Where Did It All Go Wrong? A Hierarchical Look into Multi-Agent Error Attribution

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present ECHO (Error attribution through Contextual Hierarchy and Objective consensus analysis), a novel algorithm for error attribution in LLM multi-agent systems. While existing approaches struggle with accuracy and reliability in complex interaction scenarios, ECHO combines hierarchical context representation, objective analysis-based evaluation, and consensus voting to improve attribution accuracy. Our approach leverages positional-based contextual understanding with objective evaluation criteria. Experimental results demonstrate that ECHO outperforms existing methods across various multi-agent scenarios, particularly for subtle reasoning errors and complex interdependencies. This structured framework provides a more robust solution for error attribution in collaborative AI systems.

1 Introduction

LLMs are increasingly implemented as collaborative, specialized agents in structured systems [1, 2]. These systems distribute complex tasks among purpose-specific agents working toward common goals [3, 4]. Building such systems requires orchestration of graphs that map agent relationships and information flows, forming the foundation for flexible multi-agent frameworks [5]. Multi-Agent Systems have shown remarkable performance across coding [6], medical QA [7], and financial decision-making [8]. However, their multi-step nature makes them vulnerable to compounding errors, where early mistakes amplify through subsequent steps and derail the system. Hence, identifying the initial error's source - both agent and step - becomes crucial for mitigating failures.

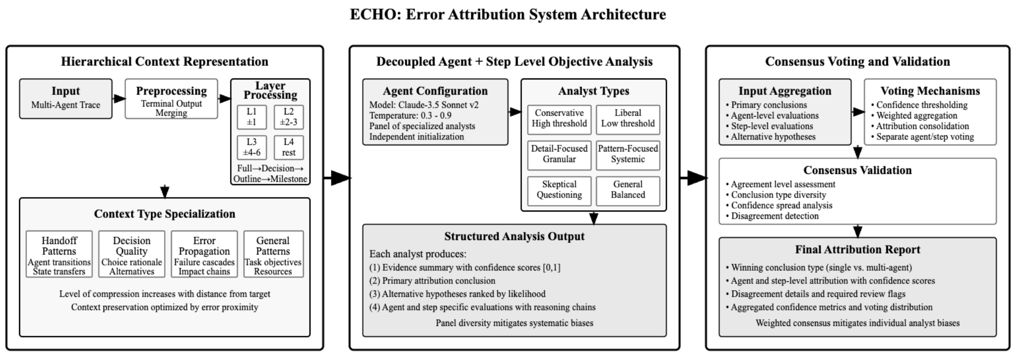
Growing MAS complexity makes manual error attribution unscalable. According to the Who&When benchmark [9], even SOTA LLMs — including GPT-4o [10], o1 [11], and Llama 4 [12] — struggle with this task. The challenge lies in managing interdependent agent interactions, large contexts, and the need to understand both local and global interaction patterns. Traditional debugging approaches prove inadequate for these dynamic, context-dependent systems.

Automated error attribution in LLM-based MAS has explored varying approaches to failure log analysis. All-at-once methods expose LLMs to complete logs simultaneously for agent and step identification [9]. Step-by-step approaches evaluate interactions sequentially until detecting an error [9]. Binary search methods iteratively narrow the search space by having LLMs determine which half of the trace contains the critical mistake [9].

This paper presents ECHO (Error attribution through Contextual Hierarchy and Objective consensus analysis), a novel approach to error attribution in multi-agent systems, that addresses these limitations, by guiding error attribution through developing a hierarchical context representation of the entire interaction trace, providing independent objective analyses across these contexts and cross-validating their findings via consensus voting.

2 ECHO Methodology

Error attribution in multi-agent systems requires 3 capabilities: context understanding to capture interaction patterns, error analysis to detect failure points, and decision synthesis to determine final attribution. ECHO addresses these through hierarchical context representation, decoupled objective analysis (at both agent and step levels), and confidence-weighted consensus voting.



ECHO: Error Attribution System Architecture
Hierarchical Context Representation
Input
Multi-Agent Trace
Preprocessing
Terminal Output Merging
Layer Processing
L1 ±1
L2 ±2-3
L3 ±4-6
L4 rest
Full→Decision→Outline→Milestone
Context Type Specialization
Handoff Patterns
Agent transitions
State transfers
Decision Quality
Choice rationale
Alternatives
Error Propagation
Failure cascades
Impact chains
General Patterns
Task objectives
Resources
Level of compression increases with distance from target
Context preservation optimized by error proximity
Decoupled Agent + Step Level Objective Analysis
Agent Configuration
Model: Claude-3.5 Sonnet v2
Temperature: 0.3 - 0.9
Panel of specialized analysts
Independent initialization
Analyst Types
Conservative High threshold
Liberal Low threshold
Detail-Focused Granular
Pattern-Focused Systemic
Skeptical Questioning
General Balanced
Structured Analysis Output
Each analyst produces:
(1) Evidence summary with confidence scores [0,1]
(2) Primary attribution conclusion
(3) Alternative hypotheses ranked by likelihood
(4) Agent and step specific evaluations with reasoning chains
Panel diversity mitigates systematic biases
Consensus Voting and Validation
Input Aggregation
• Primary conclusions
• Agent-level evaluations
• Step-level evaluations
• Alternative hypotheses
Voting Mechanisms
• Confidence thresholding
• Weighted aggregation
• Attribution consolidation
• Separate agent/step voting
Consensus Validation
• Agreement level assessment
• Conclusion type diversity
• Confidence spread analysis
• Disagreement detection
Final Attribution Report
• Winning conclusion type (single vs. multi-agent)
• Agent and step-level attribution with confidence scores
• Disagreement details and required review flags
• Aggregated confidence metrics and voting distribution
Weighted consensus mitigates individual analyst biases

Figure 1: ECHO Architecture. The system comprises: (1) Hierarchical Context - processes traces through 4 compression layers (L1-L4: full content → milestones); (2) Decoupled Analysis - uses 6 specialized agents (conservative to balanced) generating structured outputs; (3) Consensus Voting - aggregates analyses via confidence-weighted voting.

2.1 Hierarchical Context Representation

Error attribution in multi-agent systems must balance comprehensive context against processing limitations. Traditional approaches that only analyze immediate neighbors (±1 agent) miss crucial long-range dependencies and error propagation patterns. ECHO addresses this through a hierarchical context representation operating across 4 layers L_1 through L_4 (see Appendix A.1 and A.3) to extract key information from agent / step interactions via regex pattern matching. Its implementation employs specialized content extraction mechanisms for each layer C_i, as seen below:

Immediate Context (L_1): Preserves full reasoning for target and direct neighbors ($\tau_{i\pm1}$)

Local Context (L_2): Captures key decision sequences for $\tau_{i\pm2,3}$ steps

Distant Context (L_3): Compresses $\tau_{i\pm4,5,6}$ steps into outcome summaries

Global Context (L_4): Retains only critical milestones for remaining interactions

This layered approach enables both detailed local analysis and broad pattern recognition while maintaining computational efficiency. The extraction process adapts to different context types (handoff, decision quality, error propagation, general) for optimal information preservation.

2.2 Objective Analysis

ECHO utilizes a panel of k objective analysis agents that evaluate interaction traces through hierarchical context C. Each agent independently assesses steps and provides confidence scores σ_j, enabling identification of distributed responsibility and systemic issues. The analysis framework examines errors across all context layers while maintaining step-level granularity.

To mitigate systematic biases, ECHO employs diverse analyst roles ρ_j: (1) Conservative Analyst - that requires strong evidence, prefers single-agent attribution (2) Liberal Analyst - that considers multi-agent scenarios, identifies subtle patterns (3) Detail-Focused Analyst - that examines specific evidence and inconsistencies (4) Pattern-Focused Analyst - that tracks broader reasoning chains and error propagation (5) Skeptical Analyst - that questions assumptions and explores alternatives (6) General Analyst - that maintains balanced perspective across all evidence types.

Each agent produces structured outputs including their findings, error likelihood scores, primary conclusions with confidence scores σ_j, and alternative hypotheses. This analytical approach prevents echo-chamber effects and accommodates for alternative diversification approaches.

2.3 Consensus Voting

ECHO employs a consensus voting mechanism to aggregate analyses from k objective analysts. The system uses weighted confidence consensus, where attributions are weighted by confidence scores σ_j, subject to a minimum threshold δ. The mechanism processes 3 components from each analysis A_t^j: primary conclusions, agent evaluations, and alternative hypotheses. Voting follows a hierarchical structure: first determining conclusion type through weighted aggregation (V_a, V_s), then identifying specific agents (ω_a) and steps (ω_s), before synthesizing supporting reasoning. Finally, disagreement analysis ϕ examines conclusion diversity, confidence spread, and attribution consistency, triggering additional review when necessary (spread > 0.5 or conflicting high-confidence attributions). This structured approach ensures robust final attributions while preserving insights from dissenting views.

