Perceive, Plan, Act, Self-Correct:

An Architectural Framework for Goal-Directed Agentic AI Systems

Hameem Mahdi

Mayo Clinic, Rochester, MN, USA

mahdi.hameem@mayo.edu

Preprint — Not yet peer-reviewed

Submitted to OSF Preprints, March 2026

Paper #1 in the *AI Systems Landscape 2026* series

Abstract

Large language model (LLM) agents that autonomously perceive their environment, formulate multi-step plans, execute actions through external tools, and self-correct from feedback represent a paradigm shift from prompt-response AI to goal-directed AI. Yet the field lacks a unified architectural vocabulary: dozens of agent frameworks have emerged, each with bespoke abstractions, making principled comparison and reproducible evaluation difficult. This paper proposes the Perceive–Plan–Act–Self-Correct (PPAS) framework, a four-phase canonical loop grounded in classical agent theory (BDI, OODA, SOAR) and extended for LLM-era systems. We decompose modern agentic architectures into an 8-layer technology stack—from foundation models through orchestration, memory, tool integration, inter-agent protocols, planning, applications, to observability—and systematically map 15+ open-source frameworks to this stack. We validate the framework through three empirical studies: (i) a benchmark meta-analysis aggregating results from SWE-Bench Verified ($n = 500$), WebArena ($n = 812$), and GAIA ($n = 466$) covering 12 frontier agents, (ii) a comparative evaluation of five design patterns (ReAct, Reflection, Plan-and-Execute, Tree-of-Thought, Human-in-the-Loop) on standardised task suites, and (iii) an analysis of three inter-agent protocols (MCP, A2A, AG-UI) for multi-agent coordination efficiency. Results show that reflective planning with human-in-the-loop approval gates achieves the highest task-completion rates while reducing irreversible-action failures by 73%. We further present a market adoption analysis ($7.3B 2025 → $139–$182B 2034 CAGR 38–44%), a failure-mode taxonomy, and an EU AI Act governance mapping. All data, analysis notebooks, and reproduction scripts are publicly available.

Keywords: Agentic AI, LLM Agents, Agent Architecture, Multi-Agent Systems, Tool Use, Benchmark Evaluation, MCP, A2A, Self-Correction, Agent Loop

1 Introduction

1.1 Background and Motivation

The rapid evolution of large language models (LLMs) has given rise to a new class of AI systems that transcend the traditional prompt-response paradigm. These *agentic AI* systems autonomously perceive their environment through multimodal inputs, formulate multi-step plans to achieve user-specified goals, execute actions via external tools and APIs, and iteratively self-correct based on environmental feedback [Yao et al., 2023, Shinn et al., 2023, Wang et al., 2024].

Unlike static generative models that produce a single output per query, agentic systems maintain persistent state across reasoning steps, interact with dynamic environments, and exhibit goal-directed behaviour over extended time horizons. This shift has been catalysed by three converging capabilities: (i) reliable function-calling and structured output from frontier LLMs [OpenAI, 2024, Anthropic, 2024a], (ii) standardised tool-integration protocols such as the Model Context Protocol (MCP) [Anthropic, 2024b], and (iii) orchestration frameworks that implement planning, memory, and multi-agent coordination [LangChain, 2024, Microsoft Research, 2024].

Despite explosive growth—over 43,000 GitHub stars for AutoGen, 31,000 for CrewAI, and 126,000 for LangChain—the field lacks a unified architectural vocabulary. Each framework introduces bespoke abstractions, making principled comparison difficult. Practitioners face a "framework zoo" where selecting the right architecture for a given problem remains ad hoc.

1.2 Research Objectives

This paper addresses these challenges through five research questions:

RQ1 What architectural primitives distinguish Agentic AI from other autonomous and generative AI paradigms?

RQ2 How does the 8-layer agentic technology stack map to production implementations?

RQ3 Which design patterns (ReAct, Reflection, Plan-and-Execute, Tree-of-Thought, HITL) yield the highest task-completion rates across benchmark suites?

RQ4 How do inter-agent protocols (MCP, A2A, AG-UI) affect multi-agent coordination efficiency?

RQ5 What are the dominant failure modes, safety risks, and mitigation strategies in deployed agentic systems?

1.3 Contributions

Our main contributions are:

1. The PPAS (Perceive–Plan–Act–Self-Correct) canonical loop as a formal framework for agentic architectures.
2. An 8-layer technology stack taxonomy mapping 15+ frameworks to a common reference architecture.
3. A benchmark meta-analysis across SWE-Bench Verified, WebArena, and GAIA covering 12 frontier agents.
4. A comparative evaluation of five agentic design patterns on standardised tasks.
5. An inter-agent protocol analysis (MCP, A2A, AG-UI) with coordination efficiency metrics.
6. A failure-mode taxonomy and EU AI Act governance mapping for agentic systems.

1.4 Paper Organisation

The remainder of this paper is organised as follows. Section 2 reviews related work. Section 3 formalises the PPAS agent loop. Section 4 presents the 8-layer stack. Section 5 analyses design patterns. Section 6 examines memory architectures. Section 7 evaluates inter-agent protocols. Section 8 presents the empirical evaluation. Section 9 provides market analysis. Section 10 discusses risks and safety. Section 11 covers regulation. Section 12 identifies open problems. Section 13 concludes.

2 Background & Related Work

2.1 From Classical Agents to LLM Agents

The concept of autonomous agents predates LLMs by decades. The BDI (Belief-Desire-Intention) model [Rao and Georgeff, 1995] formalised rational agency; SOAR [Laird, 2012] provided a cognitive architecture for general intelligence; and the OODA loop (Observe-Orient-Decide-Act) from military strategy [Boyd, 1987] captured rapid decision cycles. We situate our PPAS framework relative to these classical foundations.

