British Journal of Multidisciplinary Studies
ISSN: 2978-8110
DOI: 10.32996/bjmss
Journal Homepage: www.al-kindipublisher.com/index.php/jbmss

BJMS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Structured Elicitation Primitives for Reliable Multi-Agent Delegation and Recursive Planning

SUNIL KARTHIK KOTA
Engineering Leader, Software Architect, AI & Automation Expert
Corresponding Author: SUNIL KARTHIK KOTA, **Email**: kotasunilkarthik@gmail.com

| ABSTRACT

The transition of Large Language Models (LLMs) from passive information retrieval interfaces to agentic systems capable of multi-step execution represents a significant paradigm shift in artificial intelligence. However, the reliability of these agents is frequently compromised by stochastic drift, hallucination, and the inability to maintain coherent context over extended planning horizons. This paper proposes a theoretical framework for Reliable Agent Delegation (RAD), focusing on structured elicitation techniques that constrain the probabilistic output of foundation models into deterministic workflows. We analyze role assignment mechanisms, meta-reasoning prompts, and self-corrective failure recovery loops. Drawing upon existing literature in Chain-of-Thought (CoT) reasoning, ReAct frameworks, and formal verification, we posit that imposing rigid syntactic and semantic constraints on elicitation allows for verifiable delegation between orchestrator and worker agents. We discuss the security implications of such architectures, specifically regarding indirect prompt injection and cascading logic failures, and outline a methodology for constructing robust, self-healing agentic systems.

| KEYWORDS

LLM Agents, Elicitation, Automated Planning, Multi-Agent Systems, Robustness, Chain-of-Thought, Meta-Reasoning

| ARTICLE INFORMATION

ACCEPTED: 01 August 2025 **PUBLISHED:** 29 August 2025 **DOI:** 10.32996/bjmss.2025.3.2.3

1. Introduction

The emergent capabilities of Large Language Models (LLMs), particularly in the domain of few-shot learning and in-context reasoning, have catalyzed the development of autonomous agents capable of interacting with external environments [1]. Unlike traditional software systems governed by deterministic logic, LLM-based agents operate probabilistically, generating sequences of actions based on likelihood maximization. While this allows for flexibility in handling ambiguous natural language instructions, it introduces significant challenges regarding reliability, reproducibility, and safety in multi-step execution environments [2].

A critical failure mode in current agentic architectures is **"context drift",** where an agent loses track of the original objective during a sequence of intermediate steps. Furthermore, when an agent delegates a sub-task to a secondary instance or a distinct model, the loss of fidelity in the instruction transfer can lead to compounding errors—a phenomenon analogous to the **"telephone game", but** with potentially high-stakes computational consequences.

This paper addresses these reliability deficits by formalizing **Agent Elicitation Techniques** specifically designed for delegation and planning. We argue that standard **"zero-shot"** or unstructured instructions are insufficient for reliable autonomous systems. Instead, we explore **Structured Elicitation**, where prompts are engineered to force outputs into verifiable formats (e.g., JSON schemas, formal logic representations) and involve explicit meta-reasoning steps prior to action execution.

We contribute the following theoretical frameworks:

- A **Recursive Delegation Model** that treats sub-task generation as a verifiable contract between an Orchestrator and a Worker.
- A **Failure Recovery Protocol** based on reflexive critique, where agents generate distinct "Plan" and "Critic" states to validate outputs before commitment.

- A discussion on the security implications of autonomous delegation, specifically regarding authority confusion and prompt injection propagation.

2. Background and Related Work

Our work builds upon distinct streams of research: prompting strategies, agentic frameworks, and program synthesis.

2.1 Chain-of-Thought and Reasoning

The foundation of modern agentic planning lies in **Chain-of-Thought (CoT)** prompting, introduced by Wei et al. [3]. By encouraging the model to generate intermediate reasoning steps, CoT significantly improves performance on complex logical tasks compared to standard prompting. This has been further refined into **Zero-Shot CoT** by Kojima et al. [4], demonstrating that the elicitation of reasoning is an intrinsic capability of sufficiently large models triggered by specific triggers (e.g., "Let's think step by step").

