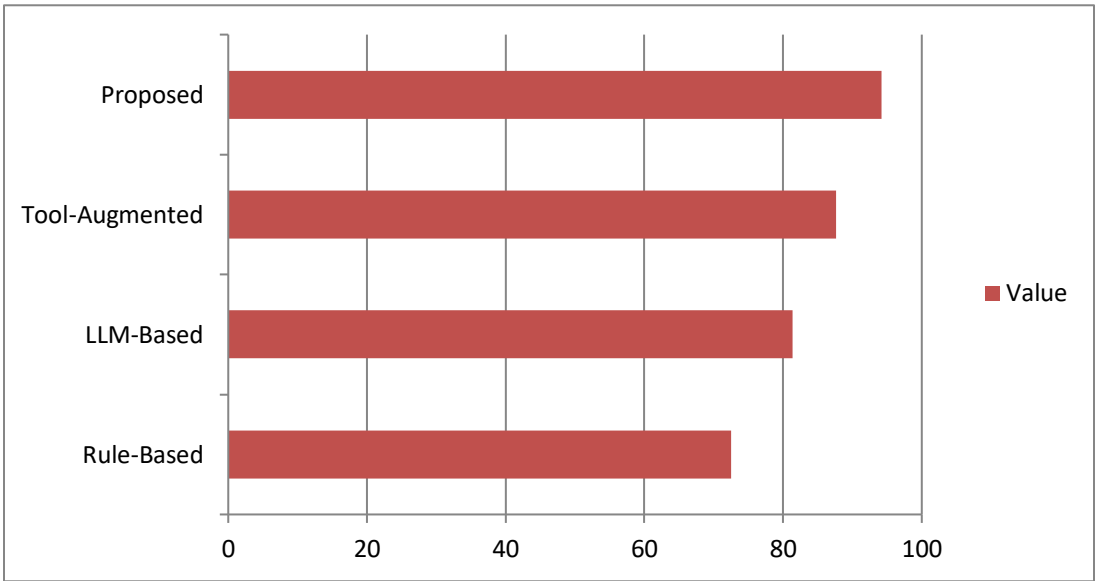


Figure 6. Graphical Task Completion Rate across Agentic Architectures

The overall agentic execution performance measures the overall effectiveness of the proposed three-layer reference architecture in complex task completion, varying execution conditions. [25] The evaluation takes into account the task completion rate, the task execution success rate, the evidences verification accuracy, the consistency of the task workflows, and the reliability of end-to-end task execution. The integrated architecture is analyzed by analyzing the contribution of the runtime harness, the evidence-based execution loops and explicit workflow graphs to the successful execution of tasks. Experimental observations are analyzed to find what improvements are needed to the way results are achieved: in terms of the execution coordination, intermediate results validation, and operational consistency in comparison to the conventional agentic approach. The results are an overall estimation of how well the proposed architecture can ensure reliable task execution independently of the other tasks as well as maintain the task execution transparency and systematic control in multiple stages of the task execution.

11.2. Runtime Harness Reliability Evaluation

To assess runtime harness reliability, the stability and consistency of agent runtime in the face of varying workload conditions and operational scenarios are considered. A key point of the evaluation is the agent's initialisation, context management, tool invocation, execution control, state preservation, exception handling and resource management. Runtime failures include: tool timeouts, unavailable services, invalid inputs, unexpected failure during execution... etc. to help evaluate the effectiveness of the runtime monitoring and error-management mechanisms. Execution logs and run-time status information are used to examine the various failures that may happen while performing an operation and the ability of the harness to keep the operation controlled. The results show the impact of the runtime management on the stability of the operations, uniformity of agent behavior and reliable integration with external tools and services.

