The Temporal Horizon: Engineering Long-Running Autonomous AI Agents for Durable Execution

1. Introduction: The Shift from Conversation to Continuity

The trajectory of artificial intelligence has historically been defined by the expansion of capability in discrete, episodic interactions. From the early era of classification models to the generative revolution of 2023, the dominant paradigm was "statelessness"—a model receives an input, processes it, generates an output, and resets. However, the period between late 2024 and early 2026 has witnessed a fundamental phase transition in AI systems engineering: the emergence of **long-running autonomous agents**. These are systems designed not merely to respond, but to exist, persist, and execute complex workflows spanning hours, days, or weeks with minimal human supervision.

This report investigates the architectural imperatives required to sustain AI cognition over these extended temporal horizons. As of 2026, the primary constraint on AI utility in enterprise environments is no longer reasoning capability per se, but the **temporal stability** of that capability. An agent that can solve a difficult coding problem in ten minutes is useful; an agent that can maintain the architectural coherence of a large-scale refactoring project over three days without succumbing to "context amnesia" or "hallucinated progress" is transformative.

The engineering challenges inherent in this shift are profound. They require moving beyond the naive "loop-and-prompt" architectures that characterized early agent frameworks. Production-grade long-running agents demand a synthesis of distributed systems engineering—specifically **durable execution** patterns—and cognitive architecture. We must solve for the stochastic nature of Large Language Models (LLMs) within the deterministic rigidities of enterprise software environments. This necessitates robust state management, standardized interoperability protocols like the **Model Context Protocol (MCP)** and **Agent2Agent (A2A)**, and novel memory systems that utilize **Knowledge Graphs** and **Nested Learning** to preserve intent across massive context windows.

This analysis draws upon the latest research from METR (Model Evaluation & Threat Research), Anthropic, Microsoft, and Google to provide a comprehensive blueprint for building agents that do not just think, but work. We explore the exponential growth of the "time horizon"—the duration of tasks agents can reliably complete—and dissect the "harnesses" that make such horizons possible. From the "Initializer/Worker" patterns to neurosymbolic guardrails, this report details the mechanisms that turn probabilistic models into reliable digital employees.[1]

---

2. Quantifying Autonomy: The Science of Time Horizons

To engineer for the long term, we must first measure it. Traditional benchmarks—such as MMLU (Massive Multitask Language Understanding) or HumanEval—measure static reasoning capabilities on isolated tasks. They fail to capture the degradation of performance that occurs as a task extends in time and complexity. The defining metric for the current generation of agents is the **Time Horizon**.

2.1 The Exponential Growth of the Effective Horizon

Research conducted by METR has established a rigorous methodology for quantifying agent capability in terms of human-equivalent time. The "Time Horizon" is defined as the duration of a task (as performed by a human expert) that an AI system can complete with a 50% success rate. This metric serves as a proxy for the agent's ability to maintain context, plan hierarchically, and recover from errors over time.[1]

The data reveals a robust exponential trend. The effective time horizon of frontier models has been doubling approximately every 7 months since 2019. In 2024, models struggled with tasks requiring more than 15 minutes of independent work. By early 2025, systems like Claude 3.7 Sonnet and GPT-5 pushed this boundary to nearly an hour. Extrapolations based on this trend suggest that before the end of the decade, we will witness agents capable of autonomously executing month-long projects.[4]

This growth is not merely a function of larger context windows (though 1M+ token windows are a prerequisite). It is driven by improvements in **reliability** and **error recovery**. An agent working for a week will inevitably make mistakes; the "horizon" is determined by its ability to detect those mistakes (self-correction) rather than proceeding down a path of divergent failure.[6]

2.2 Benchmarking the Messy Reality: HCAST and RE-Bench

To validate these horizons, the industry has moved toward benchmarks that simulate the unstructured "messiness" of real-world engineering.

- **HCAST (High-Complexity Autonomous Software Tasks):** This benchmark evaluates agents on tasks ranging from cybersecurity exploits to low-level algorithmic optimization. It reveals a stark bifurcation: frontier agents achieve near-perfect success on tasks equivalent to <5 minutes of human effort but degrade to <20% success on tasks exceeding 4 hours. This "cliff" represents the current frontier of autonomous reliability.[7]
- **RE-Bench (Research Engineering Benchmark):** Designed to test AI R&D capabilities,

RE-Bench compares agents against human experts with 8-hour time budgets. This benchmark exposes the "last mile" problem: agents can often generate a theoretical solution faster than humans but fail to navigate the subtle environment configuration issues or dependency conflicts that a human would intuitively resolve. Interestingly, humans still show better returns on time investment; given more time, humans improve their score more reliably than agents, which tend to plateau or diverge.[9]

2.3 The Evolution of SWE-bench Verified

The most visible indicator of this progress is the evolution of **SWE-bench**. The original benchmark, comprised of GitHub issues, suffered from underspecification. The release of **SWE-bench Verified** in late 2024 addressed this by filtering for tasks with clear, unambiguous success criteria.

