AI Agents for Economic Research: August 2025 Update to "Generative AI for Economic Research: Use Cases and Implications for Economists," Published in the *Journal of Economic Literature* 61(4)[†]

Anton Korinek*

The objective of this paper is to demystify AI agents—autonomous LLM-based systems that plan, use tools, and execute multi-step research tasks—and to provide hands-on instructions for economists to build their own, even if they do not have programming expertise. As AI has evolved from simple chatbots to reasoning models and now to autonomous agents, the main focus of this paper is to make these powerful tools accessible to all researchers. Through working examples and step-by-step code, it shows how economists can create agents that autonomously conduct literature reviews across myriads of sources, write and debug econometric code, fetch and analyze economic data, and coordinate complex research workflows. The paper demonstrates that by "vibe coding" (programming through natural language) and building on modern agentic frameworks like LangGraph, any economist can build sophisticated research assistants and other autonomous tools in minutes. By providing complete, working implementations alongside conceptual frameworks, this guide demonstrates how to employ AI agents in every stage of the research process, from initial investigation to final analysis.

*I would like to thank Kevin Bryan, Kody Carmody, Ricardo Correa, Dawis Kim, Peter McCrory, Patrick McKelvey, Dylan Ryfe, and Merlin Stein for helpful comments on this update, and David Romer for wisely asking me to commit to producing semi-annual updates of this rapidly evolving material when publishing the original paper in the JEL. I have learned so much from working on these updates. Disclaimer: I am serving as a member of the Anthropic Economic Advisory board but I am doing this in an unpaid capacity.

† Please cite this as Korinek, Anton. 2025. "AI Agents for Economic Research: August 2025 Update to 'Generative AI for Economic Research: Use Cases and Implications for Economists,' published in the *Journal of Economic Literature* 61(4)." Check the journal website for future updates and for the original version of the paper, which provides an introduction to generative AI for economists.

Reader's guide:

For readers of earlier versions of "Generative AI for Economic Research:" This paper is new in its entirety.

For new readers : This paper provides an overview of the state of AI Agents in Economic Research as of mid-2025. Readers may also want to consult two related resources: The December 2024 JEL update on "LLMs Learn to Collaborate and Reason" covers advances in LLM reasoning and collaboration as well as several dozen use cases. The original December 2023 JEL article on "Generative AI for Economic Research" offers an introduction to LLMs in economic research (Sections 1 and 2) and discusses longer-term implications for economists (Section 4).

1. Introduction

AI agents are software systems that combine large language models with planning capabilities, memory, and access to external tools to autonomously pursue complex goals through multi-step actions. Unlike traditional chatbots that respond to individual prompts in isolation, AI agents can decompose research questions, gather information from multiple sources, execute code, and iteratively refine their approach based on intermediate findings. They function as semi-autonomous research assistants capable of conducting end-to-end analyses: downloading datasets, cleaning and processing data, running econometric analyses, creating visualizations, interpreting, and synthesizing results into coherent narratives.

Since the original "Generative AI for Economic Research" was published (Korinek, 2023), the landscape of AI capabilities has shifted dramatically. What were experimental features and proofs-of-concept just two years ago have matured into production-ready tools that economists can increasingly rely upon for their daily research workflows. The chatbots that could help draft emails and summarize papers have evolved into sophisticated systems that can autonomously conduct literature reviews, write and debug complex econometric code, and orchestrate multi-faceted research investigations. This transformation reflects not merely incremental improvements but fundamental architectural advances in how AI systems operate.

This paper documents and analyzes these advances, providing both a conceptual framework for the evolution of AI capabilities and practical guidance for leveraging these tools in economic research. I cover five major themes:

The Evolution from Chatbots to Agents The paper documents the progression of AI systems through three distinct paradigms. Traditional LLMs, which have been broadly available since ChatGPT's launch in November 2022, excel at language-based tasks through pattern recognition and generation. These systems operate like Kahneman's "System 1" thinking—fast, intuitive, but limited to single-pass responses without the ability to reflect or revise (Kahneman, 2011).

Reasoning models, introduced in September 2024, bring "System 2" capabilities—deliberate, step-by-step problem solving that can perform complex mathematical derivations and logical analysis. These models can now solve multi-step problems that consistently challenged earlier systems, from deriving optimal control conditions to implementing

sophisticated numerical methods. The ability to "think" through problems systematically has opened new possibilities for theoretical economic work. Korinek (2024) provides a guide to how these capabilities can benefit economic research.

The third paradigm, agentic chatbots, emerged in December 2024. It synthesizes fluent language generation, reasoning and autonomous action. Agentic systems can plan sequences of operations, actively gather information, and adapt their strategies based on what they discover. For economists, this means AI systems that don't just respond to questions but actively investigate them—a shift from reactive tools to proactive research partners.

Breakthrough Agents: Deep Research and Claude Code Perhaps no capability better illustrates the potential (and limitations) of AI agents for research than Deep Research systems. As described in Section 3, these systems employ multi-agent architectures to process and synthesize hundreds of sources in minutes, producing comprehensive research reports with accurate citations. When given a research question, an orchestrator agent decomposes it into subtasks, spawns specialized agents to investigate different aspects in parallel, and compiles their findings into a coherent narrative. While these systems compile existing knowledge rather than generating novel insights, they speed up an important part of the research process by automating the time-intensive work of information gathering and initial synthesis. Literature reviews that once took weeks can now be completed in under an hour. However, these systems compile existing knowledge as it is available on the internet and struggle at times to identify the most impactful papers on a given question, especially in less established areas of literature—a critical skill for researchers who work at the frontier.

Anthropic's Claude Code was released as a coding agent in February 2025 to create sophisticated software through natural language descriptions—a practice known as "vibe coding." Many of the examples in this paper were generated with this tool, in minutes rather than hours. However, Claude Code's usefulness extends beyond coding to other areas of cognitive work, including research as it offers an agent that can automate complex workflows simply from natural language descriptions. By some measures, Claude Code has emerged as the most powerful agent in recent months. I felt that using it gave me a glimpse of the future in which more and more cognitive work will be automated.

Democratizing Technical Implementation A striking development documented throughout this paper is how AI agents are making sophisticated technical work accessible to researchers without significant programming expertise. Vibe-coding offers a new paradigm for any research tasks that benefit from computer code. Section 3.3.3 demonstrates how to use vibe-coding to build complete econometric tools from simple English descriptions, handling everything from file uploads to regression analysis to visualization.

