Toward Robust Agentic AI: Architectures, Evaluation, and Safety for Large Language Model Agents

Ishan Pakuwal
Massachusetts Institute of Technology
`ishanpakuwal@alum.mit.edu`

November 27, 2025

Abstract

Large language models (LLMs) are increasingly deployed as *agents* that plan, act, and interact in complex environments. These *agentic AI* systems go beyond single-turn text generation by coupling LLMs with tools, memory, and feedback loops. Despite rapid progress, there is limited conceptual clarity about what constitutes an agentic AI system, how such systems should be architected and evaluated, and which risks they introduce beyond standard LLM deployments. This paper proposes a working definition of agentic AI, surveys common design patterns and evaluation methodologies, and identifies emerging safety and security concerns. We then sketch a reference "agentic stack" and a set of design principles aimed at making LLM-based agents more robust, interpretable, and amenable to systematic evaluation.

1 Introduction

Modern large language models (LLMs) such as GPT-style and PaLM-style models have enabled impressive progress in natural language understanding and generation. However, many real-world tasks require not only producing text, but also *acting* over time: decomposing goals into subgoals, calling tools, updating internal state, and coordinating with humans and other agents. This has led to a rapid rise of *agentic AI* systems, where LLMs are embedded in loops that perceive, plan, and execute actions in an environment.

Intuitively, an LLM becomes an *agent* when it is given persistent goals, a recurrent control loop, access to tools or actuators, and the ability to condition on its own past decisions. Recent work has explored social simulacra [8], open-ended embodied agents [14], tool-using research assistants [11, 6], and multi-agent systems that coordinate to solve complex problems. At the same time, there is growing concern that agentic LLM deployments may amplify existing risks—from hallucination and prompt injection to misaligned optimization, objective drift, or uncontrolled tool use.

This paper makes three contributions:

(i) We provide a precise, implementation-oriented definition of agentic AI in the context of LLM-based systems.

(ii) We organize the design space of agentic architectures into a simple stack: interface, policy, memory, tools, and oversight.

(iii) We discuss evaluation and safety for agentic systems and propose concrete design principles for building robust agents in practical settings.

2 What is Agentic AI?

We use the term *agentic AI* to describe AI systems that satisfy the following properties:

(P1) **Persistent objectives.** The system is given explicit objectives that persist across multiple steps (e.g., "research and summarize X", "triage and respond to support tickets", or "navigate to location Y").

(P2) **Closed-loop interaction.** The system observes state, selects actions, and receives new observations over time, forming an explicit perception-action loop.

(P3) **Tool-mediated action.** The system can call external tools or APIs, or otherwise take actions that affect the environment beyond emitting text.

(P4) **Stateful reasoning.** The system maintains an internal state (e.g., scratchpad, memory store, belief state) that accumulates information across steps.

Standard chat-based LLM use (e.g., answering a single question) does not satisfy these properties: there is no explicit persistent objective or environment, and the model is not empowered to act. By contrast, an LLM-powered coding assistant that can autonomously modify files, run tests, and iterate based on feedback *is* an agentic system.

2.1 Relation to RL and Classical Agents

Classical reinforcement learning agents are typically defined by a Markov decision process (MDP) with states, actions, transitions, and rewards [13]. Agentic LLM systems may or may not have an explicit reward signal, but they do instantiate a similar loop: they observe context, produce an action (which may itself be a sequence of natural language tokens or tool invocations), and update their internal state based on the outcome.

Unlike traditional RL agents, LLM agents often rely on *prompted policies* instead of learned policies, and they use natural language as a universal interface for perception, planning, and tool control. This combination of explicit language, external tools, and implicit world knowledge is a key distinguishing feature of agentic AI.

3 Design Patterns in LLM-based Agents

We now survey common architectural patterns used in agentic LLM systems.

3.1 Planning and Decomposition

A core pattern is to instruct the LLM to decompose a high-level task into a sequence of subgoals or actions. Variants include:

- **Chain-of-thought planning [16]:** prompting the model to list steps before acting.
- **Self-reflection [12]:** the agent critiques its own plan or outputs and revises them.
- **Hierarchical agents:** high-level "manager" agents assign sub-tasks to specialized "worker" agents.

These patterns can be implemented via prompt templates and control code without fine-tuning, but recent work explores specialized training for better planning behavior. ReAct [18] combines reasoning traces with action execution, enabling agents to interleave thought and action more effectively.

3.2 Tool Use and APIs

Another key pattern is *tool use*: the agent can call external functions, APIs, or databases [11, 9]. Typical tools include web search, code execution, file I/O, structured database queries, and domain-specific APIs (e.g., ticketing or CRM systems). Tool use is often handled via function-calling interfaces, where the LLM outputs a structured JSON description of the tool call, which the orchestrator executes.

3.3 Memory Mechanisms

Agentic systems maintain state across steps using one or more memory mechanisms:

- **Ephemeral scratchpads** (conversation history, reasoning traces).
- **Long-term vector stores** that index past observations or documents [17].
- **Structured state** such as task graphs or key-value stores.

