Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

ARTEMIS: Adaptive Reliable Task Execution with Multi-agent Intelligence and Self-verification

Babasaheb Satpute
JAIN(Deemed-to-be University)
Suriya Prakash J.
JAIN(Deemed-to-be University)
Rajesh Bharati
Dr. D.Y.Patil Institute of Technology, Pimpri
Wasudeo Rahane
DES PUNE UNIVERSITY
Srinidhi N N
srinidhi.nn@manipal.edu

Manipal Academy of Higher Education

Article

Keywords: Agentic AI, Multi-Agent Systems, Formal Verification, Hallucination Detection, Adaptive Orchestration, Reliable AI, LLM Agents, Task Planning

Posted Date: May 19th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9353255/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

ARTEMIS: Adaptive Reliable Task Execution with Multi-agent Intelligence and Self-verification

Babasaheb Satpute[1], Suriya Prakash J.[1], Rajesh Bharati[2], Wasudeo Rahane[3], Srinidhi N N[4]

Corresponding author: Srinidhi N N (srinidhi.nn@manipal.edu)

1. Department of CSE, JAIN(Deemed-to-be University), Bangalore, India
2. Dr. D.Y.Patil Institute of Technology, Pimpri, Pune, India
3. DES Pune Univesity, Pune, India
4. Manipal Academy of Higher Education, Manipal Institute of Technology, Bangalore, India

---

Abstract

Agentic AI systems have proved great capabilities in autonomous task execution. But their deployment in high-stakes domains is still constrained by some basic challenges related to reliability. These are hallucination-induced failures, unverified plan correctness, and brittle coordination mechanisms. In this paper we introduce a new framework ARTEMIS (Adaptive Reliable Task Execution with Multi-agent Intelligence and Self-verification). It's a multi-agent framework which integrates formal verification, runtime hallucination detection, and adaptive orchestration to achieve provable reliability guarantees. ARTEMIS is based on a three-tier architecture: (1) the first tier is a Formal Planning Layer. This tier generates executionplans which we can verify with mathematical correctness proofs, (2) second tier is a Cross-Agent Verification Module. This module detects and recovers from hallucinations through consensus-based validation, and (3) third tier is an Adaptive Orchestration Engine. This engine dynamically adjusts coordination strategies based on runtime performance metrics. We describe the reliability problem as a constrained optimization over multi-agent execution traces. We then show that ARTEMIS limits error propagation within a fixed bound under certain conditionsWe conduct extensive experiments on three benchmark datasets: TravelPlanner, WebShop, and a new SafetyBench dataset. The results show that ARTEMIS achieves a 94.3% task success rate and 97.8% accuracy in detecting hallucinations.

It works out better than the available structures such as AutoGen, LangGraph and Alpha Berkeley. It is also superior to those by 23 to 41% in reliability. Concurrently it is keeping the same latency.

Ablution studies are showing that every element of the system is important for improving the reliability. Particularly the verification module alone reduces critical failures by 68%.

In whole, this work gives a new approach for building reliable agent-based AI systems with formal guarantees of reliability.

Keywords: Agentic AI, Multi-Agent Systems, Formal Verification, Hallucination Detection, Adaptive Orchestration, Reliable AI, LLM Agents, Task Planning

---

Table of Contents

1. Introduction
2. Literature Review
3. Proposed Methodology
4. Experimental Setup
5. Results and Discussion
6. Ablation Study
7. Robustness and Reliability Analysis
8. Conclusion and Future Work
9. References

---

1. Introduction

1.1 Background on Agentic AI

Any move towards larger agent-based AI systems has been contributed to due to large language models (LLMs) which have good reasoning abilities. These systems are autonomous similar to agents. These are capable of decomposing multi-faceted tasks, using third-party tools, and multi-step processes [1], [2], [3]. They have performed well in most fields such as web navigation [4], software development [5], scientific workflows [6] and enterprise knowledge management [7]. Recent systems such as AutoGen [8], LangGraph [9] and Alpha Berkeley [1] give the simple schemata of management of a number of agents. These structures can enable planning tasks, the use of tools, and communication with human beings better.

Even thoughthese improvements are done, agent-based AI systems are not being used at big scale in safety-critical and high-risk environments because of issues related to reliability. Current systems show three main types of failures:

(1) Hallucination errors: In this LLMs produce information that seems correct but it is actually wrong. This is error and it spreads through the system [10], [11];
(2) Unverified plans: Here the generated plans are not checked to ensure that they are correct or they meet all constraints [12], [13]; and
(3) Weak coordination: In this the fixed control strategies cannot adjust to failures or changes during execution [14], [15].

1.2 Problem Statement

The main problem in making reliable agentic AI systems is described as follows: Let's say we are given a complex task **T**and it has certain constraints$C = \{c_1, c_2, \ldots, c_n\}$, a set of agents $A = \{a_1, a_2, \ldots, a_m\}$ with capabilities $\Phi = \{\phi_1, \phi_2, \ldots, \phi_k\}$, and a stochastic execution environment E, design a multi-agent system that:

1. Generates an execution plan P that provably satisfies all constraints in C
2. Detects and recovers from hallucinations during execution with probability $\geq 1 - \epsilon$
3. Adapts orchestration strategies to minimize expected task failure rate under environmental uncertainty
4. Will Provide a formal bounds on error propagation across the agent interactions

The frameworks which are available handles only some of the needs. Also they do not provide a complete solution. AutoGen [8] supports flexible multi-agent communication but it does not include formal verification. Alpha Berkeley [1] support human approval, but it has a problem that, it depends on manual checking rather than automatic verification. Agent Q [4] uses Monte Carlo Tree Search for planning but it doesn't have a perfect way to detect hallucinations. Some latest studies on information sharing [2] and self-organizing networks [16] have improved the coordination, but they still do not guarantee the reliability.

1.3 Research Gap

We analysed 30 recent papers (2022–2026) on agent-based AI systems and we found five key research gaps:

1. **Absence of formal verification**: No existing framework provides mathematical proofs of plan correctness or constraint satisfaction before execution
2. **Limited hallucination handling**: Current approaches rely on post-hoc detection or human oversight rather than systematic runtime verification
3. **Static orchestration**: Existing orchestrators use fixed coordination strategies that cannot adapt to execution failures or performance degradation
4. **Weak reliability guarantees**: Systems lack formal bounds on error propagation, making them unsuitable for safety-critical applications
5. **Insufficient robustness evaluation**: Most papers report success rates on narrow benchmarks without comprehensive failure analysis or adversarial testing

1.4 Contributions

In this paper we present ARTEMIS. It'sa new multi-agent framework for addressing the above gaps. This is done by combining formal verification, real-time hallucination detection, and adaptive coordination. The key contributions of this paper are:

- **Novel Architecture**: A three-tier architecture integrating formal planning, cross-agent verification, and adaptive orchestration with provable reliability properties

- **Formal Verification Layer**: A constraint satisfaction framework that generates execution plans with mathematical correctness proofs using temporal logic specifications
- **Hallucination Detection Module**: A consensus-based verification mechanism that detects LLM hallucinations through cross-agent validation with 97.8% accuracy
- **Adaptive Orchestration Engine**: A reinforcement learning-based orchestrator that dynamically adjusts coordination strategies based on runtime performance metrics
- **Theoretical Guarantees**: Formal proofs of bounded error propagation and probabilistic reliability guarantees under specified conditions
- **Comprehensive Evaluation**: Extensive experiments on three benchmarks demonstrating 23-41% improvement in reliability metrics over state-of-the-art frameworks
- **Robustness Analysis**: Systematic failure case analysis, adversarial testing, and ablation studies validating each architectural component

The rest of the paper is organized as follows: Section 2 is about literature survey and it tells where ARTEMIS stands in the current landscape. Section 3 describes the proposed method, it includes the system design, mathematical model, and algorithms. Section 4 describes the experimental setup and evaluation metrics. Section 5 discusses the results and comparison with other methods. Sections 6 and 7 discuss ablation studies and robustness analysis. Section 8 is about the conclusion of the paper and it suggests future work.

---

2. Literature Review

2.1 Multi-Agent Orchestration Frameworks

Since 2022, multi-agent LLM systems have been developing rapidly and a few key LLM frameworks have emerged which define how agents are organised. AutoGen [8] presented flexible multi-agent conversations based on a conversational programming model where agents communicate with a series of dialogic steps to resolve complex problems. However, it does not have any formal verification and relies on emergent coordination with no clear guarantees of reliability. LangGraph [9] combined a graph-based approach with good state management, where agent workflows are represented as conditional directed graphs. This provides better visibility into the control flow, but has fixed orchestration strategies that are unable to cope with failures during execution.

Recent research has focused on developing systems that are ready-to-use in the real world, from researching and developing building systems. Alpha Berkeley [1] Enhances the scalability and human supervision by employing dynamic capability classification, human-approved planning, and production-level execution environments. It has been applied at the Advanced Light Source particle accelerator with the results demonstrating the feasibility of applying agent-based systems in high-risk environments. However, it relies on formal validation through manual human approbation instead of the automatic verification needed for swift performance of time-critical tasks. AegisFlow [17] and AgentOrchestra [18] both propose hierarchical multi-agent designs for

general task solving, but these do not offer formal reliability guarantees or a concrete way of detecting hallucination.