This democratization extends beyond simple scripts. Researchers can now build custom data pipelines, implement complex estimation procedures, and create interactive research tools without writing a single line of code. The examples in this paper show agents not just writing code but debugging it, adding features based on feedback, and producing well-documented, professional-quality software. For a discipline where computational methods are increasingly central but programming skills remain unevenly distributed, some suggest that this may lead to a leveling of the technical playing field. However, although all may benefit, a common alternative perspective is that those power users who

know best how to deploy sophisticated AI agents may benefit far more from agentic AI than regular users.

Building Custom Research Agents Moving beyond using pre-built agents, Section 4 demonstrates that creating specialized AI agents for research is surprisingly accessible. Through concrete examples—from a simple FRED data retrieval agent to a sophisticated Deep Research system—the paper shows how researchers can build agents tailored to their specific needs. The tools required are not mysterious: with frameworks like LangGraph and a few hundred lines of code (fully generated with natural language prompts), researchers can create systems that embody the same architectural principles as commercial offerings.

The paper's implementation examples reveal an important recursive truth: these agent systems are themselves largely "vibe coded," with AI assistants helping write the code for new AI agents. This capability spiral, where each generation of tools enables the creation of still more sophisticated successors, suggests an accelerating pace of development. The emergence of standardized protocols like MCP (Model Context Protocol) and A2A (Agent-to-Agent) further reduces barriers, enabling agents to connect to institutional databases and collaborate across platforms without complex integration work.

Economic and Strategic Considerations Our analysis also brings up interesting economic considerations. The computational cost of advanced capabilities has driven new pricing models, with premium subscriptions reaching $200 per month—a tenfold increase from standard subscriptions just a year ago. This raises concerns about unequal access to frontier AI capabilities. The concentration of advanced reasoning and agentic capabilities among well-resourced labs, evidenced by their leadership in benchmarks, suggests that cutting-edge AI tools may increasingly become a function of ability-to-pay.

Yet there are also countervailing forces. Open-source models from players like DeepSeek and Alibaba's Qwen offer near-frontier capabilities at dramatically lower costs. The example of Moonshot's Kimi-K2, offering GPT-4-class performance at 1/100th the price, illustrates how competition and innovation can broaden access. For researchers, these dynamics suggest maintaining flexibility across model providers rather than committing to a single ecosystem.

As these capabilities continue to evolve at a remarkable pace, this paper aims to provide economists with both the conceptual understanding and practical tools needed to effectively leverage AI agents in their research. The examples and implementations are designed to be immediately useful while also providing a foundation for understanding future developments. In a field where the half-life of specific tools may be measured in months, the architectural patterns and strategic insights that I document should prove more durable guides for navigating the AI-augmented research landscape.

Despite their remarkable capabilities, AI agents still face significant practical limitations that researchers must be aware of. Current systems still suffer from hallucinations, generating plausible but incorrect content, for example citations or statistics. Sometimes agentic systems detect and correct mistakes, but at other times, mistakes compound through multi-agent workflows and may give rise to computational cascades where errors, tool failures, or incomplete data can propagate undetected and enter the final output. Moreover, AI agents sometimes demonstrate remarkable brittleness to small variations in prompts that make evaluation and reproducibility challenging. Also, they are exposed

to the risk of prompt injection attacks, allowing malicious actors to manipulate agent behavior through hidden instructions. Combined with costs that can quickly add up in multi-agent workflows, these limitations pose significant challenges. Perhaps most critically for economists, current LLM-based AI agents still struggle with genuine economic reasoning when pushed to operate at the level of a researcher—they sometimes misapply theoretical frameworks and reproduce common misconceptions from their training data rather than maintaining rigorous economic logic and are not yet capable of independent investigation. These limitations underscore that current AI agents require careful and responsible human oversight.

A useful analogy for how to productively employ AI agents in research is to treat them like a professor would treat a team of research assistants: they require careful planning of what is to be done, oversight during execution, and detailed vetting of the final results. Importantly, although there are also similarities, the mistakes made by LLMs and AI agents differ from the mistakes typically made by humans—for example, LLMs will generate hallucinations while sounding supremely confident in their correctness—and productively integrating them into research workflows requires getting used to such subtleties.

The paper covers several practical examples and use cases for AI agents in the sections below, ranging from intuitive chatbot-based interactions for everyday use to pedagogical hands-on examples of how to code AI agents. The examples include:

- Drawing and analyzing Beveridge curves
- Implementing simple agent workflows
- Agent workflows for plotting economic data
- Deep Research literature reviews
- Vibe coding an econometric tool
- Building an economic data retrieval agent
- Building a multi-agent deep research system

The remainder of this paper provides both conceptual frameworks and practical tools for economists engaging with AI agents. Section 2 describes the evolution from traditional LLMs to reasoning models and agentic chatbots over the past year and documents the current offerings of leading AI labs. Section 3 focuses on AI agents, explaining their architecture, exploring Deep Research systems, and examining coding agents that are reshaping software development. Section 4 goes "under the hood," providing working code examples and showing how to build research agents using modern frameworks. The appendix provides additional technical details and complete code implementations for researchers who wish to build their own systems.

2. Changing Paradigms: From LLMs to Reasoners and Agentic Chatbots

This section covers the evolution of generative AI systems from traditional LLMs, to reasoning models, and ultimately to agentic chatbots. It describes the most recent progress and state of the art in each type of system. Traditional LLMs have been broadly available since the launch of ChatGPT in November 2022, reasoning models were introduced in September 2024, and agentic chatbots followed soon after, in December 2024.

Each paradigm offers specific capabilities that make it particularly suited for different research tasks. Traditional LLMs excel at language-based tasks through pattern recognition

and generation, reasoning models bring structured problem-solving capabilities that enable complex mathematical and logical analysis, while agentic chatbots combine these strengths with the ability to autonomously employ tools such as web search or data analysis to perform end-to-end research workflows. We cover each type of system in further detail in the following subsections. Moreover, Table 1 provides a practical comparison of how each paradigm performs across the seven key research categories laid out by Korinek (2023): traditional LLMs dominate in pure language and writing tasks, reasoning models excel where logical precision is paramount such as coding and math, and agentic chatbots are most valuable for tasks requiring interaction with external systems or real-time information such as background research and data analysis.