Performance on SWE-bench Verified has served as a bellwether for the industry. Between April and November 2025, success rates for top systems (like **Amazon Q Developer** and **Claude 3.7**) doubled from ~25% to over 50%.[11] This metric is critical because it measures "real" engineering work—navigating an existing codebase, reproducing a bug, and implementing a fix without breaking existing functionality. The rapid saturation of this benchmark suggests that for tasks with a horizon of 1-2 hours, agents have reached near-parity with junior engineers.[12]

2.4 Economic Implications of Long-Running Agents

The shift to long-running agents necessitates a reimagining of AI economics. The traditional "per-seat" SaaS model is incompatible with agents that consume variable, non-deterministic amounts of compute.

Pricing Model	Description	Suitability for Agents
Action-Based	Charges per discrete tool call or API interaction.	**High:** Aligns cost with the agent's "activity" but can be unpredictable for users.
Outcome-Based	Charges only for successful resolution (e.g., bug fixed).	**Medium:** High risk for the provider if the agent spins for days and fails.
Agent-as-an-Employee	Flat monthly fee for an autonomous entity (e.g., AWS Kiro).	**High:** Mimics the human labor model; incentivizes the provider to optimize

		efficiency.[14]
Token Pass-Through	Direct billing for inference + tool costs.	**Low:** Too complex for enterprise procurement; creates "bill shock" risk.

As "inference costs drop by orders of magnitude" (9x-900x per year), the margin protection strategies for AI companies are shifting from raw token arbitrage to the value of the **agentic workflow** itself.[15]

---

3. The Engineering of Context: Architecture Patterns

The fundamental technical challenge of long-running agents is **Context Amnesia**. LLMs are stateless processors; they have no inherent memory of previous states unless that history is re-injected into the context window. Even with 1M+ token windows, "context stuffing" leads to degradation in reasoning quality (the "lost-in-the-middle" phenomenon) and prohibitive latency. Successful long-running architectures, therefore, rely on **Harnesses**—external control systems that manage the agent's focus and memory.[3]

3.1 The "Initializer" and "Worker" Pattern

Anthropic's engineering teams have formalized a "two-fold" solution to bridge the context gap across sessions, a pattern that has become a standard for production-grade systems.[17]

The Initializer Agent

This agent runs only once at the genesis of a project. Its specific role is **Environment Scaffolding**. It does not attempt to solve the user's problem directly; instead, it prepares the environment for the agents that will follow.

- **Artifact Generation:** It creates an init.sh script to set up dependencies and a claude-progress.txt (or equivalent) to log high-level milestones.
- **Feature Decomposition:** Crucially, it generates a feature_list.json. This file breaks the high-level goal (e.g., "Build a Tetris clone") into granular, verifiable steps (e.g., "Initialize canvas," "Create block structure," "Implement gravity"). This prevents the model from attempting to "one-shot" complex applications—a primary cause of failure.[17]

The Coding (Worker) Agent

Subsequent sessions are handled by ephemeral Worker agents. These agents are designed to be stateless execution units.

- **The Re-orientation Phase:** Every Worker begins its lifecycle by reading the claude-progress.txt and the feature_list.json. It executes pwd and git status to "ground" itself in the current reality of the file system.

- **Incremental Execution:** The agent is prompted to select *one* feature from the JSON list, implement it, verify it, and update the status to passes: true.
- **The Commit Checkpoint:** The session concludes with a git commit. This "commit-as-checkpoint" strategy is vital; if an agent hallucinates or destroys code, the damage is contained to a single commit that can be reverted by the next worker.[17]

3.2 Structured State Handoffs: JSON vs. Markdown

The format of the "memory file" significantly impacts agent reliability.