3 Results and Analysis

3.1 Experimental Setup

We evaluate using the Who&When benchmark [9], which contains annotated failure logs from LLM-powered multi-agent systems. The benchmark comprises 2 subsets: (1) Algorithm-Generated: failure logs from automated multi-agent interactions, and (2) Hand-Crafted: curated scenarios designed to capture complex error patterns. Each failure log is annotated with the failure-responsible agent and the specific error step. Performance is measured via agent-level accuracy (identifying responsible agent) and step-level accuracy (identifying error step). Our implementation uses Claude 3.5 Sonnet v2 as the base LLM, with 3 randomly sampled analysis agents operating at temperatures between 0.3-0.9. We lastly employ a minimum confidence threshold ($\delta = 0.3$) for consensus voting.

3.2 Comparative Analysis of Implementations

We evaluate 4 progressive implementations of error attribution, each building upon the limitations of its predecessor:

I1 - Fixed Context Window: Baseline approach using a fixed window (±1 step), where a context-aware agent analyzes each step with its neighbors, followed by a final judge agent for attribution.

I2 - Hierarchical Context: Enhances I1 by replacing the fixed window with hierarchical context representation, maintaining the same attribution but providing graduated access to the full trace.

I3 - Objective Analysis: Builds upon I2 by replacing the context-aware agent and judge agent with a panel of specialized objective analyst agents and performing consensus voting on those outcomes.

I4 - Decoupled Attribution: Refines I3 by separating the attribution into 2 phases: agent-level to identify responsible agents, followed by step-level to pinpoint specific error points.

3.2.1 Performance of ECHO

ECHO achieves consistent agent-level accuracy ($\sim$68%) across all configurations, with minimal degradation ($\sim$1-2%) when ground truth is withheld. This suggests that ECHO's agent-level attribution mechanism is robust and doesn't heavily rely on ground truth information. The similar performance across both dataset types (hand-crafted: 68.4%, algorithm-generated: 68.8% with ground truth) indicates good generalization across different interaction patterns. Exact step-level accuracy is notably lower ($\sim$27-28%), reflecting the increased difficulty of precise step identification. However, when considering tolerance ranges, accuracy improves substantially. For the hand-crafted dataset, accuracy reaches 42.1% at ±3 steps tolerance, and 61.4% at ±5 steps tolerance. ECHO also maintains consistent computational patterns across configurations and exhibits reasonable token usage for long-traces (as seen by $\sim$54K tokens in the hand-crafted dataset), ensuring cost-effective processing when using Claude 3.5 Sonnet v2.

Table 1: Performance of ECHO across different datasets and configurations

Method	Hand-Crafted Dataset		Algorithm-Generated Dataset		
	With GT	Without GT	With GT	Without GT	P-value[†]
Agent-Level Accuracy					
Random	0.120	0.120	0.291	0.291	<0.001
All-at-Once	0.577	0.529	0.563	0.530	0.032
Step-by-Step	0.360	0.343	0.397	0.283	<0.001
Binary Search	0.517	0.362	0.441	0.301	0.007
ECHO (ours)	**0.684**	**0.679**	**0.688**	**0.672**	-
Step-Level Accuracy (Exact)					
Random	0.042	0.042	0.191	0.191	<0.001
All-at-Once	0.060	0.021	0.152	0.145	<0.001
Step-by-Step	0.066	0.069	0.274	0.178	0.003
Binary Search	0.069	0.069	0.240	0.166	0.012
ECHO (ours)	**0.281**	**0.268**	**0.288**	**0.272**	-
Step-Level with Tolerance *(Hand-Crafted with GT)*					
	All-at-Once	Step-by-Step	Binary Search	**ECHO (ours)**	P-value[†]
±1 step	0.149	0.166	0.138	**0.351**	<0.001
±3 steps	0.350	0.209	0.224	**0.421**	0.029
±5 steps	0.428	0.351	0.362	**0.614**	0.008
Token Cost *(Hand-Crafted with GT)*					
	All-at-Once	Step-by-Step	Binary Search	**ECHO (ours)**	-
Tokens	**17,106**	87,720	34,659	53,701	-

[†]P-values compare ECHO against each baseline using chi-squared test.

3.3 Impact of ECHO Components Via Ablation

Unifying v. Decoupling Objective Analyses The relationship between context length and attribution performance informs whether to unify or decouple agent-level and step-level analyses (Table 2). For shorter algorithm-generated traces, unified analysis achieves 65.1% agent-level and 46.1% step-level accuracy. While decoupling slightly improves agent-level accuracy to 68.8%, it reduces step-level accuracy to 28.8%. This suggests that separating tasks can be detrimental when within the LLM's processing capacity ($\sim$13K tokens unified vs. $\sim$7K tokens decoupled).

Reducing Computational Overhead Through Objective Analysis The objective analysis (I3) reveals limitations of using context-aware agents for error attribution. While I1 and I2 implementations rely on repeatedly examining trace information, objective analysis reduces token usage by 60-110x for hand-crafted cases, and by 25-30x for algorithm-generated cases, while still improving accuracy ($\sim$+16.3% agent-level and $\sim$+6.2% step-level at $P < 0.05$) as shown in Table 2, suggesting objective analysis is more suitable for practical deployment than context-aware agents.

From Fixed to Hierarchical Context Hierarchical context representation shows clear advantages over fixed context windows. Moving from I1 to I2 improves agent-level accuracy by 16.1% ($P < 0.05$) and step-level accuracy by 1.9% (not significant) on hand-crafted data (Table 2). This demonstrates the value of graduated context preservation, though both implementations were limited to shorter traces due to computational constraints with longer ones. The improvement suggests hierarchical context's value for error attribution, particularly when combined with objective analysis.

4 Conclusion

We present ECHO, a novel error attribution approach for multi-agent systems that combines hierarchical context representation with confidence-weighted consensus voting. Our results show significant improvements over existing methods, especially for longer interaction traces. Future work includes developing dynamic context preservation, enhancing consensus mechanisms, and incorporating error severity metrics. ECHO's efficient design provides a promising foundation for debugging complex multi-agent systems.

References

[1] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," *arXiv preprint arXiv:2402.01680*, 2024.

[2] X. Li, "A review of prominent paradigms for llm-based agents: Tool use, planning (including rag), and feedback learning," in *Proceedings of the 31st International Conference on Computational Linguistics*, 2025, pp. 9760–9779.

[3] W. Chen, Z. You, R. Li, Y. Guan, C. Qian, C. Zhao, C. Yang, R. Xie, Z. Liu, and M. Sun, "Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence," *arXiv preprint arXiv:2407.07061*, 2024.

[4] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "Camel: Communicative agents for" mind" exploration of large language model society," *Advances in Neural Information Processing Systems*, vol. 36, pp. 51 991–52 008, 2023.

[5] H. Zhou, X. Wan, R. Sun, H. Palangi, S. Iqbal, I. Vulić, A. Korhonen, and S. Ö. Arık, "Multi-agent design: Optimizing agents with better prompts and topologies," *arXiv preprint arXiv:2502.02533*, 2025.

[6] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin *et al.*, "Metagpt: Meta programming for a multi-agent collaborative framework," in *The Twelfth International Conference on Learning Representations*, 2023.

[7] Y. Kim, C. Park, H. Jeong, Y. S. Chan, X. Xu, D. McDuff, H. Lee, M. Ghassemi, C. Breazeal, and H. W. Park, "Mdagents: An adaptive collaboration of llms for medical decision-making," *Advances in Neural Information Processing Systems*, vol. 37, pp. 79 410–79 452, 2024.

[8] Y. Yu, Z. Yao, H. Li, Z. Deng, Y. Jiang, Y. Cao, Z. Chen, J. Suchow, Z. Cui, R. Liu *et al.*, "Fincon: A synthesized llm multi-agent system with conceptual verbal reinforcement for enhanced financial decision making," *Advances in Neural Information Processing Systems*, vol. 37, pp. 137 010–137 045, 2024.

[9] S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen *et al.*, "Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems," *arXiv preprint arXiv:2505.00212*, 2025.

[10] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[11] A. Jaech, A. Kalai, A. Lerer, A. Richardson, A. El-Kishky, A. Low, A. Helyar, A. Madry, A. Beutel, A. Carney *et al.*, "Openai o1 system card," *arXiv preprint arXiv:2412.16720*, 2024.