Table 1
Primary Usefulness of LLM Model Types for Research Tasks

Research Category	Traditional LLMs	Reasoning Models	Agentic Chatbots
Ideation & Feedback	Good for initial brainstorming.	**Best for structured feedback and identifying logical flaws.**	Actively scans literature for novelty and grounding.
Writing	**Excellent for drafting, summarizing, and rephrasing existing text.**	Ensures logical flow in complex arguments.	Incorporates real-time web information.
Background Research	*Shortcoming: No live web access.*	Synthesizes info from provided texts.	**Best for live web searches and up-to-date literature reviews.**
Coding	Generates basic code snippets.	**Excellent for writing and debugging complex algorithms.**	**Executes code, tests hypotheses, and interacts with data files.**
Data Analysis	*Shortcoming: Cannot execute code.*	Helps interpret data and suggest approaches.	**Best for end-to-end analysis: data cleaning, coding, visualization.**
Math	*Shortcoming: Unreliable for complex math.*	**Solves multi-step problems and formal proofs at PhD level.**	Can leverage external computational tools to ensure accuracy.
Promoting Research	Drafts initial promotional content.	Tailors summaries for specific audiences.	**Automates creating summaries and posts for multiple platforms.**

Note: The most useful model type in each research category is bolded.

In early 2025, it was common that users had to explicitly select which type of AI system to interact with. As of mid-2025, leading chatbots such as ChatGPT, Claude, or Gemini

have the funtionality to automatically select in which of the three paradigms they operate, but they offer the user the choice to make the system "think longer" to explicitly turn on a reasoning mode that delivers better coding and math results.

The following subsections examine each paradigm in further detail, summarizing their underlying architectures, capabilities, and limitations for economic research. This is followed by a summary of the current offerings of leading AI labs.

2.1. Traditional LLM-Based Chatbots

Traditional LLM-based chatbots have become wildly popular since the release of ChatGPT in November 2022, although they have improved vastly since. They still represent the foundational paradigm of modern generative AI, generating text through token-by-token prediction based on their training data and fine-tuning. They build on deep neural networks that are trained on vast amounts of text to represent the conditional probability distribution of words given preceding context. This process enables these systems to develop abstract representations of concepts and their relationships, forming what can be understood as implicit world models that allow them to generate coherent and contextually appropriate text.

The speed, low transaction costs, and broad accessibility of traditional LLMs have made them invaluable tools for automating numerous micro-tasks in the research workflow. As already documented in the original version of this article (Korinek, 2023), traditional LLMs have proven remarkably versatile for economic research, being useful for tasks requiring language understanding and generation, including synthesizing text, editing, translating, and explaining economic concepts.

Many modern LLMs are multimodal, implying that they can process not only text but also other modalities such as images, videos, and sound. This is extremely helpful both when using chatbots as assistants for small tasks (e.g., transcribing spoken text, or taking a picture of an equation and translating it to latex) and for systematic data collection for economic research (e.g., evaluating large quantities of images or videos).

Table 2
Top models of leading AI labs on LMSYS-Arena

AI Lab	Best model	Latest Update	LMSYS	URL
Google DeepMind	Gemini 2.5 Pro	2025-06-17	1457	gemini.google
OpenAI	GPT-5 high	2025-08-07	1455	chatgpt.com
Anthropic	Claude Opus 4.1	2025-08-05	1451	claude.ai
xAI	Grok-4	2025-07-09	1425	x.com
Alibaba	Qwen3	2025-05	1422	chat.qwen.ai
Moonshot AI	Kimi K2	2025-07-11	1421	kimi.com

Source: https://lmarena.ai/?leaderboard. See Chiang et al. (2024). Last accessed on Aug 20th, 2025.

Table 2 shows that the current landscape of traditional LLMs is highly competitive, with

Google's Gemini 2.5 Pro leading the LMSYS rankings at 1465, followed closely by models from major AI labs including OpenAI, Anthropic, and emerging players like Alibaba, xAI and DeepSeek. LMSYS, short for Large Model Systems Organization, maintains a widely used leaderboard that ranks LLMs by pitting randomly-selected models against each other and employing user ratings to compile an Elo-like score for each model.[1,2]The table is ranked by the score of each provider's leading models in the LMSYS leaderboard (column 4). The proximity in the models' scores listed in the table suggests that the differences between leading models are relatively minor – high-quality language models have almost become commodities, democratizing access. For example, using the Elo formula from footnote 1, the difference in scores between the first and last model listed in the table implies relative winning probabilities of 60%/40% when the two models are pitted against each other.

However, traditional LLMs operate fundamentally as pattern recognition systems along the lines of what Kahnemann described as "system-1" thinking, which leads to characteristic limitations: they process text in a unidirectional stream without the ability to ponder on and revise earlier output, they lack access to real-time information beyond their training cutoff, and they struggle with tasks requiring multi-step logical reasoning or mathematical derivations. The two newer paradigms discussed in the next two subsections have made major leaps to remedy these shortcomings.

2.2. Reasoning Models

Reasoning models are designed to overcome the analytic shortcomings of traditional LLMs through structured thinking processes. These models are still next-token predictors but are trained via reinforcement learning to generate tokens that imitate "system-2" thinking in humans—deliberate, step-by-step problem solving that can perform complex and systemic analysis while also identifying and correcting errors. The reinforcement learning teaches the models to follow techniques such as producing chains of thought and tree search to solve complex analytic problems step-ty-step. The reasoning steps produced by the models are often kept invisible to the user. Proprietary models typically show just a summary as their final output. For a detailed description, see Section 2.1 of Korinek (2024).

To unlock the highest level of reasoning capabilities in the leading chatbots, user typically need a paid subscription from one of the leading model providers. Gemini 2.5 Pro and ChatGPT-5 automatically default to "slow thinking" mode when complex analytic questions are entered. In Claude, users can activate the setting "Extended thinking" to achieve the highest possible reasoning performance.

The architecture of reasoning models allows them to excel at tasks that have consistently challenged traditional LLMs. They can solve multi-step mathematical problems, write and debug complex code, and construct formal proofs at levels approaching or exceeding human

[1]The Elo-system was designed by the physicist Arpad Elo to rank chess players by their relative skills. It is designed so that a score difference of D points between two players (or LLMs) corresponds to the higher-ranked one having a probability of $1/1+10^{D/400}$ of winning in a direct match-up.