- **JSON (feature_list.json):** Research indicates that JSON is superior for structural integrity. Models are less likely to hallucinate changes to a JSON schema than to free-text Markdown. It enforces a binary state ("passes": false) that requires an explicit action to flip, acting as a forcing function for verification.[17]
- **Markdown (todo.md):** While less rigid, Markdown is essential for "Chain of Thought" persistence. It allows the agent to leave qualitative notes for its future self (e.g., "Tried library X, failed due to version conflict, pivoting to library Y"). This acts as an **externalized working memory**, mitigating the loss of reasoning chains between sessions.[19]
- **Git History:** The ultimate source of truth. Advanced harnesses train agents to read git log -p to understand *exactly* what changed in the previous session, rather than relying on the previous agent's potentially unreliable summary.[17]

3.3 Context Compaction and "Recency Bias"

For workflows that exceed the context window, **compaction** algorithms are employed. A naive sliding window (deleting the oldest messages) is catastrophic because it often removes the original System Prompt or the root goal.

- **The Compaction Strategy:** The harness summarizes the "middle" of the conversation history while preserving:
  1. **The System Prompt:** The immutable instructions.
  2. **The Initial Goal:** The user's original request.
  3. **The Recent Context:** The last N messages, leveraging the model's "recency bias" to ensure immediate responsiveness.
- **Tool Output Pruning:** Tool outputs (e.g., cat on a large file or verbose logs) are the primary consumers of tokens. Compaction replaces these raw outputs with semantic summaries (e.g., "Read file data.csv. 500 rows. Columns: A, B, C.").[20]

3.4 The "Definition of Done" Protocol

A common failure mode is **Premature Completion**, where an agent declares a task finished after writing the code but before verifying it works. To mitigate this, the harness enforces a **Definition of Done**.

- **Verification Loops:** The agent is structurally prevented from emitting a "Task Complete" signal until it has executed a verification command (e.g., pytest, npm test) and received a passing exit code.
- **Zero-Score Samples:** Training datasets for agents now include "negative examples" where an agent is penalized for marking a task complete without verification, calibrating the model's internal confidence threshold.[22]

---

4. Durable Execution & State Persistence

While the harness manages the *context* (what the AI sees), the underlying infrastructure must manage the *execution state* (what the system is doing). In a system running for days, failures are inevitable: servers crash, APIs time out, and models hit rate limits. The agent must be resilient to these interruptions.

4.1 Checkpointing: The LangGraph Approach

LangGraph, a dominant framework in the Python ecosystem, introduces a formal **Checkpointer** interface to solve persistence.[24]

- **The Super-Step:** LangGraph treats the agent's execution as a graph traversal. At every "super-step" (a node execution), the system automatically saves a StateSnapshot.
- **Thread ID as Primary Key:** State is keyed by a thread_id. This allows the system to be stopped, serialized to a database (Postgres, Redis), and resumed days later on a different server.
- **Time Travel & Forking:** Because every step is checkpointed, operators can "rewind" an agent to a previous state. If an agent deletes a critical file, the human operator can revert the graph to the state *before* that action, edit the state to correct the agent's assumption, and resume execution. This is critical for **Human-in-the-Loop (HITL)** debugging workflows.[26]

4.2 Durable Execution: The Temporal.io Model

For enterprise-grade reliability, the **Durable Execution** paradigm—popularized by **Temporal.io**—offers a more robust guarantee than simple database checkpointing. Temporal abstracts the execution into a workflow that is immune to hardware failure.[27]

- **Code as Workflow:** In Temporal, the agent's logic is written as a Workflow. The "thinking" steps (LLM calls) and "acting" steps (Tool calls) are modeled as **Activities**.
- **Event Sourcing & Replay:** Temporal does not just save the current state; it saves the entire **Event History**. If a workflow crashes, Temporal restarts it on a healthy worker and "replays" the history.
  - *Determinism:* When replaying, Temporal skips the actual execution of already-completed activities (e.g., it won't charge the credit card twice) and instead injects the *result* recorded in the history.

  - *Resilience:* This effectively makes the agent "immortal." It can sleep for a month (e.g., waiting for a user approval email) and wake up with its full state intact, without consuming resources during the wait.[29]

4.3 Database Patterns for Persistence

Different persistence layers serve different needs in the agent stack:

- **Redis:** Ideal for **Short-Term Memory** and "hot" state (e.g., the current conversation history). Its low latency supports the high-speed read/write loops of the agent.[31]
- **PostgreSQL/SQLite:** Used for **Checkpoints** and structured logs. LangGraph's AsyncPostgresSaver is the standard for production deployments requiring durable, queryable state.[32]
- **Vector Stores (Pinecone/Weaviate):** Used for **Semantic Search**, allowing the agent to retrieve relevant snippets from massive documentation repositories. However, vector search is increasingly being augmented by GraphRAG for better precision (see Section 6).