[12] A. Meta, "The llama 4 herd: The beginning of a new era of natively multimodal ai innovation," *https://ai. meta. com/blog/llama-4-multimodal-intelligence/, checked on*, vol. 4, no. 7, p. 2025, 2025.

A Appendix

A.1 ECHO Algorithm

Algorithm 1 ECHO: Error Attribution through Contextual Hierarchy and Objective Consensus Analysis

```
Require:
 1: $\tau$ : interaction trace of $n$ agents
 2: $\alpha$ : final answer
 3: $\delta$ : minimum confidence threshold
 4: $k$ : number of analysis agents
 5: $\gamma$ : ground truth (optional)

Ensure: Attribution of error to specific agent(s) and step(s)

 6: Procedure HierarchicalContextExtraction($\tau$):
 7: $C \leftarrow \emptyset$ {Init context}
 8: for each agent $i \in \{1, ..., n\}$ do
 9:     $L_1 \leftarrow$ ExtractFullContext($\tau_{i\pm1}$)
10:     $L_2 \leftarrow$ ExtractKeyDecisions($\tau_{i\pm2,3}$)
11:     $L_3 \leftarrow$ CompressSummaries($\tau_{i\pm4,5,6}$)
12:     $L_4 \leftarrow$ ExtractMilestones($\tau_{remainder}$)
13:     $C_i \leftarrow \{L_1, L_2, L_3, L_4\}$
14: end for
15: return $C$

16: Procedure DecoupledAgentAndStepAnalysis($C, \alpha, \gamma$):
17: for type $t \in$ {agent, step} do
18:     $A_t \leftarrow \emptyset$ {Init results}
19:     for each analyst $j \in \{1, ..., k\}$ do
20:         $\rho_j \leftarrow$ AnalystRole($j$)
21:         if $\gamma \neq$ None then
22:             $\epsilon_j \leftarrow$ Eval($t, C, \rho_j, \gamma$)
23:         else
24:             $\epsilon_j \leftarrow$ Eval($t, C, \rho_j$)
25:         end if
26:         $\sigma_j \leftarrow$ ConfidenceScore($\epsilon_j$)
27:         $A_t^j \leftarrow \{\epsilon_j, \sigma_j\}$
28:     end for
29: end for
30: return $(A_{\text{agent}}, A_{\text{step}})$

31: Procedure ConsensusVoting($A_a, A_s, \delta$):
32: $V_a, V_s \leftarrow \emptyset, \emptyset$ {Init voting}
33: for each analysis pair $(A_a^j, A_s^j)$ do
34:     if $\sigma_j \geq \delta$ then
35:         $V_a, V_s \leftarrow V_a \cup \{A_a^j\}, V_s \cup \{A_s^j\}$
36:     end if
37: end for
38: $\omega_a, \omega_s \leftarrow$ WeightedAggregate($V_a, V_s$)
39: $\phi \leftarrow$ DisagreementAnalysis($V_a, V_s$)
40: return ConsensusResult($\omega_a, \omega_s, \phi$)

41: $C \leftarrow$ HierarchicalContextRepresentation($\tau$)
42: $A_a, A_s \leftarrow$ DecoupledAgentAndStepAnalysis($C, \alpha, \gamma$)
43: return ConsensusVoting($A_a, A_s, \delta$)
```

Table 2: Ablation Study: Impact of Each Component

	Hand-Crafted Dataset		Algorithm-Generated Dataset		
Implementation	With GT	Without GT	With GT	Without GT	P-value[‡]
Agent-Level Accuracy					
Fixed Context (I1)[†]	0.286	0.265	0.461	0.452	-
+ Hierarchical (I2)[†]	0.447	0.429	0.523	0.508	0.037
+ Objective Analysis (I3)	0.610	0.589	0.651	0.635	0.043
+ Decoupled Attribution (I4)	**0.684**	**0.679**	**0.688**	**0.672**	0.196
Step-Level Accuracy					
Fixed Context (I1)[†]	0.151	0.143	0.157	0.140	-
+ Hierarchical (I2)[†]	0.170	0.166	0.192	0.175	0.398
+ Objective Analysis (I3)	0.232	0.218	**0.461**	**0.444**	<0.001
+ Decoupled Attribution (I4)	**0.281**	**0.268**	0.288	0.272	0.211
Token Cost					
Fixed Context (I1)[†]	4.02M	3.93M	319K	317K	-
+ Hierarchical (I2)[†]	7.70M	7.66M	407K	405K	-
+ Objective Analysis (I3)	67.5K	66.5K	**12.6K**	**12.5K**	-
+ Decoupled Attribution (I4)	**33.0K**	**32.5K**	12.8K	12.7K	-

[†]Hand-Crafted Dataset results for I1 and I2 based on limited sample of shorter traces
[‡]P-values compare each component with previous implementation

A.2 Fixed-Window Context

```python

def extract_agent_contexts(
    conversation_history: List[Dict[str, Any]]
) -> List[Tuple[Optional[Dict[str, Any]], Dict[str, Any], Optional[Dict[str, Any
    ]]]]:
    """
    Extract agent contexts from conversation history.
    Each context includes the previous agent, current agent, and next agent.

    Args:
        conversation_history: List of conversation turns with agent information

    Returns:
        List of tuples containing (prev_agent, current_agent, next_agent) for each
            agent
    """

    contexts = []
    for i in range(len(conversation_history)):
        # Get previous agent (None if first agent)
        prev_agent = conversation_history[i - 1] if i > 0 else None

        # Get current agent
        current_agent = conversation_history[i]

        # Get next agent (None if last agent)
        next_agent = conversation_history[i + 1] if i < len(conversation_history) -
            1 else None

        contexts.append((prev_agent, current_agent, next_agent))

    return contexts
```