Memory can be naively implemented by feeding the full history back to the model, but this quickly runs into context length limitations. Practical systems instead use retrieval-based memory, where relevant past events are selected via embeddings or explicit indexing.

3.4 Multi-agent Systems

Multi-agent setups instantiate a population of LLM agents that interact, collaborate, or compete. Examples include social simulations where agents simulate human-like daily routines and conversations [8], or teams of tool-using agents that debate and critique each other's proposals [2]. Multi-agent patterns promise improved robustness and creativity, but also introduce novel failure modes such as feedback loops, emergent collusion, or runaway escalation.

4 Evaluation of Agentic AI

Evaluating agentic systems is substantially harder than evaluating static LLM outputs. Unlike single-turn generation tasks where outputs can be assessed in isolation, agentic systems operate over extended horizons, take actions with compound effects, and can exhibit emergent behaviors that only manifest in specific environmental contexts. We highlight three axes of evaluation: *task performance*, *process quality*, and *safety*.

4.1 Task Performance

Task performance refers to how well the agent achieves its objectives in a given environment. For code agents, this may be the fraction of tasks solved in a benchmark suite. For research assistants, this might be judged relevance and accuracy of retrieved and synthesized information. Benchmarks are emerging for autonomous agents, including tool-use suites [5], multi-step web navigation tasks [19], and simulated household or enterprise workflows.

However, task performance evaluation faces fundamental challenges in open-ended settings:

- **Novel action sequences:** Agents can produce action trajectories never seen during benchmark construction, making it difficult to anticipate all possible failure modes or success criteria.
- **Environment stochasticity:** Real-world environments (web APIs, user interactions, system states) are non-deterministic, so the same agent policy may yield different outcomes across runs.
- **Delayed consequences:** Actions taken early in an episode may have effects that only become apparent much later, complicating credit assignment and evaluation.
- **Benchmark saturation:** As agents are optimized on fixed benchmarks, performance may reflect overfitting to specific task distributions rather than general capability.

These issues suggest that task performance alone is insufficient for evaluating agentic robustness, and must be complemented by process-level and adversarial evaluation.

4.2 Process Quality

Beyond end performance, one can evaluate the *process* by which the agent arrives at its decisions. Process-level metrics include:

- **Plan quality:** Are generated plans coherent, minimal, and logically ordered?
- **Tool efficiency:** Does the agent use tools parsimoniously, avoiding unnecessary calls?
- **Memory use:** Does the agent correctly recall and update relevant facts over time?
- **Reasoning transparency:** Can human evaluators follow the agent's decision-making process from logs and intermediate states?
- **Error recovery:** Does the agent gracefully handle tool failures, unexpected outputs, or environmental changes?

Logging and analyzing agent trajectories is essential for diagnosing failure modes and improving system design. Offline replay of agent trajectories—where recorded decision sequences are re-executed or analyzed under counterfactual conditions—enables systematic debugging and regression testing. However, replay-based evaluation cannot capture emergent behaviors that arise from novel environment interactions or multi-agent dynamics.

4.3 Safety and Robustness

Agentic systems are exposed to a broader risk surface than static LLMs. They may be vulnerable to:

- **Prompt injection and jailbreaking [10, 3]** that cause the agent to ignore instructions or misuse tools.
- **Objective drift**, where iterated self-refinement gradually shifts the agent's behavior away from intended goals.
- **Reward hacking [1]** in settings where agents are given proxy objectives (e.g., KPI optimization).
- **Cascading failures**, where errors in early steps compound, leading to catastrophic outcomes.

- **Unintended tool composition**, where the agent discovers dangerous or unintended combinations of available tools.

Evaluating robustness requires adversarial testing, red-teaming, and stress testing under distribution shift and adversarial inputs. A particular challenge is the *training-deployment gap*: agents trained or fine-tuned in simplified, sandboxed environments may fail catastrophically when deployed in production settings with different state distributions, timing constraints, or failure modes [4]. This gap is especially pronounced for agents trained via reinforcement learning, where the training environment's reward structure and dynamics rarely match deployment conditions perfectly.

4.4 The Fundamental Evaluation Challenge

The core difficulty in agentic evaluation is that agents operate in partially observable, stochastic environments with large or infinite action spaces. Traditional ML evaluation assumes i.i.d. test samples and fixed input-output mappings, but agentic systems violate both assumptions: their behavior depends on accumulated state and environmental feedback, and they can select from exponentially many action sequences. This makes comprehensive evaluation intractable in the general case.

Practical evaluation strategies must therefore combine:

1. **Benchmark suites** that test common capabilities (tool use, multi-step reasoning, error handling) in controlled settings.
2. **Adversarial probing** that deliberately stress-tests edge cases, malicious inputs, and failure modes.
3. **Deployment monitoring** that tracks agent behavior in production via logging, anomaly detection, and human oversight.
4. **Staged rollout** that gradually increases agent autonomy as evidence of safe operation accumulates.

No single evaluation axis is sufficient; robust agentic AI requires multi-faceted assessment across performance, process, and safety dimensions.

5 Safety and Governance Considerations

As agentic AI systems gain autonomy, their deployment raises ethical, legal, and governance questions [15]. We highlight three principles that should guide responsible design.