---

5. Interoperability & The Protocol Layer

As the number of specialized agents grows, the "Silo Problem" emerges: a coding agent from Vendor A cannot talk to a QA agent from Vendor B. 2025/2026 has seen the crystallization of standard protocols to solve this.

5.1 Model Context Protocol (MCP)

Developed by Anthropic and adopted by major players (including AWS and Block), **MCP** is the "USB-C for AI applications".[33]

- **Standardization:** MCP defines a standard way for agents to discover and invoke **Resources** (data sources like files or database rows), **Prompts** (templates), and **Tools** (executable functions).
- **Decoupling:** Prior to MCP, an agent needed a custom integration for Google Drive, another for Slack, and another for GitHub. With MCP, a "Slack MCP Server" exposes a standardized interface. Any MCP-compliant agent (whether it's Claude, a VS Code extension, or a custom internal tool) can connect to it without bespoke code.
- **Security:** MCP operates at the transport layer, enabling strict **Authorization**. The user explicitly grants the agent access to specific resources (e.g., "You can read *this* folder, but not *that* one"). This creates a security boundary that prevents the agent from overreaching.[35]

5.2 Agent2Agent (A2A) Protocol

While MCP connects Agents to Tools, Google's **Agent2Agent (A2A)** protocol connects

Agents to *Agents*.[37]

- **Capability Discovery:** A2A allows agents to broadcast an "Agent Card"—a JSON-LD manifest describing their capabilities, inputs, and outputs. A "Client Agent" (e.g., a Project Manager) can dynamically discover a "Service Agent" (e.g., a Python Expert) to delegate a task.
- **Negotiation & Handshake:** Agents utilize A2A to negotiate the terms of the interaction. For example, the Client Agent might request the output in JSON format, while the Service Agent defaults to Markdown. A2A handles this content negotiation.
- **Cross-Vendor Orchestration:** The protocol is designed to be framework-agnostic, allowing a LangGraph agent to sub-contract a task to a CrewAI swarm or an AutoGen agent, breaking down the walled gardens of proprietary frameworks.[39]

5.3 Comparison of Protocols

Feature	**Model Context Protocol (MCP)**	**Agent2Agent (A2A)**
Primary Scope	Agent <-> Tool/Data	Agent <-> Agent
Focus	Context injection, Tool discovery	Task delegation, Collaboration
Transport	JSON-RPC over Stdout/HTTP	HTTP / SSE (Server-Sent Events)
Key Adopters	Anthropic, AWS, Replit, Zed	Google, Vertex AI Partners
State Management	Stateless (mostly)	Stateful (supports long-running tasks)

---

6. Cognitive Architecture: Memory, Learning, & Neurosymbolic Systems

To achieve multi-day horizons, agents require a "cognitive architecture" that goes beyond simple context retention. They need structured long-term memory and the ability to learn from experience.

6.1 Knowledge Graphs and GraphRAG

For agents running for weeks, unstructured vector memory is insufficient—it is "fuzzy" and prone to hallucinating connections. **Knowledge Graphs** (e.g., Neo4j) provide a deterministic, structured memory.[40]

- **The Schema:** Agents map their environment into Nodes (Entities: Users, Files, Functions) and Edges (Relationships: CALLS, MODIFIES, OWNED_BY).
- **GraphRAG:** Retrieval Augmented Generation is enhanced by the graph structure. Instead of searching for "files about login," the agent queries: MATCH (f:File)-->(a:Agent {id: "Worker-1"}) RETURN f. This provides precise, auditable recall of previous actions.[41]
- **Contextual Anchoring:** The graph serves as an anchor. Even if the context window is flushed, the graph persists the *relationships* between the artifacts the agent has created.

6.2 Nested Learning & Continual Improvement

Google's **Nested Learning** paradigm addresses the "Plasticity-Stability Dilemma"—how an agent can learn new things (plasticity) without forgetting old instructions (stability).[43]

- **Architectural Separation:** The model is treated as a stack of learning loops operating at different timescales.
  - *Inner Loop:* Fast adaptation to the immediate task (e.g., learning the coding style of the current repo).
  - *Outer Loop:* Slow consolidation of general strategies (e.g., "This type of error usually requires a database migration").
- **Practical Application:** In a long-running agent, the "Inner Loop" is often implemented via a dynamic "System Prompt" that is updated by a "Reflector" agent at the end of each session, condensing the day's lessons into rules for tomorrow.[26]

