From Text Generators to Autonomous Agents: How Large Language Models Are Being Extended with Tools, Memory, and Planning

Sakshya Bhattacharya

A Research Paper on Modern AI: Technical Advancements in LLM Agents

***Abstract*—Not long ago, a large language model was essentially a very sophisticated autocomplete a system that predicted what word should come next, scaled to an almost incomprehensible degree. That description, while technically accurate, no longer captures what these systems actually do. Today's most capable LLMs can browse the web, write and run code, manage files, coordinate with other AI systems, and pursue goals across dozens of sequential steps without a human holding their hand at each turn. This paper traces the technical innovations that made this shift possible. We examine tool-use mechanisms, memory and retrieval architectures, planning and reasoning frameworks, and the emerging ecosystem of multi-agent systems. We take a close look at the platforms that have defined this space LangChain, AutoGPT, Claude Code, and others weighing their design choices honestly. The paper's central argument is simple: the move from passive text generation to goal-directed, tool- equipped agency is not a minor upgrade. It is a fundamental change in what artificial intelligence is for.**

I. INTRODUCTION

There is a tendency in technology journalism to treat every AI announcement as either catastrophic or miraculous. The actual story of how large language models became agents is more interesting than either framing it is a story about engineering creativity, unexpected emergent behavior, and the slow accumulation of good ideas that suddenly, in aggregate, change what a technology is capable of.

Start with what LLMs were designed to do: predict the next token in a sequence. At small scales, this produces autocomplete. At massive scales hundreds of billions of parameters, trained on vast swaths of human-written text it produces something that resembles understanding, reasoning, and even creativity. But through 2021 and most of 2022, the dominant paradigm was still essentially one-shot: you give the model a prompt, it gives you a response, and that is where the interaction ends.

The shift toward agency required asking a different question. Instead of 'what text should follow this prompt?' researchers and developers began asking 'what should I do next to accomplish this goal?' That change in framing from completion to action pulled on a thread that unraveled the old paradigm entirely. Suddenly, models needed memory that persisted across steps. They needed tools real software they could invoke to interact with the world. They needed the ability to plan sequences of actions, check whether those actions succeeded, and adjust when they did not. And increasingly, they needed to work alongside other AI systems rather than in isolation.

This paper is a technical account of how those capabilities were built. We work through four interconnected layers of LLM agency tool use, memory, planning, and multi-agent coordination before surveying the frameworks that have put these ideas into practice. The goal is not to be exhaustive but to be genuinely useful to someone who wants to understand what is actually happening under the hood of a modern AI agent.

II. WHAT WE ARE BUILDING ON: THE LLM SUBSTRATE

Before getting into how agents work, it is worth being precise about what they are built on because the strengths and limitations of the underlying language model shape everything an agent can and cannot do.

The transformer architecture, which Vaswani and colleagues introduced in 2017, solved a problem that had plagued sequential neural networks: parallelism. Earlier recurrent architectures processed tokens one at a time, which made training slow and made capturing

long-range dependencies difficult. The self-attention mechanism at the transformer's core allows the model to relate any token to any other token in the input simultaneously making both training and inference dramatically more efficient.

What followed was a decade-defining scaling experiment. Research teams at OpenAI, Google, Meta, Anthropic, and elsewhere trained progressively larger models on progressively larger datasets and watched, with a mixture of excitement and unease, as capabilities they had not explicitly trained for began to emerge. A model large enough would suddenly be able to perform few-shot arithmetic. A larger one would spontaneously generate working code. The phenomenon became known as emergence, and while the underlying mathematics are still debated, the practical implication was clear: scale unlocks capabilities in non-linear, sometimes unpredictable ways.

But even the most capable base LLM has a fundamental architectural limitation. It is stateless. Every inference starts fresh. The model has no awareness of what it did five minutes ago unless that history is explicitly included in the current context window. It cannot reach out to a database, check a website, or run a program unless some external system mediates that interaction. And its context window the maximum amount of text it can process at once is finite, however generously engineers have expanded it in recent years.

Agentic systems are, at their core, a set of answers to these limitations. They give the model memory, tools, and structure. They wrap the LLM's raw generation capability in scaffolding that transforms a reactive text predictor into something that can actually get things done.