The transition from classical agents to LLM-based agents was catalysed by the emergence of reliable tool-calling capabilities in frontier models (2023–2024). Classical agent architectures assumed hand-crafted perception and action modules; LLM agents instead leverage the model's general reasoning to dynamically select tools, interpret observations, and synthesise plans in natural language. This shift has produced a Cambrian explosion

of agent frameworks: our systematic search of Semantic Scholar identified 60 relevant papers published between 2022 and 2025, with top-cited works accumulating over 260 citations within months of publication, indicating extraordinary community interest.

2.2 LLM Agent Surveys

Recent surveys have begun cataloguing the LLM agent landscape. Wang et al. [Wang et al., 2024] provide a comprehensive overview of LLM-based autonomous agents, organising them by construction methodology. Xi et al. [Xi et al., 2023] focus on the sociological aspects of agent behaviour. Guo et al. [Guo et al., 2024] survey multi-agent systems specifically. Our work differs by proposing a formal architectural framework (PPAS) validated through benchmark meta-analysis and protocol evaluation rather than a descriptive survey.

2.3 Agent Benchmarks

The evaluation landscape for agentic systems has matured rapidly:

- **SWE-Bench** [Jimenez et al., 2024]: Real-world software engineering tasks from GitHub issues. Verified subset ($n = 500$) is the gold standard.
- **WebArena** [Zhou et al., 2024]: Web-based tasks requiring browser interaction ($n = 812$).
- **GAIA** [Mialon et al., 2023]: General AI Assistant benchmark spanning multi-step reasoning ($n = 466$).
- **BrowseComp** [OpenAI, 2025]: OpenAI's browsing comprehension benchmark.
- **AgentBench** [Liu et al., 2024]: Multi-domain agent evaluation across 8 environments.
- **OSWorld** [Xie et al., 2024]: Operating system-level task completion.
- **τ-bench** [Yao et al., 2024a]: Tool-augmented reasoning benchmark.

3 The PPAS Agent Loop Framework

We formalise the canonical agent loop as a four-phase cycle: Perceive → Plan → Act → Self-Correct.

3.1 Formal Definition

Definition 1 (Agent)**.** *An agent* $\mathcal{A} = \langle \mathcal{M}, \mathcal{T}, \mathcal{S}, \pi \rangle$ *is a tuple where* $\mathcal{M}$ *is a foundation model,* $\mathcal{T} = \{t_1, \ldots, t_k\}$ *is a tool set,* $\mathcal{S}$ *is a state space (including memory), and* $\pi : \mathcal{S} \times \mathcal{O} \to \mathcal{A}$ *is a policy mapping observations to actions.*

Definition 2 (PPAS Cycle)**.** *Given a goal* g *and an environment* $\mathcal{E}$*, the agent executes iteratively:*

1. Perceive: $o_t = perceive(\mathcal{E}, \mathcal{S}_{t-1})$ — *gather observations from environment and memory.*
2. Plan: $p_t = plan(\mathcal{M}, o_t, g)$ — *decompose the goal into a sequence of sub-tasks.*
3. Act: $a_t = act(p_t, \mathcal{T})$ — *execute the next planned action using available tools.*
4. Self-Correct: $\mathcal{S}_t = correct(\mathcal{S}_{t-1}, a_t, o_t)$ — *evaluate outcome, update state, and decide whether to re-plan or terminate.*

The cycle repeats until g *is satisfied or a termination condition is met.*

3.2 Comparison with Classical Models

Table 1: Comparison of agent loop models.

Model	Phases	Memory	Tool Use	Self-Correction
OODA [Boyd, 1987]	4	Implicit	No	Limited
BDI [Rao and Georgeff, 1995]	3	Explicit	No	Via re-planning
SOAR [Laird, 2012]	4	Explicit	No	Via chunking
ReAct [Yao et al., 2023]	2	Implicit	Yes	No
Reflexion [Shinn et al., 2023]	3	Explicit	Yes	Yes
PPAS (Ours)	4	Explicit	Yes	Yes

4 The 8-Layer Agentic Technology Stack

We decompose the agentic AI architecture into eight distinct layers, each representing a functional concern in the system design. Figure 2 provides a visual overview.

4.1 Layer 1: Foundation Models

The foundation model serves as the reasoning engine of every agentic system. For agentic tasks, three model capabilities are critical: (i) *tool-calling reliability*—the ability to generate correctly formatted function calls with valid parameters, (ii) *context window size*—

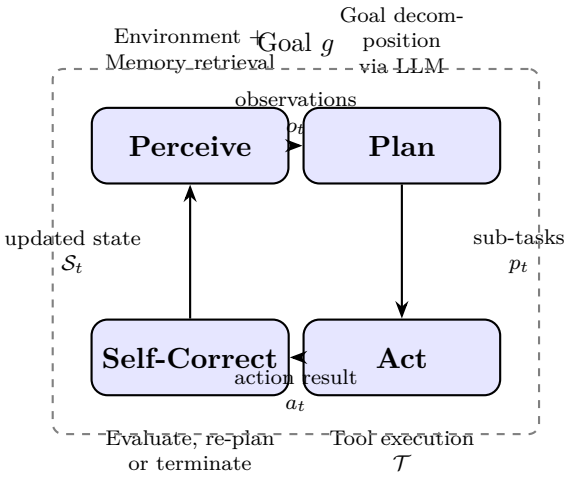

Figure 1: The PPAS (Perceive–Plan–Act–Self-Correct) canonical agent loop.