11.3. Evidence-Driven Execution Performance

Table 3. Evidence Verification Accuracy Comparison

Agent Architecture	Value
Rule-Based	68.4
LLM-Based	76.8
Tool-Augmented	85.2
Proposed	96.3

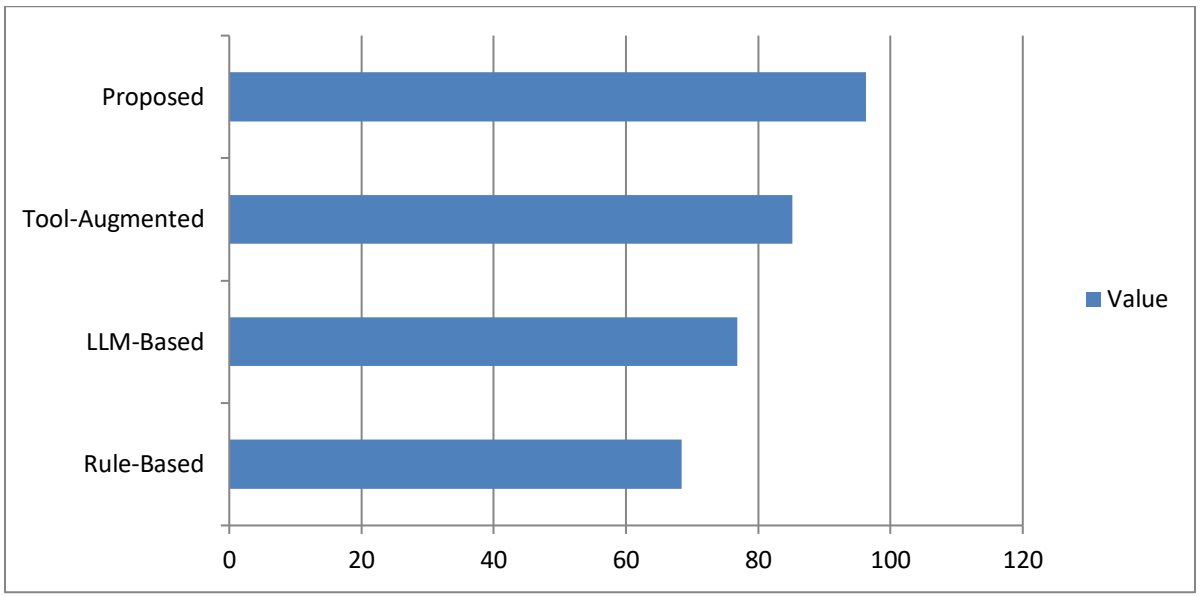


Figure 7. Graphical Evidence Verification Accuracy Comparison

Evidence-driven execution performance is a measure of how effective action–observation–verification mechanisms are for validating agent operations and reliable execution results. [26] The evaluation focuses on accuracy of evidence collection, verification of evidence, consistency of evidence in intermediate steps, corrective feedback, and successful recovery from incorrect, incomplete, operations. Various execution situations are examined to see if the framework distinguishes between verified outcomes and unsupported/unsuccessful outcomes. Special emphasis is placed on the link between the evidence collected and the decisions made by the agents so that the steps in the workflow are based on evidence from what the agents observe as they execute. The evaluation shows how the use of evidence-driven mechanisms aids execution transparency, result correctness and decreased acceptance of invalid intermediate results.

11.4. Workflow Graph Execution Performance

For the multi-step agentic tasks, an explicit graph-based orchestration has to be effective in executing the workflows, and this can be evaluated by this measure. Task decomposition, dependency management, node activation, conditional transitions, parallel

execution, and state synchronization in the workflow are taken into account. Various workflow configurations are explored to see if the orchestration mechanism is able to guarantee the workflow's execution order as well as provide verified outputs for dependent operations. The analysis also takes into account the completion of the workflow, the accuracy of the transition, the consistency of the execution and the ability to recover from a failed node. The results offer insight into the benefits of explicit representations of workflows in terms of task coordination, task ambiguity reduction and predictable task progression across complex autonomous tasks.