III. TOOL USE: GIVING THE MODEL HANDS

The single most important step toward LLM agency was giving models the ability to invoke external tools not as a prompt trick, but as a first-class architectural feature. Without tools, an LLM is confined to the world of text. With them, it can reach out and interact with software, data, and services that exist beyond its training data.

3.1. How Function Calling Actually Works

Modern tool use is built on a mechanism called function calling (or, in some frameworks, tool use). The developer defines a set of functions the model is allowed to invoke each described by a name, a plain-English description of what it does, and a schema of the arguments it accepts. These definitions are included in the model's system context at inference time.

When the model determines, mid-generation, that it needs to invoke a tool, it stops generating natural language and instead outputs a structured object typically JSON specifying which function to call and with what arguments. This structured output is intercepted by the application layer, which executes the actual function, captures the result, and feeds that result back to the model as a new context entry. The model then resumes its reasoning with the tool's output incorporated.

This loop think, act, observe, think again is the heartbeat of any LLM agent. It is conceptually simple. Getting it to work reliably across diverse tasks and tools is considerably less simple, and much of the engineering in modern agentic systems is concerned with exactly that challenge.

OpenAI's June 2023 release of function calling in the GPT-4 API was a watershed moment because it standardized this pattern at the API level. Before that, developers were hacking together tool use through clever prompt engineering, which worked inconsistently and was hard to maintain. Once function calling became a native API feature, it unlocked a wave of serious agentic development. Anthropic's tool-use API for Claude and Google's equivalent for Gemini followed in short order, effectively establishing function calling as the industry standard.

3.2. Code Execution: The Tool That Makes All Other Tools

Among the tools an LLM can be given, a code execution environment occupies a special position. A model that can write code and observe its output can, in principle, construct any other computational capability on the fly. It does not need a pre-defined tool for every task it can just write the tool it needs.

OpenAI's Code Interpreter, later rebranded as Advanced Data Analysis, made this concrete. Hand the feature a messy CSV and ask it to identify trends it will write Python, run it, look at the output, realize something is off, revise the code, and run it again until it has an answer worth sharing. The entire scientific

computing workflow hypothesis, code, execution, observation, revision compressed into a single conversation turn.

This capability has turned out to be one of the most practically valuable features of modern LLMs, particularly for knowledge workers who deal with data regularly but are not professional programmers. The model handles the programming; the human handles the judgment about what to ask and whether the answer makes sense.

IV. MEMORY: HELPING THE MODEL REMEMBER

A major constraint of working with LLMs in agentic contexts is the context window the finite span of text a model can consider at any one time. Early models capped out at a few thousand tokens; contemporary models push this to hundreds of thousands. But even a million-token context window has limits, and there are tasks managing a large codebase, tracking a project over weeks, drawing on an organization's entire document archive where no context window is ever going to be enough.

The field has developed several approaches to extending what an agent can effectively remember. They map onto four rough categories.

4.1. Four Kinds of Agent Memory

In-context memory is the simplest: information that lives directly in the current prompt or conversation history. It is immediately accessible and requires no retrieval infrastructure, but it disappears the moment the context window is full. For short interactions, it is usually sufficient. For long-running agents, it is not.

External memory refers to information stored outside the model in databases, vector stores, or filesystems and retrieved on demand. This is the architecture underlying most knowledge-

intensive agent applications, and it is discussed at length below. Its primary advantage is that it scales without limit; its challenge is building retrieval systems that surface the right information at the right moment.

In-weights memory is the knowledge baked into the model's parameters through training and fine- tuning. It is the most deeply integrated form of memory the model does not need to retrieve it because it is already part of how the model processes everything. But it is also the least flexible: updating it requires retraining, which is expensive and slow.

KV cache memory is a more technical concept but worth understanding. During inference, the computations associated with each token's key and value representations can be cached and reused. This allows repeated context prefixes a long system prompt, for example to be computed once and reused across many requests, reducing both latency and cost. For agentic applications that share substantial context across many interactions, cache efficiency matters.

4.2. Retrieval-Augmented Generation

Retrieval-Augmented Generation almost always referred to as RAG is the dominant approach to equipping LLM agents with external memory. The core idea, introduced by researchers at Facebook AI Research in 2020, is conceptually elegant: rather than trying to bake all potentially relevant knowledge into the model's weights, you store it in a searchable index and retrieve relevant chunks at inference time.