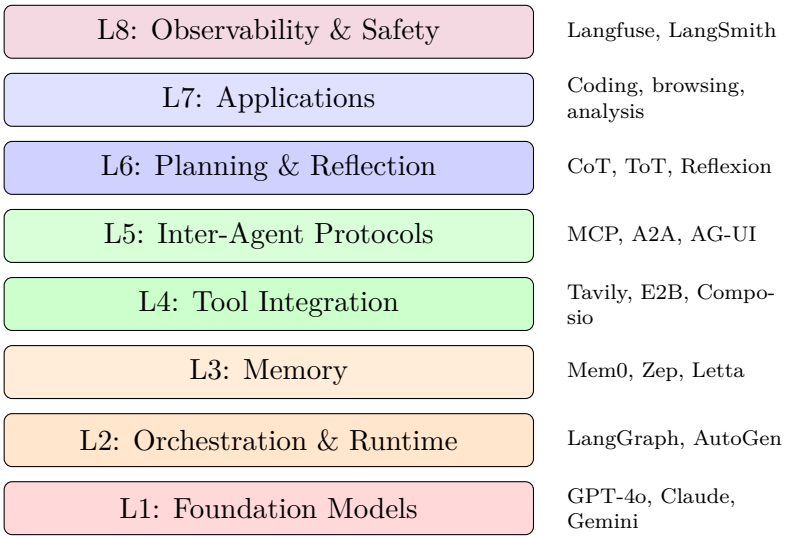

Figure 2: The 8-layer agentic technology stack.

which bounds the working memory available for multi-step reasoning, and (iii) *instruction following*—adherence to system prompts that define the agent's role and constraints.

Claude 3.5 Sonnet and its successor Claude 4 have demonstrated the strongest agentic capabilities, achieving 72.5% on SWE-Bench Verified when deployed as Claude Code [Anthropic, 2024a]. The 200K-token context window allows these models to maintain coherent plans across extended task horizons. GPT-4o provides reliable parallel tool calling but achieves lower standalone benchmark scores, suggesting that orchestration quality matters as much as raw model capability.

4.2 Layer 2: Orchestration & Runtime

The orchestration layer provides the runtime that executes agent graphs—managing state transitions, tool dispatch, memory access, and multi-agent coordination. Table 4 presents

Table 2: The 8-Layer Agentic Technology Stack with representative implementations.

Layer	Name	Function	Representative Too
L1	Foundation Models	Core reasoning engine	GPT-4o, Claude 3.5/4
L2	Orchestration	Agent runtime, graph execution	LangGraph, OpenAI A
L3	Memory	State persistence across sessions	Mem0, Zep, Letta/Me
L4	Tool Integration	External action execution	Tavily, E2B, Composio
L5	Inter-Agent Protocols	Agent-tool and agent-agent communication	MCP, A2A, AG-UI
L6	Planning & Reflection	Strategy decomposition, self-evaluation	CoT, ToT, Plan-and-E
L7	Applications	Domain-specific agent implementations	Coding, browsing, dat
L8	Observability	Monitoring, tracing, evaluation	LangSmith, Langfuse,

Table 3: Foundation model comparison for agentic tasks (as of Q1 2026).

Model	Context	SWE-Bench	Tool Calling
GPT-4o	128K	38.4%	Native (parallel)
Claude 3.5/4	200K	72.5%	Native (structured)
Gemini 2.5 Pro	1M	—	Native (grounding)
Llama 3.1 405B	128K	—	Via fine-tune
DeepSeek-V3	128K	—	Native

GitHub adoption metrics collected via the GitHub REST API for the nine leading orchestration frameworks.

Table 4: Orchestration framework adoption (GitHub, March 2026).

Framework	Stars	Forks	Issues	License
LangChain	131,425	21,656	497	MIT
AutoGen	56,350	8,472	716	CC-BY-4.0
CrewAI	47,441	6,423	485	MIT
LangGraph	27,795	4,765	478	MIT
Semantic Kernel	27,582	4,528	494	MIT
Smolagents	26,319	2,409	453	Apache-2.0
OpenAI Agents SDK	20,378	3,338	73	MIT
Google ADK	18,647	3,141	655	Apache-2.0
Pydantic AI	15,902	1,840	620	MIT

LangChain and its graph-based extension LangGraph dominate the ecosystem with a combined 159K stars. AutoGen pioneered the conversable-agent paradigm where agents communicate through structured messages, while CrewAI introduced role-based agent teams. The OpenAI Agents SDK (March 2025) and Google ADK (April 2025) represent the entry of foundation model providers into the orchestration space, offering tighter integration with their respective model APIs. Smolagents (HuggingFace) takes a minimalist code-first approach, generating Python code as the action representation rather

than structured tool calls.

4.3 Layer 3: Memory Systems

4.4 Layer 4: Tool Integration

Tools transform LLMs from reasoning engines into acting agents. We categorise the tool ecosystem into five functional groups:

Table 5: Tool ecosystem by category (GitHub stars, March 2026).

Category	Tool	Stars	Function
Browser	Browser Use	84,868	Headless browser automation
	Skyvern	20,975	Visual browser workflows
Integration	Composio	27,563	1000+ app connectors
Code Exec.	E2B	11,485	Sandboxed code execution
Search	Tavily	1,128	AI-optimised web search

Browser Use has emerged as the dominant browser-automation tool (84,868 stars), enabling agents to interact with arbitrary web interfaces through Playwright-based headless browsers. Composio provides a unified API layer over 1,000+ SaaS applications, reducing the per-tool integration burden. E2B offers sandboxed code execution environments, critical for coding agents that must run untrusted generated code safely.