[2]Like all ranking systems that condense the capabilities of candidates who differ across many dimensions into a single dimension, the LMSYS score offers only a partial and imperfect snapshot of LLM capabilities and has been subject to considerable controversy. I chose to use for my ranking of traditional LLM-based chatbots here because it has almost universal coverage, is updated in close to real-time, and aggregates many different types of use cases when evaluating models.

expert performance. For economic research, this capability is transformative: reasoning models can perform sophisticated mathematical derivations, such as log-linearizing dynamic equilibrium conditions or solving optimal control problems, tasks that were beyond the reach of earlier systems. The Fall 2024 update documents how o1-preview, the first publicly available reasoning model, successfully coded a complete solution to the Ramsey growth model, including the shooting method for finding the optimal consumption path—a task that had stumped traditional LLMs just months earlier (Korinek, 2024).

However, reasoning models come with trade-offs. Their deliberative process requires significantly more computational resources during inference, as the quality of their outputs scales with the number of reasoning tokens generated. This makes them slower and more expensive to operate than traditional LLMs, particularly for tasks that require extensive reasoning chains. And while they excel at well-defined problems with clear logical structures, they offer little advantage over traditional LLMs for tasks primarily requiring language fluency or creative generation. For economists, reasoning models are most valuable when accuracy and logical coherence are paramount, such as in theoretical derivations, coding, and algorithm development.

Table 3
Top Reasoning and Agentic Capabilities by Lab, GPQA & SWE-Bench-V Scores

Lab	Model	Last Updated	GPQA ♢ Score	SWE-Bench V
OpenAI	GPT-5	2025-08-07	89.4%	75%
xAI	Grok-4	2025-07-10	88.9%	72-75%*
Google DeepMind	Gemini 2.5 Pro	2025-06-17	84.0%	63.8%
Anthropic	Claude Opus 4.1	2025-08-05	83.3%	72.7%
DeepSeek	DeepSeek-R1	2025-03-24	71.5%	49.2%

Source: Compiled by author. Last update: Aug 20th, 2025; * marks preliminary data.

Table 3 depicts the performance of reasoning models on the graduate-level Google-Proof Q&A (GPQA) benchmark (Rein et al., 2023) – a leading benchmark that measures expertise and reasoning in a range of scientific domains. PhDs in the relevant subjects typically score about 65%; the best LLM when the benchmark was originally released in November 2023 reached only 39%. Yet at the time of writing, the GPQA benchmark has almost become saturated (i.e., fully solved, accounting for estimates that about 10% of the questions contain errors): OpenAI's GPT-5 and xAI's Grok 4 lead with scores of 89.4% and 88.9% respectively; Google DeepMind's Gemini 2.5 Pro and Anthropic's Claude 4 follow closely at 84% and 83.3%, while a significant gap separates these top performers from the rest of the field. This distribution suggests that achieving human-expert-level reasoning remains computationally intensive and technically challenging, with only the most resource-rich labs able to compete at the frontier.

By June 2025—nine months after the first public release of a reasoning model—a

gathering of the world's leading mathematicians found that the most advanced reasoning models were capable of solving some of the hardest solvable math problems in existence (Chiou and Moskowitz, 2025). In July 2025, both an experimental reasoning model by OpenAI and DeepMind's Gemini Deep Think achieved gold-medal level performance at the International Mathematical Olympiad (Castelvecchi, 2025). These systems are now likely to exceed the math capabilities of all but a handful of scientists in the world.

2.3. Agentic Chatbots

Agentic chatbots represent the newest paradigm in generative AI, characterized by the ability to autonomously take actions and call on external tools to accomplish the goals specified by the user. Unlike traditional LLMs that respond to individual prompts in isolation, agentic systems can plan sequences of actions, actively gather information from multiple sources, and iteratively refine their approach based on intermediate results. This new paradigm is made possible by the combination of several technological advances: larger context windows that enable memory across interactions, improved reasoning capabilities that support multi-step planning, and the integration of tools that allow LLMs to interact with external systems to query the internet or execute code. Still, it is useful to remember the underlying LLMs fundamentally operate as next-token predictors. They have been taught via reinforcement learning to generate tokens that call up external tools from a list of available tools when this looks promising. We will cover one of the culminations of these advances – autonomous research agents – in the following two sections of the paper. For now we keep the focus on agentic chatbots.

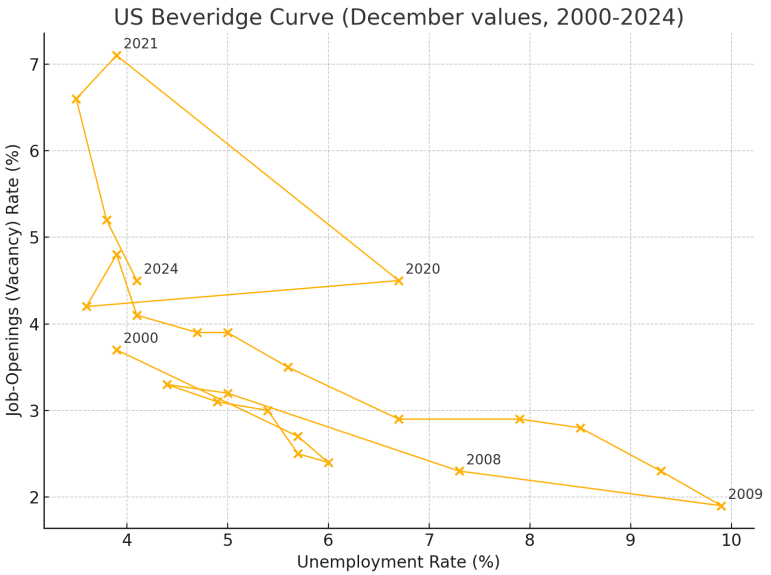
Agentic chatbots can act as semi-autonomous research assistants rather than merely responsive tools. They can perform web searches, interact with databases, execute code, manipulate files, and even control computer interfaces to accomplish complex tasks. For economic researchers, this means an AI system that can conduct end-to-end analyses: downloading datasets, cleaning and processing data, running econometric analyses, creating visualizations, and synthesizing results into coherent narratives.