A standard RAG pipeline works roughly as follows. First, a document corpus is chunked into manageable pieces and each chunk is converted into a dense vector representation using an embedding model. These vectors are indexed in a vector store Pinecone, Weaviate, Chroma, or pgvector are common choices which enables fast approximate nearest-neighbor search. When a user query arrives, it is embedded in the same vector space, and the most semantically similar chunks are retrieved. Those chunks are then inserted into the model's context alongside the query.

The practical advantages of this setup are substantial. The knowledge base can be updated without touching the model. Retrieved chunks can be cited, making the system's outputs more transparent and auditable. Domain-specific knowledge that was never in the model's training data can be incorporated without fine-tuning. For enterprise applications customer support systems, internal knowledge bases, document analysis tools RAG has become close to a standard architectural choice.

More sophisticated RAG variants have emerged as the field has matured. Agentic RAG treats retrieval as just another tool in the agent's kit, letting the model decide dynamically when retrieval would help and how many retrievals rounds to perform. Corrective RAG adds a validation step: if the retrieved documents are not actually relevant to the query, the system escalates to web search rather than generating an answer from poor inputs. These refinements address the practical failure

modes irrelevant retrieval, missing context, outdated information that plagued first- generation RAG deployments.

V. PLANNING AND REASONING: TEACHING THE MODEL TO THINK AHEAD

Having tools and memory is not enough to make an agent capable of handling complex, multi-step tasks. An agent also needs to be able to plan to break a goal down into sub-tasks, sequence those sub-tasks intelligently, and adapt the plan when something goes wrong. This section examines the reasoning architectures that researchers have developed to give LLMs some version of this capability.

5.1. Chain-of-Thought: Thinking Out Loud

The insight that launched a thousand planning architectures was almost embarrassingly simple: if you ask an LLM to show its work to reason step by step before giving an answer it performs dramatically better on tasks that require multi-step reasoning. This is chain-of-thought prompting, formalized by Wei et al. in 2022, and its implications ran deeper than anyone initially expected.

The finding suggested that the generation process itself could serve as a working memory for computation. Each intermediate reasoning step, once generated, becomes part of the context for the next step. The model is, in a sense, thinking on paper externalizing cognition that would otherwise have to happen implicitly within a single forward pass. This is why chain-of-thought works: it breaks a hard problem into a sequence of easier ones, each grounded in what the model has already established.

Tree of Thoughts extends this idea into a tree-structured search. Instead of following a single reasoning path, the model explores multiple branches in parallel, evaluates each for promise, and prunes the less productive ones. This is computationally expensive but has shown strong performance on tasks requiring search over a large solution space puzzle, creative writing with structural constraints, mathematical reasoning. The key innovation is that the model can backtrack, something linear chain-of-thought cannot do.

ReAct, short for Reason and Act, interleaves reasoning traces with tool invocations. The model does not plan in full and then execute; it alternates between thinking about what to do next and actually doing it. This makes React more adaptive than purely planning-then-acting approaches if a tool call returns unexpected results, the model can immediately adjust its reasoning rather than being locked into a predetermined plan.

5.2. Reflection and Error Recovery

One of the more stubborn problems in agentic AI is error propagation. A wrong assumption early in a multi-step task does not stay contained it shapes every subsequent step, often in ways that are hard to detect until the final output is clearly wrong. By then, correcting course requires unwinding multiple steps of faulty reasoning.

Reflexion, a technique introduced by Shinn and colleagues in 2023, addresses this by giving agents an explicit self-critique step. After completing a task or failing to do so, the agent is prompted to reflect verbally on what went wrong and articulate what it would do differently. These reflections are stored in memory and incorporated into future attempts. The results showed meaningful improvement on coding tasks, decision-making scenarios, and reasoning benchmarks not because the underlying model changed, but because it was given a structured opportunity to learn from its own mistakes within a single deployment session.

Self-RAG, which Asai et al. introduced in 2023, applies similar logic to the retrieval process. The model is trained to generate special tokens that signal in the middle of generation whether retrieval is needed, whether retrieved documents are actually relevant, and whether its own outputs are grounded in the evidence it was given. This creates a self-correcting retrieval loop that produces more reliable, more citation-aware generations than simpler RAG architectures.