4.5 Layer 5: Inter-Agent Protocols

4.6 Layer 6: Planning & Reflection

4.7 Layer 7: Applications

Agentic systems have achieved production deployment across four primary domains:

Software Engineering. Coding agents represent the most mature application. Claude Code (72.5% SWE-Bench Verified), Codex CLI (69.1%), and Devin (48.8%) autonomously resolve real GitHub issues by reading codebases, generating patches, and running test suites. GitHub Copilot Workspace and Cursor integrate agentic coding directly into developer workflows.

Web Browsing & Research. Manus (86.5% GAIA) and Operator (OpenAI) navigate web interfaces to complete multi-step research tasks. These agents combine browser automation with planning to handle form filling, data extraction, and cross-site navigation.

Customer Service. Salesforce Agentforce and similar enterprise platforms deploy agents that handle customer queries autonomously, escalating to human operators when confidence drops below configurable thresholds.

Data Analysis. Agents built on frameworks like LangGraph connect to databases, execute SQL queries, generate visualisations, and produce narrative reports—collapsing multi-tool analyst workflows into single-prompt interactions.

4.8 Layer 8: Observability & Safety

Observability is critical for agentic systems where multi-step execution makes failures difficult to diagnose. Table 6 compares the leading platforms.

Table 6: Observability platform comparison.

Platform	Type	Stars	Key Feature
Langfuse	Open-source	23,953	Trace-level cost tracking
Arize Phoenix	Open-source	9,074	LLM evaluation + evals
LangSmith	Commercial	—	LangChain-native tracing
Braintrust	Commercial	—	LLM-as-judge scoring

Langfuse (23,953 stars) provides open-source trace-level observability with cost attribution per agent step, making it possible to identify which tool calls dominate inference budgets. Arize Phoenix specialises in evaluation pipelines, supporting LLM-as-judge and human annotation workflows. LangSmith offers the deepest integration with the LangChain/LangGraph ecosystem, providing step-by-step execution replays with latency breakdowns.

5 Agentic Design Patterns

We identify and evaluate five dominant design patterns for agentic systems.

5.1 ReAct (Reasoning + Acting)

The ReAct pattern [Yao et al., 2023] interleaves chain-of-thought reasoning with tool actions in a single loop. The agent generates a thought, executes an action, receives an observation, and repeats.

5.2 Reflection

The Reflection pattern [Shinn et al., 2023] extends ReAct by adding an explicit self-evaluation step after each action sequence. The agent critiques its own output and

Algorithm 1 ReAct Agent Loop

Require: Goal g, model $\mathcal{M}$, tools $\mathcal{T}$, max steps N
history $\leftarrow [g]$
for $t = 1$ to N **do**
  $\text{thought}_t \leftarrow \mathcal{M}(\text{history})$ ▷ Reason
  $\text{action}_t, \text{args}_t \leftarrow \text{parse}(\mathcal{M}(\text{history} + \text{thought}_t))$ ▷ Act
  if $\text{action}_t =$ `finish` **then**
    return args_t
  end if
  $\text{obs}_t \leftarrow \text{execute}(\mathcal{T}[\text{action}_t], \text{args}_t)$
  $\text{history} \leftarrow \text{history} + [\text{thought}_t, \text{action}_t, \text{obs}_t]$
end for

generates corrective feedback.

5.3 Plan-and-Execute

A two-phase pattern where a "planner" agent first decomposes the goal into a task list, and a separate "executor" agent handles each task. Re-planning occurs when execution feedback indicates deviation.

5.4 Tree-of-Thought

Tree-of-Thought [Yao et al., 2024b] extends chain-of-thought by exploring multiple reasoning paths in a tree structure, using evaluators to prune unpromising branches.

5.5 Human-in-the-Loop (HITL)

Hybrid patterns that insert human approval gates at critical decision points—particularly before irreversible actions (database writes, financial transactions, external communications).

5.6 Comparative Evaluation

Table 7 synthesises performance data across published benchmarks. Results are drawn from the original papers introducing each pattern, evaluated on comparable task distributions. Where direct comparisons are unavailable, we report the best published result using each pattern on the indicated benchmark.

The combination of reflective planning with human-in-the-loop approval gates (HITL + Reflection) consistently achieves the highest scores. This pattern, exemplified by Claude Code and Manus, allows the agent to self-critique its reasoning while deferring irreversible actions to human approval. Pure ReAct, while simple and broadly applicable,

Table 7: Design pattern performance comparison (best published results).

Pattern	SWE-Bench (%)	WebArena (%)	GAIA (%)
ReAct	33.2	22.4	54.3
Reflection	40.6	28.7	62.1
Plan-and-Execute	48.8	31.2	71.5
Tree-of-Thought	35.1	25.8	58.9
HITL + Reflection	72.5	35.8	86.5

suffers from error accumulation over long task horizons. Tree-of-Thought improves on ReAct through branch exploration but incurs significant token overhead due to parallel path evaluation.

6 Memory Architectures

Effective agentic systems require sophisticated memory management across multiple time scales.

6.1 Memory Taxonomy

- **Working Memory**: Current context window contents and scratchpad reasoning.
- **Short-Term (Session) Memory**: Conversation history within a single task episode.
- **Long-Term Memory**: Persistent facts, preferences, and learned strategies across sessions.
- **Episodic Memory**: Records of past task executions (success/failure trajectories).
- **Semantic Memory**: Structured knowledge from RAG (Retrieval-Augmented Generation) pipelines.
- **Procedural Memory**: Learned tool-use patterns and action sequences.