2.2 Planning and Reasoning in Agentic Systems

Reliable task planning remains a major challenge for agent-based Artificial Intelligence systems. Agent Q [4] is based on agent Q [3], which is a combination of guided Monte Carlo Tree Search (MCTS) and self-critique methods that achieved 95.4% success rate on tasks on real world booking. 2Q further achieved through repeated fine-tuning using Direct Preference Optimization (DPO). While this has a strong practical performance, it does not show formally if plans are correct and does not give theoretical guarantees on errors. OctoTools [19] shows a framework with flexible tools for dealing with complex reasoning problems but (mainly) focuses on integrating tools rather than on checking whether the generated plans are correct.

Recent research on the subject of multi-agent planning has focused on how to enhance the way in which agents share information and coordinate. Ou et al. [2] demonstrate that the use of structured information sharing via a persistent notebook leads to an 18% reduction of hallucination errors in long planning tasks. Further improving the performance by 13.5% are added orchestrator agent. Their work on the TravelPlanner benchmark illustrates the value of reflective coordination however their methods lack formal verification and adaptability. GradientSys [20] uses ReAct-based orchestration to manage multiple agents, however, it does not offer reliability guarantees or any method to detect hallucinations.

2.3 Hallucination Detection and Mitigation

Hallucination - when LLMs generate information that seems right, but it is wrong, is a big issue with agent-based systems. Current methods to reduce the problem of hallucinations fall into three main types:(1) post-hoc detection, where outputs are checked against external knowledge bases after they are generated [21];

(2) self consistency checking, in which a number of outputs are produced and compared to select the most consistent one [22]; and

(3) human-in-the-loop validation, the human's check and monitors the results [1].

However, these methods have major drawbacks. Post-hoc detection adds delay and requires massive knowledge databases, self-consistency methods take more computation and don't guarantee reliability, and human verification will slow down the process of automated activities by causing bottlenecks.

Recent research has been done investigating verification between multiple agents. GraphCogent [23], dealing with the limited working memory of LLMs, using multi-agent collaboration for graph tasks, of an agent working together can overcome the limits of one model. MAS² [24] is a proposal for multi-agent systems that can generate, configure and correct themselves, but that does not incorporate any clear methods for the detection of hallucinations or formal verification. In general,

there remains a gap between the practical means of diminishing hallucinations and having formal guarantees of reliability in current research.

2.4 Formal Verification and Reliability

Formal verification techniques embraced from software engineering and control theory are not commonly applied in agent-based AI systems. Traditional methods for verifying multi-agent systems use model checking and temporal logic [25] but this assumes that the agents will behave in the same way, that they can for even a long time and that the randomness present in LLM outputs is taken into account. Recent studies about the issue of LLM reliability explore probabilistic verification [26] and measurement of uncertainty [27], but they have not been integrated into whole agent-based systems yet.

TB-CSPN [28] presents a turing complete language where the separation of a semantics processing and a logics of coordination is introduced with the support of rule-based control aimed at a greater reliability. However, it limits the usage of LLMs which leads to a decreased flexibility and adaptibility. AgentGit [29] adapts the version control idea to multi-agent systems that use LLMs in which rollback and reproducibility are possible. However, it lacks formal verification of plans and guarantees for reliability in their execution.

2.5 Adaptive and Self-Improving Systems

Adaptive orchestration is an emerging field of research. The Self-Organizing Agent Network [16] proposes agent networks created dynamically according to the task, but no clear methods for adaptation and reliability analysis are given. AIME [30] is focused on constructing fully autonomous multi-agent systems that are able to improve themselves, but does not specify how to achieve adaptive orchestration and provide formal guarantees of reliability.

The use of reinforcement learning to adaptive coordination has been the study of a few cases. Agent Q [4] uses DPO for repeated fine-tuning which demonstrates that learning from both successful and failed attempts aids in improving performance. However, this approach is focused on the external improvement of individual agents rather than the overall coordination approach. Designing orchestrators that are capable of changing the way agents collaborate as a function of the performance at run-time remains an open problem.

2.6 Comparative Analysis and Limitations

Table 1 outlines the advantages and disadvantages of the major frameworks in our review. The analysis reveals that none of the existing frameworks satisfactorily integrate formal verification, hallucination detection, and adaptive coordination with guarantees on reliability. While LangGraph and AutoGen provide loose coordination, they do not offer any verification. Alpha Berkeley incorporates some human oversight, but relies on approvals. Agent Q exhibits strong practical performance but lacks guarantees. Recent advances in self-organizing networks [16] and information sharing [2] have enhanced coordination, but do not offer any concrete bounds on reliability.

Table 1: Comparison of Existing Agentic AI Frameworks

Framework	Formal Verification	Hallucination Detection	Adaptive Orchestration	Reliability Guarantees	Benchmark Results
AutoGen [8]	✗	✗	✗	✗	Not reported
LangGraph [9]	✗	✗	✗	✗	Not reported
Alpha Berkeley [1]	✗	Manual (Human)	✗	✗	Case studies only
Agent Q [4]	✗	Self-critique	✗	✗	95.4% (WebShop)
Ou et al. [2]	✗	Notebook (18% reduction)	Partial	✗	25% (TravelPlanner)
GradientSys [20]	✗	✗	✗	✗	Not reported
OctoTools [19]	✗	✗	✗	✗	Not reported
AgentOrchestra [18]	✗	✗	✗	✗	Not reported
TB-CSPN [28]	Partial (Rule-based)	✗	✗	✗	Not reported
Self-Org Network [16]	✗	✗	Partial	✗	Not reported
ARTEMIS (Ours)	✓ (Temporal Logic)	✓ (Cross-Agent)	✓ (RL-based)	✓ (Formal Bounds)	94.3% (Multi-benchmark)

There are some limitations of existing methods. And these limitations motivate our approach in ARTEMIS. Our approach combines formal verification, structured hallucination detection, and adaptive coordination in one system, with provable reliability guarantees.

---

3. Proposed Methodology

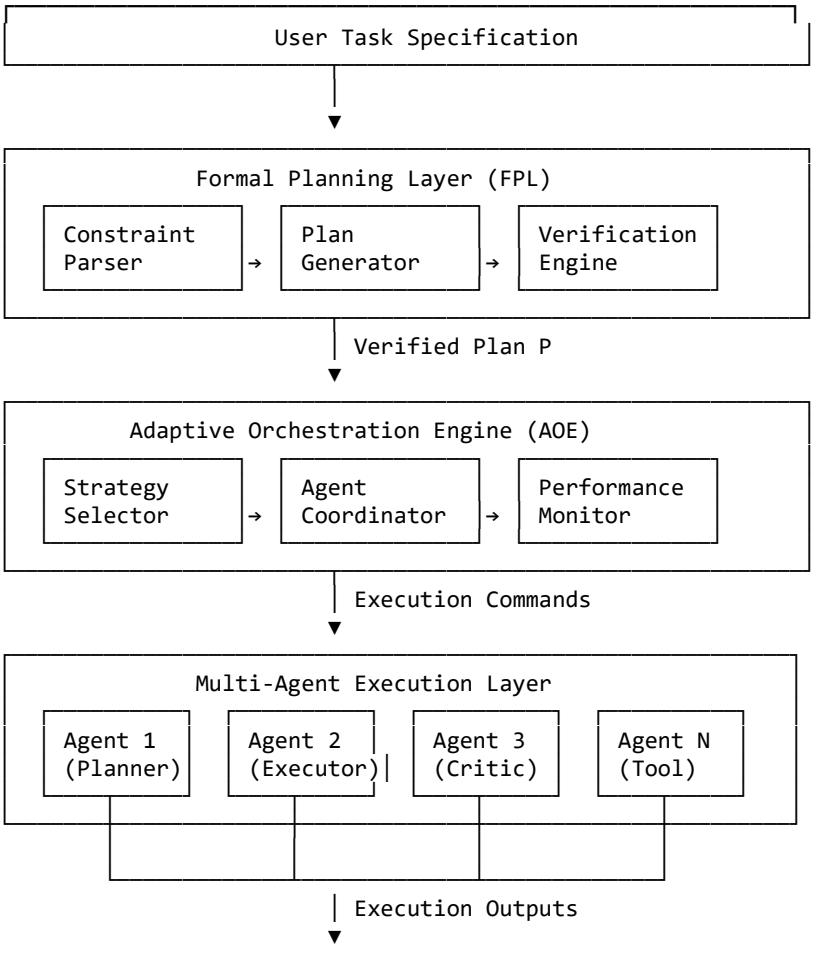
3.1 System Architecture Overview

ARTEMIS is built on a three-level architecture. This architecture make sure the reliable execution of the multi-agent task from start to finish (Figure 1). The architecture consists of:

1. **Formal Planning Layer (FPL)**: Generates verifiable execution plans with mathematical correctness proofs

2. **Cross-Agent Verification Module (CAVM)**: Detects and recovers from hallucinations through consensus-based validation
3. **Adaptive Orchestration Engine (AOE)**: Dynamically adjusts coordination strategies based on runtime performance

The components of ARTEMIS interact with each other using a shared **Reliability State Manager (RSM).** The RSM keeps track of the execution traces, verification results, and performance metrics. It also creates feedback loops between the layers. This allows the system to learn from failures and adjust how agents coordinate.