6.3 SiriuS: Self-Improving Systems

The **SiriuS** framework (NeurIPS 2025) demonstrates how agents can bootstrap their own capability without human fine-tuning.[45]

- **Trajectory Mining:** The system maintains an **Experience Library** of interaction logs. Successful trajectories are preserved.
- **Failure Augmentation:** When an agent fails, a "Critic" agent analyzes the failure, generates a correction, and the system re-simulates the task using this feedback. If the re-simulation succeeds, this "synthetic" trajectory is added to the library.
- **Bootstrapping:** The agent is periodically fine-tuned on this library, effectively creating a "self-healing" cycle where the agent becomes more expert in its specific domain (e.g., a proprietary codebase) over time.[45]

6.4 Neurosymbolic Guardrails

In high-stakes domains (Finance, DevOps), probabilistic safety is inadequate. **Neurosymbolic** architectures hybridize neural intuition with symbolic logic.[47]

- **The Sandwich Pattern:**
  1. **Neural (Planner):** The LLM proposes a plan (e.g., "Delete unused tables").
  2. **Symbolic (Verifier):** A deterministic rule engine checks the plan against strict logic (e.g., IF command == DROP_TABLE AND env == PRODUCTION THEN REJECT).
  3. **Neural (Executor):** If verified, the LLM executes the plan.
- **Impact:** This guarantees that no matter how much the model hallucinates, it cannot violate hard constraints defined in the symbolic layer.[49]

---

7. The Framework Landscape (2025/2026)

The ecosystem of agent frameworks has consolidated significantly. The "framework wars" have settled into a few dominant platforms, each serving a distinct niche.

7.1 Microsoft Agent Framework (Unified)

Microsoft has merged **AutoGen** and **Semantic Kernel** into a single, unified **Microsoft Agent Framework**.[50]

- **Enterprise Focus:** It combines the event-driven orchestration of AutoGen (ideal for multi-agent chat patterns) with the plugin/kernel architecture of Semantic Kernel (ideal for integration with enterprise apps).
- **Features:** It offers native support for MCP, deep integration with Azure AI Foundry for observability, and rigorous compliance features (SOC 2, HIPAA) for enterprise deployment. It is the default choice for.NET/Azure shops.

7.2 LangGraph

LangGraph has emerged as the standard for **stateful, cyclic orchestration** in the Python/Open Source ecosystem.[24]

- **Cyclic Graphs:** Unlike earlier DAG-based frameworks (like basic LangChain), LangGraph supports loops (Plan $\rightarrow$ Execute $\rightarrow$ Critique $\rightarrow$ Plan). This cyclic capability is essential for the self-correction loops required for long horizons.
- **Control:** It provides low-level control over state and persistence, making it the preferred choice for engineers building custom, highly complex agent runtimes.[26]

7.3 CrewAI

CrewAI has carved out a niche in **Role-Based** abstractions.[53]

- **Abstraction Level:** It excels at high-level role definition (e.g., "You are a Researcher," "You are a Writer"). It is less about the low-level graph and more about the "business logic" of the team.
- **Adoption:** With broad adoption in the Fortune 500, it is often used for rapid prototyping

and business process automation where the rigid state control of LangGraph is overkill.

Framework	Primary Philosophy	Best For	Key Strength
MS Agent Framework	**Unified Enterprise**	Corporate IT, Azure Ecosystem	Compliance,.NET support, AutoGen/SK merger.
LangGraph	**Stateful Control**	Python Engineers, Complex Runtimes	Low-level state control, cyclic graphs, persistence.
CrewAI	**Role-Based**	Business Logic, Rapid Prototyping	Ease of use, high-level abstractions, "Team" metaphor.

8. Operationalizing Agents: Economics & Observability

Running long-running agents introduces operational challenges distinct from standard LLM apps.

8.1 Token Economics & Optimization

The cost of an agent is not linear; it is recursive. A loop that runs 10 times costs 10x the tokens.