5.3. Long-Horizon Planning: The Hard Problem

Completing a task that requires fifty coherent sequential actions is qualitatively different from completing one that requires five. State must be maintained across all steps. Earlier decisions constrain later options. Errors accumulate. Context windows fill up. The model's attention becomes diluted across an increasingly long history.

Early fully-autonomous systems like AutoGPT struggled badly with this. Tasks that looked impressive in early demos would often unravel after a dozen steps the model would loop, lose track of what it had already done, or pursue a sub-goal so

relentlessly that it forgot the original goal entirely.

Hierarchical planning offers a principled way to manage this complexity. Rather than trying to plan every action in a single flat sequence, the system decomposes the goal into high-level sub- goals, plans at the sub-goal level, and only expands into detailed action sequences when each sub- goal is being actively pursued. This limits the planning complexity at each level and provides natural checkpoints moments where the system can assess progress, detect failures, and course- correct without needing to re-examine the entire task history.

Systems like HuggingGPT demonstrated what hierarchical planning could look like in practice: an LLM serves as the high-level planner and orchestrator, delegating specific sub-tasks image generation, speech synthesis, object detection to specialized models. Each specialist handles what it is good at; the LLM coordinates the whole. This division of cognitive labor has become a foundational pattern in more recent agentic frameworks.

VI. MULTI-AGENT SYSTEMS: WHEN ONE MODEL IS NOT ENOUGH

The single-agent paradigm one LLM, one context window, one thread of execution has a natural ceiling. Some tasks are simply too large or too complex for any single agent to handle well. Multi-agent systems address this by distributing the work across multiple specialized AI instances that communicate, check each other's outputs, and collaborate toward a shared objective.

6.1. The Case for Multiple Agents

The argument for multi-agent systems is straightforward when you think about how complex work actually gets done. A legal brief is not drafted by a single person who simultaneously researches the law, interviews clients, writes the argument, checks citations, and edits for clarity. It is divided among specialists who each bring focused attention to one aspect of the problem. Multi-agent AI systems follow the same logic.

Specialization matters because a model operating under a tightly scoped system prompt 'you are a code reviewer focused exclusively on security vulnerabilities' consistently outperforms a generalist model asked to do the same task alongside a dozen others. Parallelism matters because independent sub-tasks can be executed simultaneously, collapsing wall-clock completion time. Adversarial checking matters because one agent catching another's errors is more reliable than self-review, for roughly the same reason that code review catches bugs that the original author misses.

6.2. Communication Architectures

How agents communicate is one of the more consequential design decisions in multi-agent systems. The simplest approach natural language message-passing is flexible and requires no special infrastructure, but it can be lossy. Nuance gets dropped. Instructions get misinterpreted. The ambiguity inherent in natural language becomes a reliability problem when agents need to coordinate precisely.

Structured communication protocols, where agents exchange typed schemas rather than free text, reduce this ambiguity at the cost of flexibility. Shared memory architectures, where agents read and write to a common state object, work well for tasks with clear, discrete state tracking task completion, accumulating findings but introduce concurrency challenges when multiple agents need to update state simultaneously.

Microsoft's AutoGen framework, released in 2023, made a significant contribution to this space by providing a flexible conversation infrastructure for multi-agent systems. AutoGen agents are defined by system prompts, and the framework supports several conversation topologies: two- agent dialogues, group chats with a moderator agent that routes messages, and hierarchical patterns where an orchestrating agent delegates to specialized sub-agents. The fact that AutoGen quickly accumulated tens of thousands of GitHub stars suggest the framework struck the right balance between flexibility and usability.

6.3. Safety Is Not an Afterthought

Multi-agent systems introduce safety challenges that do not exist in single-agent contexts. When one AI agent instructs another, the receiving agent has no way to verify that the instruction is legitimate, authorized, or sensible. A compromised orchestrator one that has been manipulated by malicious content in its environment could direct subordinate agents to take harmful actions. If the subordinate agents simply comply because an AI system told them to, the damage can cascade quickly.

Anthropic's guidance on multi-agent safety makes this

explicit: a Claude agent acting as a sub- agent should evaluate instructions from an orchestrating agent by the same standards it would apply to instructions from a human user. Claimed authority does not change what is or is not safe to do. An orchestrator that instructs a sub-agent to bypass safety guidelines or access unauthorized systems should be refused, regardless of how the instruction is framed.