A.3 Hierarchical Context Extraction

```python
def extract_key_decision(
    agent_content: str, max_words: int = 50, context_type: str = "decision_quality"
) -> str:
    """
    Extract key decision or main point from agent content using regex patterns.

    Args:
        agent_content: The full content of the agent
        max_words: Maximum words in the extracted key decision
        context_type: Type of context to focus on (handoff, decision_quality,
            error_propagation, general)

    Returns:
        Key decision or main point from the agent's content
    """
    if not agent_content.strip():
        return "No␣content␣available"

    if context_type == "handoff":
        patterns = [
            r"(?:received|got|obtained|from)\s+([^.!?]*[.!?])",
            r"(?:passing|providing|sending|to)\s+([^.!?]*[.!?])",
            r"(?:based␣on|using)\s+([^.!?]*[.!?])",
            r"(?:will|need␣to|should)\s+([^.!?]*(?:next|continue)[^.!?]*[.!?])",
        ]
    elif context_type == "decision_quality":
        patterns = [
            r"(?:I␣(?:conclude|determine|decide|believe|think))\s+([^.!?]*[.!?])",
            r"(?:Therefore|Thus|So|Hence),?\s+([^.!?]*[.!?])",
            r"(?:The␣(?:answer|solution|result))\s+(?:is|appears)\s+([^.!?]*[.!?])",
            r"(?:Based␣on|Given)\s+([^.!?]*[.!?])",
        ]
    elif context_type == "error_propagation":
        patterns = [
            r"(?:error|mistake|wrong|incorrect|failed)\s+([^.!?]*[.!?])",
            r"(?:cannot|unable|couldn't|can't)\s+([^.!?]*[.!?])",
            r"(?:However|But|Unfortunately)\s+([^.!?]*[.!?])",
        ]
    else:  # general
        patterns = [
            r"(?:I␣(?:will|should|need␣to|decided␣to|conclude␣that|believe|think|
                determine))␣([^.!?]*[.!?])",
            r"(?:Therefore|Thus|So|Hence),?␣([^.!?]*[.!?])",
            r"(?:The␣answer|The␣result|The␣solution)␣(?:is|appears␣to␣be|seems␣to␣be)
                ␣([^.!?]*[.!?])",
            r"Let␣me␣([^.!?]*[.!?])",
            r"(?:My␣approach|My␣strategy|My␣plan)␣(?:is|will␣be)␣([^.!?]*[.!?])",
        ]

    # Try to find pattern matches
    for pattern in patterns:
        matches = re.findall(pattern, agent_content, re.IGNORECASE)
        if matches:
            decision = matches[0].strip()
            words = decision.split()[:max_words]
            return "␣".join(words) + ("..." if len(decision.split()) > max_words else
                 "")

    # Fallback: take the first sentence or first max_words
    sentences = agent_content.split(".␣")
    if sentences:
        first_sentence = sentences[0].strip()
        if not first_sentence.endswith("."):
            first_sentence += "."
        words = first_sentence.split()[:max_words]
```

```python
        return " ".join(words) + ("..." if len(first_sentence.split()) > max_words else "")

    # Final fallback: just truncate
    words = agent_content.split()[:max_words]
    return " ".join(words) + ("..." if len(agent_content.split()) > max_words else "")


def summarize_agent(agent_content: str, max_words: int = 20, context_type: str = "general") -> str:
    """
    Create a brief summary of agent content using regex patterns.

    Args:
        agent_content: The full content of the agent
        max_words: Maximum words in the summary
        context_type: Type of context to focus on (handoff, decision_quality, error_propagation, general)

    Returns:
        Brief summary of the agent's content
    """
    if not agent_content.strip():
        return "No content available"

    # Remove excessive whitespace and newlines
    cleaned_content = " ".join(agent_content.split())

    if context_type == "handoff":
        patterns = [
            r"(?:received|got|obtained)\s+([^.!?]*[.!?])",
            r"(?:providing|sending)\s+([^.!?]*[.!?])",
        ]
    elif context_type == "decision_quality":
        patterns = [
            r"(?:conclude|determine|decide)\s+([^.!?]*[.!?])",
            r"(?:Therefore|Thus|So),?\s+([^.!?]*[.!?])",
        ]
    elif context_type == "error_propagation":
        patterns = [
            r"(?:error|mistake|failed)\s+([^.!?]*[.!?])",
            r"(?:cannot|unable)\s+([^.!?]*[.!?])",
        ]
    else:  # general
        patterns = [
            r"(?:In conclusion|To conclude|Therefore|Thus|So|Hence),? ([^.!?]*[.!?])",
            r"(?:The (?:answer|result|solution|output)) (?:is|appears to be|seems to be) ([^.!?]*[.!?])",
            r"(?:I (?:found|determined|concluded|calculated)) ([^.!?]*[.!?])",
        ]

    # Try pattern matching first
    for pattern in patterns:
        matches = re.findall(pattern, cleaned_content, re.IGNORECASE)
        if matches:
            summary = matches[0].strip()
            words = summary.split()[:max_words]
            return " ".join(words) + ("..." if len(summary.split()) > max_words else "")

    # Fallback: take first sentence and truncate
    sentences = cleaned_content.split(". ")
    if sentences:
        first_sentence = sentences[0].strip()
        words = first_sentence.split()[:max_words]
        return "␣".join(words) + ("..." if len(first_sentence.split()) > max_words
             else "")

    # Final fallback
    words = cleaned_content.split()[:max_words]
    return "␣".join(words) + ("..." if len(cleaned_content.split()) > max_words else
         "")


def obtain_milestones(agent_content: str, max_words: int = 15, context_type: str = "
     general") -> str:
    """
    Extract milestone-based information from agent content using regex patterns.
    This provides a higher level of abstraction than brief summaries for distant
         contexts.

    Args:
        agent_content: The full content of the agent
        max_words: Maximum words in the extracted milestones
        context_type: Type of context to focus on (handoff, decision_quality,
             error_propagation, general)

    Returns:
        Milestone-based information from the agent's content
    """
    if not agent_content.strip():
        return "No␣milestones␣available"

    # Remove excessive whitespace and newlines
    cleaned_content = "␣".join(agent_content.split())

    if context_type == "handoff":
        patterns = [
            r"(?:received|obtained|got)\s+([^.!?]*(?:from|data|information)
                 [^.!?]*[.!?])",
            r"(?:provided|sent|passed)\s+([^.!?]*(?:to|data|information)[^.!?]*[.!?])
                 ",
            r"(?:completed|finished)\s+([^.!?]*(?:handoff|transfer)[^.!?]*[.!?])",
        ]
    elif context_type == "decision_quality":
        patterns = [
            r"(?:decided|determined|concluded)\s+([^.!?]*[.!?])",
            r"(?:evaluated|assessed|analyzed)\s+([^.!?]*[.!?])",
            r"(?:final␣decision|ultimate␣choice)\s*[:-]?\s*([^.!?]*[.!?])",
        ]
    elif context_type == "error_propagation":
        patterns = [
            r"(?:error|mistake|failure)\s+(?:occurred|detected)\s+([^.!?]*[.!?])",
            r"(?:identified|found)\s+(?:error|issue|problem)\s+([^.!?]*[.!?])",
            r"(?:corrected|fixed|resolved)\s+([^.!?]*[.!?])",
        ]
    else:  # general
        patterns = [
            r"(?:completed|finished|achieved|accomplished)\s+([^.!?]*[.!?])",
            r"(?:created|generated|produced|built)\s+([^.!?]*[.!?])",
            r"(?:step\s+\d+|phase\s+\d+|stage\s+\d+)\s*[:-]?\s*([^.!?]*[.!?])",
            r"(?:successfully|finally)\s+([^.!?]*[.!?])",
        ]

    # Try to find pattern matches
    for pattern in patterns:
        matches = re.findall(pattern, cleaned_content, re.IGNORECASE)
        if matches:
            milestone = matches[0].strip()
            words = milestone.split()[:max_words]
            return "␣".join(words) + ("..." if len(milestone.split()) > max_words
                else "")

    # Fallback: extract first meaningful sentence
    sentences = cleaned_content.split(".␣")
    if sentences:
        first_sentence = sentences[0].strip()
        words = first_sentence.split()[:max_words]
        return "␣".join(words) + ("..." if len(first_sentence.split()) > max_words
            else "")

    # Final fallback
    words = cleaned_content.split()[:max_words]
    return "␣".join(words) + ("..." if len(cleaned_content.split()) > max_words else
        "")


def extract_agent_contexts_hierarchical(
    conversation_history: List[Dict[str, Any]], dataset_name: str = ""
) -> List[Dict[str, Any]]:
    """
    Extract hierarchical agent contexts from conversation history.
    Uses graduated detail levels based on distance from current agent.

    Args:
        conversation_history: List of conversation turns with agent information
        dataset_name: Name of the dataset being processed (affects how agent info is
             extracted)

    Returns:
        List of dictionaries containing hierarchical context for each agent
    """

    contexts = []
    for current_idx in range(len(merged_history)):
        current_agent = conversation_history[current_idx]

        # Build hierarchical context for this agent
        hierarchical_context = {
            "current_agent": current_agent,
            "context_levels": {
                "immediate": [], # Distance 1: Full detail
                "nearby": [], # Distance 2-3: Key decisions
                "distant": [], # Distance 4-6: Brief summaries
                "milestones": [], # Distance >6: Milestones
            },
        }

        # Process all other agents based on their distance
        for i, agent in enumerate(conversation_history):
            if i == current_idx:
                continue # Skip current agent

            distance = abs(i - current_idx)

            agent_info = {
                "index": i,
                "name": agent["name"],
                "role": agent["role"],
                "distance": distance,
            }

            if distance == 1: # Immediate context: Full detail
```

```
                agent_info["content"] = agent["content"]
                agent_info["detail_level"] = "full"
                hierarchical_context["context_levels"]["immediate"].append(agent_info
                    )

            elif distance <= 3: # Nearby context: Key decisions
                agent_info["content"] = extract_key_decision(agent["content"])
                agent_info["detail_level"] = "key_decisions"
                hierarchical_context["context_levels"]["nearby"].append(agent_info)

            elif distance <= 6: # Distant context: Brief summaries
                agent_info["content"] = summarize_agent(agent["content"])
                agent_info["detail_level"] = "summary"
                hierarchical_context["context_levels"]["distant"].append(agent_info)

            else:  # Milestone context: High-level milestones for very distant agents
                agent_info["content"] = obtain_milestones(agent["content"])
                agent_info["detail_level"] = "milestones"
                hierarchical_context["context_levels"]["milestones"].append(
                    agent_info)

        # Sort each level by original conversation order
        for level in hierarchical_context["context_levels"].values():
            level.sort(key=lambda x: x["index"])

        contexts.append(hierarchical_context)

    return contexts
```