- **Prompt Caching:** This is the single most effective optimization. By placing static context (codebase summaries, API docs) at the *start* of the prompt, providers like Anthropic and OpenAI can cache the attention states. This reduces the cost of input tokens by up to 90% and significantly reduces latency for subsequent steps in the loop.[21]
- **Speculative Decoding:** For agents that generate structured data (JSON), **Speculative Decoding** utilizes a small "draft" model to predict the next tokens, which are then verified by the larger model. This accelerates inference and reduces the "wall clock time" of the agent, which is crucial for multi-step workflows.[55]

8.2 Observability with LangSmith & AgentCore

You cannot improve what you cannot see. Observability platforms like **LangSmith** and **Amazon Bedrock AgentCore** provide the necessary X-ray vision into the agent's thought process.[57]

- **Tracing:** Every step (LLM call, tool invocation, parser error) is traced. This allows engineers to pinpoint exactly *where* the agent went off the rails—was it a bad retrieval, a hallucination, or a tool failure?
- **Insights Agent:** New features like "Insights Agent" in LangSmith automatically categorize failure modes across thousands of runs, helping engineers identify systemic issues (e.g., "The agent fails 40% of the time when using the SQL tool").[59]

---

9. Future Directions: The Agentic IDE

The culmination of these technologies is the **Agentic IDE**. Tools like **AWS Kiro** and **Cursor** are transforming the IDE from a text editor into a stateful, agentic environment.[14]

- **Kiro's "Powers":** Kiro introduces modular expertise packages called "Powers" (MCP servers + steering files). A developer can "install" the "Datadog Power," instantly giving the IDE agent the ability to query metrics and understand SRE best practices.[14]
- **The Shift:** This represents the move from "Chat" to "Integration." The agent is not a side panel; it is deeply woven into the file system, the terminal, and the cloud infrastructure.

10. Conclusion

The transition to long-running autonomous agents represents a maturation of AI engineering. It is a shift from the "magical" thinking of 2023—where the prompt was everything—to the "structural" thinking of 2026. Success in this new era requires a disciplined approach to **state management**, **protocols**, and **verification**.

The "Time Horizon" is the new benchmark. Extending it requires:

1. **Harnesses** (like the Initializer/Worker pattern) to manage context.
2. **Durable Execution** (like Temporal) to manage state.
3. **Standard Protocols** (MCP/A2A) to manage interoperability.
4. **Neurosymbolic Memory** (GraphRAG) to manage knowledge.

For the AI engineer, the task is no longer just to "prompt the model," but to "architect the employee." The systems described in this report—capable of working for days, recovering from failure, and learning from experience—are the early prototypes of the digital workforce that will define the next decade of software development.

Works cited

1. Measuring AI Ability to Complete Long Tasks - METR, accessed February 7, 2026, https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/
2. Enterprise Agentic Architecture and Design Patterns | Salesforce Architects, accessed February 7, 2026, https://architect.salesforce.com/fundamentals/enterprise-agentic-architecture
3. Long-running Agents - What It Means in AI | Glossary - TeamDay.ai, accessed February 7, 2026, https://www.teamday.ai/de/ai/glossary/long-running-agents
4. [2503.14499] Measuring AI Ability to Complete Long Tasks - arXiv, accessed February 7, 2026, https://arxiv.org/abs/2503.14499
5. METR: Measuring AI Ability to Complete Long Tasks - LessWrong, accessed February 7, 2026, https://www.lesswrong.com/posts/deesrjitvXM4xYGZd/metr-measuring-ai-ability-to-complete-long-tasks
6. Measuring AI Ability to Complete Long Tasks - arXiv, accessed February 7, 2026, https://arxiv.org/html/2503.14499v1
7. HCAST: Human-Calibrated Autonomy Software Tasks - arXiv, accessed February 7, 2026, https://arxiv.org/html/2503.17354v1
8. HCAST: Human-Calibrated Autonomy Software Tasks - METR, accessed February 7, 2026, https://metr.org/hcast.pdf
9. ICML Poster RE-Bench: Evaluating Frontier AI R&D Capabilities of Language Model Agents against Human Experts, accessed February 7, 2026, https://icml.cc/virtual/2025/poster/46519
10. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts - METR, accessed February 7, 2026, https://metr.org/AI_R_D_Evaluation_Report.pdf
11. Between April and now, SWE-bench Verified scores (percentage of problems solved) doubled from less than 25% to over 50% : r/singularity - Reddit, accessed February 7, 2026, https://www.reddit.com/r/singularity/comments/1gjqxdz/between_april_and_now_swebench_verified_scores/
12. SWE-BENCH+: ENHANCED CODING BENCHMARK FOR LLMS - OpenReview, accessed February 7, 2026, https://openreview.net/pdf?id=pwIGnH2LHJ
13. What skills does SWE-bench Verified evaluate? | Epoch AI, accessed February 7, 2026, https://epoch.ai/blog/what-skills-does-swe-bench-verified-evaluate
14. AWS re:Invent 2025: Amazon announces Nova 2, Trainium3, frontier ..., accessed February 7, 2026, https://www.aboutamazon.com/news/aws/aws-re-invent-2025-ai-news-updates
15. AI Agent Cost-Based Pricing, accessed February 7, 2026, https://nevermined.ai/blog/ai-agent-cost-based-pricing
16. Quick reality check. Is this “working memory” approach for Copilot Agent Mode worth pursuing? : r/GithubCopilot - Reddit, accessed February 7, 2026, https://www.reddit.com/r/GithubCopilot/comments/1p2vz7c/quick_reality_check_is_this_working_memory/