An example is the following:

Chat 1 (ChatGPT o3): *Can you draw and analyze the Beveridge curve for the US using data for the past 25 years?*

Thought for 7m 5s >

Analyzed >



US Beveridge Curve (December values, 2000–2024)

What the chart shows

Phase	Typical pattern on the curve	Economic story
2000–07 expansion	Slides down the *same* downward–sloping locus: unemployment fell from ≈ 4% to ≈ 41/2%, while the vacancy rate hovered around 3–3 1/4%.	Labour–market matching was reasonably efficient; firms filled openings quickly when jobseekers were plentiful.
2008–09 Great Recession	The curve *pivoted outwards*: the economy jumped from ≈ 3% vacancies/5% unemployment in 2007–12 to the lower–right corner (1.9% vacancies/9.9% unemployment) in 2009–12.	A negative aggregate–demand shock produced both fewer openings and far more jobseekers.
2010–19 recovery	A slow crawl *back up* the pre–crisis curve: vacancies regained pre–crisis levels by 2018 while unemployment fell below 4%.	Matching efficiency and worker reallocation gradually normalised.
2020 pandemic shock	Huge clockwise jump: unemployment spiked to 6.7% (Dec–2020) yet vacancies stayed *high* at 4 1/2%.	Lockdowns and fear effects severed matches, but fiscal/monetary support let firms keep posting jobs.
2021–22 boom	Vacancy rate hit a record 7.1% with unemployment already back below 4% (Dec–2021) – a dramatic *outward shift* of the curve.	Explosive demand, labour–supply constraints (health, caregiving, early retirements, immigration shortfalls) and sectoral mismatches made hiring unusually difficult.

2023–24 cooling	The economy slides *down* the new, higher curve: vacancies retreat to 4 1/2% (Dec–2024) while unemployment edges up only to about 4%.	Fed tightening and fading pandemic distortions eased labour–market overheating but matching efficiency remains weaker than pre–2020.

Key take–aways

1) **Downward slope is intact** – periods with more job openings still coincide with fewer unemployed workers, reflecting basic search–and–matching dynamics.

2) **The whole curve has shifted outward since 2020**, meaning that for any given unemployment rate, today's vacancy rate is about 1 1/2–2 percentage points higher than in the 2000–19 period.

   Possible explanations: lower worker–job matching efficiency (skill or geographic mismatch), lingering health–related absences, increased job–shopping in a tight market, and a higher "reservation" wage after pandemic support.

3) **Clockwise loops mark recessions** (2001, 2008–09, 2020): vacancies collapse first, then unemployment rises. Counter–clockwise moves mark expansions.

4) **Policy implication** – Bringing labour demand back into balance will probably require either: (a) further cooling of aggregate demand (reducing postings) *or* (b) structural measures that raise matching efficiency (skills training, childcare to lift participation, immigration). Otherwise, the outward–shifted Beveridge curve implies continued wage–price pressure even with unemployment near 4%.

5) **Watch 2025 data**: early releases show the vacancy rate easing toward 4 1/4% while unemployment flattens around 4.1%, tentatively continuing the slow move down the pandemic–shifted curve.

The model's analysis is insightful and adheres closely to the popular consensus on the topic. Depending on the use case, the latter can be a feature or a limitation. For researchers who are aware of it, obtaining in-depth analysis from agentic chatbots at high speed is highly valuable. However, there is also a risk that non-experts are not sufficiently exposed to alternative viewpoints. Another notable point about the given example is that the chatbot does not explictly report its data source, which makes it more difficult to audit the results. It is advisable to ask agentic chatbots for their sources in data analysis tasks and verify it. I have experienced on more than one occasion that chatbots generated pseudo data that looked plausible but was made up when such systems cannot gain access to the data they would like to fetch.

An important development that has amplified the power of agentic chatbots is the introduction of "connections"—integrations that allow AI systems to directly access and interact with external applications and services. These connections may enable chatbots to read a user's emails, navigate the file system on their harddisk, query databases, access calendars, and interact with other productivity tools. This transforms the AI from a system limited to its training data into one that can draw upon a user's entire digital ecosystem. For economic researchers, this means an AI assistant that can instantly locate past paper drafts, cross-reference email discussions with co-authors, pull data from institutional databases, and check upcoming deadlines—all within a single interaction. The combination of reasoning capabilities, tool use, and persistent connections to a researcher's

working environment creates a system that understands not just general knowledge but the specific context of an individual's research workflow, making AI an integrated research partner.

Over time, the potential of agentic chatbots will extend beyond automating individual tasks to reshaping research workflows more significantly. These systems can maintain research context across extended projects, proactively identify relevant literature, test hypotheses through computational experiments, and even generate comprehensive research outputs. However, this autonomy also introduces new challenges. Researchers must carefully oversee agentic systems to ensure they pursue the right objectives and maintain research integrity. Privacy and security concerns become paramount when systems have broad access to computational resources and data. Additionally, the current generation of agentic chatbots still requires significant human guidance to navigate complex research decisions and ensure that their outputs meet academic standards. Yet as these systems continue to evolve, they promise to transform from tools that assist with research tasks to collaborators that can independently pursue research objectives under human direction.

A model's performance on the SWE-Bench V coding benchmark, displayed in the last column of Table 3, reflects its ability to function as a software engineering agent that is called to fix software bugs in verified real-world use cases. The top tier of models – OpenAI's GPT-5, xAI's Grok 4, and Anthropic's Claude Opus 4.1 – achieve scores above 70%, demonstrating the ability to independently resolve complex software issues that require understanding codebases, identifying bugs, and implementing solutions. These scores correlate with models' broader agentic capabilities as well as with reasoning capabilities, as the skills required for SWE-Bench – planning, tool use, iterative problem-solving, and code execution – mirror those needed for autonomous research assistance. The rapid improvement in these scores over the past year suggests that fully autonomous AI research assistants may soon be built.

2.4. Leading AI Labs and Their Model Portfolios

Whereas the three paradigms above provide a functional view of generative AI capabilities, the following provides an overview of how these capabilities are packaged in the chatbots of leading AI labs. Each of the leading labs offers portfolios of models of different sizes that reflect different trade-offs between model performance, speed, and cost. Larger models are more "intelligent" and generally offer better performance and greater capabilities, but they also require more computational resources and take longer to process requests, making them more expensive. Smaller models, on the other hand, are faster and more cost-effective, but may not provide the same level of quality in their outputs.