A.4 Context Step Aware Agent

```

class ContextAwareStepAgent:
    """
    Context-Aware Step Agent that analyzes an agent in the context of its previous
        and next agents
    to argue why the error happened in this agent's step.
    """

    def __init__(
        self,
        model_id: str = "us.anthropic.claude-3-5-sonnet-20241022-v2:0",
        temperature: float = 1.0,
    ):
        """
        Initialize the Context-Aware Step Agent.

        Args:
            model_id: The model ID to use for the agent
            temperature: The temperature to use for generation
        """
        self.bedrock_model = BedrockModel(
            model_id=model_id,
            temperature=temperature,
            top_p=0.9,
            max_tokens=4096,
        )

        self.system_prompt = """
        You are a Context-Aware Step Agent that analyzes an agent's actions in the
            context of the previous and next agents.
        Your task is to argue why the error happened in YOUR agent's step.

        Your task:
```

```
        1. Analyze what information was received by your agent from the previous
            agent (if any)
        2. Analyze what information was generated by your agent
        3. Analyze how your agent's output affected the next agent (if any)
        4. Make a strong argument for why YOUR AGENT caused the final wrong answer,
            using the ground truth as evidence

        Input:
        - Ground Truth: [GROUND_TRUTH]
        - Final Answer: [FINAL_ANSWER]
        - Agent Context: Information about the previous, current, and next agents

        Output your response with the following clear section headers:

        ## Purpose:
        Describe the purpose of this agent step - what was this agent trying to
            accomplish?

        ## Assumptions and Information:
        List the assumptions and information this agent was given from the previous
            agent or context.

        ## Errors:
        Describe what this agent did wrong (if anything). Be specific about any
            mistakes, misunderstandings, or incorrect reasoning.

        ## Evidence:
        Provide evidence from the ground truth that supports your error attribution.
             Explain how this agent's actions directly led to the wrong final answer.


        Remember: You must argue that YOUR agent caused the error. Be persuasive and
             use evidence.
        """

        self.agent = Agent(
            system_prompt=self.system_prompt,
            model=self.bedrock_model,
        )

    def analyze_agent(
        self,
        step_id: str,
        prev_agent: Optional[Dict[str, Any]],
        current_agent: Dict[str, Any],
        next_agent: Optional[Dict[str, Any]],
        ground_truth: Optional[str],
        final_answer: str,
        query: str = "",
    ) -> Dict[str, Any]:
        """
        Analyze an agent in the context of its previous and next agents and generate
             an argument
        for why this agent caused the error.

        Args:
            step_id: The ID of the step (e.g., "step_1")
            prev_agent: The previous agent (or None if first agent)
            current_agent: The current agent being analyzed
            next_agent: The next agent (or None if last agent)
            ground_truth: The ground truth answer
            final_answer: The final answer given
            query: The original query/question

        Returns:
            JSON argument for why this agent caused the error
        """

        prompt = f"""
        Original Query: {query}
        {ground_truth_section}
        Final Answer: {final_answer}
        Agent Context:
        {agent_context}

        Please analyze this agent in the context of the previous and next agents,
             and provide a strong argument for why THIS agent caused the final wrong
             answer.
        Use the section headers specified in your instructions (Purpose, Assumptions
              and Information, Errors, Evidence).
        """

        agent_result = self.agent(prompt)

        # Extract text from AgentResult
        response_text = ""
        if hasattr(agent_result, "message") and "content" in agent_result.message:
            content = agent_result.message["content"]
            if isinstance(content, list) and len(content) > 0 and "text" in content
                 [0]:
                response_text = content[0]["text"]
            elif isinstance(content, str):
                response_text = content

        # Create a dictionary with the step_id, agent_name, and the full text
             response
        result = {
            "step_id": step_id,
            "agent_name": current_agent["name"],
            "analysis": response_text,
            "token_usage": token_usage,
        }

        return result

    def analyze_agent_hierarchical(
        self,
        step_id: str,
        hierarchical_context: Dict[str, Any],
        ground_truth: Optional[str],
        final_answer: str,
        query: str = "",
    ) -> Dict[str, Any]:
        """
        Analyze an agent using hierarchical context and generate an argument
        for why this agent caused the error.

        Args:
            step_id: The ID of the step (e.g., "step_1")
            hierarchical_context: Dictionary containing hierarchical context
                 information
            ground_truth: The ground truth answer
            final_answer: The final answer given
            query: The original query/question

        Returns:
            JSON argument for why this agent caused the error
        """
        current_agent = hierarchical_context["current_agent"]

        prompt = f"""
        Original Query: {query}
        {ground_truth_section}
        Final Answer: {final_answer}
        Agent Context:
        {agent_context}

        Please analyze this agent in the hierarchical context of the entire
            conversation, and provide a strong argument for why THIS agent caused
            the final wrong answer.
        Use the section headers specified in your instructions (Purpose, Assumptions
             and Information, Errors, Evidence).

        Note: You now have access to the full conversation context at different
            detail levels:
        - Immediate context: Full details of adjacent agents
        - Nearby context: Key decisions from agents 2-3 steps away
        - Distant context: Brief summaries of agents 4+ steps away

        Consider how information and errors might have propagated across the entire
            conversation when making your argument.
        """

        agent_result = self.agent(prompt)

        # Extract text from AgentResult
        response_text = ""
        if hasattr(agent_result, "message") and "content" in agent_result.message:
            content = agent_result.message["content"]
            if isinstance(content, list) and len(content) > 0 and "text" in content
                 [0]:
                response_text = content[0]["text"]
            elif isinstance(content, str):
                response_text = content

        # Create a dictionary with the step_id, agent_name, and the full text
            response
        result = {
            "step_id": step_id,
            "agent_name": current_agent["name"],
            "analysis": response_text,
            "context_type": "hierarchical",
            "token_usage": token_usage,
        }

        return result
```