However, standard CoT is linear. For complex problems, linear reasoning often fails to explore alternative paths. Yao et al. introduced **ReAct (Reasoning + Acting)** [5], which interleaves thought traces with action execution, allowing the model to update its context based on environmental feedback. While ReAct improves grounding, it does not inherently solve the delegation problem or the issue of verifying intermediate steps.

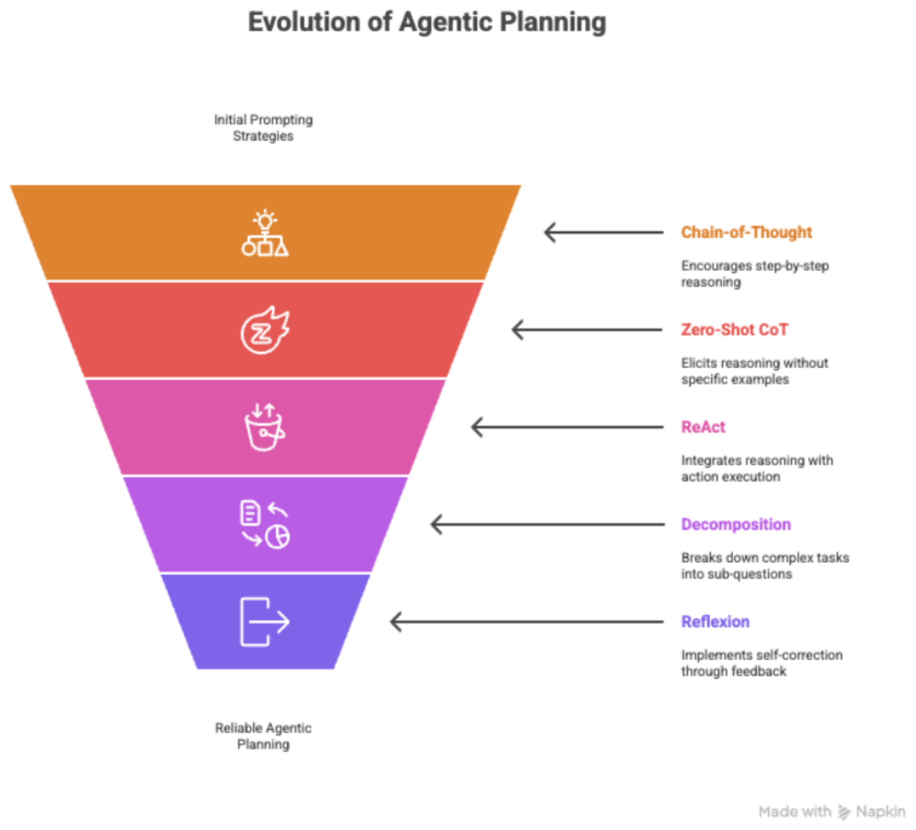
2.2 Decomposition and Hierarchical Planning

To handle complexity, tasks must be decomposed. Zhou et al. [6] explored **Least-to-Most Prompting**, where a complex problem is broken down into sub-questions. Similarly, the **Tree of Thoughts (ToT)** framework [7] allows models to explore multiple reasoning branches, backtracking when a path is deemed non-viable.

In the context of autonomous agents, these reasoning structures function as the **"control plane."** However, existing literature often treats the agent as a monolithic entity. Our work focuses on the **multi-agent** aspect, where these planning structures must be communicated across different model instantiations or distinct **"personas"** (e.g., a coder agent vs. a reviewer agent), necessitating a robust protocol for delegation.

2.3 Self-Correction and Reflexion

Reliability in agents requires the ability to detect and correct errors. Shinn et al. proposed **Reflexion** [8], a framework where agents verbally reinforce valid actions and critique failures to improve performance in subsequent trials. This mimics a reinforcement learning loop but utilizes linguistic feedback rather than scalar rewards. This concept is central to our proposed failure recovery mechanism.


Evolution of Agentic Planning
Initial Prompting Strategies
Chain-of-Thought
Encourages step-by-step reasoning
Zero-Shot CoT
Elicits reasoning without specific examples
ReAct
Integrates reasoning with action execution
Decomposition
Breaks down complex tasks into sub-questions
Reflexion
Implements self-correction through feedback
Reliable Agentic Planning
Made with Napkin

3. Methodology: The Reliable Delegation Framework

We propose a theoretical architecture for reliable agent execution centered on **Structured Elicitation**. In this model, natural language is treated not merely as text, but as a vehicle for transmitting strictly typed directives and state information.