Prompt injection where adversarial content embedded in a document, webpage, or tool output attempts to hijack an agent's behavior is perhaps the most practically concerning attack vector in deployed agentic systems. Defense strategies include executing agents in sandboxed environments, requiring human confirmation before high-impact actions, and designing systems with minimal footprint: agents should request only the permissions they actually need, and no more.

VII. THE FRAMEWORKS IN PRACTICE

The theoretical architecture of LLM agents has been instantiated in a variety of real-world frameworks, each shaped by different design philosophies and target audiences. Understanding these frameworks concretely helps ground the abstractions of the previous sections.

7.1. LangChain: The Framework That Defined the Space

LangChain appeared in October 2022, almost simultaneously with the explosion of developer interest that followed ChatGPT's launch. Its timing was fortuitous but its design was genuinely

useful: it gave developers a modular, composable vocabulary for thinking about LLM applications. Models, prompts, chains, agents, memory, and tools were all first-class concepts with clean interfaces. You could swap a different model in, add a new tool, or replace the memory backend without rewriting the whole application.

The framework grew fast almost too fast. Critics pointed out that LangChain's abstractions could obscure what was actually happening in a pipeline, making debugging difficult. Breaking changes in the rapidly evolving API frustrated developers who had built production systems on earlier versions. The codebase, trying to support every possible integration, became sprawling in ways that made navigating it intimidating for newcomers.

LangChain's responses to these criticisms were substantive. LangChain Expression Language (LCEL) introduced a more principled, more performant way to define chains. LangSmith addressed the observability gap, giving developers the tools to trace, evaluate, and debug complex LLM pipelines with the same rigor they would apply to traditional software. Whatever its rough edges, LangChain built the scaffolding on which a generation of LLM applications was constructed, and its influence on how the field thinks about agent architecture is undeniable.

7.2. AutoGPT: The Vision, Imperfectly Realized

AutoGPT's appearance in March 2023 was genuinely exciting. Here was a demonstration, clumsy and unreliable as it often was, of what a fully autonomous AI agent might actually look like. You gave it a goal 'research the competitive landscape for electric vehicles and write me a report' and watched it spin up its task queue, start searching the web, make notes, revise its plan, and keep going without you holding its hand.

In practice, AutoGPT was unreliable in ways that taught the field important lessons. Long tasks degraded badly. The model would loop on sub-tasks, lose track of what it had already established, or decide that some absurdly tangential action was relevant to the original goal. The gap between the vision the early demos suggested and the performance users actually experienced was substantial.

But dismissing AutoGPT as a failure misses its historical significance. It was the first system that many developers and researchers had actually seen try to behave like an autonomous agent. The failure modes it exhibited goal drift, error accumulation, context degradation became the research agenda for the next two years. You cannot solve a problem you have not seen, and AutoGPT made these problems visible in a way that academic papers alone could not.

7.3. Claude Code: Narrow Focus, Deep Capability

Claude Code, released by Anthropic in 2025, represents a deliberate counter-philosophy to the broad-scope autonomy that AutoGPT pursued. Rather than trying to build a general-purpose agent for arbitrary goals, Anthropic focused on a single domain software engineering and built something that is genuinely excellent at it.

Operating as a command-line tool, Claude Code

integrates directly with the developer's actual working environment. It reads files, writes code, executes shell commands, manages git history, and runs test suites all in the real codebase, not a sandboxed copy. When a developer asks it to fix a failing test, it does not simulate the fix in isolation; it actually finds the broken code, understands why it is broken, writes the correction, runs the tests, verifies that they pass, and commits the change. This is the complete feedback loop of software development, automated.

The design choices reflect Anthropic's emphasis on safety and user control. Claude Code asks permission before taking actions with significant consequences deleting files, making network requests, modifying configuration. It logs every file change it makes. It can be restricted to operate only within specified directories. These constraints are not limitations so much as deliberate trust- building: a tool that acts transparently and within defined boundaries is a tool developers are willing to give more autonomy over time.

When Claude Code was extended to support the Model Context Protocol in 2025, it became both an MCP server exposing its capabilities to other agents and an MCP client able to invoke tools from other MCP servers. This positioned it within a broader ecosystem of interoperable agent capabilities rather than as a standalone product.

