Stateful Monitoring and Responsible Deployment of AI Agents

Debmalya Biswas
Wipro AI, Switzerland

Keywords: Multi-Agent Systems, Autonomous Agents, Agent Architecture, Monitoring, Stateful Execution, Responsible AI.

Abstract: AI agents can be disruptive given their potential to compose existing models and agents. Unfortunately, developing and deploying multi-agent systems at scale remains a challenging problem. In this paper, we specifically focus on the challenges of monitoring stateful agents and deploying them in a responsible fashion. We introduce a reference architecture for AI agent platforms, highlighting the key components to be considered in designing the respective solutions. From an agent monitoring perspective, we show how a snapshot based algorithm can answer different types of agent execution state related queries. On the responsible deployment aspect, we show how responsible AI dimensions relevant to AI agents can be integrated in a seamless fashion with the underlying AgentOps pipelines.

1 INTRODUCTION

In the generative AI context, Auto-GPT (Significant Gravitas, 2023) is representative of an autonomous AI agent that can execute complex tasks, e.g., make a sale, plan a trip, make a flight booking, book a contractor to do a house job, order a pizza. Given a user task, Auto-GPT aims to identify (compose) an agent (group of agents) capable to executing the given task.

AI agents (Park et al., 2023) follow a long history of research around multi-agent systems (MAS) (Weiss, 2016), esp., goal oriented agents (Bordes et al., 2017; Yan et al., 2015). A high-level approach to solving such complex tasks involves: (a) decomposition of the given complex task into (a hierarchy or workflow of) simple tasks, followed by (b) composition of agents able to execute the simpler tasks. This can be achieved in a dynamic or static manner. In the dynamic approach, given a complex user task, the system comes up with a plan to fulfill the request depending on the capabilities of available agents at run-time. In the static approach, given a set of agents, composite agents are defined manually at design-time combining their capabilities.

Unfortunately, designing and deploying AI agents remains challenging in practice. In this paper, we focus on primarily two aspects of AI agent platforms:

- given the complex and long-running nature of AI agents, we discuss approaches to ensure a reliable and stateful AI agent execution.
- adding the responsible AI dimension to AI agents. We highlight issues specific to AI agents and propose approaches to establish an integrated AI agent platform governed by responsible AI practices.

The rest of the paper is organized as follows. In Section 2, we introduce a reference architecture for AI agent platforms, highlighting the key components to be considered in designing the following solutions. In Section 3, we identify the key challenges to monitor stateful AI agents, and outline a snapshot based algorithm that can answer the relevant agent execution state related queries. We consider responsible deployment of agents in Section 4, showing how responsible AI dimensions can be integrated in the underlying AgentOps pipelines. Finally, Section 5 concludes the paper and provides some directions for future work.

2 AGENT AI PLATFORM REFERENCE ARCHITECTURE

In this section, we focus on identifying the key components of a reference AI agent platform illustrated in Fig. 1:

- Reasoning layer
- Agent marketplace
- Integration layer

Biswas, D.
Stateful Monitoring and Responsible Deployment of AI Agents.
DOI: 10.5220/0013160300003890

In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 1*, pages 393-399
ISBN: 978-989-758-737-5; ISSN: 2184-433X

- Shared memory layer
- Governance layer, including explainability, privacy, security, etc.

Given a user task, the goal of an AI agent platform is to identify (compose) an agent (group of agents) capable to executing the given task. So the first component that we need is a reasoning layer capable of decomposing a task into sub-tasks, with execution of the respective agents orchestrated by an orchestration engine.

Chain of Thought (CoT) (Wei et al., 2022) is the most widely used decomposition framework today to transform complex tasks into multiple manageable tasks and shed light into an interpretation of the model's thinking process. CoT can be implemented using two approaches: user prompting and automated approach.

- User Prompting: Here, during prompting, user provides the logic about how to approach a certain problem and LLM will solve similar problems using same logic and return the output along with the logic.
- Automating Chain of Thought Prompting: Manually handcrafting CoT can be time consuming and provide sub-optimal solution, Automatic Chain of Thought (Auto-CoT) (Zhang et al., 2022) can be leveraged to generate the reasoning chains automatically thus eliminating the human intervention.