Proprietary Model Providers

The growing computational cost of reasoning models and agentic chatbots has also given rise to new pricing models in recent months, as illustrated in the table below. Each of the top AI labs is now offering premium subscriptions that cost $200/month or more – a tenfold increase from the standard subscription cost of $20 that gave users access to the best AI models a year ago. As the table illustrates, a power user who buys the most expensive subscription for each of the leading labs would spend close to $1000/month.

This raises concerns that access to top intelligence for researchers (and everyone else) may increasingly become a function of their ability-to-pay. Sam Altman has already floated the possibility that he may soon sell access to a Ph.D.-level scientist system at a cost of $20,000 per year (Palazzolo and Weinberg, 2025).

Table 5
Pricing tiers for chatbot subscriptions of leading AI labs
(USD per month, July 2025)

Lab / Provider	Basic tier	$/mo	Most expensive tier	$/mo
OpenAI ChatGPT	Plus	$20	Pro	$200
Google DeepMind Gemini	AI Pro	$20	AI Ultra	$250
Anthropic Claude	Pro	$20	Max 20× usage	$200
xAI Grok	X Premium	$8	SuperGrok Heavy	$300
Microsoft Copilot	Copilot Pro	$20		

Google DeepMind has emerged as a consistent leader across all three paradigms with its *Gemini 2.5 Pro* model, last updated in June 2025, achieving top-tier performance in traditional language tasks (LMSYS score: 1457) and excellent reasoning and agentic capabilities. The model's most distinguishing feature is its 2-million token context window, enabling it to process multiple books or dozens of research papers simultaneously. Google also offers some limited access to its Pro model even to free subscribers of its chatbot. Aside from the Gemini chatbot, researchers can obtain access to Google's most advanced LLMs with a signfiicant free usage quote at https://aistudio.google.com. Moreover, Google also offers a smaller model, *Gemini 2.5 Flash*, that is very capable (LMSYS score: 1416), placing it above the top models of several other leading labs. According to an analysis by latent.space, Gemini currently offers the best price/performance ratio across different model sizes.

OpenAI released its GPT-5 model in August 2025, which offers perhaps the smoothest agentic chatbot experience among leading providers and top performance in reasoning and agentic coding. ChatGPT-5 analyzes uses queries and automatically decides what level of reasoning is necessary to generate a satisfactory response. (Sometimes it is useful to add "think hard" in one's prompt to ensure that the correct level of thinking is employed.) Even the free tier of ChatGPT offers access to the cutting-edge GPT-5. The most expensive "Pro" subscription tier gives users access to GPT-5 pro, which uses extra compute for the best answers to hard questions. ChatGPT's Agent mode, available since late July 2025, is an extra capability layer that gives GPT-5 access to a virtual computer, browser, and app connectors, allowing it to autonomously plan and execute multi-step tasks like travel booking while keeping the user in the loop.

Anthropic continues to be a leader for its coding capabilities and its nuanced writing style while maintaining industry-leading safety standards with its Claude Opus 4.1 model, updated in August 2025. The lighter-weight Claude Sonnet 4 offers almost the same

performance at lower compute cost. Both models feature an "Extended Thinking" mode that turns on in-depth reasoning capabilities. Claude can also be easily connected to external apps such as email applications.

xAI is the newest entrant, founded by Elon Musk in March 2023, but has achieved remarkable progress with its Grok-4 model, which was state-of-the-art when released in July 2025. The model ranks highly in reasoning capabilities (GPQA: 88.0%) and is among the top for agentic tasks (SWE-Bench: 72-75%), while maintaining competitive traditional language performance. The company's rapid ascent, leveraging its integration with the X platform and Elon Musk's "free speech absolutism" philosophy, represents a distinct approach to AI development that prioritizes accelerating advances in AI capabilities.

Microsoft occupies a unique position in the AI landscape through its partnership with OpenAI while also advancing their own internal AI research efforts. While Microsoft does not appear independently in the benchmark tables above, it offers access to OpenAI's leading models via its Microsoft Copilot subscription and has become instrumental in bringing frontier AI capabilities to enterprise and research users by integrating Copilot into Microsoft Office. Through these offerings, Microsoft has arguably achieved the widest deployment of AI assistance in professional workflows.

Global and Open-Source Players

Whereas the models above are proprietary, there is also a growing number of open-source players who make their models freely available to download, use, modify, and distribute.[3] This offers several benefits for economic research. Firstly, the transparency of open-source models allows researchers to examine the underlying architecture, enabling them to better understand the model's structure. Secondly, open-source projects allow anybody to innovate upon the model. This can help accelerate the development of LLMs tailored to specific needs. Thirdly, if researchers have access to low-cost computing resources, they can leverage open-source models for their work without incurring financial costs. Fourthly, open-source models that are operated locally offer significant privacy benefits as sensitive data does not need to be channeled over the internet to be processed on the servers of proprietary model providers. Finally, open-source models allow for greater reproducibility, which is helpful for ensuring scientific integrity in research as it enables other researchers to verify and build upon the reported results. These benefits make open-source language models an attractive choice for natural language processing. The main downside of open-source models is that their performance may lag behind the leading proprietary offerings and that operating them requires more expertise. Moreover, many of the models are too large to operate on typical desktop computers and require expensive server infrastructure.

Meta is one of the leading provider of open-source models in the US but has found itself lagging behind the Chinese open-source competition in recent months, with the LLaMA 4 series that was released in April 2025 delivering disappointing benchmark results. In June 2024, Mark Zuckerberg restructured Meta's AI organization and created Meta Superintelligence Labs (MSL), acqui-hiring Scale AI's founder Alexandr Wang for $14.3 billion and reportedly offering pay packages worth hundreds of millions of dollars

[3]More precisely, the models are "open weights," which means that the weights and software to run inference on the LLM can be freely downloaded but not the training source code and data.

to leading AI researchers at competing labs. Given Meta's unparalleled computational resources, vast social media training data, and aggressive hiring of top AI talent, they remain a formidable long-term player.

OpenAI also released two open-weight models gpt-oss-120B and gpt-oss-20B in August 2025 that deliver impressive performance for their size, especially in code generation. By offering both proprietary and open-weight models, OpenAI is making their products available to a wider range of users and addressing the growing competition to their business faced from open-source competitors.

DeepSeek represents China's growing presence in frontier AI development, with its R1 model achieving competitive traditional language scores (LMSYS: 1414) despite operating under semiconductor export restrictions. The company's focus on efficiency and novel architectures demonstrates that computational constraints can drive innovation, potentially offering a path for researchers with limited resources.