7.4. The Major Labs Enter the Space

By 2025, every significant AI organization had released its own agentic framework, making it clear that agent infrastructure had become strategic rather than experimental.

OpenAI's Agents SDK, launched in early 2025, introduced two concepts that influenced the broader ecosystem. Handoffs formalize the transfer of control between specialized agents rather than an orchestrator trying to be everything, it can explicitly hand a task to a better-suited specialist and resume control once that task is complete. Guardrails provide input and output validation layers that enforce safety policies at the framework level rather than relying solely on model alignment. The SDK's design aesthetic prioritized production readiness: sensible defaults, clear error messages, and strong observability built in from the start.

Google's Agent Development Kit, released in April 2025, leaned into the company's existing strengths. Integration with Google Search, Maps, Gmail, Calendar, and the Vertex AI platform gave ADK-built agents access to a uniquely rich tool ecosystem. Its multi-agent orchestration layer supported both hierarchical and peer-to-peer agent topologies with built-in state management reflecting Google's experience with distributed systems at scale.

Microsoft's Semantic Kernel predated the current wave of agentic frameworks but evolved alongside it, eventually supporting sophisticated multi-agent workflows through its Processes and Plans abstractions. Its enterprise orientation deep Azure integration, strong identity and access management hooks, compliance tooling made it the natural choice for organizations deploying AI agents within Microsoft's infrastructure.

VIII. MEASURING PROGRESS: BENCHMARKS AND WHAT THEY ACTUALLY TELL US

Evaluating an agent's capability is genuinely harder than evaluating a base language model. A standard NLP benchmark asks the model to answer questions, complete sentences, or classify text tasks with clear right answers that can be scored automatically. Agent benchmarks need to measure something more complex: whether the model can take the right sequence of actions, across multiple steps, in a dynamic environment, to accomplish a goal that may have no single correct solution.

Three benchmarks have become particularly influential. WebArena, introduced by Zhou and colleagues in 2023, evaluates agents on realistic web navigation tasks tasks that a competent human assistant could complete, like booking a flight, submitting a form on a government website, or navigating an e-commerce interface to find a specific product. The environments are sandboxed but behave like real websites, including the messiness and inconsistency of real web interfaces.

SWE-Bench, from Jimenez et al., takes a more domain-specific approach. It presents agents with real GitHub issues from popular open-source repositories and measures whether the agent can produce a patch that resolves the issue and passes the repository's existing test suite. This is a demanding benchmark because it requires understanding unfamiliar codebases, correctly diagnosing bugs, writing working fixes, and verifying that the fix does not break other

functionality. Early agent systems in 2024 resolved fewer than five percent of SWE-Bench issues. By early 2025, the best systems exceeded forty-nine percent a remarkably rapid improvement that reflects both model capability gains and better agentic scaffolding.
GAIA tests general-purpose agents on tasks that require combining web search, document analysis, code execution, and multi-step reasoning. The questions are designed to be tractable for a capable human with internet access but challenging for AI systems that struggle with multi-hop reasoning or tool coordination.
These benchmarks are useful, but they should be read with appropriate caution. A system that performs well on SWE-Bench may still fail on real-world software engineering tasks that differ in subtle ways from the benchmark distribution. Benchmark leakage where systems are inadvertently or deliberately optimized to perform well on specific benchmarks is a recurring concern. And for open-ended, long-horizon tasks, ground truth is often ambiguous: there are many ways to correctly build a web scraper, and a benchmark that evaluates only one of them will undercount capable agents.

IX. WHAT REMAINS UNSOLVED

Progress in LLM agency has been rapid enough that it is easy to lose sight of how much remains genuinely difficult. This section is an honest accounting of the open problems.

9.1. Reliability Under Pressure

Current agent systems work reasonably well under favorable conditions. They work much less well when conditions are unfavorable ambiguous instructions, unexpected tool failures, noisy retrieved content, or tasks that are genuinely at the edge of the model's capability. The brittleness is not random; it tends to manifest in predictable ways at predictable task complexities. But predicting exactly when an agent will fail, and building systems that fail gracefully rather than catastrophically, remains an open engineering challenge.

9.2. Staying on Task Over Many Steps

Long-horizon coherence maintaining consistent state, consistent goals, and consistent reasoning quality across fifty or a hundred sequential actions is hard. Context windows fill up and information gets pushed out. Errors accumulate. The model's attention distributes itself unevenly across a long history, with recent events receiving more weight than earlier ones even when earlier events are more relevant. Hierarchical memory systems and structured state representations help, but the problem is not solved.