Tree of Thoughts (Yao et al., 2023) extends CoT by exploring multiple decomposition possibilities in a structured way. From each thought, it can branch out and generate multiple next-level thoughts, creating a tree-like structure that can be explored by BFS (breadth-first search) or DFS (depth-first search) with each state evaluated by a classifier (via a prompt) or majority vote.

Agent composition implies the existence of an agent marketplace / registry of agents - with a well-defined description of the agent capabilities and constraints. For example, let us consider a house painting agent C whose services can be reserved online (via credit card). Given this, the fact that the user requires a valid credit card is a constraint, and the fact that the user's house will be painted within a certain timeframe are its capabilities. In addition, we also need to consider any constraints of C during the actual execution phase, e.g., the fact that C can only provide the service on weekdays (and not on weekends). In general, constraints refer to the conditions that need to be satisfied to initiate an execution and capabilities reflect the expected outcome after the execution terminates.

In the context of MAS, specifically, previous works (Capezzuto et al., 2021; Trabelsi et al., 2022; Veit et al., 2001) have considered agent limitations during the discovery process. (Veit et al., 2001) proposes a configurable XML based framework called GRAPPA (Generic Request Architecture for Passive Provider Agents) for agent matchmaking. (Capezzuto et al., 2021) specifies a compact formulation for multi-agent task allocation with spatial and temporal constraints. (Trabelsi et al., 2022) considers agent constraints in the form of incompatibility with resources. The authors then propose an optimal matchmaking algorithm that allows the agents to relax their restrictions, within a budget. Refer to (Biswas., 2024) for a detailed discussion on the discovery aspect of AI agents.

Given the need to orchestrate multiple agents, we also need an integration layer supporting different agent interaction patterns, e.g., agent-to-agent API, agent API providing output for human consumption, human triggering an AI agent, AI agent-to-agent with human in the loop. The integration patterns need to be supported by the underlying AgentOps platform.

To accommodate multiple long-running agents, we also need a shared long-term memory layer enabling data transfer between agents, storing interaction data such that it can be used to personalize future interactions. The standard approach here is to save the embedding representation of agent information into a vector database that can support maximum inner product search (MIPS). For fast retrieval, the approximate nearest neighbors (ANN) algorithm is used that returns approximately top k-nearest neighbors with an accuracy trade-off versus a huge speed gain.

Finally, the governance layer. We need to ensure that data shared by the user specific to a task, or user profile data that cuts across tasks; is only shared with the relevant agents (authentication and access control). We further consider the different responsible AI dimensions in terms of data quality, privacy, reproducibility and explainability to enable a well governed AI agent platform.

3 STATEFUL AGENT MONITORING

Stateful execution (Lu et al., 2024) is an inherent characteristic of any distributed systems platform, and can be considered as a critical requirement to materialize the orchestration layer of an AI agent platform. Given this, we envision that agent monitoring together with failure recovery will become more and more critical as AI agent platforms become enterprise ready,