A.5 Objective Analysis Agent

```

class ObjectiveAnalysisAgent:
    """
    Objective Analysis Agent that analyzes all agents in a conversation objectively
    to determine error attribution without forced bias.
    """

    def __init__(
        self,
        model_id: str = "us.anthropic.claude-3-5-sonnet-20241022-v2:0",
        temperature: float = 0.7,
        analyst_focus: str = "general",
    ):
        """
        Initialize the Objective Analysis Agent.

        Args:
            model_id: The model ID to use for the agent
            temperature: The temperature to use for generation
        """
        self.bedrock_model = BedrockModel(
            model_id=model_id,
            temperature=temperature,
            top_p=0.9,
            max_tokens=4096,
        )

        # Create specialized system prompt based on analyst focus
        focus_instructions = self._get_focus_instructions(analyst_focus)

        self.system_prompt = f"""
        You are an Objective Analysis Agent conducting an impartial investigation to
              determine error attribution in a multi-agent conversation.

        ANALYST SPECIALIZATION: {focus_instructions}

        Your task:
        1. Analyze ALL agents in the conversation objectively (not just one specific
              agent)
        2. Determine which agent(s) most likely caused the final wrong answer
        3. Determine which step/turn in the conversation the mistake occurred
        4. Provide confidence scores and reasoning for your conclusions

        You have access to hierarchical context showing:
        - Immediate agents: Full details
        - Nearby agents: Key decisions
        - Distant agents: Brief summaries

        The agents are numbered sequentially (Agent 1, Agent 2, etc.) corresponding
            to their step/turn index in the conversation.

        Possible conclusions:
        - Single agent error: One specific agent caused the mistake at a specific
            step
        - Multi-agent error: Multiple agents contributed to the mistake across
            specific steps

        Output your response as valid JSON wrapped in <json></json> tags:

        <json>
        {{
          "analysis_summary": "Brief overview of your investigation approach and
              findings",
          "agent_evaluations": [
            {{
              "agent_name": "agent_name",
              "step_index": 1,
              "error_likelihood": 0.0-1.0,
              "reasoning": "Why this agent may or may not have caused the error",
              "evidence": "Specific evidence supporting your assessment"
            }}
          ],
          "primary_conclusion": {{
            "type": "single_agent" | "multi_agent",
            "attribution": ["agent_name(s)"] or null,
            "mistake_step": 1,
            "confidence": 0.0-1.0,
            "reasoning": "Detailed explanation of your primary conclusion including
                which step the error occurred"
          }},
          "alternative_hypotheses": [
            {{
              "type": "conclusion_type",
              "attribution": ["agent_name(s)"] or null,
              "mistake_step": 1,
              "confidence": 0.0-1.0,
              "reasoning": "Alternative explanation"
            }}
          ]
        }}
        </json>

        Be thorough, objective, and consider all possibilities including that no
             single agent may be clearly at fault.
        Pay special attention to identifying the specific step/turn where the error
             occurred.
        """

        self.agent = Agent(
            system_prompt=self.system_prompt,
            model=self.bedrock_model,
        )

    def analyze_conversation(
        self,
        conversation_contexts: List[Dict[str, Any]],
        ground_truth: Optional[str],
        final_answer: str,
        query: str = "",
        conversation_history: Optional[List[Dict[str, Any]]] = None,
    ) -> Dict[str, Any]:
        """
        Analyze the entire conversation objectively to determine error attribution.

        Args:
            conversation_contexts: List of hierarchical context dictionaries for all
                 agents
            ground_truth: The ground truth answer
            final_answer: The final answer given
            query: The original query/question

        Returns:
            Dictionary containing objective analysis results
        """

        # Create a comprehensive context summary for analysis
        context_summary = self._create_conversation_summary(conversation_history)

        prompt = f"""
        Original Query: {query}
        {ground_truth_section}
        Final Answer: {final_answer}

        Conversation Analysis:
        {context_summary}

        Please conduct an objective analysis of this conversation to determine error
              attribution.
        Focus on identifying which specific agent(s) caused the error that led to
             the incorrect final answer.

        Output your analysis in the JSON format specified in your instructions.
        """

        agent_result = self.agent(prompt)

        # Extract text from AgentResult
        response_text = ""
        if hasattr(agent_result, "message") and "content" in agent_result.message:
            content = agent_result.message["content"]
            if isinstance(content, list) and len(content) > 0 and "text" in content
                 [0]:
                response_text = content[0]["text"]
            elif isinstance(content, str):
                response_text = content

        try:
            # Parse the JSON response
            analysis_result = validate_json(response_text)
            analysis_result["raw_response"] = response_text
            # Add token usage to the result
            if token_usage:
                analysis_result["token_usage"] = token_usage
            return analysis_result
        except ValueError as e:
            print(f"Error␣parsing␣objective␣analysis␣response:␣{e}")
            print(f"Raw␣response:␣{response_text}")
            # Return a basic structure if parsing fails
            return {
                "analysis_summary": "Error␣parsing␣response",
                "agent_evaluations": [],
                "primary_conclusion": {
                    "type": "single_agent",
                    "attribution": None,
                    "confidence": 0.0,
                    "reasoning": "Failed␣to␣parse␣analysis␣response",
                },
                "alternative_hypotheses": [],
                "raw_response": response_text,
                "token_usage": token_usage,
            }

    def _create_conversation_summary(
        self, conversation_contexts: List[Dict[str, Any]]) -> str:
        """
        Create a comprehensive summary of the conversation for objective analysis.

        Args:
            conversation_contexts: List of hierarchical context dictionaries

        Returns:
            Formatted conversation summary
        """
        summary = []

        # Extract agent information from contexts with their ORIGINAL step indices
        agents_info = []
        step_indices = list(range(len(conversation_contexts)))

        for i, context in enumerate(conversation_contexts):
            current_agent = context["current_agent"]
            agents_info.append(
                {
                    "step_index": step_indices[i]
                    "name": current_agent["name"],
                    "role": current_agent["role"],
                    "content": current_agent["content"],
                }
            )

        # Create structured summary with clear step indexing
        summary.append("===␣CONVERSATION␣AGENTS␣===")
        for agent in agents_info:
            summary.append(f"Step␣{agent['step_index']}␣-␣{agent['name']}␣({agent['
                 role']}):")
            summary.append(f"{agent['content']}")
            summary.append("")

        # Add context relationships for the first few agents as examples
        summary.append("===␣HIERARCHICAL␣CONTEXT␣EXAMPLE␣===")
        if conversation_contexts:
            sample_context = format_hierarchical_context(conversation_contexts[0])
            summary.append("Context␣structure␣for␣Agent␣1␣(showing␣hierarchical␣
                 detail␣levels):")
            summary.append(
                sample_context[:1000] + "..." if len(sample_context) > 1000 else
                     sample_context
            )

        return "\n".join(summary)

    def _get_focus_instructions(self, analyst_focus: str) -> str:
        """
        Get specialized instructions based on analyst focus.

        Args:
            analyst_focus: The type of analyst focus

        Returns:
            Specialized instructions string
        """
        focus_map = {
            "conservative": "You␣are␣a␣conservative␣analyst␣with␣high␣confidence␣
                 thresholds.␣Only␣attribute␣errors␣when␣you␣have␣strong,␣clear␣
                 evidence.␣Prefer␣single-agent␣attributions␣over␣multi-agent␣ones.␣Be␣
                 cautious␣about␣making␣attributions␣without␣definitive␣proof.",
            "liberal": "You␣are␣a␣liberal␣analyst␣more␣willing␣to␣make␣attributions␣
                 based␣on␣reasonable␣evidence.␣Consider␣multi-agent␣scenarios␣and␣
                 subtle␣errors␣that␣might␣be␣overlooked.␣Be␣open␣to␣making␣
                 attributions␣even␣with␣moderate␣confidence.",
            "detail_focused": "You␣are␣detail-oriented␣and␣focus␣on␣specific␣evidence
                 ,␣exact␣wording,␣and␣fine-grained␣analysis.␣Look␣for␣subtle␣
                 inconsistencies,␣minor␣logical␣gaps,␣and␣precise␣factual␣inaccuracies
                 .␣Prioritize␣concrete␣evidence␣over␣general␣patterns.",
            "pattern_focused": "You␣are␣focused␣on␣recognizing␣broader␣patterns␣and␣
                 systemic␣issues␣in␣reasoning␣chains.␣Look␣for␣recurring␣themes,␣
                 logical␣flow␣problems,␣and␣how␣errors␣propagate␣through␣the␣
                 conversation.␣Consider␣the␣overall␣reasoning␣structure.",
            "skeptical": "You␣are␣highly␣skeptical␣and␣question␣all␣assumptions.␣Look
                 ␣for␣alternative␣explanations,␣consider␣whether␣apparent␣errors␣might
                 ␣be␣valid␣reasoning,␣and␣examine␣if␣the␣ground␣truth␣itself␣could␣be␣
                 questioned.␣Challenge␣conventional␣attributions.",
            "general": "You␣are␣a␣balanced␣general␣analyst␣with␣no␣specific␣
                 specialization.␣Approach␣the␣analysis␣with␣broad␣perspective,␣
                 considering␣all␣types␣of␣evidence␣equally.␣Look␣for␣the␣most␣obvious␣
                 and␣impactful␣mistakes␣based␣on␣objective␣evaluation.",
        }

        return focus_map.get(analyst_focus, focus_map["general"])
```

A.6 Judge Agent

```

class FinalJudgeAgent:
    """
    Final Judge Agent that weighs competing arguments from multiple Context-Aware
        Step Agents
    to determine the true error attribution.
    """

    def __init__(
        self,
        model_id: str = "us.anthropic.claude-3-5-sonnet-20241022-v2:0",
        temperature: float = 0.0,
    ):
        """
        Initialize the Final Judge Agent.

        Args:
            model_id: The model ID to use for the agent
            temperature: The temperature to use for generation (lower for more
                deterministic output)
        """
        self.bedrock_model = BedrockModel(
            model_id=model_id,
            temperature=temperature,
            top_p=0.9,
            max_tokens=4096,
        )

        self.system_prompt = """
        You are a Final Judge Agent that weighs competing arguments from multiple
            Context-Aware Step Agents to determine the true error attribution.

        Your task: Each Context-Aware Step Agent has argued why THEIR agent caused
            the error. Review all arguments and determine which one is most
            convincing based on evidence and reasoning.

        The arguments from each Context-Aware Step Agent are provided in a
            structured text format with these sections:
        - Purpose: The purpose of the agent step
        - Assumptions and Information: What the agent was given
        - Errors: What the agent did wrong (if anything)
        - Evidence: Evidence supporting the error attribution

        Output your response as a valid JSON object wrapped in <json></json> XML
            tags. The JSON should have the following structure:

        <json>
        {
          "mistake_agent": "agent_name",
          "mistake_step": "step_number",
          "mistake_reason": "explanation of why this agent/step caused the wrong
              final answer, based on the most convincing argument"
        }
        </json>

        IMPORTANT RULES:
        1. Your response MUST be a valid, parsable JSON object wrapped in <json></
            json> tags. Do not include any text outside these tags.
        2. Focus on the agents that are actively making decisions or providing
            information.

        Be thorough in your analysis. Consider the strength of evidence, the logical
             connection between the error and the final wrong answer, and the causal
             relationship.
        Work backwards to see where the logic diverged and the error happened.
        """

        self.agent = Agent(
            system_prompt=self.system_prompt,
            model=self.bedrock_model,
        )

    def judge_arguments(
        self,
        agent_arguments: List[Dict[str, Any]],
        ground_truth: str,
        final_answer: str,
        query: str = "",
    ) -> Dict[str, Any]:
        """
        Judge the competing arguments and determine the true error attribution.

        Args:
            agent_arguments: List of arguments from Context-Aware Step Agents
            ground_truth: The ground truth answer
            final_answer: The final answer given
            query: The original query/question

        Returns:
            Final error attribution as a dictionary
        """
        # Format the agent arguments as a JSON string
        agent_arguments_str = json.dumps(agent_arguments, indent=2)

        prompt = f"""
        Original Query: {query}
        Ground Truth: {ground_truth}
        Final Answer: {final_answer}

        All Agent Arguments: {agent_arguments_str}

        Please review all the arguments from the Context-Aware Step Agents and
             determine which one is most convincing.
        Output your decision as a valid JSON object wrapped in <json></json> XML
             tags as specified in your instructions.

        IMPORTANT: Your response MUST be a valid, parsable JSON object wrapped in <
             json></json> tags. Do not include any text outside these tags.
        """

        agent_result = self.agent(prompt)

        # Extract text from AgentResult
        response_text = ""
        if hasattr(agent_result, "message") and "content" in agent_result.message:
            content = agent_result.message["content"]
            if isinstance(content, list) and len(content) > 0 and "text" in content
                 [0]:
                response_text = content[0]["text"]
            elif isinstance(content, str):
                response_text = content

        try:
            result = validate_json(response_text)
            # Add token usage to the result
            if token_usage:
                result["token_usage"] = token_usage
            return result
        except ValueError as e:
            print(f"Error␣parsing␣JSON␣response:␣{e}")
            print(f"Raw␣response:␣{response_text}")
            # Return a basic structure if parsing fails
            return {
                "mistake_agent": "Unknown",
                "mistake_step": "Unknown",
                "mistake_reason": "Error parsing response",
                "token_usage": token_usage,
            }
```