9.3. Safety in High-Stakes Contexts

As LLM agents are deployed in contexts where their actions have real consequences executing transactions, modifying production systems, communicating with external parties the cost of mistakes scales accordingly. Building agents that are reliably safe in these contexts is not just a matter of model alignment; it requires thoughtful system design, robust human oversight mechanisms, and adversarial testing that tries to find the edge cases before they find themselves in production. Prompt injection, goal misspecification, and unauthorized action are all active research and engineering problems without fully satisfying solutions.

9.4. The Cost of Agency

Agentic workflows are expensive. A task that requires thirty tool calls, each triggering its own LLM inference, can cost two orders of magnitude more than a single-turn question and answer. For many applications, this cost is acceptable if an agent saves a knowledge worker four hours of work, it is worth whatever it costs to run. But for applications that require many agents running many tasks concurrently, inference cost is a serious constraint. Progress on model efficiency distillation, quantization, speculative decoding, better KV cache utilization matters enormously for the practical scalability of agentic systems.

9.5. The Interoperability Problem

A developer building an agent today must choose a framework, a model provider, a vector store, and a set of tool integrations and hope that these components work well together. The ecosystem is fragmented, and switching costs are high. Anthropic's Model Context Protocol is a genuine
attempt to address this by standardizing the interface between AI models and external tools. If MCP achieves broad adoption, it could enable a composable ecosystem in which agent capabilities are as interchangeable as software libraries. Whether that vision materializes, and on what timeline, is one of the more interesting open questions in the field.

X. CONCLUSION

There is something genuinely novel about what LLM agents represent. Not because AI systems have never been goal-directed before classical AI was full of goal-directed systems, from chess engines to constraint solvers. But those systems were goal-directed in narrow, brittle ways. They could pursue one kind of goal in one kind of environment, and they had no capacity to generalize.

What LLM agents bring to the table is flexible reasoning in natural language, applied to action in the real world. An agent built on a capable foundation model can pursue goals that were never explicitly specified in its design, in environments that were never explicitly anticipated, using tools that can be added at any time. The combination of that generality with real-world action capability is genuinely new.

The technical pillars that make this possible tool use, retrieval-augmented memory, chain-of- thought planning, multi-agent coordination are individually elegant but collectively transformative. Frameworks like LangChain, Claude Code, AutoGen, and the emerging offerings from major AI labs have moved these ideas from research papers into production systems used by real organizations to do real work.

The limitations are real and the challenges are significant. Reliability needs to improve. Long-horizon coherence needs to improve. Safety in high-stakes deployments needs frameworks that do not yet fully exist. The cost of agentic inference needs to come down. The ecosystem needs standardization it does not yet have.

But the direction of travel is clear. Language models learned to act. The question the field is now working on one careful answer at a time is how to make sure they act well.

REFERENCES

[1] Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," *arXiv preprint arXiv:2310.11511*, 2023.[2] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" *arXiv preprint arXiv:2310.06770*, 2023.

[2] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, *et al*., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[3] G. Mialon, C. Fourrier, C. Swift, T. Wolf, Y. LeCun, and T. Scialom, "GAIA: A Benchmark for General AI Assistants," *arXiv preprint arXiv:2311.12983*, 2023.

[4] OpenAI, "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, 2023.

[5] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, *et al*., "Toolformer: Language Models Can Teach Themselves to Use Tools," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[6] 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 (NeurIPS)*, vol. 36, 2023.

[7] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: Solving AI Tasks with ChatGPT and Its Friends in Hugging Face," *arXiv preprint arXiv:2303.17580*, 2023.

[8] Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, *et al*., "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5998–6008.

[9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, *et al*., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 24824–24837.

[10] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, *et al*., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," *arXiv preprint arXiv:2308.08155*, 2023.

[11] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," *arXiv preprint arXiv:2210.03629*, 2022.

[12] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of Thoughts: Deliberate Problem Solving with

Large Language Models," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[13] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, S. Sridhar, *et al*., "WebArena: A Realistic Web Environment for Building Autonomous Agents," *arXiv preprint arXiv:2307.13854*, 2023.