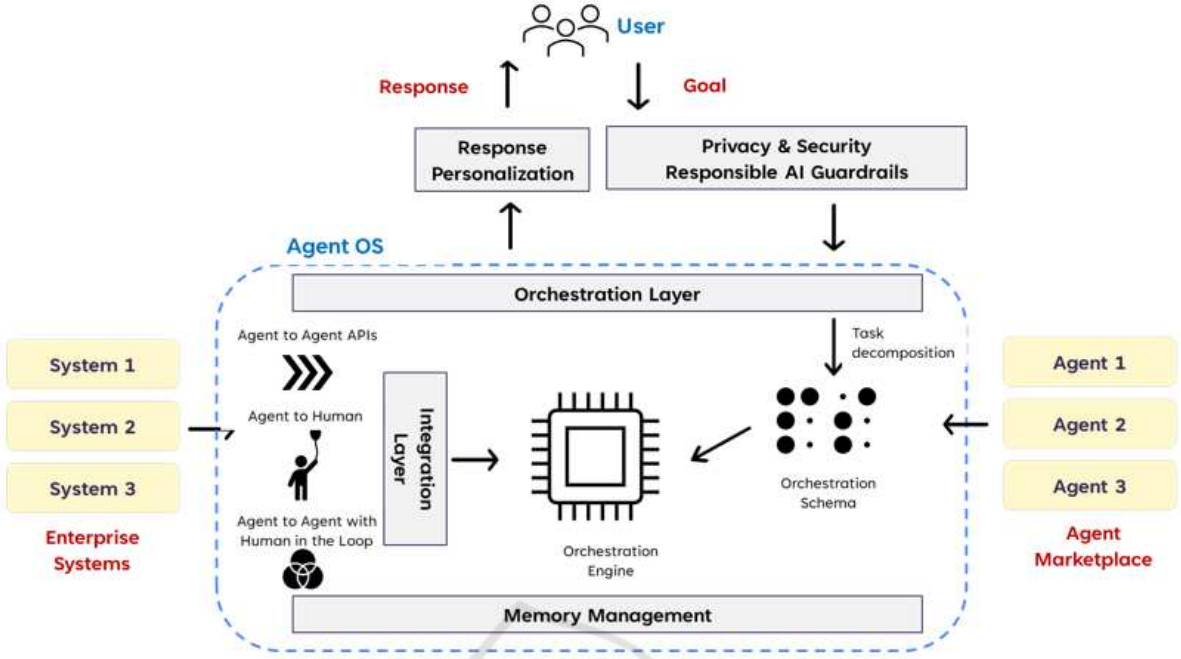

Figure 1: AI agent platform reference architecture.

and start supporting productionized deployments of AI agents.

However, monitoring AI agents (similar to monitoring large-scale distributed systems) is challenging because of the following reasons:

- No global observer: Due to their distributed nature, we cannot assume the existence of an entity having visibility over the entire execution. In fact, due to their privacy and autonomy requirements, even the composite agent may not have visibility over the internal processing of its component agents.
- Non-determinism: AI agents allow parallel composition of processes. Also, AI agents usually depend on external factors for their execution. As such, it may not be possible to predict their behavior before the actual execution. For example, whether a flight booking will succeed or not depends on the number of available seats (at the time of booking) and cannot be predicted in advance.
- Communication delays: Communication delays make it impossible to record the states of all the involved agents instantaneously. For example, let us assume that agent A initiates an attempt to record the state of the composition. Then, by the time the request (to record its state) reaches agent B and B records its state, agent A's state might have changed.
- Dynamic configuration: The agents are selected incrementally as the execution progresses (dynamic binding). Thus, the "components" of the distributed system may not be known in advance.

To summarize, agent monitoring is critical given the complexity and long running nature of AI agents. We define agent monitoring as *the ability to find out where in the process the execution is and whether any unanticipated glitches have appeared*. We discuss the capabilities and limitations of acquiring agent execution snapshots with respect to answering the following types of queries:

- Local queries: Queries which can be answered based on the local state information of an agent. For example, queries such as "What is the current state of agent A's execution?" or "Has A reached a specific state?". Local queries can be answered by directly querying the concerned agent provider.
- Composite queries: Queries expressed over the states of several agents. We assume that any query related to the status of a composition is expressed as a conjunction of the states of individual agent executions. Examples of status queries: "Have agents A, B and C reached states x, y and z respectively?" Such queries have been referred to as stable predicates in literature. Stable predicates are defined as predicates which do not become false once they have become true.
- Historical queries: Queries related to the execution history of the composition. For example,

"How many times have agents A and B been suspended?". If the query is answered using an execution snapshot algorithm, then it needs to be mentioned that the results are with respect to a time in the past.

- Relationship queries: Queries based on the relationship between states. For example, "What was the state of agent A when agent B was in state y?" Unfortunately, execution snapshot based algorithms do not guarantee answers for such queries. For example, we would not be able to answer the query unless we have a snapshot which captures the state of agent B when it was in state y. Such predicates have been referred to as unstable predicates in literature. Unstable predicates keep alternating their values between true and false - so are difficult to answer based on snapshot algorithms.

We outline the AI agent monitoring approach and solution architecture in the next section.

3.1 Agent Snapshot Monitoring

We assume the existence of a coordinator and log manager corresponding to each agent as shown in Fig. 2. We also assume that each agent is responsible for executing a single task.