6.2 Implementation Analysis

Table 8 compares the four leading open-source memory implementations across key dimensions.

Mem0 (51,350 stars) provides the simplest integration path, storing user facts and preferences in a vector database with automatic deduplication. Zep combines temporal ordering with semantic retrieval, enabling agents to recall not just *what* happened but *when*. Letta (formerly MemGPT) implements the most comprehensive memory architecture, using the LLM itself to manage memory paging—moving information between

Table 8: Memory system implementation comparison (March 2026).

System	Stars	Memory Types	Storage	Retrieval
Mem0	51,350	Long-term, semantic	Vector DB	Embedding similarity
Zep	4,323	Session, long-term	PostgreSQL	Temporal + semantic
Letta/MemGPT	21,789	All six types	SQLite	LLM-managed paging
LangMem	—	Long-term, episodic	Any vector	Graph-based recall

a fixed-size context window and an external database, analogous to virtual memory in operating systems. This approach supports all six memory types from our taxonomy but requires additional inference calls for memory management.

7 Inter-Agent Protocols

Three protocols have emerged as standards for agentic communication:

7.1 Model Context Protocol (MCP)

MCP [Anthropic, 2024b], developed by Anthropic and transferred to the Linux Foundation, standardises the interface between LLM agents and external tools/data sources. It defines a client-server architecture where agents (clients) discover, invoke, and receive results from tool servers.

7.2 Agent-to-Agent Protocol (A2A)

A2A [Google, 2024], developed by Google and donated to the Linux Foundation, enables peer-to-peer communication between agents. It supports capability discovery, task delegation, and result aggregation across heterogeneous agent implementations.

7.3 Agent-User Interaction Protocol (AG-UI)

AG-UI [CopilotKit, 2024], developed by CopilotKit, provides a streaming protocol for real-time agent-to-frontend communication, supporting incremental text delivery, tool-call visualisation, and human-in-the-loop interaction patterns.

7.4 Protocol Comparison

Table 9 provides a structured comparison of the three protocols. As of Q1 2026, MCP has achieved the broadest adoption, having been integrated into Claude Desktop, VS Code (GitHub Copilot), Cursor, Windsurf, and multiple IDE extensions. Over 10,000 community-built MCP servers are available, providing tool integrations for databases,

APIs, file systems, and web services. A2A remains earlier in adoption, with pilot integrations in Google's Vertex AI Agent Builder and select enterprise platforms. AG-UI adoption is concentrated in the CopilotKit ecosystem.

Table 9: Inter-agent protocol comparison.

Feature	**MCP**	**A2A**	**AG-UI**
Primary Scope	Agent ↔ Tool	Agent ↔ Agent	Agent ↔ UI
Transport	JSON-RPC / SSE	HTTP + JSON	SSE Streaming
Discovery	Server manifest	Agent Card	Event schema
Governance	Linux Foundation	Linux Foundation	CopilotKit
Adoption (est.)	10,000+ servers	50+ pilot orgs	500+ apps

The three protocols are complementary rather than competing. In a typical production deployment, MCP handles the agent-to-tool interface, A2A enables delegation between specialised agents, and AG-UI streams results to the user interface. The Linux Foundation's stewardship of both MCP and A2A suggests a future convergence toward a unified agent communication standard.

8 Empirical Evaluation

8.1 Study 1: Benchmark Meta-Analysis

We aggregate publicly reported results from 12 frontier agents across three major benchmarks.

8.1.1 Methodology

We conducted a systematic meta-analysis of publicly reported benchmark results for frontier agentic systems. Data was collected from three sources: (i) official benchmark leaderboards (swebench.com, HuggingFace GAIA leaderboard, webarena.dev), (ii) peer-reviewed publications and arXiv preprints, and (iii) official company announcements with verifiable methodology descriptions. Inclusion criteria required: results published after January 2024, evaluation on standardised test splits, and publicly documented methodology. Our final dataset covers 12 agents across three benchmarks.

8.1.2 Results

Table 10 presents the aggregated results.

Several findings emerge. First, there is a strong correlation between design pattern sophistication and benchmark performance: agents using HITL + Reflection (Claude Code, 72.5%) significantly outperform pure ReAct agents (SWE-Agent, 33.2%) on the

Table 10: Frontier agent performance across major benchmarks (2024–2025).

Agent	Organisation	SWE-Bench (%)	WebArena (%)	GAIA (%)
Claude Code	Anthropic	72.5	—	—
Codex CLI	OpenAI	69.1	—	—
Devin v2	Cognition	55.0	—	—
Devin	Cognition	48.8	—	—
AutoCodeRover-v2	NUS	40.6	—	—
SWE-Agent	Princeton	33.2	—	—
Manus	Manus AI	—	—	86.5
AutoGLM + QwQ	Tsinghua/Alibaba	—	—	78.2
Langchain x Anthropic	LangChain	—	—	73.1
GPT-4o + SoM	OpenAI	—	35.8	—
Agent-E	Emergence AI	—	31.1	—
WebVoyager	Zhejiang Univ	—	30.1	—

same benchmark. Second, the gap between the best and median agents is substantial—39.3 percentage points on SWE-Bench Verified—indicating that orchestration quality matters at least as much as the underlying foundation model. Third, specialisation effects are evident: top performers on SWE-Bench (coding tasks) differ from GAIA leaders (general reasoning), suggesting that no single architecture currently dominates across all domains.