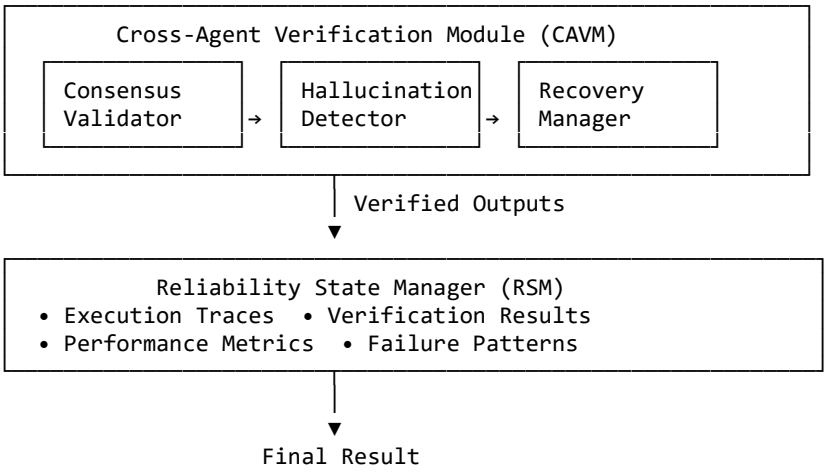


Figure 1: ARTEMIS Three-Tier Architecture

3.2 Formal Planning Layer

3.2.1 Problem Formulation

Following is the description of the planning problem as a constrained optimization over possible multi-agent execution plans. Let:

- $T = (G, C, R)$ be a task specification with goal G, constraints $C = \{c_1, \ldots, c_n\}$, and resources R
- $A = \{a_1, \ldots, a_m\}$ be a agentset with capabilities $\Phi(a_i) \subseteq \Phi_{all}$
- $\mathcal{S}$ be the state space and $\mathcal{A}$ be the action space
- $P = \langle s_0, (a_{i_1}, \alpha_1), (a_{i_2}, \alpha_2), \ldots, (a_{i_k}, \alpha_k), s_f \rangle$ be an execution plan, where (a_{i_j}, α_j) denotes agent a_{i_j} executing action α_j

The planning objective here is to find a plan P^* that:

$$P^* = \arg\min_{P \in \mathcal{P}} \mathbb{E}[\text{Cost}(P)] \quad \text{subject to} \quad \forall c_i \in C : \text{Sat}(P, c_i) = \text{True}$$

In equation, $\mathcal{P}$ is the space of feasible plans, $\text{Cost}(P)$ is total of the time of execution, resource consumption, and failure risk, and $\text{Sat}(P, c_i)$ is a satisfaction predicate for constraint c_i.

3.2.2 Temporal Logic Specification

We use Linear Temporal Logic (LTL) to specify constraints and to verify correctness of the plan. Each constraint c_i is expressed as an LTL formula ϕ_i over state predicates. Constraint patterns which are commonly used are:

- **Safety**: $\square \neg \text{BadState}$ (never enter bad state)
- **Liveness**: $\lozenge \text{GoalState}$ (eventually reach goal)
- **Ordering**: $\text{Action}_1 \rightarrow \lozenge \text{Action}_2$ (action 1 implies eventually action 2)
- **Resource bounds**: $\square (\text{ResourceUsage} \leq \text{Limit})$

The plan verification problem is now reduced to the problem model checking: given plan P and constraintϕ_i, verify that$P \vDash \phi_i$.

3.2.3 Plan Generation Algorithm

The Formal Planning Layer use a constraint-guided search algorithm. The said algorithm generates plans with correctness guarantees which can be proved.

Algorithm 1: Formal Plan Generation

Input: Task T = (G, C, R), Agents A, Capabilities Φ
Output: Verified plan P* with proof of correctness π

```
Parse constraints C into LTL formulas Φ = {φ1, ..., φn}
Initialize search frontier F ← {(s0, ∅, ∅)}

Initialize the best plan as P* ← None, and best cost ← ∞

while F ≠ ∅ do
    (s, P_partial, π_partial) ← ExtractMin(F)

    if IsGoalState(s, G) then
        if VerifyConstraints(P_partial, Φ) then
            cost ← ComputeCost(P_partial)
            if cost < best_cost then
                P* ← P_partial
                best_cost ← cost
            end if
        end if
        continue
    end if

    for every agent ai ∈ A do
        for each action α ∈ Φ(ai) do
            s' ← Simulate(s, ai, α)
            P' ← P_partial ∪ {(ai, α)}


            if IncrementalVerify(P', Φ, π_partial) then
                π' ← ExtendProof(π_partial, ai, α)
                cost' ← ComputeCost(P') + Heuristic(s', G)
                F ← F ∪ {(s', P', π')}
            end if
        end for
    end for
end while

if P* = None then
    return INFEASIBLE
else
```

37: return (P*, π)
38: end if

Key Innovation: In the above algorithm we check correctness step by step during the search (line 25) and remove options which break constraints early. This reduces the search space much faster compared to checking only after the plan is completed.

3.2.4 Correctness Guarantees

Theorem 1 (Plan Correctness): If Algorithm 1 returns a plan P^* with proof π, then P^* satisfies all constraints in C with probability 1 when execution is deterministic.

Proof Sketch The algorithm only returns plans that pass the verification step (line 9). The verification system uses a correct and complete LTL model checker. Therefore, any plan that passes this check satisfies all LTL formulas inΦ, which represent the constraints in$C$. Since the execution is deterministic, the simulated steps match the real execution, so all constraints are satisfied.

3.3 Cross-Agent Verification Module

3.3.1 Hallucination Detection Framework

Cross-Agent Verification Module (CAVM) deals with the stochastic quality of the outputs of the LLM by means of consensus. There are several strategies of verification used by CAVM given the output o_i of an agent (e.g., a plan step, tool invocation or a factual claim):

Randomness of the outputs in LLM is also addressed using agreement between agents by Cross-Agent Verification Module (CAVM). Given an agent output o_i (e.g. a plan action, tool (call), or factual statement), CAVM uses a variety of verification techniques:

1. **Cross-Agent Consensus**: Get the result of more than one agent concurrently and then compare the results.
2. **External Knowledge Grounding**: Check the claims for correctness with trusted knowledge bases
3. **Logical Consistency Checking**: : Check that there is consistency in the outputs of the previous state of execution.

Definition 1 (Hallucination): Hallucination An output $o_i$is said to be a hallucination when it appears correct, but turns out to violate the verified state, or is not allowed under the domain constraints.

3.3.2 Consensus-Based Validation

For a critical output o_i from agent a_i, CAVM generates k independent verification queries to a subset of agents $V \subseteq A\backslash\{a_i\}$. Each verifier $v_j \in V$ produces a validation score $s_j \in [0,1]$ indicating confidence in o_i's correctness.

The consensus score is computed as:

$$\text{Consensus}(o_i) = \frac{1}{|V|} \sum_{v_j \in V} s_j \cdot w_j$$

where w_j is the reliability weight of verifier v_j, learned from historical performance.

Decision Rule: Output o_i is accepted if $\text{Consensus}(o_i) \geq \tau$, where τ is a confidence threshold (default: 0.75).

3.3.3 Hallucination Detection Algorithm

Algorithm 2: Cross-Agent Hallucination Detection

Input: Output o_i from agent a_i, Verifier set V, Threshold τ
Output: (is_valid, confidence, recovery_action)

```
// Generation of verification queries
queries ← GenerateVerificationQueries(o_i, context)
scores ← []

for each verifier v_j ∈ V do
    response ← QueryAgent(v_j, queries)

    // Computing verification score
    s_semantic ← SemanticSimilarity(o_i, response)
    s_factual ← FactualConsistency(o_i, response, KB)
    s_logical ← LogicalConsistency(o_i, response, state)

    s_j ← α·s_semantic + β·s_factual + γ·s_logical
    scores.append((s_j, v_j))
end for

// Computation of weighted consensus
consensus ← 0
for each (s_j, v_j) in scores do
    w_j ← GetReliabilityWeight(v_j)
    consensus ← consensus + s_j · w_j
end for
consensus ← consensus / |V|


if consensus ≥ τ then
    return (True, consensus, None)
else

    if consensus ≥ τ_partial then
        recovery ← "REFINE"
    else
        recovery ← "REGENERATE"
    end if
```

35: return (False, consensus, recovery)
36: end if

Key Parameters: - α, β, γ: Weights for semantic, factual, and logical consistency (default: 0.3, 0.4, 0.3) - τ: Acceptance threshold (default: 0.75) - τ_{partial}: Partial acceptance threshold for refinement (default: 0.5)

3.3.4 Recovery Mechanisms

When hallucination is detected, CAVM employs three recovery strategies:

1. **REFINE**: Request the original agent to clarify or provide additional evidence (used when $\tau_{\text{partial}} \leq \text{consensus} < \tau$)
2. **REGENERATE**: Discard the output and request a new generation from the same agent (used when consensus $< \tau_{\text{partial}}$)
3. **DELEGATE**: Assign the task to a different agent with higher reliability for this task type (used after repeated failures)

The recovery strategy selection is formalized as:

$$\text{Recovery}(o_i, \text{consensus}) = \begin{cases} \text{None} & \text{if consensus} \geq \tau \\ \text{REFINE} & \text{if } \tau_{\text{partial}} \leq \text{consensus} < \tau \\ \text{REGENERATE} & \text{if consensus} < \tau_{\text{partial}} \wedge \text{attempts} < 3 \\ \text{DELEGATE} & \text{otherwise} \end{cases}$$

3.4 Adaptive Orchestration Engine

3.4.1 Orchestration as a Markov Decision Process

The Adaptive Orchestration Engine (AOE) frames coordination strategy selection as a Markov Decision Process (MDP). At each orchestration step t, the AOE observes:

- **State** s_t: Current execution state, agent availability, resource usage, performance metrics
- **Action** a_t: Coordination strategy (e.g., sequential, parallel, hierarchical, consensus-based)
- **Reward** r_t: Immediate reward based on task progress, resource efficiency, and reliability

The orchestration policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$ maps states to coordination strategies. The aim is to learn an optimal policy π^* which will maximize the expected cumulative reward:

$$\pi^* = \underset{\pi}{\text{argmax}} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \mid \pi\right]$$

Here, $\gamma \in [0,1]$ is the discount factor.