17. Effective harnesses for long-running agents \ Anthropic, accessed February 7, 2026, https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents
18. Agent Harnesses: From DIY Patterns to Product - Emergent Minds | paddo.dev, accessed February 7, 2026, https://paddo.dev/blog/agent-harnesses-from-diy-to-product/
19. Building Long-Running Deep Research Agents: Architecture, Attention Mechanisms, and Real-World Applications | by Madhur Prashant | Medium, accessed February 7, 2026, https://medium.com/@madhur.prashant7/building-long-running-deep-research-agents-architecture-attention-mechanisms-and-real-world-11f559614a9c
20. Context windows - Claude API Docs, accessed February 7, 2026, https://platform.claude.com/docs/en/build-with-claude/context-windows
21. Context Engineering: 4 Ways Your Agent's Context Fails - Firecrawl, accessed February 7, 2026, https://www.firecrawl.dev/blog/context-engineering
22. Robo-Dopamine: General Process Reward Modeling for High-Precision Robotic Manipulation - arXiv, accessed February 7, 2026, https://arxiv.org/html/2512.23703v1
23. ATOD: An Evaluation Framework and Benchmark for Agentic Task-Oriented Dialogue Systems - arXiv, accessed February 7, 2026, https://arxiv.org/html/2601.11854v2
24. Persistence - Docs by LangChain, accessed February 7, 2026, https://docs.langchain.com/oss/javascript/langgraph/persistence
25. Persistence - Docs by LangChain, accessed February 7, 2026, https://docs.langchain.com/oss/python/langgraph/persistence
26. Mastering Persistence in LangGraph: Checkpoints, Threads, and Beyond | by Vinod Rane, accessed February 7, 2026, https://medium.com/@vinodkrane/mastering-persistence-in-langgraph-checkpoints-threads-and-beyond-21e412aaed60
27. Temporal + LangGraph: A Two-Layer Architecture for Multi-Agent Coordination - lightsong - 博客园, accessed February 7, 2026, https://www.cnblogs.com/lightsong/p/19530436
28. How To Build a Durable AI Agent with Temporal and Python, accessed February 7, 2026, https://learn.temporal.io/tutorials/ai/durable-ai-agent/
29. How Retool built robust Workflow and Agents products on Temporal, accessed February 7, 2026, https://temporal.io/resources/case-studies/how-retool-built-robust-workflow-agents-products
30. Lindy Boosts Reliability and Observability of AI Agents with Temporal Cloud, accessed February 7, 2026, https://temporal.io/resources/case-studies/lindy-reliability-observability-ai-agents-temporal-cloud
31. LangGraph & Redis: Build smarter AI agents with memory & persistence, accessed February 7, 2026,