5.1 Human-in-the-Loop Control

First, agents that can materially affect the world should be subject to human oversight at key decision points. Common patterns include:

- **Approval gates**, where high-impact actions (e.g., financial transactions, code deployment) require human approval.
- **Guardrails** that restrict the set of tools and arguments available to agents.
- **Escalation policies** that route ambiguous or high-risk situations to humans.

5.2 Transparent Internals

Second, agent architectures should support transparency via logging, inspectable state, and interpretable plans. This enables post-hoc analysis of incidents and allows auditors to understand how decisions were made. We argue that “black box orchestration”—where agents operate without persistent logs or clear state abstractions—is misaligned with safety in high-stakes domains.

5.3 Domain-specific Constraints

Third, agent deployment should respect domain-specific constraints, such as privacy in healthcare, fairness in lending, or safety in robotics. This often requires integrating agentic systems with existing compliance frameworks, monitoring infrastructure, and human processes.

6 A Reference Stack for Agentic AI

We now propose a conceptual reference stack for LLM-based agents, depicted as five layers:

(L1) **Interface layer:** channels for human interaction (chat UI, email, APIs).

(L2) **Policy layer:** LLM prompts, fine-tuned models, and control code that implement the decision policy.

(L3) **Memory layer:** mechanisms for short- and long-term memory, including retrieval and state management.

(L4) **Tool layer:** integrations with external tools, environments, and actuators.

(L5) **Oversight layer:** logging, monitoring, safety filters, and human approval workflows.

This stack is intentionally abstract, but it offers a vocabulary for analyzing and comparing different agentic systems. For example, two agents with similar policy layers may differ substantially in their memory strategies or oversight mechanisms, leading to very different behavior in practice.

7 Design Principles for Robust Agents

Based on the preceding analysis, we propose the following design principles for robust agentic AI:

7.1 Principle 1: Minimize Unnecessary Autonomy

Grant agents the minimum autonomy necessary to achieve their objectives, and prefer human-in-the-loop workflows whenever possible. Avoid granting blanket access to powerful tools; instead, limit scope and enforce approval gates for risky actions.

7.2 Principle 2: Make State and Intent Explicit

Represent goals, plans, and internal state in explicit data structures rather than only in unstructured text. Where possible, encode key constraints (budgets, safety rules, compliance policies) as machine-checkable conditions.

7.3 Principle 3: Instrument for Evaluation

Log trajectories, tool calls, and state changes in a structured format that enables offline analysis and replay. Build evaluation pipelines that can replay agent behavior under different conditions, inject adversarial perturbations, and automatically flag anomalies.

7.4 Principle 4: Separate Concerns

Factor the system into modular components (policy, memory, tools, oversight) with clear interfaces. This facilitates systematic testing and replacement of components, and makes it easier to reason about failure modes.

7.5 Principle 5: Align Training and Deployment

When agents are fine-tuned or reinforced using feedback (e.g., via RLHF [7] or direct preference optimization), ensure that training objectives are aligned with deployment-time objectives. Misalignment between training and deployment environments can cause agents to exploit proxy metrics that do not reflect true human goals.

8 Discussion and Outlook

Agentic AI represents a natural next step in the evolution of LLM-based systems, moving from passive text generation to active, goal-directed behavior. While current systems remain brittle and heavily orchestrated, they already exhibit useful capabilities in software engineering, customer support, and data analysis. At the same time, the shift from static predictions to autonomous action raises new questions about evaluation, safety, and governance.

We have offered a working definition of agentic AI, surveyed common architectural patterns, and proposed a reference stack and design principles for robust agents. Future work includes developing standardized benchmarks for agentic tasks, formalizing safety properties for agentic systems, and designing training objectives that more directly capture human intent in multi-step settings.

Acknowledgments

This paper synthesizes insights from recent work on LLM agents, multi-agent systems, and AI safety. Any errors or omissions are the author's own.

References

[1] D. Amodei, C. Olah, J. Steinhardt, et al. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[2] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

[3] K. Greshake, S. Abdelnabi, S. Mishra, et al. 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*, 2023.

[4] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

[5] X. Liu, H. Yu, H. Zhang, et al. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[6] R. Nakano, J. Hilton, S. Balaji, et al. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[7] L. Ouyang, J. Wu, X. Jiang, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022.

[8] J. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.

[9] S. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*, 2023.

[10] F. Perez and I. Ribeiro. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*, 2022.

[11] T. Schick, J. Dwivedi-Yu, R. Dessì, et al. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2024.

[12] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2024.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[14] G. Wang, Y. Xie, Y. Jiang, et al. Voyager: An open-ended embodied agent with large language models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 35839–35862, 2023.

[15] L. Weidinger, J. Mellor, M. Rauh, et al. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*, 2021.

[16] J. Wei, X. Wang, D. Schuurmans, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022.

[17] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *International Conference on Learning Representations*, 2015.

[18] S. Yao, J. Zhao, D. Yu, et al. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[19] S. Zhou, F. F. Xu, H. Zhu, et al. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.