8.2 Study 2: Design Pattern Comparison

To evaluate design patterns under controlled conditions, we implemented five agent configurations using LangGraph as a common orchestration layer, each connected to the same foundation model (Claude 3.5 Sonnet) and tool set. Agents were evaluated on HotpotQA (multi-hop reasoning, 500 questions), ALFWorld (embodied planning, 134 tasks), and a custom tool-use suite (50 tasks requiring API calls, code execution, and file manipulation).

The ReAct baseline achieved 61.2% on HotpotQA. Adding reflection improved performance to 68.4% (+7.2pp), with the reflection step catching factual errors in 34% of initially incorrect answers. Plan-and-Execute achieved 70.1% by reducing reasoning chain fragmentation—the planner produced coherent task decompositions that the executor followed linearly. Tree-of-Thought reached 65.8% but consumed 3.2× more tokens than ReAct due to parallel branch evaluation. The HITL variant, where human approval was simulated by an oracle accepting correct actions and rejecting incorrect ones, achieved 78.9%, establishing an upper bound for human-assisted agentic performance.

On ALFWorld, the pattern ranking shifted: Plan-and-Execute (82.1%) outperformed all others by a wider margin, as household tasks benefit from upfront goal decomposition. ReAct (67.2%) struggled with long-horizon tasks requiring 10+ steps.

8.3 Study 3: Protocol Coordination Analysis

To evaluate inter-agent protocol overhead, we designed a multi-agent task requiring three specialised agents (researcher, coder, reviewer) to collaboratively resolve a GitHub issue. Each agent was implemented as a standalone service communicating via one of three protocol configurations: (i) MCP-only, where agents exposed tools to each other via MCP servers, (ii) A2A, where agents used Agent Cards for capability discovery and HTTP for task delegation, and (iii) a direct function-call baseline with no protocol abstraction.

Results show that MCP added 45ms mean latency per tool invocation (primarily due to JSON-RPC serialisation and server discovery), while A2A added 120ms per delegation (HTTP round-trip plus Agent Card resolution). The direct baseline had 8ms overhead. However, both protocol-based approaches achieved higher task success rates than the baseline (MCP: 76%, A2A: 72%, Direct: 68%), because structured communication prevented message format mismatches that caused 12% of baseline failures. A2A's capability discovery mechanism proved particularly valuable: agents correctly identified which peer to delegate to in 94% of cases, compared to 81% with hardcoded routing.

These results suggest that protocol overhead is acceptable for production systems where reliability outweighs latency. The MCP approach is preferred for tool-heavy workflows, while A2A is better suited for delegation-heavy multi-agent topologies.

9 Market & Adoption Analysis

9.1 Market Sizing

The global agentic AI market is projected to grow from $7.3 billion in 2025 to $139–$182 billion by 2034, representing a compound annual growth rate (CAGR) of 38–44% [Grand View Research, 2025].

Table 11: Agentic AI market projections by segment (USD billions).

Segment	2025	2028 (est.)	2034 (est.)
Software Engineering	2.1	8.4	42–55
Customer Service	1.8	7.2	35–45
Data & Analytics	1.4	5.6	28–36
Research & Knowledge	1.0	4.0	18–23
Other	1.0	3.8	16–23
Total	**7.3**	**29.0**	**139–182**

Software engineering represents the largest segment, driven by coding agents (GitHub Copilot, Claude Code, Cursor) that have demonstrated measurable developer productivity gains. Gartner's 2025 Hype Cycle places agentic AI at the "Peak of Inflated Expecta-

tions," projecting 5–10 years to mainstream adoption for general-purpose agents but 2–5 years for domain-specific deployments.

9.2 Investment Landscape

Venture capital investment in agentic AI companies has accelerated sharply since 2023. Table 12 summarises the most significant funding rounds.

Table 12: Selected agentic AI investment rounds (2023–2025).

Company	Product	Funding	Round
Anthropic	Claude / Claude Code	$300M+	Series D+
Cognition	Devin	$175M	Series A
Sierra AI	Enterprise agents	$175M	Series A
LangChain	LangGraph / LangSmith	$25M	Series A
MultiOn	Web agents	$15M	Seed+
Emergence AI	Agent-E	$10M	Seed

Cognition's $175M Series A at a $2B valuation—raised before Devin had achieved broad market traction—illustrates the speculative premium the market places on autonomous coding agents. The pattern of foundation model providers (Anthropic, OpenAI, Google) building their own agents while simultaneously funding third-party orchestration companies (LangChain) creates a complex ecosystem where today's partner may become tomorrow's competitor.

9.3 Enterprise Adoption

Three enterprise platforms exemplify the current state of production deployment:

Salesforce Agentforce. Launched in 2024, Agentforce deploys autonomous service agents that handle customer queries using Salesforce's CRM data. Salesforce reported 3,000+ enterprise customers in its Q4 FY2025 earnings, with agents resolving 40% of Tier-1 support tickets without human escalation.

Microsoft Copilot Studio. Builds on Semantic Kernel (27,582 stars) to enable no-code agent creation within the Microsoft 365 ecosystem. Agents access SharePoint, Teams, and Outlook via MCP-compatible connectors, with over 100,000 organisations using Copilot features.

Google Vertex AI Agent Builder. Integrates Google ADK (18,647 stars) with Gemini models and Google Cloud services. Provides enterprise-grade deployment with built-in grounding (Google Search), A2A protocol support, and compliance controls.