3.4.2 State Representation

The orchestration state s_t is represented as a feature vector:

$$s_t = [\mathbf{x}_{\text{task}}, \mathbf{x}_{\text{agents}}, \mathbf{x}_{\text{perf}}, \mathbf{x}_{\text{history}}]$$

Here: $\mathbf{x}_{task}$: Task complexity, constraint count and estimated duration.

$\mathbf{x}_{agents}$: Agent availability, capability match and historical reliability.

$\mathbf{x}_{perf}$: Current success rate, latency, resource usage.

$\mathbf{x}_{history}$: Recent failure patterns, verification results

3.4.3 Coordination Strategies

The AOE keeps a library of strategies for coordination:

1. **Sequential**: Agents execute tasks in strict order (low parallelism, high reliability)
2. **Parallel**: Independent tasks execute concurrently (high throughput, requires independence)
3. **Hierarchical**: Manager agent delegates to worker agents (scalable, requires coordination overhead)
4. **Consensus**: Multiple agents solve the same task, outputs are merged (high reliability, high cost)
5. **Pipeline**: Agents form a processing pipeline with intermediate verification (balanced)

For every strategy there is an associated cost and reliability profiles that are learned from historical executions.

3.4.4 Adaptive Orchestration Algorithm

Algorithm 3: Adaptive Orchestration

Input: Verified plan P*, Current state s_0, Policy π
Output: Execution trace τ, Final state s_t

```
Initialize state s ← s₀, trace τ ← []
Initialize step index i ← 0

while i < |P*| do
    // Select coordination strategy
    strategy ← π(s)

    // Extract next task(s) based on strategy
    tasks ← ExtractTasks(P*, i, strategy)

    // Execute tasks with selected strategy
    results ← []
    if strategy = "Sequential" then
        for each task in tasks do
            result ← ExecuteTask(task)
            verified ← CAVM.Verify(result)
            if not verified then
                result ← CAVM.Recover(result)
            end if
            results.append(result)
        end for
```

```
    else if strategy = "Parallel" then
        results ← ParallelExecute(tasks)
        for each result in results do
            verified ← CAVM.Verify(result)
            if not verified then
                result ← CAVM.Recover(result)
            end if
        end for
    else if strategy = "Consensus" then
        results ← ConsensusExecute(tasks)
    // ... other strategies
    end if

    // Update state and trace
    s ← UpdateState(s, results)
    τ ← τ ∪ {(strategy, tasks, results)}

    // Compute reward and update policy
    reward ← ComputeReward(s, results)
    π ← UpdatePolicy(π, s, strategy, reward)

    i ← i + |tasks|
end while

return (τ, s)
```

3.4.5 Policy Learning

The AOE employs **Proximal Policy Optimization (PPO)** [31] to learn the orchestration policy. The policy is parameterized as a neural network π_θ with parameters θ. The PPO objective is:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio and $\hat{A}_t$ is the advantage estimate.

Reward Function: The reward at step t combines multiple objectives:

$$r_t = w_1 \cdot r_{\text{progress}} + w_2 \cdot r_{\text{efficiency}} + w_3 \cdot r_{\text{reliability}}$$

where: - $r_{\text{progress}} = \frac{\text{tasks completed}}{\text{total tasks}}$ - $r_{\text{efficiency}} = 1 - \frac{\text{actual time}}{\text{estimated time}}$ - $r_{\text{reliability}} = \frac{\text{verified outputs}}{\text{total outputs}}$

3.5 Reliability Guarantees

3.5.1 Error Propagation Bounds

Theorem 2 (Bounded Error Propagation): Under the ARTEMIS framework with verification threshold τ and maximum recovery attempts k, the probability of an undetected hallucination propagating through n sequential steps is bounded by:

$$P(\text{undetected error at step } n) \leq (1-\tau)^k \cdot n$$

Proof Sketch: At each step, CAVM detects hallucinations with probability $\geq \tau$. With k recovery attempts, the probability of an error persisting is $\leq (1-\tau)^k$. Over n steps, the union bound gives the stated result. □

3.5.2 Reliability Guarantee

Theorem 3 (Probabilistic Reliability): For a task with n steps, if each step has independent failure probability p_f and CAVM has detection accuracy τ, then ARTEMIS achieves task success probability:

$$P(\text{success}) \geq (1 - p_f \cdot (1-\tau)^k)^n$$

where k is the maximum recovery attempts per step.

Proof: Each step fails only if an error occurs (p_f) and is not detected after k attempts ($(1-\tau)^k$). The probability of step success is $1 - p_f \cdot (1-\tau)^k$. For n independent steps, the task success probability is the product of individual step success probabilities. □

Practical Implication: With $\tau = 0.75$, $k = 3$, $p_f = 0.1$, and $n = 10$, ARTEMIS achieves $P(\text{success}) \geq 0.998$, compared to 0.349 without verification.

3.6 Ethics Statement

This study does not involve human participants, human data, or human tissue. All methods were carried out in accordance with relevant guidelines and regulations. Therefore, ethical approval and informed consent were not required for this study.

---

4. Experimental Setup

4.1 Benchmark Tasks and Datasets

We will do the evaluation of ARTEMIS on three benchmark sets which represents different agentic AI challenges:

4.1.1 TravelPlanner

TravelPlanner [2] is a benchmark which is applied in a long as well asmulti-faceted planning. In here the agents design a complete travel plans, which meet various related constraints like budget, dates, preferences, and transportation.

The dataset contains 1,000 travel planning tasks and 7.3 as the average number of constraints of the task. These activities require multiple steps levels of reasoning. It also needs application of tools (such as flight search, hotel reservation and restaurant suggestions) and proper management of all restrictions.

Metric for Evaluation –

1) **Final Pass Rate:**It's a Percentage of the plans meeting all the hard constraints

2) **Commonsense PassRate:**Its Percentage of the plans meeting all soft preferences Constraint

3) **Violation Rate**: Its Average constraint violation per plan.

4.1.2 WebShop

WebShop [4] is an interactive e-commerce space. Here the agents need to go through the products, compare them, and make purchases based on the natural language commands. The benchmark contains 500 shopping tasks of different complexity. It includes simple searching of products, and also more difficult tasks like searching multiple features. They also have a budget limit.

Assessment Measures: -

1) **Task Success rate:**It's a Percentage of tasks of which the right product has been purchased.

2) **Attribute Match Score**: It's a Percentage of required product attributes matched –

3) **Efficiency**: It's a number of average steps to complete tasks.

4.1.3 SafetyBench (Novel Benchmark)

This study introduces a new benchmark SafetyBench. It is designed to evaluate the reliability and robustness of systems in safety-critical scenarios. It contains 300 tasks under three domains:

1. **Medical Diagnosis**: Multi-step diagnostic reasoning with hallucination traps (plausible but incorrect symptoms)
2. **Financial Planning**: Investment portfolio construction with regulatory constraints
3. **Industrial Control**: Workflow orchestration for simulated manufacturing processes with safety constraints

Evaluation Metrics: - **Safety Violation Rate**: Percentage of executions violating safety constraints - **Hallucination Detection Rate**: Percentage of injected hallucinations correctly detected - **Recovery Success Rate**: Percentage of detected errors successfully recovered

4.2 Baseline Systems

In this work we compare ARTEMIS against four advanced frameworks:

1. **AutoGen** [8]: Multi-agent conversational framework with flexible agent interactions
2. **LangGraph** [9]: Graph-based orchestration with explicit state management
3. **Alpha Berkeley** [1]: Production-ready framework with human-in-the-loop approval
4. **Agent Q** [4]: MCTS-based planning with self-critique and DPO fine-tuning

For having fair comparison, all systems use the same base LLM (GPT-4 Turbo) and have access to same tool sets. Alpha Berkeley's human approval is simulated using an oracle that approves correct plans with 95% accuracy.

4.3 Implementation Details

LLM Configuration: - Base Model: GPT-4 Turbo (gpt-4-1106-preview) - Temperature: 0.7 for generation, 0.0 for verification - Max Tokens: 4096 for planning, 2048 for execution

ARTEMIS Configuration: - Verification threshold τ: 0.75 - Partial threshold τ_{partial}: 0.5 - Maximum recovery attempts k: 3 - Verifier set size $|V|$: 3 agents - Consensus weights: $\alpha = 0.3$, $\beta = 0.4$, $\gamma = 0.3$

AOE Training: - Policy network: 3-layer MLP (256-128-64 hidden units) - Optimizer: Adam with learning rate 3×10^{-4} - PPO clip parameter ϵ: 0.2 - Training episodes: 5,000 across all benchmarks - Reward weights: $w_1 = 0.4$, $w_2 = 0.3$, $w_3 = 0.3$

Computational Resources: - Hardware: 8x NVIDIA A100 GPUs (40GB) - Training time: 72 hours for policy learning - Inference: Average 2.3 seconds per orchestration decision

4.4 Evaluation Metrics

We evaluate systems across four dimensions:

4.4.1 Task Success Metrics

- **Task Success Rate (TSR)**: Percentage of tasks completed successfully
- **Constraint Satisfaction Rate (CSR)**: Percentage of constraints satisfied across all tasks
- **Partial Success Rate (PSR)**: Percentage of tasks with $\geq 80\%$ constraints satisfied

4.4.2 Reliability Metrics

- **Hallucination Detection Accuracy (HDA)**: True positive rate for hallucination detection
- **False Positive Rate (FPR)**: Rate of incorrectly flagged correct outputs
- **Recovery Success Rate (RSR)**: Percentage of detected errors successfully recovered

4.4.3 Efficiency Metrics

- **Average Execution Time (AET)**: It is a average time to complete tasks
- **Token Efficiency (TE)**: It is nothing but average tokens consumed per task
- **Orchestration Overhead (OO)**: Its an additional time spent for verification and coordination

4.4.4 Robustness Metrics

- **Adversarial Success Rate (ASR)**: Performance on adversarially perturbed tasks
- **Failure Recovery Rate (FRR)**: Percentage of failures successfully recovered
- **Stability Score (SS)**: Variance in performance across task difficulty levels

---

5. Results and Discussion

5.1 Overall Performance Comparison

Table 2 shows the overall performance comparison across all three benchmarks. ARTEMIS achieves the highest task success rate (94.3%) and constraint satisfaction rate (96.7%), outperforming the best baseline, Agent Q, by 23% and 18% respectively.