Alibaba's Qwen series (short for Tongyi Qianwen, which translates to "Unified Thousand Questions") offers a comprehensive portfolio of open-source models, spanning from mobile-deployable versions to a highly capable flagship 235B parameter model, with strong multilingual capabilities. This offers researchers many different choices along the size/capability frontier.

Kimi-K2, released by Alibaba-backed startup Moonshot in July 2025, is the most recent Chinese open-source model that challenges the leading AI labs. The model has impressive benchmark results but offers dramatically lower pricing—just 15 cents per million input tokens compared to, e.g., Claude's $15. This 100x price differential helps to make high performance far more widely available.

Mistral AI, founded by former Google DeepMind and Meta employees, has carved out a niche in the European AI ecosystem, emphasizing efficient, privacy-conscious models. Their Magistral Medium achieves respectable reasoning performance (GPQA: 70.8%) while maintaining lower computational requirements than competing models, appealing to European researchers prioritizing data sovereignty and computational efficiency.

Strategic Implications for Researchers

Let me summarize three observations on the current AI landscape. First, traditional LLM capabilities have become commoditized, with differentiation now occurring in reasoning and agentic capabilities. Second, the significant performance gaps in GPQA and SWE-Bench scores indicate that advanced capabilities remain concentrated among well-resourced labs. Third, the emergence of strong open-source alternatives, particularly from Chinese developers, is democratizing access to capabilities below but near the frontier.

For economic researchers, a paid subscription to the offerings of any of the leading labs in Table 2 is a reasonable choice. It provides access to top-tier reasoning models for complex analytical work and to agentic capabilities for workflow automation. However, for researchers with intermittent needs—such as occasional coding assistance through Cursor or Claude Code—using models via API on a pay-per-use basis may be more economical than maintaining multiple monthly subscriptions. The rapid pace of improvement across all labs counsels against over-commitment to any single provider.

Data Security Considerations

Researchers working with sensitive data—whether due to institutional requirements (such as central banks handling market-sensitive information) or the confidential nature of their research materials—must carefully evaluate the security implications of different AI deployment options. OpenAI and Anthropic, in particular, provide enterprise security (including SOC 2 Type 2) and strong data controls. For API and enterprise use, both default to no training on customer content, and OpenAI offers zero-data-retention options; by contrast, consumer chat usage can contribute to training unless users opt out in settings. These safeguards suggest that concerns about data leakage through major commercial providers may sometimes be overstated, particularly when proper API configurations are used rather than consumer chatbot interfaces.

For maximum security, however, deploying open-source models on internal infrastructure remains the gold standard. Models like OpenAI's gpt-oss or Meta's Llama 4 series deliver near-frontier capabilities while allowing complete control over data handling. These models can perform sophisticated tasks including agentic workflows, code generation, and complex reasoning entirely within institutional firewalls. The trade-off between the convenience and somewhat superior performance of cloud-based proprietary models and the security of locally-deployed open-source alternatives ultimately depends on the specific sensitivity of the research data and institutional compliance requirements. However, it is crucial that institutions base their compliance policies on evidence-based security assessments rather than outdated assumptions, ensuring that researchers can fully leverage AI capabilities without unnecessarily handicapping their productivity.

3. AI Agents for Economic Research

3.1. Basic Architecture

AI agents are software systems that combine LLMs with planning tools, function-calling capabilities, and memory systems to autonomously pursue complex goals through multi-step actions.

The architecture of simple AI agents is illustrated in Figure 1: an orchestrator passes the original objective (for example a user prompt) and the list of available external tools to a reasoning LLM. This LLM represents the digital equivalent of the system's brain: it strategizes how to pursue the objective and decides what external tools to call. These tools provide the system with an interface with the external world, giving it the digital equivalent of eyes to see its environment and hands to perform actions. Common examples of such tools are search engines, web browsers, code execution, database queries, or LLM subagents, all of which we will explore below. Each time the reasoning engine wants to call a tool, it generates tokens that indicate to the orchestrator to call upon the designated external tool and feed the result back to the reasoning engine before continuing the token generation process. Moreover, a memory system allows the agent to store context and build upon past results.[4]

[4]Observe the close analogy to more traditional definitions of the term "agents:" In computer science and robotics, agents have traditionally been described as entities that perceive their environment via sensors and act on it via actuators. In our example, these sensors and actuators are given by external tools that the reasoning engine can call upon. In economics, agents maximize an objective subject to information and resource

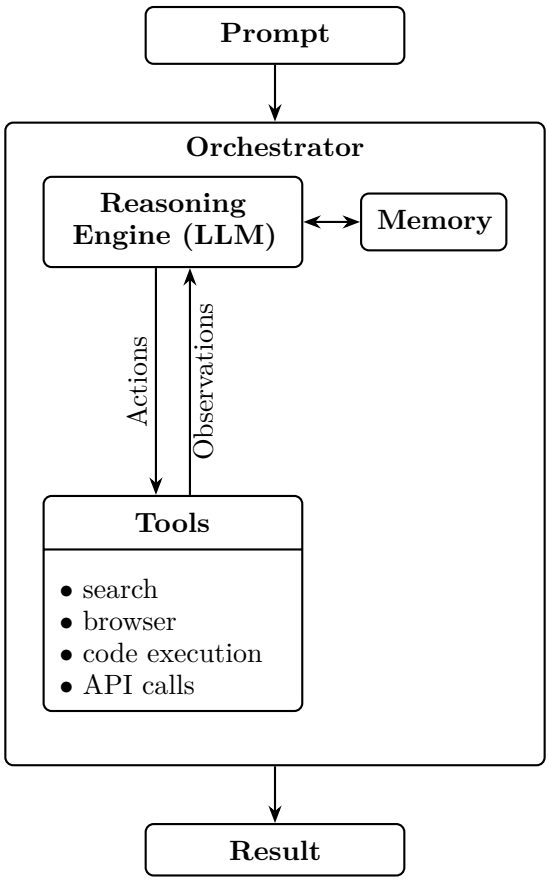


Figure 1. AI Agent Architecture

Whereas traditional chatbots engage with human users in the form of turn-by-turn conversations that require regular human interaction to provide the agent with data or instructions for the next step, AI agents automate some of this process: once they are started, they can be thought of as autonomous conversations between LLMs and different computer tools that pursue a specific goal without constant human interaction.