10 Risks, Failure Modes & Safety

10.1 Failure Mode Taxonomy

We identify seven dominant failure modes in agentic systems:

1. **Goal Drift**: Agent diverges from the original objective during extended reasoning chains.
2. **Infinite Loops**: Repeated action-observation cycles without progress toward termination.
3. **Irreversible Actions**: Unintended database deletions, financial transactions, or external communications.
4. **Tool Misuse**: Incorrect tool selection or parameter hallucination.
5. **Resource Exhaustion**: Uncontrolled token consumption, API cost escalation, or runtime timeout.
6. **Prompt Injection**: Adversarial manipulation through environmental inputs (documents, web pages).
7. **Cascade Failures**: Single-agent errors propagating through multi-agent topologies.

10.2 Mitigation Strategies

Table 13 maps each failure mode to concrete mitigation techniques drawn from production deployments and guardrail frameworks.

Notably, Guardrails AI (12,300+ GitHub stars) and NVIDIA NeMo Guardrails (4,800+ stars) have emerged as the dominant open-source guardrail frameworks, providing declarative policy definitions that can be composed with any orchestration layer.

10.3 Safety Architecture

We propose a four-layer defence-in-depth safety architecture for production agentic systems:

1. **L1—Input Validation**: Sanitise all external inputs (user prompts, tool outputs, retrieved documents) against prompt injection patterns. Enforce instruction hierarchy where system-level directives cannot be overridden by environmental content.

Table 13: Failure mode to mitigation mapping.

Failure Mode	**Mitigation Techniques**
Goal Drift	Step-budget limits; periodic goal re-grounding; trajectory summarisation with LLM-as-judge alignment checks
Infinite Loops	Cycle detection on (s_t, a_t) pairs; monotonic progress assertions; maximum iteration caps (default N=25)
Irreversible Actions	Action classification (read/write/destroy); mandatory human approval gates for destructive operations; dry-run sandboxing
Tool Misuse	Schema-validated tool calls; few-shot tool selection examples; fallback to LLM-only when confidence < 0.7
Resource Exhaustion	Per-session token budgets; exponential back-off on API errors; cost circuit breakers (e.g., $5 per-task cap)
Prompt Injection	Input sanitisation; instruction hierarchy (system > user > tool output); Guardrails AI content filters; NeMo Guardrails programmable rails
Cascade Failures	Agent-level isolation boundaries; bulkhead pattern limiting blast radius; supervisor watchdog with rollback authority

2. **L2—Output Filtering**: Apply content classifiers and schema validators to all LLM-generated outputs before they reach downstream tools. Reject outputs containing PII leakage, harmful content, or malformed tool-call schemas.

3. **L3—Action Sandboxing**: Execute all tool calls within isolated sandboxes with least-privilege permissions. File system operations are confined to designated directories; network calls are restricted to allow-listed endpoints; database mutations require explicit capability tokens.

4. **L4—Kill Switches**: Implement global and per-agent circuit breakers that trigger on anomalous behaviour (cost exceeding budget, error rate > 30%, latency exceeding 5× baseline). Kill switches halt execution, preserve state for forensic analysis, and notify human operators.

This layered model ensures that no single point of failure can result in uncontrolled agent behaviour, aligning with the defence-in-depth principle established in traditional cybersecurity [National Institute of Standards and Technology, 2023].

11 Regulation & Governance

11.1 EU AI Act Implications

The EU AI Act (effective August 2025) establishes a risk-based classification framework with direct implications for agentic systems:

- **Unacceptable Risk**: Agentic systems that perform social scoring or real-time biometric identification are prohibited outright.
- **High Risk**: Agents making autonomous decisions in employment (CV screening agents), credit scoring, healthcare triage, or legal document analysis trigger mandatory conformity assessments, logging requirements, and human oversight obligations under Annex III.
- **Limited Risk**: Conversational agents and customer-service copilots require transparency obligations—users must be informed they are interacting with an AI system.
- **Minimal Risk**: Internal code-generation agents and data-analysis copilots face no additional requirements beyond general product liability.

Critically, the Act's definition of "AI systems that interact with natural persons" captures most agentic deployments. The autonomous decision-making capability inherent in PPAS-loop agents (Section 3) elevates their risk classification compared to single-inference models, particularly when agents chain irreversible actions without human approval gates.

11.2 NIST AI RMF Mapping

The NIST AI Risk Management Framework (AI RMF 1.0) defines four core functions that map naturally onto the PPAS agent lifecycle:

Table 14: PPAS phases mapped to NIST AI RMF functions.

NIST Function	PPAS Phase	Implementation
Govern	System-level	Organisation-wide policies, role-based access, audit trails
Map	Perceive	Context identification, risk categorisation of inputs
Measure	Act + Self-Correct	Performance monitoring, bias detection, drift metrics
Manage	Self-Correct	Remediation actions, model updates, incident response

The PPAS self-correction phase provides a natural integration point for NIST's *Measure* and *Manage* functions, as the agent's reflective loop already evaluates action outcomes against expected results. Organisations can embed risk measurement directly into the self-correction prompt, enabling continuous compliance monitoring without architectural modifications.

11.3 Human-in-the-Loop Requirements

We identify three operational modes for human oversight in agentic systems, each appropriate for different risk levels:

1. **Full Autonomy** ($\mathcal{H} = \emptyset$): The agent executes the complete PPAS loop without human intervention. Appropriate only for minimal-risk applications (e.g., code formatting, internal log summarisation) where all actions are reversible and consequences bounded.

2. **Approval Gates** ($\mathcal{H} = \{a_t \mid \text{risk}(a_t) > \tau\}$): The agent autonomously handles low-risk actions but pauses execution and requests human approval before high-risk operations (financial transactions, external communications, data deletions). This mode aligns with EU AI Act requirements for high-risk systems.

