*International Journal of Computational and Experimental Science and ENgineering (IJCESEN)*

*Vol. 12-No.1 (2026) pp. 618-623*
http://www.ijcesen.com

**ISSN: 2149-9144**

**Research Article**

# Tool Use and External System Integration in Conversational AI: From Function Calling to Autonomous Agents

**Panneer Selvam Viswanathan***

S Tech Mahindra Americas Inc, USA
* **Corresponding Author Email:** reachpanneerselvamv@gmail.com - **ORCID:** 0000-0002-0047-9950

**Article Info:**

**DOI:** 10.22399/ijcesen.4872
**Received :** 02 December 2025
**Revised :** 30 January 2026
**Accepted :** 02 February 2026

**Keywords**

Tool-Augmented Language Models,
Function Calling Architectures,
Autonomous Agent Coordination,
Compositional Reasoning Frameworks,
Multi-Agent Orchestration Systems

**Abstract:**

Conversational AI agents have evolved from text-based agents into tool-using agents that can perform computation and physical actions. This builds on the capabilities of pre-trained language models by integrating them into tool-using systems, performing accurate calculations, and supporting multi-agent systems, sometimes in conjunction with human operators and sometimes independently. They all follow a similar pattern of using self-supervised learning to allow agents to self-invoke and control external tools, verifying tool execution using reasoning models, using hierarchical recovery to counteract tool-using system failures, and accessing thousands of real-world APIs using semantic retrieval mechanisms, as well as a wide variety of protocol standards. Compositional emergence, a subset of emergent behavior, is the idea that complex behaviors can emerge in a system composed of simple components engaging in pipelining and parallelism. Examples include tree-based deliberative strategies that converge on a solution with efficient exploration, world model planning, micro- to meta-level error recovery, and graceful degradation in production contexts. Multi-agent orchestration mechanisms allow modular components to communicate via a declarative requirement and event-driven message systems. Limitations in the scalability, hallucination, and explainability of these systems need to be addressed to create effective multi-agent architectures. Future work could research online reinforcement learning, quantum planning algorithms, and representational robotics that transfer knowledge of tool use from simulations to real-world applications.

## 1. Introduction

LLMs provide a new way of building conversational agents, not only by producing coherent programmatic text one token at a time, but also by controlling external tools, by contrast with conventional language models that can only encode parametric knowledge learned during the training phase. They do not have access to knowledge of current events, to information other than what they were trained on, to accurate arithmetic, or to invoke and control other software systems like LLMs can. The ability to use external tools like the web or orchestrate multiple special tools provides conversational agents and chatbots with important capabilities. The Toolformer paper showed that no tool-calling mechanism is necessary: LMs can simply learn to call tools by self-supervised learning on a corpus of API calls. In this self-deciding setup, the models decide which answer is the better answer to the question, whether it is generated using parametric knowledge or using tools.Second, the architecture of the model becomes more complex, and it becomes easy to build end-to-end tools for all real-world APIs, so that conversational agents are able to learn how to interact with thousands of APIs across many domains and ecosystems [2]. This leads to tool-enabled agents. Given the natural language goal of analyzing quarterly sales data, using market signals to predict quarterly growth, and building a quarterly dashboard, automatically created workflows can use vector loaders to ingest files, discover multimodal patterns and relationships, query live APIs, and synthesize visualizations. These can be done at any level of resolution using parallel dispatching, speculative execution, and runtime registration, respectively, while the Model Context Protocol permits plug-and-play interoperability between heterogeneous systems. Agent-to-Agent protocols encourage spontaneous coordination models in multi-agent systems where agents dynamically choose coordination mechanisms tailored to the current task and environmental conditions.