https://redis.io/blog/langgraph-redis-build-smarter-ai-agents-with-memory-persistence/
32. Human-in-the-loop - Docs by LangChain, accessed February 7, 2026, https://docs.langchain.com/oss/python/langchain/human-in-the-loop
33. Specification and documentation for the Model Context Protocol - GitHub, accessed February 7, 2026, https://github.com/modelcontextprotocol/modelcontextprotocol
34. What is the Model Context Protocol (MCP)? - Model Context Protocol, accessed February 7, 2026, https://modelcontextprotocol.io/
35. Authorization - Model Context Protocol, accessed February 7, 2026, https://modelcontextprotocol.io/specification/draft/basic/authorization
36. Tools - Model Context Protocol, accessed February 7, 2026, https://modelcontextprotocol.io/docs/concepts/tools
37. accessed February 7, 2026, https://agent2agent.ren/#:~:text=A2A%20Protocol%20Documentation&text=A2A%20(Agent2Agent)%20is%20an%20open,more%20complex%20cross%2Dapplication%20automation.
38. Announcing the Agent2Agent Protocol (A2A) - Google Developers ..., accessed February 7, 2026, https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
39. Getting Started with Agent2Agent (A2A) Protocol: A Purchasing Concierge and Remote Seller Agent Interactions on Cloud Run and Agent Engine | Google Codelabs, accessed February 7, 2026, https://codelabs.developers.google.com/intro-a2a-purchasing-concierge
40. Meet Lenny's Memory: Building Context Graphs for AI Agents | by William Lyon - Medium, accessed February 7, 2026, https://medium.com/neo4j/meet-lennys-memory-building-context-graphs-for-ai-agents-24cb102fb91a
41. Knowledge Graph Generation - Graph Database & Analytics - Neo4j, accessed February 7, 2026, https://neo4j.com/blog/developer/knowledge-graph-generation/
42. LLM Knowledge Graph Builder: From Zero to GraphRAG in Five Minutes - Neo4j, accessed February 7, 2026, https://neo4j.com/blog/developer/graphrag-llm-knowledge-graph-builder/
43. What Is Continual Learning? (And Why It Powers Self-Learning AI Agents) - Beam AI, accessed February 7, 2026, https://beam.ai/agentic-insights/what-is-continual-learning-(and-why-it-powers-self-learning-ai-agents)
44. Nested Learning Changes the Game for Continual AI | by Marco Mastrodonato - Medium, accessed February 7, 2026, https://medium.com/@m.mastrodonato/nested-learning-changes-the-game-for-continual-ai-8cfe44060ef6
45. NeurIPS Poster SiriuS: Self-improving Multi-agent Systems via Bootstrapped Reasoning, accessed February 7, 2026, https://neurips.cc/virtual/2025/poster/118834

46. zou-group/sirius: SiriuS: Self-improving Multi-agent ... - GitHub, accessed February 7, 2026, https://github.com/zou-group/sirius
47. Neurosymbolic AI as an antithesis to scaling laws | PNAS Nexus - Oxford Academic, accessed February 7, 2026, https://academic.oup.com/pnasnexus/article/4/5/pgaf117/8134151
48. From Logic to Learning: The Future of AI Lies in Neuro-Symbolic Agents, accessed February 7, 2026, https://builder.aws.com/content/2uYUowZxjkh80uc0s2bUji0C9FP/from-logic-to-learning-the-future-of-ai-lies-in-neuro-symbolic-agents
49. Autonomous Business System via Neuro-symbolic AI - ResearchGate, accessed February 7, 2026, https://www.researchgate.net/publication/400002889_Autonomous_Business_System_via_Neuro-symbolic_AI
50. Introduction to Microsoft Agent Framework, accessed February 7, 2026, https://learn.microsoft.com/en-us/agent-framework/overview/agent-framework-overview
51. Semantic Kernel and Microsoft Agent Framework, accessed February 7, 2026, https://devblogs.microsoft.com/semantic-kernel/semantic-kernel-and-microsoft-agent-framework/
52. A Detailed Comparison of Top 6 AI Agent Frameworks in 2025 - Turing, accessed February 7, 2026, https://www.turing.com/resources/ai-agent-frameworks
53. Top 7 Agentic AI Frameworks in 2026: LangChain, CrewAI, and Beyond, accessed February 7, 2026, https://www.alphamatch.ai/blog/top-agentic-ai-frameworks-2026
54. Top 6 AI Agents for Enterprises in 2026 - Activepieces, accessed February 7, 2026, https://www.activepieces.com/blog/ai-agents-for-enterprises
55. 6 Production-Tested Optimization Strategies for High-Performance LLM Inference - BentoML, accessed February 7, 2026, https://www.bentoml.com/blog/6-production-tested-optimization-strategies-for-high-performance-llm-inference
56. Speculative Actions: A Lossless Framework for Faster AI Agents | OpenReview, accessed February 7, 2026, https://openreview.net/forum?id=P0GOk5wslg
57. AI Agent Observability & Monitoring Platform - LangSmith - LangChain, accessed February 7, 2026, https://www.langchain.com/langsmith/observability
58. Amazon Bedrock AgentCore: How AWS Just Solved the AI Agent Production Problem, accessed February 7, 2026, https://www.refactored.pro/blog/2025/7/19/ai-agents-aws-bedrock-agentcore
59. Introducing Insights Agent & Multi-turn Evals, new features in LangSmith!, accessed February 7, 2026, https://forum.langchain.com/t/introducing-insights-agent-multi-turn-evals-new-features-in-langsmith/1914
60. AWS' Kiro launches autonomous agents for individual developers, accessed February 7, 2026, https://www.constellationr.com/insights/news/aws-kiro-launches-autonomous-agents-individual-developers