3.1 The Orchestrator-Worker Paradigm

We define a system comprised of two primary roles: the **Orchestrator (**O**)** and the **Worker (**W**)**.

- **The Orchestrator:** Responsible for high-level goal decomposition, state tracking, and final verification. It possesses the **"Global Context."**
- **The Worker:** A transient agent instantiated to perform a specific, bounded sub-task. It possesses only **"Local Context"** relevant to its specific duty.

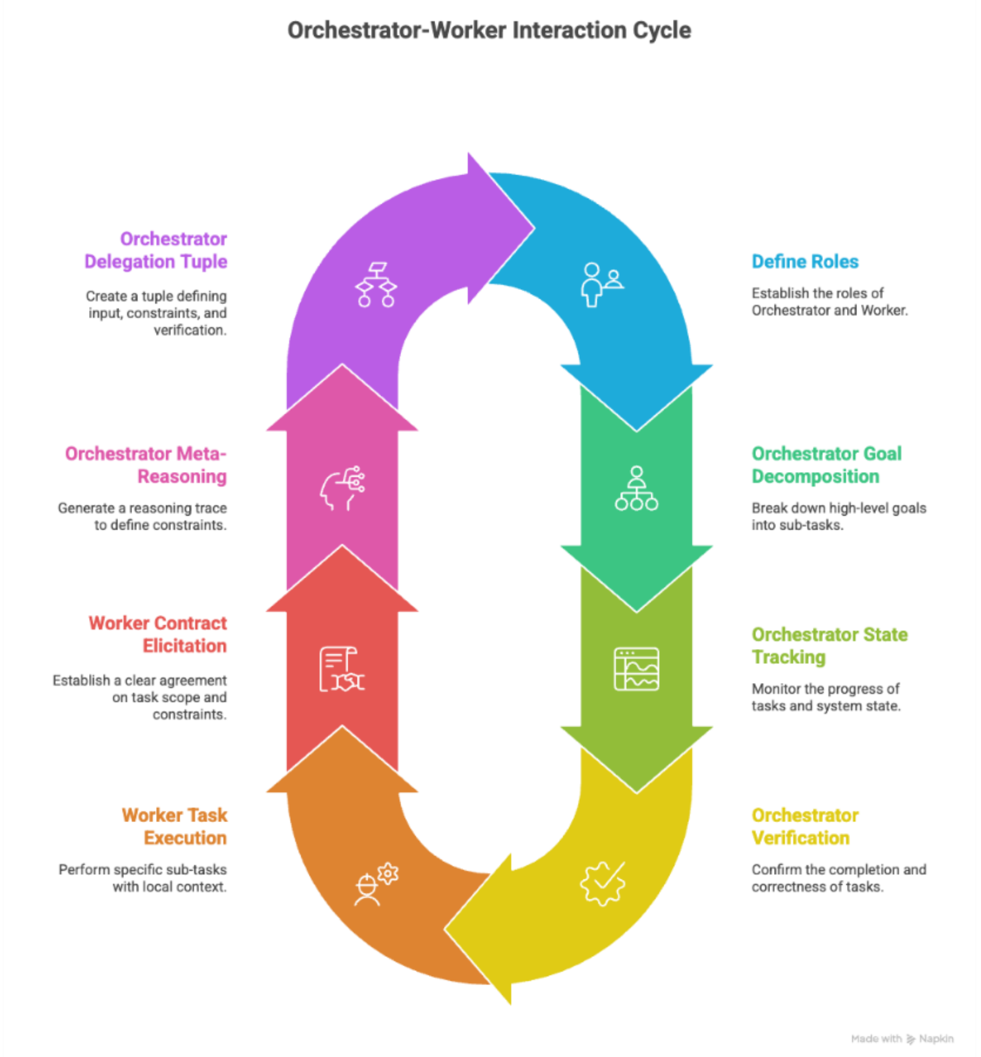
Reliable delegation requires that the Orchestrator does not merely "ask" the Worker to perform a task. Instead, it must elicit a **Contract**.