## 2. Architectural Foundations and Protocol Standardization

Technical substrates include function-calling architectures, which allow LLMs to call out to external tools. These architectures expose tools to LLMs via schemas that include the name of the tool, typed parameters (validators and default values), and async schemas for any output generated by the LLM. Guided decoding is used in state-of-the-art methods as a way of generating compatible JSON, XML, or Pydantic schemas that produce syntactically valid inputs to the tools. Techniques for guiding LMs to use tools have led to a variety of training models, as LMs have been shown to generalize well across a variety of specifications of APIs [1]. However, through the use of comparisons to see whether calling a tool improves model responses, it can be determined when models actually learn to use tools.Platform convergence has occurred around several models that balance expressivity and computational efficiency. OpenAI's tools framework automatically batches and caches tools used multiple times. Anthropic's Model Context Protocol dynamically discovers tools, while Gemini includes multimodal extensions for different data modalities. Open-source frameworks such as LangChain, Haystack, CrewAI (agent orchestration based on roles), and AutoGen (conversational agents) implement the canonical loop of tool registration, request parsing, parallel invocation, stream-processing of tool outputs, and reflective evaluation. As the number of tools in the tooling system grows to tens of thousands of tools, new challenges arise, such as contextualizing tool documentation, instructing the agent to use tools, and retrieving tools from large inventories. Conversely, new approaches use semantic tool retrieval, such as tool vector embeddings, allowing agents to search for tools with similar capabilities rather than enumerating them all [2].The Model Context Protocol also supports semantic search, stateful sessions, and federated discovery for distributed tool registries, which are used by agents to find models that have a certain capability available to carry out a task (e.g., vector store, retrieval-augmented generation (RAG) pipelines). Agent-to-Agent protocols simplify higher-order communication, including swarm coordination via gossip protocols, collective reasoning via blackboard architectures, and contract-based delegation subject to service-level agreement (SLA) constraints. Communication is further supplemented by hierarchical agent topologies enforced by planner agents via goal decomposition. Meanwhile, the atomic actions performed by worker agents are validated or rejected by the critic agents against the quality requirements. Thereby, the specialization of the agents is achieved while maintaining the coherence of their overall behavior. Initial architectures in the tool learning literature have shown that diverse documentation formats and instruction schemes can considerably impact learning performance [2]. However, a more uniform and predictable documentation format of diverse APIs can ease a model's generalization and make it easier for users to adapt to new tool specifications with low cognitive costs. Such runtime patching mechanisms provide support for schema drift and can maintain API compatibility against changes in tool specifications.

## 3. Reasoning Mechanisms and Decision Frameworks

A question in the design of conversational agents is whether to reason over some external or internal parametric knowledge. Hybrid reasoning systems have been proposed using several sources of knowledge and incorporating logit-based probabilities to express epistemic uncertainty when deciding between using external tools or internal parametric knowledge. Dependency graphs identify shortcomings in the agent's reasoning, allowing RL policies to optimize correctness, latency, and tool costs (if it is more efficient to call the tool rather than reason [125, 129, 130]). Planning and action interleaved in structured reasoning models have changed how agents tackle complex reasoning problems [3]. Such a log would help with transparency and would make it easier to debug unexpected outcomes when a particular method is called and does not return the expected result.Modern reasoning architectures beyond the simple React model also support multi-shot reasoning, interleaving, and reasoning traces for interpretability and error diagnosis. They provide advantages over internal reasoning, including the ability to access information in real-time beyond training cutoffs, exact computation, compositionality across arbitrary subdomains, and grounding rationales externally for verification or falsification. They also help when combined with reasoning to solve knowledge-intensive tasks, where the model needs to retrieve and combine information about different facts to answer a question [3]. For instance, the empirical results on the knowledge-intensive tasks in E2E show that the model is more successful at solving a task when it reasons before taking the action rather than predicting the action directly.Tool selection is based on various ranking functions, which may rely on semantic information or performance. It is driven by similarity metrics based on embedding methods and the success rate of the tools. Hierarchical planning algorithms manage action dependencies on previous actions, and can parallelize independent actions by using fan-out to quickly complete highly parallelizable actions. The engine implements several parameter binding techniques, including entity linking (mapping natural language utterances to structure identifiers), translating natural language into system types, and checking calls for invalid ones prior to a call's execution. The two systems constrain tool invocation by user intent (what they want) and system capability (what they can do). Systems must check whether their intended use of a tool would be correct before using it to avoid run-time failures. In addition, reasoning systems offer configurational advantages, especially in choosing the best tool for the job, while external tools provide grounded, fact-based outputs [3]. These components can be combined, with reasoning and knowledge, to create agents that are more capable and stronger across several task domains.The balance between the reasoning performed with the model and the tool is task-dependent. For example, if a task demands speed, then it might be better to use the tool to obtain updated information rather than instantiating reasoning in the model, whereas the opposite may be true for older queries. Exact tools are required for computation, but approximate reasoning suffices for qualitative reasoning problems. Benchmarks have shown that using tools outperforms directly generating computations with language models in arithmetic and symbolic reasoning tasks. The differences arise due to (i) scope: internal knowledge is limited to what the model knows about the training domains, while the tools support compositionality for arbitrary problems; and (ii) verification: internal reasoning requires internal coherence, while tool use provides external verification that the reasoning has occurred correctly via modification of the state of the system. These frameworks allow agents to consider the trade-offs between the cost of using a tool and the value it provides.