12. Discussion

12.1. Impact of Layer Integration on Production Reliability

The presence of the runtime harness, evidence-driven execution loops, and explicit workflow graph offers a coordinated basis for enhancing the reliability of production agentic systems. The runtime harness provides controlled execution environments; controls execution states of tools and agent; monitors execution activities; and the evidence-driven layer validates execution activities and intermediate actions. The workflow graph represents the relationships between tasks, the order in which they are executed, conditionally linked transitions, and recovery actions. These capabilities combine to overcome some of the issues that arise with agent-based limitations on execution mechanisms and the need to ensure more predictable task completion, enhanced fault handling, and increased execution traceability. The integrated architecture thus ensures a structured management of complex autonomous operations, lowers execution uncertainty and increases production reliability.

12.2. Contributions of Evidence-Driven Execution

By using only the results of the execution, as opposed to excluding reasoning and assumpitions solely based on generated results, evidence driven execution helps to make production more reliable. Responses to tools, outputs from APIs, database rows, execution logs, and state changes are collected and checked to ensure intermediate operations and outputs are correct and the final outcomes are as expected. If evidence suggests that the operation was not completed or did not work, the architecture can take corrective action, repeat the operation or choose alternate operations before executing the next operation. This is a better way to verify results, helps the decision making process be transparent, and gives an audit trail of agent's actions. This means that autonomous systems in complex production environments are more trusted and accountable when they are executed properly based on evidence.

13. Limitations and Future Research

13.1. Architectural and Implementation Limitations

The proposed three-layer reference architecture offers a well-defined architecture and development of more reliable production agents; however, there are some limitations in the architecture and implementation. The use of runtime harnesses, evidence-driven execution loops and explicit workflow graphs could potentially add to the complexity of the system and introduce further configuration, coordination and maintenance challenges. The quality of the workflow definitions, evidence collection mechanisms, verification criteria and runtime management policies are also crucial for the effectiveness of the architecture. In addition, highly dynamic tasks could necessitate frequent changes in workflow and/or incomplete or inconsistent evidence could have an impact on verification decisions. The restrictions present the necessity for flexible architecture designs, standard interfaces, and flexible execution mechanisms which are able to support various production requirements and applications for agents.

13.2. Scalability and Computational Overhead

The architecture proposed might incur extra computation and operational overhead due to monitoring at run time, evidence collection, result verification, processing the workflow state and recovery processes. More agents, tools, workflow nodes and parallel tasks can lead to issues with coordination and synchronization, which can impact on execution time and resource usage. Storing evidence, and repeatedly verifying it, can also require additional memory, processor and data management resources. Moving forward, lightweight verification mechanisms, distributed application workflow management, efficient evidence storage, and adaptive resource allocation need to be explored to further enhance the scalability of applications. More testing in larger production systems is required to assess the architecture's behavior with varying workload levels and concurrency.

14. Conclusion

In this paper, a three-layer reference architecture supporting agentic engineering was introduced, combining a runtime harness, an evidence-driven execution loop and an explicit workflow graph, to enhance the reliability of production agentic systems. The proposed architecture provides a coordinated framework that includes a runtime harness to manage agent execution, tools and

operational states, evidence-driven execution loops to check actions and outcomes with observations and verification, and explicit workflow graphs to define and manage task dependencies and execution sequences and recovery paths. The layers integrated enable systematic execution management, enhanced traceability, transparent decision making and structured failure recovery. The paper also presented an experimental approach and evaluation indicators to evaluate task completion, task execution success, evidence verification accuracy, workflow consistency and recovery effectiveness. While there are many challenges such as implementation complexity, computational overhead, scalability, and dependence on external tools, the proposed architecture offers a starting point for the development of more controllable, observable, and reliable agentic systems. Research will be focused on implementing the solution at a large scale, the quantitative validation of the solution, the adaptive orchestration of the workflow, and the extension of the security mechanisms for production deployment in future research.