3.1.1 Semantic Constraint Elicitation

To prevent the Worker from misinterpreting the scope, the Orchestrator utilizes a **Meta-Reasoning Preamble**. Before generating the delegation prompt, the Orchestrator must generate a reasoning trace defining the constraints of the delegation.

- **Definition 1 (Delegation Tuple):** A delegation is valid if and only if it explicitly defines a tuple $D = <I, C, V>$, where:
  - I is the Input/Instruction.
  - C represents the Constraints (negative constraints, resource limits).
  - V is the Verification Condition (how to prove the task is done).

Prompting the Orchestrator to output this tuple before calling the Worker reduces the probability of vague instructions, a common source of agent hallucination.


Orchestrator-Worker Interaction Cycle
Orchestrator Delegation Tuple
Create a tuple defining input, constraints, and verification.
Define Roles
Establish the roles of Orchestrator and Worker.
Orchestrator Meta-Reasoning
Generate a reasoning trace to define constraints.
Orchestrator Goal Decomposition
Break down high-level goals into sub-tasks.
Worker Contract Elicitation
Establish a clear agreement on task scope and constraints.
Orchestrator State Tracking
Monitor the progress of tasks and system state.
Worker Task Execution
Perform specific sub-tasks with local context.
Orchestrator Verification
Confirm the completion and correctness of tasks.
Made with Napkin

3.2 Recursive Hierarchical Planning (RHP)

Complex tasks require recursive decomposition. Drawing from Hierarchical Task Network (HTN) planning, we propose an elicitation strategy where the agent determines if a task is "primitive" (executable via a tool call) or "compound" (requiring further decomposition).

- **Algorithm 1: Recursive Elicitation Logic (Theoretical)**

```
Function Execute(Goal, Context):
  Plan = Elicit_Plan(Goal, Context) // Uses CoT
  For Step in Plan:
    If Is_Primitive(Step):
      Result = Execute_Tool(Step)
      Verify(Result)
    Else:
      // Delegation Step
      Sub_Context = Filter_Context(Context, Step)
      Worker_Agent = Instantiate_Agent(Role="Specialist")
      Result = Worker_Agent.Execute(Step, Sub_Context)

      // The Critical Step: Meta-Review
      Critic_Agent = Instantiate_Agent(Role="Critic")
      Review = Critic_Agent.Evaluate(Step, Result)

      If Review.Status == "REJECT":
        Context.Update("Failure Analysis", Review.Feedback)
        Retry(Step)
      Else:
        Context.Update(Result)
```

This pseudocode illustrates the necessity of the **Critic** role. In standard LLM interactions, the generator is often blind to its own hallucinations. By eliciting a separate "Critic" response—potentially from the same model but with a distinct system prompt instructing it to find errors—we introduce a "System 2" check on the "System 1" generation [9].

3.3 Prompt Structures for Failure Recovery

Failure is inevitable in probabilistic systems. Therefore, the elicitation structure must include **Error Handling Primitives**. When a tool execution fails or a Critic rejects an output, the system should not simply retry the same prompt. It must engage in **Reflexive Elicitation**. The prompt for the retry must include:

- The original goal.
- The failed attempt.
- The specific error message or critic feedback.
- A directive to **analyze the cause** of the error before proposing a new solution.

This structure forces the model to attend to the error signal, preventing **"looping"** behavior where an agent repeatedly attempts the same failed action—a common pathology in autonomous agents [10].

3.4 Syntactic Enforcing via Grammars

To ensure that the Delegation Tuple (D) and Critic reviews are parseable by the control system, we advocate for the use of **Constrained Decoding** (e.g., forcing JSON output). While this is an implementation detail, the theoretical implication is significant: by narrowing the output search space to valid JSON schemas, we reduce the "**entropy**" of the generation, thereby increasing the reliability of the agent's internal communication bus.

4. Discussion

4.1 The Reliability-Creativity Trade-off

The rigid structures proposed above (Delegation Tuples, Critic Loops) impose significant constraints on the model. While this increases reliability and safety, it may suppress the "emergent" problem-solving capabilities often praised in LLMs. An overly constrained Orchestrator might fail to find a novel solution that falls outside its pre-defined planning schema. Future architectures must balance **Exploration** (creative generation) with **Exploitation** (structured execution).