## 4. Error Handling, Resilience, and Recovery Strategies

Successful and strong tool use requires wide-ranging error handling for hallucinated invocations where agents call non-existent tools, sandbox violations that cannot be triggered with the current security mechanism, and semantic mismatches between expected and actual tool behaviors. Detection strategies include handling runtime errors, validating tool invocations against schemas of expected outputs, and judges implemented as language models that identify whether an invocation is appropriate or not. Diagnostics can pinpoint the source of failure, such as a change of API, based on error codes returned. An exploration strategy based on tree search can enable agents to efficiently find a successful path in the event of failure, maintaining multiple plausible hypotheses until further evidence is available [4]. This search strategy provides additional leverage for novel tasks, where the ideal strategies for each invocation of the tool are unknown.These various recoveries are nested according to the breadth and severity of failure they attempt to recover from. The micro-level recoveries use jittered, exponential backoff retry mechanisms to cope with transient failures. Recoveries also address timeouts. The meso-level recovery strategies include strategies such as reparameterization of invocations or replacing the currently executing tool with another tool that provides the same service and can effectively complete the task. The agents perform macro-level recovery through scratchpad-based replanning, where agents reconfigure the decomposition when initial plans have failed or time out. Meta-level synthesis generates tools automatically using code generation when existing tools are not sufficient. This supports dynamic tool generation and a deliberate search through intermediate reasoning branches. This allows the system to cleverly backtrack over intermediate reasoning states when it reaches a dead end, increasing the rate of success in difficult problems [4]. Remembering the explored sequences and the actions taken allows agents to avoid repeating bad decisions and to allocate their resources to more promising branches.Self-debugging techniques are promising for addressing code execution failures. Error-guided mutation-based strategies, which are commonly applied to code correctness benchmarks, achieve high success rates by iteratively transforming the generated programs based on their execution results. Standardized benchmarks used by foundational code generation evaluation frameworks enable researchers to measure and compare the capabilities of models across various programming tasks [7]. Despite these benchmarks, researchers have identified that models are still not perfect at algorithmic reasoning and very often require multiple attempts to generate an implementation. In security deployments, it is important to use isolation mechanisms such as sandboxing and taint tracking to ensure that models cannot compromise the system by maliciously invoking the tools. Deployment of these systems has shown that, in practice, the majority of privilege escalation attempts are blocked with low latency.The error recovery regime has a big influence on the agents' robustness. The more errors an agent can predict and recover from, the more resilient its operation. In systems where error recovery is weak, the agents become brittle: after one error, an agent stops the task. Failure propagation rates also indicate this brittleness. In contrast, agents using hierarchical recovery provide a graceful degradation of functioning when a component fails, and are more suitable for use in production settings where the demands of reliability are greater than for a research prototype. Strategies for deliberate exploration form the basis for an efficient search space of possible solutions [4]. Instead of abandoning at the first signs of difficulty, they maintain active explorations of possible solutions along with the information to change their search strategy. This combination of a tree-based search with the generative capability of language models makes it possible to construct systems that navigate complex states with robustness to failures and errors.