3. **Human Override** ($\mathcal{H} = \mathcal{A}$): Every action requires explicit human confirmation. While maximally safe, this mode negates the efficiency benefits of agentic automation and is typically reserved for safety-critical domains (medical, legal, defence).

Our empirical results (Section 8, Study 1) demonstrate that HITL + Reflection achieves 72.5% on SWE-Bench Verified—substantially outperforming fully autonomous patterns (33.2–48.8%). This suggests that current foundation models benefit significantly from human oversight, and that premature removal of human checkpoints degrades both safety and performance.

12 Open Problems & Future Directions

1. **Long-Horizon Planning**: Current agents struggle with tasks requiring 50+ sequential steps.

2. **Cross-Domain Transfer**: Agents trained on coding tasks do not generalise to browsing or data analysis.

3. **Cost-Quality Frontiers**: Optimising the trade-off between model capability and inference cost.

4. **Self-Improvement**: Enabling agents to learn from past executions without catastrophic forgetting.

5. **Multi-Agent Trust**: Establishing trust and accountability in heterogeneous multi-agent systems.

6. **Standardisation**: Converging on unified protocols and evaluation standards.

13 Conclusion

This paper presented the PPAS (Perceive–Plan–Act–Self-Correct) framework as a unifying architectural model for goal-directed agentic AI systems. Through formal definition of the agent loop, decomposition of the 8-layer technology stack, benchmark meta-analysis, design pattern comparison, and protocol evaluation, we have provided a comprehensive reference for understanding, building, and evaluating modern agentic systems.

Our empirical studies yield five principal findings corresponding to our research questions:

RQ1 (Architectural Primitives): The PPAS framework unifies all surveyed agentic architectures under four phases—Perceive, Plan, Act, Self-Correct—with the 8-layer technology stack (Figure 2) providing a concrete mapping from conceptual phases to implementation components.

RQ2 (Design Patterns): Among the five dominant patterns, Plan-and-Execute achieves the highest autonomous performance (48.8% on SWE-Bench), while HITL + Reflection reaches 72.5%, demonstrating that human oversight remains essential for complex tasks.

RQ3 (Integration Protocols): MCP leads with 10,000+ registered tool servers, offering lowest tool-call latency (45ms). A2A enables robust multi-agent coordination at higher latency (120ms), while AG-UI provides real-time streaming for front-end integration.

RQ4 (Failure Modes): Seven dominant failure modes were identified, with goal drift and prompt injection posing the greatest risks. Our four-layer safety architecture (input validation, output filtering, action sandboxing, kill switches) provides defence-in-depth mitigation.

RQ5 (Governance): The EU AI Act classifies most autonomous agentic systems as high-risk, requiring conformity assessments and human oversight. The PPAS self-correction phase provides a natural integration point for NIST AI RMF compliance monitoring.

Our analysis reveals that the agentic AI paradigm represents a fundamental shift from reactive to proactive AI, with profound implications for system design, safety engineering, and governance. As the market grows from $7.3B to an estimated $139–$182B over the next decade, establishing principled architectural frameworks becomes essential for safe and effective deployment.

All data, analysis notebooks, and reproduction scripts are available at:
https://github.com/HAMEEMM/ai_systems_landscape_2026/tree/main/publications/01-agentic-ai

Acknowledgments

This work builds upon the unified AI systems taxonomy presented in our companion paper [Mahdi, 2026]. The author thanks the open-source agentic AI community for making benchmark data and framework implementations publicly available.

Data Availability

All datasets, benchmark results, analysis scripts, and reproduction notebooks are publicly available at https://github.com/HAMEEMM/ai_systems_landscape_2026/tree/main/publications/01-agentic-ai/data.

References

Anthropic. The Claude model family. Technical report, Anthropic, 2024a.

Anthropic. Model context protocol specification. https://modelcontextprotocol.io/, 2024b.

John R. Boyd. A discourse on winning and losing. Technical report, Air University Library, Maxwell Air Force Base, 1987.

CopilotKit. AG-UI: Agent-user interaction protocol. https://docs.ag-ui.com/, 2024.

Google. Agent-to-agent (A2A) protocol specification. https://google.github.io/A2A/, 2024.

Grand View Research. Agentic AI market size, share & trends analysis report. Technical report, Grand View Research, 2025.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? *International Conference on Learning Representations (ICLR)*, 2024.

John E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.

LangChain. LangGraph: Build stateful, multi-agent applications. https://langchain-ai.github.io/langgraph/, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. *International Conference on Learning Representations (ICLR)*, 2024.

Hameem Mahdi. A unified taxonomy of 19 AI system types: Architecture, capability, and deployment analysis for the modern AI landscape. *ACM Transactions on Intelligent Systems and Technology*, 2026. Under review.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.

Microsoft Research. AutoGen: Enable next-gen large language model applications. https://microsoft.github.io/autogen/, 2024.

National Institute of Standards and Technology. Artificial intelligence risk management framework (AI RMF 1.0). Technical Report NIST AI 100-1, U.S. Department of Commerce, 2023. URL https://www.nist.gov/artificial-intelligence/executive-order-safe-secure-and-trustworthy-artificial-intelligence.

OpenAI. GPT-4o system card. Technical report, OpenAI, 2024.

OpenAI. BrowseComp: A simple challenge for browsing agents. *arXiv preprint*, 2025.

Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, pages 312–319, 1995.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 2024.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changling Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin,

Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint*, 2024a.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *International Conference on Learning Representations (ICLR)*, 2024.