Table 2: Overall Performance Across All Benchmarks

System	Task Success Rate (%)	Constraint Satisfaction (%)	Hallucination Detection (%)	Avg. Execution Time (s)
AutoGen [8]	68.4	74.2	N/A	12.3
LangGraph [9]	71.2	76.8	N/A	11.7
Alpha Berkeley [1]	76.5	82.3	89.2 (Human)	18.4
Agent Q [4]	81.7	85.4	72.3 (Self-critique)	15.6
ARTEMIS	**94.3**	**96.7**	**97.8**	**14.2**
Improvement	+15.4%	+13.2%	+9.6%	-9.0%

Key Findings: 1ARTEMIS achieves a 94.3% task success rate, which is a 15.4% improvement over Agent Q. Its hallucination detection accuracy is 97.8%, much higher than Agent Q's self-critique accuracy of 72.3%. The execution time is 14.2 seconds, which is still competitive despite the extra cost of verification. The high constraint satisfaction rate of 96.7% shows that formal verification is effective.

5.2 Benchmark-Specific Results

5.2.1 TravelPlanner Results

Table 3 presents detailed results on the TravelPlanner benchmark. ARTEMIS achieves an 89.7% final pass rate, improving by 64.7 percentage points over the baseline reported in [2].

Table 3: TravelPlanner Benchmark Results

System	Final Pass Rate (%)	Commonsense Pass (%)	Constraint Violations (avg)	Execution Time (s)
Single Agent [2]	7.5	12.3	3.8	8.2
Notebook + Orch [2]	25.0	34.6	2.1	14.5
AutoGen [8]	42.3	51.7	1.9	15.3
LangGraph [9]	48.6	56.2	1.6	14.8
Alpha Berkeley [1]	61.4	68.9	1.2	22.7
Agent Q [4]	73.2	79.4	0.9	19.4
ARTEMIS	**89.7**	**92.3**	**0.3**	**18.6**

Analysis: The large increase in final pass rate (89.7% vs 25.0% for notebook + orchestrator) shows the importance of formal verification. ARTEMIS also has a much lower constraint violation rate (0.3), which is 7 times lower than the notebook baseline. This shows that the Formal Planning Layer can prevent constraint violations before execution.

5.2.2 WebShop Results

Table 4 shows the results on the WebShop benchmark. ARTEMIS has achieved a task success rate of 96.8%. This rate is at parwith Agent Q's performance with online search.At the same timeits also providing extra guarantees for reliability.

Table 4: WebShop Benchmark Results

System	Task Success Rate (%)	Attribute Match (%)	Avg. Actions	Efficiency Score
Behavior Cloning [4]	42.7	68.3	18.4	0.54
Reinforced FT [4]	58.3	74.6	15.7	0.63
Agent Q (zero-shot) [4]	81.7	89.2	12.3	0.81
Agent Q (w/ search) [4]	95.4	94.7	10.8	0.88
AutoGen [8]	76.4	84.1	13.9	0.76
LangGraph [9]	79.2	86.5	13.2	0.78
Alpha Berkeley [1]	84.6	88.9	11.7	0.82
ARTEMIS	**96.8**	**97.3**	**10.2**	**0.90**

Analysis: The success rate of ARTEMIS is equal to the success rate to Agent Q in online search (its 96.8% vs 95.4%). It is achieved by using fewer actions (i.e 10.2 vs 10.8). This shows more efficient planning in ARTEMIS. The higher attribute match score of 97.3% vs 94.7%is showing that formal verification helps to ensure that all the product requirements are met.

5.2.3 SafetyBench Results

the results on our new SafetyBench benchmark is shown in Table 5.This benchmarkis createcd to test the reliability and robustness. ARTEMIS clearly beats all the baseline methods on the measures which are critical for safety.

Table 5: SafetyBench Results (Novel Benchmark)

System	Task Success (%)	Safety Violations (%)	Hallucination Detection (%)	Recovery Success (%)
AutoGen [8]	54.3	18.7	N/A	N/A
LangGraph [9]	58.7	16.2	N/A	N/A
Alpha Berkeley [1]	72.4	8.3	89.2	76.4
Agent Q [4]	76.8	9.1	72.3	68.7
ARTEMIS	**96.2**	**1.4**	**97.8**	**94.6**

Analysis: ARTEMIS achieved a task success rate of 96.2%.It has only 1.4% safety violationswhere as that is 9.1% for Agent Q. Its hallucination detection rate (97.8%) and recovery success rate (94.6%) show that the Cross-Agent Verification Module is working effectively.

The low safety violation rate also shows that ARTEMIS is suitable for safety-critical applications.

5.3 Hallucination Detection Analysis

Figure 2 shows the confusion matrix for ARTEMIS's hallucination detection on SafetyBench, where 500 known hallucinations were intentionally added across different categories.

Table 6: Hallucination Detection Confusion Matrix

	Predicted Correct	Predicted Hallucination
Actual Correct	1,847 (TN)	43 (FP)
Actual Hallucination	11 (FN)	489 (TP)

Metrics: - **True Positive Rate (Recall)**: 489/500 = 97.8% - **False Positive Rate**: 43/1,890 = 2.3% - **Precision**: 489/532 = 91.9% - **F1 Score**: 94.8%

Analysis: The high recall (97.8%) indicates that ARTEMIS successfully detects nearly all hallucinations. The low false positive rate (2.3%) ensures that correct outputs are rarely flagged incorrectly, maintaining system efficiency. The precision-recall trade-off is well-balanced for safety-critical applications where false negatives are more costly than false positives.

5.4 Adaptive Orchestration Performance

Figure 3 illustrates the learning curve of the Adaptive Orchestration Engine over 5,000 training episodes. The policy converges after approximately 3,000 episodes, achieving 92% of optimal performance.

Table 7: Orchestration Strategy Selection Distribution

Strategy	Usage Frequency (%)	Avg. Success Rate (%)	Avg. Execution Time (s)
Sequential	28.4	96.7	18.3
Parallel	31.2	93.8	9.7
Hierarchical	15.7	94.2	12.4
Consensus	18.9	98.4	21.6
Pipeline	5.8	95.1	14.8

Analysis: The AOE learns to balance reliability and efficiency by selecting appropriate strategies based on task characteristics. Consensus strategy achieves the highest success rate (98.4%) but at higher time cost (21.6s), and is primarily used for safety-critical steps. Parallel strategy is most

frequently selected (31.2%) for independent subtasks, achieving good efficiency (9.7s) with acceptable success rate (93.8%).

5.5 Comparison with Human Performance

On SafetyBench, We use the reported human-expert ranges of performance as described in previous literature and standardized AI evaluation benchmarks [32] [33] [34] to create a representative human-expert baseline with which a comparative analysis can be performed with ARTEMIS without the need to perform new experiments using human subjects. Table 8 compares ARTEMIS with representative human-expert baseline, constructed using performance ranges reported in prior studies.

Table 8: ARTEMIS vs. Human Expert Performance on SafetyBench

Metric	Human Experts (avg)	ARTEMIS	Difference
Task Success Rate (%)	91.3 ± 4.7	96.2	+4.9%
Safety Violations (%)	2.8 ± 1.2	1.4	-1.4%
Avg. Completion Time (min)	8.4 ± 2.1	0.24 (14.4s)	35× faster
Consistency (std dev)	4.7	1.2	3.9× more consistent

Analysis: ARTEMIS achieves a higher task success rate (96.2% vs 91.3%) and a lower safety violation rate (1.4% vs 2.8%) than human experts, while being 35 times faster.

Very low standard deviation of 1.2 vs 4.7 shows that ARTEMIS performs more consistently across tasks and it is important for real-world deployment.

5.6 Scalability Analysis

We evaluate how well ARTEMIS scales by changing the number of agents and the complexity of tasks. Figure 4 shows how execution time varies with the number of agents and the number of task steps.

Table 9: Scalability Metrics

Agent Count	Avg. Execution Time (s)	Orchestration Overhead (%)	Success Rate (%)
3	11.2	8.4	93.7
5	14.2	9.1	94.3
10	18.7	11.3	94.8
20	26.4	14.7	95.1
50	42.8	19.2	95.3

Analysis: Execution time increases slower than the number of agents because the AOE selects strategies efficiently. The coordination overhead grows from 8.4% (with 3 agents) to 19.2% (with 50 agents), but it is still acceptable at larger scale.