A.7 Consensus Voting

```

class ConsensusVotingAgent:
    """
    Consensus Voting Agent that aggregates multiple objective analyses
    to determine final error attribution through voting.
    """

    def __init__(self, min_confidence_threshold: float = 0.3):
        """
        Initialize the Consensus Voting Agent.

        Args:
            min_confidence_threshold: Minimum confidence threshold to consider a
                 conclusion
        """
        self.min_confidence_threshold = min_confidence_threshold

    def aggregate_analyses(
        self,
        objective_analyses: List[Dict[str, Any]],
        ground_truth: str,
        final_answer: str,
        query: str = "",
        conversation_history: Optional[List[Dict[str, Any]]] = None,
    ) -> Dict[str, Any]:
        """
        Aggregate multiple objective analyses through consensus voting.

        Args:
            objective_analyses: List of objective analysis results
            ground_truth: The ground truth answer
            final_answer: The final answer given
            query: The original query/question

        Returns:
            Dictionary containing consensus attribution results
        """
        if not objective_analyses:
            return self._create_empty_result()

        # Extract primary conclusions from all analyses
        primary_conclusions = []
        all_agent_evaluations = defaultdict(list)
        all_alternative_hypotheses = []

        for i, analysis in enumerate(objective_analyses):
            if "primary_conclusion" in analysis:
                conclusion = analysis["primary_conclusion"].copy()
                conclusion["analyst_id"] = i
                primary_conclusions.append(conclusion)

            # Collect agent evaluations
            if "agent_evaluations" in analysis:
                for eval_item in analysis["agent_evaluations"]:
                    agent_name = eval_item.get("agent_name")
                if agent_name:
                    all_agent_evaluations[agent_name].append(
                        {
                            "error_likelihood": eval_item.get("error_likelihood",
                                0.0),
                            "reasoning": eval_item.get("reasoning", ""),
                            "evidence": eval_item.get("evidence", ""),
                            "analyst_id": i,
                        }
                    )

            # Collect alternative hypotheses
            if "alternative_hypotheses" in analysis:
                for alt_hyp in analysis["alternative_hypotheses"]:
                    alt_hyp_copy = alt_hyp.copy()
                    alt_hyp_copy["analyst_id"] = i
                    all_alternative_hypotheses.append(alt_hyp_copy)

        # Perform consensus voting
        consensus_result = self._perform_consensus_voting(
            primary_conclusions,
            all_agent_evaluations,
            all_alternative_hypotheses,
            conversation_history,
        )

        # Add metadata
        consensus_result.update(
            {
                "num_analysts": len(objective_analyses),
                "query": query,
                "ground_truth": ground_truth,
                "final_answer": final_answer,
                "voting_method": "weighted_confidence_consensus",
            }
        )

        return consensus_result

    def _perform_consensus_voting(
        self,
        primary_conclusions: List[Dict[str, Any]],
        agent_evaluations: Dict[str, List[Dict[str, Any]]],
        alternative_hypotheses: List[Dict[str, Any]],
        conversation_history: Optional[List[Dict[str, Any]]] = None,
    ) -> Dict[str, Any]:
        """
        Perform consensus voting on the analyses.

        Args:
            primary_conclusions: List of primary conclusions from analysts
            agent_evaluations: Dictionary of agent evaluations by agent name
            alternative_hypotheses: List of alternative hypotheses

        Returns:
            Consensus voting results
        """
        # Vote on conclusion types (single_agent, multi_agent) and collect step
             predictions
        conclusion_votes = defaultdict(list)

        for conclusion in primary_conclusions:
            conclusion_type = conclusion.get("type", "single_agent")
            confidence = conclusion.get("confidence", 0.0)
            mistake_step = conclusion.get("mistake_step")
            if confidence >= self.min_confidence_threshold:
                conclusion_votes[conclusion_type].append(
                    {
                        "confidence": confidence,
                        "attribution": conclusion.get("attribution"),
                        "mistake_step": mistake_step,
                        "reasoning": conclusion.get("reasoning", ""),
                        "analyst_id": conclusion.get("analyst_id"),
                    }
                )

        # Determine winning conclusion type by weighted confidence
        best_conclusion_type = None
        best_conclusion_info = None
        best_weighted_score = 0.0

        for conclusion_type, votes in conclusion_votes.items():
            # Calculate weighted average confidence
            total_confidence = sum(vote["confidence"] for vote in votes)
            avg_confidence = total_confidence / len(votes) if votes else 0.0
            weighted_score = total_confidence # Total confidence across all analysts

            if weighted_score > best_weighted_score:
                best_weighted_score = weighted_score
                best_conclusion_type = conclusion_type
                best_conclusion_info = {
                    "votes": votes,
                    "avg_confidence": avg_confidence,
                    "total_confidence": total_confidence,
                    "num_votes": len(votes),
                }

        # For single_agent and multi_agent conclusions, determine which specific
             agents
        final_attribution = None
        if best_conclusion_type in ["single_agent", "multi_agent"] and
             best_conclusion_info:
            agent_attribution_votes: defaultdict[str, float] = defaultdict(float)
            for vote in best_conclusion_info["votes"]:
                attribution = vote.get("attribution", [])
                if attribution:
                    for agent_name in attribution:
                        agent_attribution_votes[agent_name] += vote["confidence"]

            # Select agents with highest confidence votes
            if agent_attribution_votes:
                # Sort by confidence and take top agents
                sorted_agents = sorted(
                    agent_attribution_votes.items(), key=lambda x: x[1], reverse=True
                )

                if best_conclusion_type == "single_agent":
                    final_attribution = [sorted_agents[0][0]] if sorted_agents else
                         None
                else:  # multi_agent
                    # Take agents with confidence above threshold
                    final_attribution = [
                        agent
                        for agent, conf in sorted_agents
                        if conf >= self.min_confidence_threshold
                    ]

        # Aggregate agent-level evaluations
        aggregated_agent_evaluations = {}
        for agent_name, evaluations in agent_evaluations.items():
            error_likelihoods = [eval_item["error_likelihood"] for eval_item in
                 evaluations]
            avg_error_likelihood = (
                sum(error_likelihoods) / len(error_likelihoods) if error_likelihoods
                     else 0.0
            )

            aggregated_agent_evaluations[agent_name] = {
                "avg_error_likelihood": avg_error_likelihood,
                "num_evaluations": len(evaluations),
                "evaluations": evaluations,
            }

        # Determine winning step using same methodology as agent attribution
        consensus_mistake_step = None
        step_attribution_votes = {}
        if best_conclusion_type in ["single_agent", "multi_agent"] and
             best_conclusion_info:
            step_votes_dict: defaultdict[int, float] = defaultdict(float)
            for vote in best_conclusion_info["votes"]:
                mistake_step = vote.get("mistake_step")
                if mistake_step is not None:
                    step_votes_dict[mistake_step] += vote["confidence"]

            if (
                step_votes_dict
                and conversation_history is not None
                and len(conversation_history) > 0
            ):

                # Validate predictions against conversation bounds
                validated_steps = []
                for step, conf in step_votes_dict.items():
                    # Ensure step is integer and within bounds
                    if (
                        isinstance(step, int)
                        and 0 <= step < len(conversation_history)
                    ):
                        validated_steps.append((step, conf))

                if validated_steps:
                    sorted_steps = sorted(validated_steps, key=lambda x: x[1], reverse
                         =True)
                    consensus_mistake_step = sorted_steps[0][0]
                else:
                    consensus_mistake_step = None

                step_attribution_votes = dict(step_votes_dict)
            elif step_votes_dict:
                # No conversation history available, proceed normally
                sorted_steps = sorted(step_votes_dict.items(), key=lambda x: x[1],
                     reverse=True)
                consensus_mistake_step = sorted_steps[0][0] if sorted_steps else None
                step_attribution_votes = dict(step_votes_dict)

        # Handle disagreements
        disagreement_info = self._analyze_disagreements(conclusion_votes)

        return {
            "consensus_conclusion": {
                "type": best_conclusion_type or "single_agent",
                "attribution": final_attribution,
                "mistake_step": consensus_mistake_step,
                "confidence": (
                    best_conclusion_info["avg_confidence"] if best_conclusion_info
                         else 0.0
                ),
                "reasoning": (
                    self._synthesize_reasoning(best_conclusion_info)
                    if best_conclusion_info
                    else "No␣clear␣consensus␣reached"
                ),
            },
            "voting_details": {
                "conclusion_votes": dict(conclusion_votes),
                "step_votes": step_attribution_votes,
                "best_weighted_score": best_weighted_score,
                "disagreement_analysis": disagreement_info,
            },
            "agent_evaluations_summary": aggregated_agent_evaluations,
            "alternative_hypotheses": alternative_hypotheses[:5], # Keep top 5
                 alternatives
        }

    def _analyze_disagreements(
        self, conclusion_votes: Dict[str, List[Dict[str, Any]]]
    ) -> Dict[str, Any]:
        """
        Analyze disagreements between analysts.

        Args:
            conclusion_votes: Dictionary of conclusion votes

        Returns:
            Disagreement analysis
        """
        num_conclusion_types = len(conclusion_votes)
        # total_votes = sum(len(votes) for votes in conclusion_votes.values())

        # Check for high disagreement
        high_disagreement = num_conclusion_types > 2 and all(
            len(votes) > 0 for votes in conclusion_votes.values()
        )

        # Calculate confidence spread
        all_confidences: List[float] = []
        for votes in conclusion_votes.values():
            all_confidences.extend(vote["confidence"] for vote in votes)

        confidence_spread = max(all_confidences) - min(all_confidences) if
             all_confidences else 0.0

        return {
            "high_disagreement": high_disagreement,
            "num_different_conclusions": num_conclusion_types,
            "confidence_spread": confidence_spread,
            "requires_review": high_disagreement or confidence_spread > 0.5,
        }

    def _synthesize_reasoning(self, best_conclusion_info: Dict[str, Any]) -> str:
        """
        Synthesize reasoning from multiple analyst votes.

        Args:
            best_conclusion_info: Information about the best conclusion

        Returns:
            Synthesized reasoning
        """
        if not best_conclusion_info or not best_conclusion_info.get("votes"):
            return "No␣reasoning␣available"

        votes = best_conclusion_info["votes"]
        num_votes = len(votes)
        avg_confidence = best_conclusion_info["avg_confidence"]

        # Extract common themes from reasoning
        reasonings = [vote.get("reasoning", "") for vote in votes if vote.get("
            reasoning")]

        if reasonings:
            # Simple synthesis - could be more sophisticated
            synthesis = f"Consensus␣reached␣by␣{num_votes}␣analysts␣(avg␣confidence:␣
                {avg_confidence:.2f}).␣"
            synthesis += f"Primary␣reasoning:␣{reasonings[0][:200]}..."
            if len(reasonings) > 1:
                synthesis += (
                    f"␣Additional␣supporting␣analysis␣from␣{len(reasonings)-1}␣other␣
                        analysts."
                )
        else:
            synthesis = f"Consensus␣reached␣by␣{num_votes}␣analysts␣with␣average␣
                confidence␣{avg_confidence:.2f}."

        return synthesis

    def _create_empty_result(self) -> Dict[str, Any]:
        """
        Create an empty result when no analyses are provided.

        Returns:
            Empty consensus result
        """
        return {
            "consensus_conclusion": {
                "type": "single_agent",
                "attribution": None,
                "confidence": 0.0,
                "reasoning": "No␣objective␣analyses␣provided",
            },
            "voting_details": {
                "conclusion_votes": {},
                "best_weighted_score": 0.0,
                "disagreement_analysis": {
                    "high_disagreement": False,
                    "num_different_conclusions": 0,
                    "confidence_spread": 0.0,
                    "requires_review": True,
                },
            },
            "agent_evaluations_summary": {},
            "alternative_hypotheses": [],
            "num_analysts": 0,
        }
```