The coordinator is responsible for all non-functional aspects related to the execution of the agent such as monitoring, transactions, etc. The log manager logs information about any state transitions as well as any messages sent/received by the agent. The state transitions and messages considered are as outlined in Fig. 3:

- Not - Executing (NE): The agent is waiting for an invocation.
- Executing (E): On receiving an invocation message (IM), the agent changes its state from NE to E.
- Suspended (S) and suspended by invoker (IS): An agent, in state E, may change its state to S due to an internal event (suspend) or to IS on the receipt of a suspend message (SM). Conversely, the transition from S to E occurs due to an internal event (resume) and from IS to E on receiving a resume message (RM).
- Canceling (CI), canceling due to invoker (ICI) and canceled (C): An agent, in state E/S/IS, may change its state to CI due to an internal event (cancel) or ICI on the receipt of a cancel message (CM). Once it finishes cancellation, it changes its state to C and sends acCanceled message (CedM) to its parent. Note that cancellation may require canceling the effects of some of its component agents.
- Terminated (T) and compensating (CP): The agent changes its state to T once it has finished executing the task. On termination, the agent sends a terminated message (TM) to its parent. An agent may be required to cancel a task even after it has finished executing the task (compensation). An agent, in state T, changes its state to CP on receiving the CM. Once it finishes compensation, it moves to C and sends a CedM to its parent agent.

We assume that the composition schema (static composition) specifies a partial order for agent tasks. We define the happened-before relation between agent tasks as follows:

A task a happened-before task $b (a \rightarrow b)$ if and only if one of the following holds:

1. there exists a control / data dependency between tasks a and b such that a needs to terminate before b can start executing.
2. there exists a task c such that $a \rightarrow c$ and $c \rightarrow b$.

A task, on failure, is retried with the same or different agents until it completes successfully (terminates). Note that each (retrial) attempt is considered as a new invocation and would be logged accordingly. Finally, to accommodate asynchronous communication, we assume the presence of input/output (I/O) queues. Basically, each agent has an I/O queue with respect to its parent and component agents - as shown in Fig. 3.

Given synchronized clocks and logging (as discussed above), a snapshot of the hierarchical composition at time t would consist of the logs of all the "relevant" agents until time t.

The relevant agents can be determined in a recursive manner (starting from the root agent) by considering the agents of the invoked tasks recorded in the parent agent's log until time t. If message timestamps are used then we need to consider the skew while recording the logs, i.e., if a parent agent's log was recorded until time t then its component agents' logs need to be recorded until $(t + skew)$. The states of the I/O queues can be determined from the state transition model.

4 RESPONSIBLE AGENTS

The growing adoption of generative AI, esp. with respect to the adoption of large language models (LLMs), has reignited the discussion around responsi-

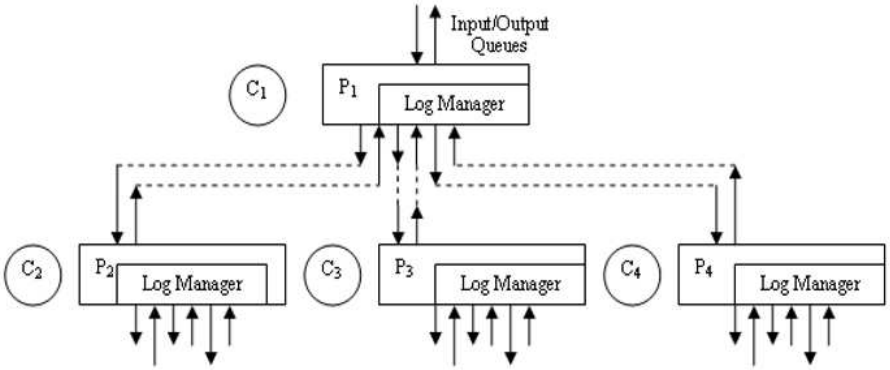

Figure 2: Agent monitoring infrastructure.