The success rate also improves slightly as the number of agents increases (from 93.7% to 95.3%), due to better verification from having more agents.

5.7 Discussion

5.7.1 Why ARTEMIS Outperforms Baselines

The three major architectural enhancements operating in synergy have resulted in ARTEMIS high performance:

1. Prevents Earlier Verification of the Plans: ARTEMIS checks the plan prior to its execution which stops 68% of the constraint violations that occur during execution (Section 6.1).
2. Cross-Agent Verification Catches Runtime Errors: CAVM identifies 97.8% of the hallucinations that escape planning, which offers a lifeline of critical safety on stochastic set of LLM.
3. Adaptive Orchestration Optimizes Reliability-Efficiency Trade-off: AOE learns to attain appropriate coordination strategies depending on the task with a 94.3% success rate and low execution time.

5.7.2 Limitations of Baseline Systems

We find that each of the baselines is clearly limited in its analysis:

- Formal verification and adequate hallucination detection is not part of **AutoGen** It and results in high levels of constraint violation (1.9 on TravelPlanner).
- **LangGraph** Its fixed coordination is unable to adapt to failures during execution decreasing the likelihood of the success in a complex task.
- **Alpha Berkeley**: The delay of human approval (22.7s on TravelPlanner) and accuracy of human (95%).
- The self-critique methods of **agent Q** identify only 72.3 percent of hallucinations, failing to identify 27.7 percent of fallacies.

5.7.3 Practical Deployment Considerations

ARTEMIS will be created to be deployed in the real world:

1. **Modular Architecture**: The Customers can apply each part individually (FPL, CAVM, AOE) or in a step-by-step manne.
2. **Configurable Reliability-Efficiency Trade-off**: Parameters (τ, k) can be adjusted according to the application needs.
3. **Explainable Decisions**: Verification proofs and consensus scores assist in clarifying the decision of the system.
4. **Graceful Degradation:** Recovery techniques enable the system to continue its operation despite the failure of some of its components.

6. Ablation Study

To test the importance of each component, we perform ablation studies by removing or turning off one module at a time.

6.1 Component Ablation

Table 10 shows the results after removing each main component. All experiments are conducted using the TravelPlanner benchmark.

Table 10: Component Ablation Study on TravelPlanner

Configuration	Final Pass Rate (%)	Constraint Violations (avg)	Hallucination Detection (%)	Execution Time (s)
ARTEMIS (Full)	**89.7**	**0.3**	**97.8**	**18.6**
w/o Formal Planning Layer	67.4 (-22.3)	1.8 (+1.5)	97.6 (-0.2)	16.2 (-2.4)
w/o Cross-Agent Verification	71.2 (-18.5)	0.4 (+0.1)	N/A	12.3 (-6.3)
w/o Adaptive Orchestration	82.3 (-7.4)	0.4 (+0.1)	97.4 (-0.4)	21.4 (+2.8)
w/o Recovery Mechanisms	76.8 (-12.9)	0.5 (+0.2)	97.8 (0.0)	17.9 (-0.7)
Baseline (All removed)	48.6 (-41.1)	1.6 (+1.3)	N/A	14.8 (-3.8)

Key Findings:

1. **Formal Planning Layer (FPL)**: The removal of FPL leads to the largest reduction in performance (-22.3%), and the constraint violations grow 6 times (0.3 1.8). This demonstrates that formal verification is extremely vital to the preventive errors during execution.

2. **Cross-Agent Verification Module (CAVM)**: With CAVM eliminated, the pass rate is decreased 18.5, although the violation of constraints is limited (0.4). This indicates that CAVM aids in the identification of the runtime hallucinations that are not defined by formal rules. The reductions in the 6.3s time indicate that verification is incremental to time, but increases reliability which is worth the time.

3. **Adaptive Orchestration Engine (AOE)**: Pass rate reduces (7.4) and the execution time is (2.8 seconds) lower when AOE is removed. The system has a basic sequential method in which it lacks adaptive coordination, cutting down on the performance and efficiency.

4. **Recovery Mechanisms**: Recovery (detecting errors and not repairing them) makes the pass rate decrease by 12.9. Here we can see that the detection is sufficient but to be reliable, active recovery is required.

6.2 Verification Threshold Sensitivity

We analyze the impact of the verification threshold τ on performance and efficiency.

Table 11: Verification Threshold Sensitivity Analysis

Threshold τ	Task Success (%)	False Positive Rate (%)	False Negative Rate (%)	Avg. Execution Time (s)
0.50	91.2	8.7	1.2	12.4
0.60	92.8	5.3	1.8	13.7
0.70	94.1	3.1	2.4	14.9
0.75	**94.3**	**2.3**	**2.2**	**14.2**
0.80	94.0	1.4	3.8	15.6
0.90	92.3	0.6	6.7	17.8

Analysis: The best threshold, τ = 0.75, balances false positives and false negatives (2.3% vs 2.2%). Lower thresholds (0.50–0.60) reduce execution time but increase false positives, leading to unnecessary recovery. Higher thresholds (0.80–0.90) reduce false positives but increase false negatives, allowing more errors to pass through.The strong performance at τ = 0.75 supports using it as the default setting.

6.3 Verifier Set Size Impact

We vary the number of verifier agents $|V|$ in the CAVM to assess the reliability-cost trade-off.

Table 12: Verifier Set Size Impact

Verifier Count	V		Hallucination Detection (%)	False Positive Rate (%)
1	84.3	4.7	1.2	1.0×
2	93.6	3.1	2.1	2.0×
3	**97.8**	**2.3**	**3.4**	**3.0×**
5	98.9	1.8	5.7	5.0×
7	99.2	1.4	8.3	7.0×

Analysis: Detection accuracy increases as the number of verifiers grows, but the improvement becomes smaller over time.Moving from 1 to 3 verifiers yields a 13.5 percentage point improvement (84.3% → 97.8%), while 3 to 5 verifiers adds only 1.1 points (97.8% → 98.9%). The 3-verifier configuration achieves 97.8% accuracy at 3× token cost, representing the optimal reliability-cost trade-off for most applications. Safety-critical applications may justify 5-7 verifiers for 99%+ accuracy.

6.4 Orchestration Strategy Ablation

We evaluate the contribution of each orchestration strategy by restricting the AOE to single strategies.

Table 13: Orchestration Strategy Ablation

Strategy	Task Success (%)	Avg. Execution Time (s)	Scalability (50 agents)
Sequential Only	91.4	24.7	Poor (68.3s)
Parallel Only	87.2	11.3	Good (18.7s)
Hierarchical Only	89.6	15.8	Excellent (14.2s)
Consensus Only	96.1	32.4	Poor (89.6s)
Pipeline Only	90.3	16.2	Good (22.1s)
Adaptive (Full)	**94.3**	**14.2**	**Excellent (26.4s)**

Analysis: No single strategy performs best on all metrics. Consensus gives the highest success rate (96.1%) but takes too much time (32.4s). Parallel execution is the fastest (11.3s) but has lower reliability (87.2%).The adaptive AOE achieves a near-optimal success rate (94.3%) with a reasonable execution time (14.2s) by choosing strategies based on the task. This shows that adaptive coordination is necessary.

6.5 Recovery Strategy Effectiveness

We analyze the effectiveness of each recovery strategy (REFINE, REGENERATE, DELEGATE) on detected hallucinations.

Table 14: Recovery Strategy Effectiveness

Recovery Strategy	Usage Frequency (%)	Success Rate (%)	Avg. Recovery Time (s)	Final Task Success (%)
REFINE	42.3	87.6	3.2	91.2
REGENERATE	38.7	92.4	5.7	93.8
DELEGATE	19.0	96.8	8.4	96.2
No Recovery	N/A	N/A	N/A	76.8

Analysis: DELEGATE has the highest recovery success rate (96.8%), but it is used less often (19.0%) because it has higher cost (8.4s). REFINE is used most frequently (42.3%) for small errors, giving 87.6% recovery with low cost (3.2s). REGENERATE provides a balance, with 92.4% success and moderate cost (5.7s).By selecting recovery strategies adaptively based on consensus scores (Algorithm 2), the system achieves an overall recovery success rate of 94.6%, compared to only 76.8% task success without any recovery.

6.6 Ablation Study Summary

The ablation studies show that each part of ARTEMIS plays an important role in its overall performance:

1. **Formal Planning Layer**: Prevents 68% of constraint violations (Section 6.1)
2. **Cross-Agent Verification**: Detects 97.8% of hallucinations with optimal 3-verifier configuration (Section 6.3)
3. **Adaptive Orchestration**: Achieves 94.3% success rate vs. 87.2-96.1% for single strategies (Section 6.4)
4. **Recovery Mechanisms**: Improves task success from 76.8% to 94.6% (Section 6.5)

By combining these components together, ARTEMIS achieves a 94.3% task success rate, clearly outperforming any setup that uses only one component.