## 5. Learning, Adaptation, and Compositional Emergence

The in-context learning and long-term policy adaptation properties of conversational agents enable the model to improve its tool use with experience. The in-context meta-learning property allows a few-shot demonstration of tool use to bootstrap the model and thus alleviate the need for elaborate fine-tuning for the many types of tools available. Meta-prompts also allow the encoding of meta patterns to reuse tool schemas across tools of similar capabilities, and novel approaches to isolate different components of compositional reasoning have had an important impact, such as decomposing complex queries to atomic sub-questions, which improve performance on multi-hop reasoning tasks [5]. This naturally gives rise to tool-enabled architectures in which complex objectives are decomposed into sequences of tool invocations to solve sub-problems.Compositional emergence is a case of higher-order emergence, referring to cases where the capabilities of the super-tool are richer than those of component tools in isolation. This can occur in chains of tools, for instance, where retrieval-augmented generation systems query knowledge bases, pass context to a language model, and execute generated code to produce an output. Parallel forks find multiple solution paths and combine them, for example, via aggregator functions that merge multiple views into one conclusion. Tool discovery mechanisms may be implemented through registry scans, dry runs that evaluate the tools and performance benchmarks for tool selection. This decomposition of the reasoning problem into sub-problems is particularly useful when the task is knowledge-intensive and involves collating evidence from multiple sources [5]. Decomposing monolithic queries into atomic queries that can be solved in isolation results in better performance than end-to-end approaches and improves the interpretability of the solver.Real-time ecosystem integration shows the benefits of tool-using agents in practical applications. For instance, the web search system combines sparse and dense retrieval and supports multiple hops of reasoning over documents, both of which require the use of several information-seeking actions. Code execution environments maintain state between invocations, federate library dependencies, support differential debugging, and support iterative refinement of complex programs. Database interfaces convert natural language queries into structured queries and have produced substantial results on complex semantic parsing benchmarks in recent years. This approach allows more flexible manipulation of reasoning operations in relation to sub-problems [5]. Systems that contain representations of which sub-problems rely on which sub-problem solutions are more efficient than those without, since new solutions can be incrementally improved and errors can be directly corrected on specific sub-problems.Orchestration frameworks enable multi-agent workloads where specialized components work together to achieve more than an individual agent can. E-commerce systems, for example, deploy agent swarms where specialized components handle inventory, dynamic pricing, demand forecasting, supplier coordination, and fulfillment optimization. Production deployments processing millions of transactions show agent-based systems can provide orders of magnitude more efficiency than rule-based systems. Compositional reasoning enables agents to be built from simpler ones, such that problems of unprecedented scale and complexity can be solved [5]. The use of modular tool ensembles and principled decomposition techniques has allowed for tools with flexibility and robustness that are unattainable with monolithic systems. The emergence of standard frameworks for declarative specification of agent crews, together with event-driven inter-agent communication architectures, has made it easier to develop complex multi-agent systems.

***Table 1:*** *Tool Learning Paradigms and API Integration Strategies [1][2]*

| **Learning Mechanism** | **Training Approach** | **Generalization Capability** | **Integration Complexity** |
|---|---|---|---|
| Self-supervised tool discovery | Autonomous determination of invocation contexts | Parametric vs. external tool decision-making | Minimal manual specification |
| API documentation processing | Structured schema interpretation | Cross-domain tool usage patterns | Heterogeneous API standardization |
| Instruction generation systems | Automated usage pattern synthesis | Novel tool bootstrapping | Dynamic documentation formats |
| Semantic tool retrieval | Vector embedding similarity | Large-scale repository navigation | Scalable discovery mechanisms |

***Table 2:*** *Reasoning Frameworks and Deliberation Strategies [3][4]*

| **Framework Component** | **Cognitive Process** | **Error Mitigation** | **Solution Quality** |
|---|---|---|---|
| Verbal reasoning traces | Explicit thought documentation | Transparent failure diagnosis | Enhanced interpretability |
| Action interleaving | Synchronous deliberation-execution | Real-time error detection | Grounded decision-making |
| Tree-based exploration | Systematic solution space search | Multiple hypothesis maintenance | Exhaustive path evaluation |
| Scratchpad replanning | Dynamic strategy restructuring | Fundamental approach revision | Adaptive problem-solving |

***Table 3:*** *Compositional Reasoning and Decomposition Techniques [5][6]*

| **Compositional Strategy** | **Decomposition Method** | **Information Synthesis** | **Architectural Pattern** |
|---|---|---|---|
| Atomic sub-question generation | Complex query segmentation | Independent component addressing | Modular problem decomposition |
| Sequential tool pipelining | Operation chaining | Context propagation across stages | Super-tool emergence |
| Parallel solution exploration | Simultaneous path investigation | Aggregator-based result merging | Multi-perspective synthesis |
| Sub-problem dependency tracking | Explicit relationship modeling | Targeted error correction | Incremental refinement |