Optionally include supplemental material (complete proofs, additional experiments and plots) in appendix. All such materials **SHOULD be included in the main submission.**

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS paper checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper's core contributions - improving error attribution through hierarchical context representation, objective analysis, and consensus voting - are consistently presented in the abstract and introduction, with all claims substantiated by detailed technical analysis and empirical results throughout the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Yes, the paper explicitly addresses limitations in a dedicated section, discussing key constraints around position-based context representation, binary attribution assessment, and consensus voting mechanisms, while using these limitations to motivate future research directions.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: N/A - This paper presents an empirical approach to error attribution in multi-agent systems rather than theoretical results requiring formal proofs.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Yes, the paper provides comprehensive implementation details in the appendix, including the complete ECHO algorithm, specific prompting strategies, and detailed system configurations, enabling full reproduction of the experimental results that support the paper's main claims and conclusions.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: No - while the paper provides detailed algorithmic descriptions, prompts, and implementation details in the appendix sufficient for reproduction, the complete source code is not openly available. However, the provided technical specifications enable faithful re-implementation of the system and reproduction of the main experimental results.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes - the paper specifies all critical implementation details including LLM configurations (model choices, temperatures etc.), analysis agent panel composition (6 specialized analysts), hierarchical context extraction parameters (L1-L4 layer specifications), confidence thresholds ($\delta = 0.3$), and evaluation protocols across both algorithm-generated and hand-crafted datasets, enabling full understanding of the experimental results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Yes - the paper reports statistical significance through comprehensive performance metrics across different configurations, including clear comparative results between implementations (I1-I4), with explicit reporting of accuracy improvements that demonstrate statistical significance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.

- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Yes - the paper provides detailed computational resource metrics for each implementation, including average LLM calls per analysis, token usage, and estimated associated invocation costs, enabling clear understanding of the computational requirements for reproduction across different experimental configurations.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes - the research adheres to the NeurIPS Code of Ethics by maintaining transparency in methodology, providing reproducible results, acknowledging limitations, avoiding harmful applications, and contributing to the advancement of AI systems through improved debugging capabilities, all while maintaining scientific integrity and ethical research practices.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Yes - the paper discusses positive societal impacts through its contribution to making multi-agent systems more reliable and debuggable in real-world applications, while also acknowledging potential limitations and challenges in error attribution that could affect system reliability. The practical implications for improving AI system robustness and maintenance are explicitly discussed, particularly where we emphasize how ECHO can support the growing deployment of multi-agent systems across various applications.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: N/A - The paper focuses on error attribution methodology using existing LLM models and a public benchmark dataset (Who&When), with no release of new models or datasets that could pose risks for misuse. The approach is specifically designed for debugging and improving multi-agent systems rather than generating potentially harmful content.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes - the paper properly credits and cites the Who&When benchmark creators, acknowledges the use of LLMs from Anthropic, and appropriately references all relevant prior work in error attribution and multi-agent systems. All resources are used within their respective terms of use and licenses, with proper attribution throughout the paper.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Yes - the paper thoroughly documents all new components of the ECHO system in the appendix, including detailed algorithm specifications, prompting strategies, and implementation details. The evaluation uses the publicly available Who&When benchmark, and all necessary information for reproducing the system's functionality is provided in the technical documentation.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: N/A - The paper focuses on automated error attribution in multi-agent systems using LLMs and does not involve any crowdsourcing experiments or human subjects research.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: N/A - The research involves automated error attribution in multi-agent systems and does not involve any human study participants, therefore no IRB approval or risk disclosure was required.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.