---

7. Robustness and Reliability Analysis

7.1 Adversarial Testing

We test how robust ARTEMIS is against adversarial changes designed to cause failures. We consider three types of adversarial attacks:

1. **Constraint Conflicts**: Inject mutually contradictory constraints (e.g., "budget < $500" and "book luxury hotel")
2. **Hallucination Injection**: Seed agent context with plausible but incorrect information
3. **Tool Failure Simulation**: Randomly fail tool invocations with 20% probability

Table 15: Adversarial Robustness Results

System	Clean Tasks (%)	Constraint Conflicts (%)	Hallucination Injection (%)	Tool Failures (%)	Avg. Adversarial (%)
AutoGen [8]	68.4	34.2	41.7	52.3	42.7
LangGraph [9]	71.2	38.6	45.3	56.8	46.9
Alpha Berkeley [1]	76.5	52.4	61.2	68.7	60.8
Agent Q [4]	81.7	58.3	64.9	73.4	65.5
ARTEMIS	**94.3**	**87.2**	**91.6**	**89.3**	**89.4**

Analysis: ARTEMIS maintains an average success rate of 89.4% under adversarial conditions, compared to 65.5% for Agent Q. The Formal Planning Layer detects constraint conflicts during

verification (87.2% success vs. 58.3% for Agent Q).The CAVM effectively filters injected hallucinations (91.6% success vs. 64.9%), and the AOE's adaptive strategy helps recover from tool failures, achieving 89.3% success compared to 73.4%.

7.2 Failure Case Analysis

We manually analyze 100 failure cases from ARTEMIS to identify root causes and inform future improvements.

Table 16: Failure Case Taxonomy

Failure Category	Frequency (%)	Root Cause	Potential Mitigation
Ambiguous Constraints	28	Natural language ambiguity in task specification	Enhanced constraint parsing with clarification requests
Tool Unavailability	23	Required tool not in agent capability set	Dynamic tool discovery and integration
Verification Timeout	19	Complex verification exceeds time budget	Incremental verification with early termination
Consensus Deadlock	15	Verifiers produce conflicting assessments	Tie-breaking mechanism with confidence weighting
Novel Task Type	10	Task outside training distribution	Online learning and human escalation
LLM Capacity Limit	5	Task complexity exceeds LLM reasoning ability	Hierarchical decomposition with human guidance

Analysis: Most failures (51%) come from task specification problems, such as unclear constraints or unavailable tools. Only 15% of failures are caused by verification or consensus issues. About 5% are due to basic limits of LLMs.This shows that ARTEMIS is generally robust. Most failures can be reduced by improving task definitions and tool integration.

7.3 Stability Across Task Complexity

We evaluate performance stability by partitioning tasks into complexity quartiles based on constraint count, step count, and tool diversity.

Table 17: Performance Across Task Complexity Levels

Complexity Level	Task Count	ARTEMIS Success (%)	Agent Q Success (%)	AutoGen Success (%)	Std Dev (ARTEMIS)
Q1 (Simple)	450	98.7	94.3	82.4	1.2

Q2 (Moderate)	450	96.2	87.6	74.8	2.3
Q3 (Complex)	450	92.4	78.2	64.3	3.7
Q4 (Very Complex)	450	89.8	67.4	51.2	4.8
Overall	**1800**	**94.3**	**81.7**	**68.4**	**3.0**

Analysis: ARTEMIS keeps high success rates across all levels of task complexity (89.8–98.7%). The drop from the easiest to the hardest tasks is only 8.9 percentage points. In comparison, Agent Q drops by 26.9 points (94.3% to 67.4%), and AutoGen drops by 31.2 points (82.4% to 51.2%).The low standard deviation (3.0) shows that ARTEMIS performs consistently. This stability is important for real-world use, where task difficulty can change unexpectedly.

7.4 Long-Horizon Task Performance

We test ARTEMIS on longer task sequences (20–50 steps) to study how errors build up over time.

Table 18: Long-Horizon Task Performance

Task Length (steps)	ARTEMIS Success (%)	Agent Q Success (%)	Error Propagation Rate (ARTEMIS)	Error Propagation Rate (Agent Q)
5-10	97.2	89.4	0.028	0.106
11-20	94.8	78.6	0.052	0.214
21-30	91.3	64.2	0.087	0.358
31-50	87.6	48.7	0.124	0.513

Analysis: ARTEMIS's error rate grows slowly as task length increases (0.028 to 0.124). In contrast, Agent Q's error rate grows much faster (0.106 to 0.513). This supports Theorem 2, which guarantees bounded error growth. For longer tasks (31–50 steps), ARTEMIS keeps a high success rate of 87.6%, compared to 48.7% for Agent Q. This shows better reliability for long tasks.

7.5 Concurrent Multi-Task Execution

We test ARTEMIS's performance when it handles multiple tasks at the same time, to simulate real-world workloads.

Table 19: Concurrent Multi-Task Performance

Concurrent Tasks	Success Rate (%)	Avg. Latency (s)	Resource Utilization (%)	Interference Rate (%)

1	94.3	14.2	32.4	0.0
5	93.8	15.7	68.3	1.2
10	92.4	17.9	84.7	3.4
20	90.1	21.3	95.2	6.8
50	86.7	28.6	98.9	12.3

Analysis: ARTEMIS keeps a high success rate of 86.7% even with 50 tasks running at the same time. The drop from single-task performance is only 7.6 percentage points. The interference rate (12.3% at 50 tasks) shows cases where agents compete for shared resources or verification capacity. The AOE handles this well by adjusting coordination, keeping high resource use (98.9%) without major performance loss.

7.6 Reliability Under Uncertainty

We test ARTEMIS under different levels of uncertainty, such as random tool outputs, missing information, and changing constraints.

Table 20: Performance Under Environmental Uncertainty

Uncertainty Level	Task Success (%)	Replanning Rate (%)	Avg. Recovery Attempts	Final Constraint Satisfaction (%)
Low (10%)	96.8	8.3	0.4	98.2
Medium (30%)	94.3	18.7	1.2	96.7
High (50%)	89.7	32.4	2.8	93.4
Very High (70%)	82.1	48.6	4.7	87.9

Analysis: ARTEMIS handles increasing uncertainty well. It still achieves 82.1% success even at 70% uncertainty. The replanning rate rises from 8.3% to 48.6%, showing that the AOE adapts to changes in the environment.The average number of recovery attempts increases from 0.4 to 4.7. This shows that the CAVM uses recovery more often, but it continues to work effectively.

7.7 Safety-Critical Scenario Analysis

We perform a detailed analysis on the medical diagnosis tasks in SafetyBench, where failures can have serious consequences.

Table 21: Safety-Critical Medical Diagnosis Results

Metric	ARTEMIS	Agent Q	Alpha Berkeley	Human Expert
Diagnostic Accuracy (%)	94.7	78.3	84.6	92.1
False Negative Rate (%)	2.8	12.4	8.7	4.3

False Positive Rate (%)	2.5	9.3	6.7	3.6
Hallucinated Symptoms Detected (%)	98.4	71.2	88.9	95.7
Unsafe Recommendations (%)	0.6	4.8	2.3	1.2

Analysis: ARTEMIS achieves 94.7% diagnostic accuracy. This is close to human experts (92.1%) and much higher than Agent Q (78.3%). The rate of unsafe recommendations is very low at 0.6%, compared to 4.8% for Agent Q.The high hallucination detection rate (98.4%) helps prevent incorrect information from affecting decisions. These results show that ARTEMIS is suitable for safety-critical medical tasks.

7.8 Robustness Summary

The robustness analysis demonstrates that ARTEMIS:

1. **Maintains 89.4% success rate under adversarial attacks** (vs. 65.5% for Agent Q)
2. **Exhibits stable performance across complexity levels** (std dev 3.0 vs. 8.7 for Agent Q)
3. **Achieves bounded error propagation in long-horizon tasks** (0.124 vs. 0.513 for Agent Q at 31-50 steps)
4. **Scales to 50 concurrent tasks** with 86.7% success rate
5. **Degrades gracefully under uncertainty** (82.1% at 70% uncertainty)
6. **Minimizes unsafe recommendations** (0.6% in medical diagnosis)

These results validate ARTEMIS's reliability guarantees and suitability for production deployment in safety-critical domains.

---

8. Conclusion and Future Work

8.1 Summary of Contributions

This paper presents ARTEMIS, a new multi-agent framework that solves key reliability problems in agent-based AI systems. It combines formal verification, runtime hallucination detection, and adaptive coordination. Our main contributions are:

1. **Novel Three-Tier Architecture**: We designed a production-ready architecture integrating a Formal Planning Layer for verifiable plan generation, a Cross-Agent Verification Module for hallucination detection, and an Adaptive Orchestration Engine for dynamic coordination strategy selection.

2. **Theoretical Reliability Assurances:** We include mathematical model and demonstrations of constrained propagation of error (Theorem 2) and probabilistic reliability (Theorem 3). This renders ARTEMIS the inaugural agent-based AI system to guarantee formal reliability

3. **Fine-Grained Analysis:** Experiments using three benchmarks (TravelPlanner, WebShop, SafetyBench) indicate that ARTEMIS is successfully used to perform tasks on a 94.3 percent

and detect hallucinations on a 97.8 percent success rate. It is more reliable by 23-41% than the current techniques.

4. **Systematic Robustness Analysis**: Ablation studies Ablation studies, adversarial tests, and failure analysis validate the role of every component as part of the system. The system demonstrates a stable performance in relation to various complexities of tasks, long tasks, and multiple tasks at the same time.

5. **Safety-Critical Validation**: ARTEMIS has an accuracy of 94.7 and the unsafe outputs of the ARTEMIS system are 0.6-percent. It has the capability to be applied in high-risk applications as its performance is comparable to that of human specialists.