***Table 4****: Code Generation and Debugging Mechanisms [7][8]*

| **Evaluation Dimension** | **Synthesis Capability** | **Error Recovery** | **Iterative Refinement** |
|---|---|---|---|
| Algorithmic reasoning | Complex logic implementation | Execution feedback analysis | Error-guided mutation |
| Benchmark standardization | Systematic capability assessment | Failure pattern identification | Multi-attempt correction |
| Domain knowledge integration | Specialized library utilization | Contextual error interpretation | Incremental code improvement |
| Explanation faithfulness | Decision process verbalization | Post-hoc rationalization detection | Transparent reasoning traces |

## 6. Conclusions

From parametric knowledge systems to tool-empowered autonomous agents, conversational artificial intelligence is a model shift in the capabilities of machine intelligence. Language models can autonomously discover optimal tool usage through self-supervised learning, engage in multi-step reasoning with deliberate path explorations in the solution space, and recover from failures with the aid of hierarchical intervention. Architectural advances enable access to a varied set of API ecosystems containing thousands of real-world tools. Compositional reasoning frameworks enable the combination of simpler components into higher-order tools for more complex tasks.

Combined with tool access, these enable a symbiosis in which reasoning and tool usage reinforce each other: reasoning informs tool usage, and tool outputs inform reasoning. Multi-agent orchestration frameworks can coordinate specialized components over standard protocols to articulate solutions to problems that would otherwise be impossible to accomplish on a single agent. When deployed in production, high-stakes environments, challenges such as hallucination mitigation, scalability, and explanation generation will be important areas of inquiry. With the additions of tree-based search, world model planning, and error-driven fine-tuning, possible applications are opened up across a wide variety of tasks. Online RL algorithms, quantum optimization methods, and represented robotics could extend the idea of tool-enabled agency beyond digital spaces and into the world of physical manipulation. Standardized declarative agent specifications and event-driven communications channels expand existing end-to-end processes while preserving domain flexibility. The unification of large language models, high-fidelity error-handling protocols, adaptive learning architectures, and compositional reasoning makes the tool use the foundational technology of artificial general intelligence. His experience in deploying services and designing architectures for goal-directed autonomy has provided the level of autonomous machine agency needed to operate in complex real-world environments, subserving human cognition through human-AI collaboration.

## Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

## References

[1] Timo Schick, et al., "Toolformer: Language models can teach themselves to use tools," arxiv, 2023. Available: https://arxiv.org/abs/2302.04761

[2] Yujia Qin, et al., "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs," ResearchGate, 2023. Available: https://www.researchgate.net/publication/372784505_ToolLLM_Facilitating_Large_Language_Models_to_Master_16000_Real-world_APIs

[3] Pan Lu, et al., "Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models," arxiv, 2023. Available: https://arxiv.org/abs/2304.09842

[4] Shishir G. Patil, et al., "Gorilla: large language model connected with massive APIs," ACM Digital Library, 2023. Available: https://dl.acm.org/doi/10.5555/3737916.3741936

[5] Lorenz Kuhn, et al., "Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation," arxiv, 2023. Available: https://arxiv.org/abs/2302.09664

[6] Shunyu Yao, et al., "Tree of thoughts: deliberate problem solving with large language models," ACM Digital Library, 2022. Available: https://dl.acm.org/doi/abs/10.5555/3666122.3666639

[7] Mark Chen, "Evaluating large language models trained on code," arXiv, 2021. Available: https://arxiv.org/abs/2107.03374

[8] Noah Shinn, et al., "Reflexion: Language agents with verbal reinforcement learning," ACM Digital Library, 2023. Available: https://dl.acm.org/doi/10.5555/3666122.3666499

[9] Shibo Hao et al., "Reasoning with language model is planning with world model," arXiv, 2023. Available: https://www.researchgate.net/publication/371009675_Reasoning_with_Language_Model_is_Planning_with_World_Model

[10] Mohammadreza Pourreza, Davood Rafiei, "DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction," arXiv, 2023. Available: https://arxiv.org/abs/2304.11015