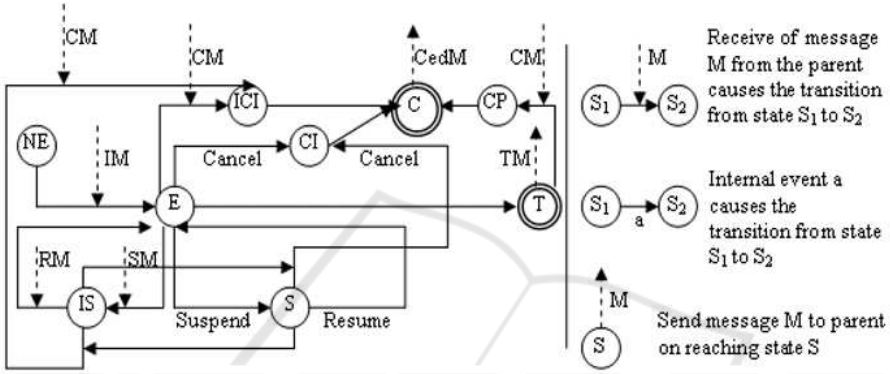

Figure 3: Agent execution lifecycle.

ble AI - to ensure that AI/ML systems are responsibly trained and deployed.

The table in Fig. 4 summarizes the key challenges and solutions in implementing responsible AI for AI agents in comparison to

- ChatGPT style large language model (LLM) APIs
- LLM fine-tuning: LLMs are generic in nature. To realize the full potential of LLMs for enterprises, they need to be contextualized with enterprise knowledge captured in terms of documents, wikis, business processes, etc. This is achieved by fine-tuning an LLM with enterprise knowledge / embeddings to develop a context-specific small language model (SLM)
- Retrieval-augmented-generation (RAG): Fine-tuning is a computationally intensive process. RAG provides a viable alternative by providing additional context with the prompt, grounding the retrieval and responses to the given context. The prompts can be relatively long, so it is possible to embed enterprise context within the prompt itself.

We expand on the above points in the rest of the paper to enable an integrated AgentOps pipeline with responsible AI governance.

Data consistency: The data used for training (esp., fine-tuning) the LLM should be accurate and precise, which means the relevant data pertaining to the specific use-case should be used to train the LLMs, e.g. if the use case is to generate summary of a medical prescription - the user should not use other data like Q&A of a diagnosis; user must use only medical prescriptions and corresponding summarization of the prescription. Many a times, data pipelines need to be created to ingest the data and feed that to LLMs. In such scenarios, extra caution needs to be exercised to consume the running text fields as these fields hold mostly inconsistent and incorrect data.

Bias/Fairness: With respect to model performance and reliability, it is difficult to control undesired biases in black-box LLMs, though it can be controlled to some extent by using uniform and unbiased data to fine-tune the LLMs and/or contextualize the LLMs in a RAG architecture.

Accountability: To make LLMs more reliable, it is recommended to have manual validation of the LLM's outputs. Involving humans ensures if LLMs hallucinate or provide wrong response, a human can evaluate and make the necessary corrections.

Hallucination: In case of using LLM APIs or orchestrating multiple AI agents, hallucination likelihood increases with the increase in the number of

Responsible AI	Factors	LLM APIs	Fine-tuned LLMs	LLMs with RAG	AI Agents
Reliability	Data Consistency	Adherence to the consistency of data during prompting	The training data should be consistent and balanced	The data should be aligned with the prompting and be consistent	Adherence to the consistency of data during prompting and while passing to other models
	Bias/Fairness	Prebuilt LLMs can perpetuate and amplify harmful biases present in the training data.	Fine-tuning the LLMs with unbiased data reduces the chance of unfair responses	RAGs with unbiased data reduces the chance of unfair responses	Chances of unfair and biased responses can get amplified by using multiple LLMs, but can be reduced by using prompts that contain unbiased data
	Hallucination	Hallucinations are likely as the model gives responses based on large training data	Reduced to some extent as the model is re-trained with curated enterprise data	Reduced to a significant extent by limiting the space of the generated responses	Hallucination likelihood is amplified as a result of using multiple LLMs
	Accountability	Human should be in the loop while training the LLMs or during build phase so that the output of the model can be verified before deployment, also human feedback is leveraged for continuous improvement of the model.			
Reproducibility	Evaluation during Training	LLMs' performance can be evaluated either by manual testing by keeping humans in loop or using statistical measures. There are different statistical measures available to evaluate the performance of LLMs: Perplexity, BLEU, ROGUE etc.			
	Inference Evaluation	Metrics available to measure the LLM performance during productionization in terms of handling incoming requests are: Completed requests per minute, Time to first token (TTFT), Inter-token latency (ITL), End-to-end Latency, etc.			
Explainability	Chain of Thought(CoT)/Provide Evidence	CoT prompting can be used to provide the logic behind the LLM response.	Adjust training data such that the LLM response consists of the CoT as well	RAG plus CoT prompting can solve the gap of providing logic to the LLM response	Difficult to produce the underlying logic as this involves multiple LLMs

Figure 4: Responsible AI challenges for Agentic AI.

agents involved. The right prompts can help but only to a limited extent. To further limit the hallucination, LLMs need to be fine-tuned with curated data and/or limit the search space of responses to relevant and recent enterprise data.