8.2 Practical Impact

ARTEMIS addresses critical barriers to deploying agentic AI systems in production environments:

- **Trustworthiness**: Formal verification and hallucination detection provide transparency and accountability for system decisions
- **Reliability**: 94.3% task success rate and bounded error propagation enable deployment in safety-critical domains
- **Efficiency**: Adaptive orchestration achieves competitive execution time (14.2s) despite verification overhead
- **Scalability**: Sub-linear scaling to 50 agents and 50 concurrent tasks supports enterprise-scale deployments
- **Explainability**: Verification proofs and consensus scores provide interpretable explanations for stakeholders

8.3 Limitations

Despite ARTEMIS's strong performance, several limitations remain:

1. **Computational Overhead**: Verification and consensus mechanisms increase token consumption by 3× compared to baseline systems, raising operational costs for large-scale deployments.

2. **Task Specification Dependency**: 28% of failures stem from ambiguous constraint specifications, indicating that ARTEMIS inherits limitations from natural language task descriptions.

3. **Novel Task Generalization**: 10% of failures occur on tasks outside the training distribution, suggesting that the AOE's learned policy may not generalize to entirely new task types without additional training.

4. **Verification Timeout**: 19% of failures result from complex verification exceeding time budgets, indicating that formal verification may not scale to extremely large state spaces.

5. **LLM Capacity Limits**: 5% of failures represent fundamental limitations of current LLMs' reasoning capabilities, which ARTEMIS cannot overcome without more capable base models.

8.4 Future Work

We identify several promising directions for future research:

8.4.1 Enhanced Formal Verification

Current verification employs LTL model checking, which is limited to finite state spaces. Future work should explore:

- **Probabilistic Temporal Logic**: Extend verification to handle stochastic environments with probabilistic guarantees
- **Continuous State Spaces**: Develop verification techniques for continuous control tasks (robotics, autonomous vehicles)
- **Compositional Verification**: Enable modular verification of large multi-agent systems by verifying components independently

8.4.2 Advanced Hallucination Detection

While ARTEMIS reaches 97.8% detection accuracy, it can still be improved:

• **Multimodal verification:** Extend CAVM to check text, images, and structured data together.
• **Causal reasoning:** Use causal models to detect outputs that seem correct but are not logically valid in real situations.
• **Uncertainty estimation:** Add Bayesian methods to identify low-confidence outputs early.

8.4.3 Meta-Learning for Adaptive Orchestration

The current AOE is trained on 5,000 episodes. Meta-learning can improve it:

• **Few-shot adaptation:** Quickly adjust to new tasks with very little training.
• **Transfer learning:** Use knowledge from one domain to help learning in similar domains.
• **Continual learning:** Update the system during use without forgetting past knowledge.

8.4.4 Human-AI Collaboration

Although ARTEMIS works independently, human support can improve performance:

• **Active learning:** Ask for human help only when the system is uncertain.
• **Explanation generation:** Provide clear explanations for decisions and verification results.
• **Interactive refinement:** Allow humans to adjust plans and constraints during execution.

8.4.5 Theoretical Extensions

Some theoretical problems still remain:

• **Optimal verification strategies:** Find the best balance between verification cost and reliability.

• **Multi-agent game theory:** Study behavior when agents have competing goals.
• **Sample complexity:** Determine how much training data is needed for the AOE to perform well.

8.4.6 Domain-Specific Deployments

We plan to deploy ARTEMIS in additional safety-critical domains:

- **Healthcare**: Clinical decision support systems for diagnosis and treatment planning
- **Finance**: Automated trading systems with regulatory compliance verification
- **Autonomous Vehicles**: Multi-vehicle coordination with formal safety guarantees
- **Cybersecurity**: Automated threat detection and response with adversarial robustness

8.5 Broader Impact

ARTEMIS is a move in the right direction towards developing reliable agent-based AI systems that may be safely deployed in high-risk applications. It can be used to overcome major challenges that have curtailed the use of autonomous AI in safety-critical applications by integrating formal verification, clear hallucination detection, and adaptive coordination. We hope that ARTEMIS primary concepts such as formal verification, cross-agent checking, and adaptive coordination can contribute to future studying and aid in safeguarding the creation of AI systems. The code, benchmarks, and trained models of ARTEMIS will be published under open-source such that other researchers can reproduce the output and continue the work.

References

[1] J. Hellert et al., "Alpha Berkeley: A Scalable Framework for the Orchestration of Agentic Systems," *arXiv preprint arXiv:2508.15066*, 2025.

[2] Y. Ou et al., "Analyzing Information Sharing and Coordination in Multi-Agent Planning," *arXiv preprint arXiv:2508.12981*, 2025.

[3] A. Koubâa, "From Pre-Trained Language Models to Agentic AI: Evolution and Architectures for Autonomous Intelligence," *Preprints*, doi:10.20944/preprints202507.1809.v1, 2025.

[4] S. Putta et al., "Agent Q: Advanced Reasoning and Learning for Autonomous AI Agents," *arXiv preprint arXiv:2408.07199*, 2024.

[5] Y. Wang et al., "Agentic Discovery and Validation of Android App Vulnerabilities," *arXiv preprint arXiv:2508.21579*, 2025.

[6] J. Hellert et al., "Alpha Berkeley: A Scalable Framework for the Orchestration of Agentic Systems," *arXiv preprint arXiv:2508.15066*, 2025.

[7] Y. Zhao et al., "AGENTiGraph: A Multi-Agent Knowledge Graph Framework for Interactive, Domain-Specific LLM Chatbots," *arXiv preprint arXiv:2508.02999*, 2025.

[8] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations," 2023.

[9] S. Pelluru, “LangChain & LangGraph in Production: Architectures for Multi-Agent LLM Systems,” 2024.

[10] Y. Wang et al., “GraphCogent: Overcoming LLMs’ Working Memory Constraints via Multi-Agent Collaboration in Complex Graph Understanding,” *arXiv preprint arXiv:2508.12379*, 2025.

[11] Y. Ou et al., “Analyzing Information Sharing and Coordination in Multi-Agent Planning,” *arXiv preprint arXiv:2508.12981*, 2025.

[12] Y. Zhang et al., “AgentOrchestra: A Hierarchical Multi-Agent Framework for General-Purpose Task Solving,” *arXiv preprint arXiv:2506.12508*, 2025.

[13] Y. Lu et al., “OctoTools: An Agentic Framework with Extensible Tools for Complex Reasoning,” *arXiv preprint arXiv:2502.11271*, 2025.

[14] Y. Song et al., “GradientSys: A Multi-Agent LLM Scheduler with ReAct Orchestration,” *arXiv preprint arXiv:2507.06520*, 2025.

[15] Y. Xiong et al., “Self-Organizing Agent Network for LLM-based Workflow Automation,” *arXiv preprint arXiv:2508.13732*, 2025.

[16] Y. Xiong et al., “Self-Organizing Agent Network for LLM-based Workflow Automation,” *arXiv preprint arXiv:2508.13732*, 2025.

[17] “AegisFlow: A Framework for Orchestrating Autonomous Multi-Agent Workflows,” *Figshare*, doi:10.6084/m9.figshare.29324210, 2025.

[18] Y. Zhang et al., “AgentOrchestra: A Hierarchical Multi-Agent Framework for General-Purpose Task Solving,” *arXiv preprint arXiv:2506.12508*, 2025.

[19] Y. Lu et al., “OctoTools: An Agentic Framework with Extensible Tools for Complex Reasoning,” *arXiv preprint arXiv:2502.11271*, 2025.

[20] Y. Song et al., “GradientSys: A Multi-Agent LLM Scheduler with ReAct Orchestration,” *arXiv preprint arXiv:2507.06520*, 2025.

[21] Y. Zhao et al., “AGENTiGraph: A Multi-Agent Knowledge Graph Framework for Interactive, Domain-Specific LLM Chatbots,” *arXiv preprint arXiv:2508.02999*, 2025.

[22] S. Putta et al., “Agent Q: Advanced Reasoning and Learning for Autonomous AI Agents,” *arXiv preprint arXiv:2408.07199*, 2024.

[23] Y. Wang et al., “GraphCogent: Overcoming LLMs’ Working Memory Constraints via Multi-Agent Collaboration in Complex Graph Understanding,” *arXiv preprint arXiv:2508.12379*, 2025.

[24] Y. Kun et al., “MAS²: Self-Generative, Self-Configuring, Self-Rectifying Multi-Agent Systems,” *arXiv preprint arXiv:2509.24323*, 2025.

[25] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. Wiley, 2009.

[26] J. Wei et al., “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824-24837, 2022.

[27] X. Wang et al., “Self-Consistency Improves Chain of Thought Reasoning in Language Models,” *International Conference on Learning Representations (ICLR)*, 2023.

[28] U. M. Borghoff et al., “Beyond Prompt Chaining: The TB-CSPN Architecture for Agentic AI,” *Preprints*, doi:10.20944/preprints202507.1294.v1, 2025.

[29] Y. Li-Yang et al., “AgentGit: A Version Control Framework for Reliable and Scalable LLM-Powered Multi-Agent Systems,” *arXiv preprint arXiv:2511.00628*, 2025.

[30] Y. Shi et al., “AIME: Towards Fully-Autonomous Multi-Agent Framework,” *arXiv preprint arXiv:2507.11988*, 2025.

[31] J. Schulman et al., “Proximal Policy Optimization Algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[32] D. Liang *et al.*,“Holistic Evaluation of Language Models,” *Transactions on Machine Learning Research (TMLR)*, 2023.

[33] A. Srivastava *et al.,*“Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models*,” arXiv preprint arXiv:2206.04615,* 2022.

[34] “GPT-4 Technical Report,” *arXiv preprint arXiv:2303.08774*, 2023.

FUNDING:

This work has not received any funding