Reference

[1] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629.
[2] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., ... & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. Advances in neural information processing systems, 36, 68539-68551.
[3] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. Advances in neural information processing systems, 36, 8634-8652.
[4] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., ... & Clark, P. (2023). Self-refine: Iterative refinement with self-feedback. Advances in neural information processing systems, 36, 46534-46594.
[5] Gou, Z., Shao, Z., Gong, Y., Yang, Y., Duan, N., & Chen, W. (2024, May). Critic: Large language models can self-correct with tool-interactive critiquing. In International Conference on Learning Representations (Vol. 2024, pp. 57734-57811).
[6] Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., ... & Neubig, G. (2023, July). Pal: Program-aided language models. In International conference on machine learning (pp. 10764-10799). PMLR.
[7] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in neural information processing systems, 33, 9459-9474.
[8] Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023, October). Generative agents: Interactive simulacra of human behavior. In Proceedings of the 36th annual acm symposium on user interface software and technology (pp. 1-22).
[9] Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., ... & Wen, J. (2024). A survey on large language model based autonomous agents. Frontiers of computer science, 18(6), 186345.
[10] Li, X., Wang, S., Zeng, S., Wu, Y., & Yang, Y. (2024). A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. Vicinagearth, 1(1), 9.
[11] Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., ... & Sun, M. (2024, May). Toolllm: Facilitating large language models to master 16000+ real-world apis. In International Conference on Learning Representations (Vol. 2024, pp. 9695-9717).
[12] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., ... & Wang, C. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation. arXiv preprint arXiv:2308.08155.
[13] Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Wang, J., ... & Schmidhuber, J. (2024, May). MetaGPT: Meta programming for a multi-agent collaborative framework. In International Conference on Learning Representations (Vol. 2024, pp. 23247-23275).
[14] Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., ... & Tang, J. (2024, May). Agentbench: Evaluating llms as agents. In International Conference on Learning Representations (Vol. 2024, pp. 52989-53046).
[15] Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., ... & Neubig, G. (2024, May). Webarena: A realistic web environment for building autonomous agents. In International Conference on Learning Representations (Vol. 2024, pp. 15585-15606).
[16] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024, May). Swe-bench: Can language models resolve real-world github issues?. In International Conference on Learning Representations (Vol. 2024, pp. 54107-54157).
[17] Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., ... & Su, Y. (2023). Mind2web: Towards a generalist agent for the web. Advances in Neural Information Processing Systems, 36, 28091-28114.
[18] Karpas, E., Abend, O., Belinkov, Y., Lenz, B., Lieber, O., Ratner, N., ... & Tenenholtz, M. (2022). Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. arXiv preprint arXiv:2205.00445.
[19] Wooldridge, M. (2009). An introduction to multiagent systems. John wiley & sons.
[20] Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., ... & Schulman, J. (2021). Webgpt: Browser-assisted question-answering with human feedback. arXiv preprint arXiv:2112.09332.
[21] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. Advances in neural information processing systems, 36, 11809-11822.
[22] Li, G., Hammoud, H., Itani, H., Khizbullin, D., & Ghanem, B. (2023). Camel: Communicative agents for" mind" exploration of large language model society. Advances in neural information processing systems, 36, 51991-52008.

[23] Baylor, D., Breck, E., Cheng, H. T., Fiedel, N., Foo, C. Y., Haque, Z., ... & Zinkevich, M. (2017, August). Tfx: A tensorflow-based production-scale machine learning platform. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1387-1395).
[24] Yu, J., & Buyya, R. (2005). A taxonomy of workflow management systems for grid computing. Journal of grid computing, 3(3), 171-200.
[25] Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., ... & Wenger, K. (2015). Pegasus, a workflow management system for science automation. Future Generation Computer Systems, 46, 17-35.
[26] Taufer, M., & Rosenberg, A. L. (2017). Scheduling DAG-based workflows on single cloud instances: High-performance and cost effectiveness with a static scheduler. The International Journal of High Performance Computing Applications, 31(1), 19-31.
[27] Benoit, A., Çatalyürek, Ü. V., Robert, Y., & Saule, E. (2013). A survey of pipelined workflow scheduling: Models and algorithms. ACM Computing Surveys (CSUR), 45(4), 1-36.
[28] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. Advances in neural information processing systems, 28.
[29] AI, N. (2023). Artificial intelligence risk management framework (AI RMF 1.0). URL: https://nvlpubs. nist. gov/nistpubs/ai/nist. ai, 1(0), 0-1.