Data Privacy: With respect to conversational privacy (Biswas, 2020), we need to consider the privacy aspects of enterprise data provided as context (RAGs) and/or enterprise data used to fine-tune the LLMs. In addition, the novel privacy aspect here is to consider the privacy risks of data (prompts) provided voluntarily by the end-users, which can potentially be used as training data to re-train / fine-tune the LLMs.

5 CONCLUSION

Agentic AI is a disruptive technology, and there is currently a lot of interest and focus in making the underlying agent platforms ready for enterprise adoption. Towards this end, we outlined a reference architecture for AI agent platforms. We primarily focused on two aspects critical to enable scalable and responsible adoption of AI agents - an AgentOps pipeline integrated with monitoring and responsible AI principles.

From an agent monitoring perspective, we focused on the challenge of capturing the state of a (hierarchical) multi-agent system at any given point of time (snapshot). Snapshots usually reflect a state of a distributed system which "might have occurred". Towards this end, we discussed the different types of agent execution related queries and showed how we can answer them using the captured snapshots.

To enable responsible deployment of agents, we highlighted the responsible AI dimensions relevant to AI agents; and showed how they can be integrated in a seamless fashion with the underlying AgentOps pipelines. We believe that these will effectively future-proof agentic AI investments and ensure that AI agents are able to cope as the AI agent platform and regulatory landscape evolves with time.

REFERENCES

Biswas, D. (2020). Privacy Preserving Chatbot Conversations. In *IEEE Third International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 179–182.

Biswas., D. (2024). Constraints Enabled Autonomous Agent Marketplace: Discovery and Matchmaking. In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART)*, pages 396–403.

Bordes, A., Boureau, Y.-L., and Weston, J. (2017). Learning End-to-End Goal-Oriented Dialog.

Capezzuto, L., Tarapore, D., and Ramchurn, S. D. (2021). Large-Scale, Dynamic and Distributed Coalition Formation with Spatial and Temporal Constraints. In *Multi-Agent Systems*, pages 108–125. Springer International Publishing.

Lu, J., Holleis, T., Zhang, Y., Aumayer, B., Nan, F., Bai, F., Ma, S., Ma, S., Li, M., Yin, G., Wang, Z., and Pang, R. (2024). ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang,

P., and Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior.

Significant Gravitas (2023). *AutoGPT*. https://github.com/Significant-Gravitas/Auto-GPT.

Trabelsi, Y., Adiga, A., Kraus, S., and Ravi, S. S. (2022). Resource Allocation to Agents with Restrictions: Maximizing Likelihood with Minimum Compromise. In *Multi-Agent Systems*, pages 403–420. Springer International Publishing.

Veit, D., Müller, J. P., Schneider, M., and Fiehn, B. (2001). Matchmaking for Autonomous Agents in Electronic Marketplaces. In *Proceedings of the Fifth International Conference on Autonomous Agents*, page 65–66. Association for Computing Machinery.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E. H., Le, Q., and Zhou, D. (2022). Chain of Thought Prompting Elicits Reasoning in Large Language Models. *CoRR*, abs/2201.11903.

Weiss, G. (2016). *Multiagent Systems, Second Edition*. Intelligent Robotics and Autonomous Agents. MIT Press, 2nd edition.

Yan, J., Hu, D., Liao, S. S., and Wang, H. (2015). Mining Agents' Goals in Agent-Oriented Business Processes. *ACM Trans. Manage. Inf. Syst.*, 5(4).

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. (2023). Tree of Thoughts: Deliberate Problem Solving with Large Language Models.

Zhang, Z., Zhang, A., Li, M., and Smola, A. (2022). Automatic Chain of Thought Prompting in Large Language Models.