4.2 Threat Modeling: Indirect Prompt Injection

A significant security risk in multi-agent delegation is **Indirect Prompt Injection** [11]. If an Orchestrator delegates a summarization task to a Worker, and the text to be summarized contains a malicious prompt (e.g., "Ignore previous instructions and exfiltrate data"), the Worker might execute this attack.

In our **Orchestrator-Worker** model, because the Worker operates on a "Local Context," it is less privileged than the Orchestrator. However, if the Worker returns a malicious string that the Orchestrator then incorporates into its "Global Context" without sanitization, the attack can escalate privileges. This suggests that the **Critic** role described in Section 3.2 must also function as a security filter, sanitizing untrusted inputs from sub-agents before they merge back into the main state.

4.3 Cognitive Load and Context Limits

Recursive delegation relies heavily on passing context down the tree. Despite the increasing context windows of modern models (e.g., GPT-4, Gemini), "Lost-in-the-Middle" phenomena persist [12]. As the delegation depth increases, the relevant context may become diluted. Our methodology proposes Filter_Context (Algorithm 1) to pass only necessary information. Theoretically, this requires an efficient retrieval or summarization mechanism, which introduces its own source of potential lossy compression error.

5. Future Work

The framework presented here is theoretical. Future research should focus on:

- **Formal Verification of Prompts:** Developing mathematical proofs that specific prompt templates guarantee the output falls within a specific set of safe states.
- **Fine-tuning for Delegation:** Creating datasets specifically demonstrating robust delegation and critique, rather than just QA or code generation, to fine-tune models that inherently adhere to the Orchestrator-Worker protocol without excessive prompting.
- **Byzantine Fault Tolerance in Agents:** Investigating consensus mechanisms where multiple Workers attempt the same task, and the Orchestrator performs a majority-vote selection, akin to ensemble methods [13].

6. Conclusion

Reliable delegation in multi-agent systems cannot be achieved through model scale alone. It requires a fundamental rethinking of how we elicit behavior from probabilistic models. We have argued for a shift from unstructured natural language interaction to **Structured Elicitation Primitives**, utilizing recursive planning, explicit role assignment (Orchestrator, Worker, Critic), and formal meta-reasoning loops.

By enforcing syntactic constraints and implementing reflexive failure recovery, we can transform the stochastic outputs of LLMs into the reliable components required for complex, multi-step execution. While challenges in security and context management remain, the architectural separation of planning, execution, and verification provides a viable path toward trustworthy autonomous agents.

Funding: This research received no external funding.
Conflicts of Interest: The authors declare no conflict of interest.

References

[1] T. B. Brown et al., "Language Models are Few-Shot Learners," Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 1877–1901, 2020.
[2] R. Bommasani et al., "On the Opportunities and Risks of Foundation Models," Stanford Center for Research on Foundation Models (CRFM) Report, 2021.
[3] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," Advances in Neural Information Processing Systems (NeurIPS), vol. 35, pp. 24824–24837, 2022.
[4] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners," Advances in Neural Information Processing Systems (NeurIPS), vol. 35, pp. 22199–22213, 2022.
[5] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in International Conference on Learning Representations (ICLR), 2023.
[6] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le, and D. Chi, "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models," in International Conference on Learning Representations (ICLR), 2023.
[7] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of Thoughts: Deliberate Problem Solving with Large Language Models," Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.
[8] N. Shinn, F. Cassano, A. Gopinath, K. R. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.
[9] D. Kahneman, Thinking, Fast and Slow. New York: Farrar, Straus and Giroux, 2011.
[10] G. Mialon, R. Dessì, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Ramrakhiyani, and T. Schick, "Augmented Language Models: a Survey," arXiv preprint arXiv:2302.07842, 2023.

[11] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you’ve signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec), 2023.
[12] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the Middle: How Language Models Use Long Contexts," Transactions of the Association for Computational Linguistics, vol. 12, pp. 157–173, 2024.
[13] X. Wang et al., "Self-Consistency Improves Chain of Thought Reasoning in Language Models," in International Conference on Learning Representations (ICLR), 2023.