The following is a very simple schematic example, following the ReAct (Reasoning and Acting) framework that was introduced by Yao et al. (2023), in which the agent's process is clearly separated into thoughts, actions, and observations. The orchestrator passes the initial query to the LLM reasoning engine, which starts by generating the tokens displayed

constraints. In our example, AI agents seek to fulfill the objective given by the initial prompt. They operate within token and memory limits and subject to the tools they have access. Given bounded rationality, they use approximate reasoning to navigate these constraints. In economics, much of the focus of agency theory is on the potential misalignment between principal and agent; for AI agents, the question of alignment is likewise receiving a growing amount of attention (Hendrycks, 2025).

in the "Thought" and "Action" parts. The latter signals to the orchestrator to call upon the external function `weather:get_current(·)`, to which it passes the current location as an argument. The weather function fetches the requested data and returns "Observation," which the orchestrator feeds back into the LLM, which understands that it has all the available information required to respond and generates the final "Result."

Chat 2 (Schematic Agent Workflow): *What's the weather in Washington DC?*

Thought: I need to check the current weather conditions in Washington, DC.
Action: `weather:get_current({"location": "Washington, DC"})`
`# [Orchestrator calls weather API and returns current conditions]`
Observation: Temperature: 88 F, Conditions: Partly cloudy
Result: The current weather in Washington, DC is 88 F with partly cloudy skies.

The next example illustrates the use of data gathering and code execution. The reasoning engine is asked to produce a plot and has access to a data-gathering tool from FRED (Federal Reserve Economic Data, St. Louis Fed) as well as to a python code interpreter that can execute computer code. When the reasoning engine has produced the respective "Action" commands, the orchestrator interrupts the token generation to call the indicated external tools, passes the results back to the reasoning engine, and lets it decide how to continue in order to satisfy the initial request. Note that the computer code that is passed on to the second "Action" was written by the reasoning engine on the fly — modern LLMs are excellent at writing computer code and, if the orchestrator enables them to execute code, they can code and perform a wide range of computer-based actions that they want to pursue.

Chat 3 (Schematic Agent Workflow): *Plot the US unemployment rate and inflation over the past 20 years.*

Thought: To answer this, I will retrieve the unemployment rate (series ID: UNRATE) and consumer price inflation (series ID: FPCPITOTLZGUSA), both from FRED.
Action: `fred_data:get_series(["UNRATE", "FPCPITOTLZGUSA"], start="2005-07-01", end="2025-07-01", frequency="monthly")`
`# [Orchestrator calls FRED API and returns data]`
Observation: Data retrieved and stored in economic_data.csv (240 rows, columns: date, unemployment, inflation)
Thought: Now I have the merged data with YoY inflation already calculated. I'll create a Phillips curve scatter plot.
Action: `python:execute({"code":`

```
    "'
    import pandas as pd; import matplotlib.pyplot as plt
    df = pd.read_csv('economic_data.csv')
    plt.figure(figsize=(10, 8))
    plt.scatter(df['unemployment'], df['inflation'], alpha=0.6)
    plt.xlabel('Unemployment Rate (%)'); plt.ylabel('Inflation Rate (%)')
    plt.title('Phillips Curve:  US Unemployment vs Inflation (2004-2024)')
    plt.savefig('phillips_curve.png')
    "'

})
# [Orchestrator executes python code to generate chart.]
```

Observation: Code executed successfully. Chart saved as phillips_curve.png
Thought: I've created the Phillips curve. Now I need to display this to the user.
Action: `display:image("file": "phillips_curve.png", "caption": "Phillips Curve showing the inverse relationship between US unemployment and inflation rates over the past 20 years")`
`# [Orchestrator renders the image in the user interface]`
Observation: Image displayed to user.
Result: The Phillips curve visualization shows the relationship between unemployment and inflation in the US economy from 2004-2024, revealing periods where the traditional inverse relationship held and periods of deviation.

As illustrated in this example, planning and tool-calling enables AI agents to handle far more complex, multi-step actions than their traditional chatbot predecessors with far greater autonomy while adapting to what they learn from the tools they use. They can decompose ambitious research goals into actionable steps, exhibiting the kind of forward-looking behavior that characterizes human problem-solving. Kwa et al. (2025) show that AI agents can now autonomously perform tasks that take humans about 50 minutes to perform and, more importantly, that the task length that AI is competent for has been doubling every seven months since 2019. At this pace of progress, AI agents would be able to autonomously perform day-long general research tasks by the end of next year.

3.2. Deep Research Agents

The most powerful applications of AI agents for research applications are currently Deep Research systems. These systems were pioneered by Google DeepMind in December 2024 and have quickly become part of the offerings of all the leading AI companies. After a user enter a research question, a Deep Research agent spends between several minutes to half an hour to search the internet on the topic, compile and evaluate the most useful materials, and prepare a comprehensive research report on the topic that is complete with well-sourced citations (OpenAI, 2025). Deep Research agents sometimes access and evaluate more than 500 internet sources for a single assignment.

To be sure, what these so-called Deep Research systems produce is what non-researchers call research, compiling existing materials rather than generating genuinely novel insights. Still, they can be enormously useful for researchers, especially in generating literature surveys or compiling background information on a topic on which the researcher may want to learn more. For example, I start many of my research projects by asking several deep research systems to provide background information on the topic. A typical example of an interaction looks as follows:

Chat 4 (Gemini Deep Research): *I am an academic economist working on a paper on "AI agents for economic research" in which I would like to cover how using and designing AI agents can benefit PhD-level researchers. Give you write a 10-page overview of the topic?*

Here's the plan I've put together. Let me know if you need any changes before I start researching.
AI Agents for Economic Research
Research Websites
(1) Establish a foundational understanding by defining 'AI agents' within the context of economic research. Differentiate them from broader AI/ML models and traditional economic agent concepts by reviewing seminal and recent academic literature on the topic.

(2) Investigate and detail how PhD researchers can *use* existing AI agents as research assistants. This includes:
(a) Their role in automating systematic literature reviews and knowledge synthesis.
(b) Their application in extracting and structuring data from unconventional sources like text, images, or satellite data.
(3) Explore the application of AI and machine learning techniques in advanced econometric analysis, focusing on benefits for researchers like automated model selection, causal inference with complex data, and forecasting.
(4) Analyze the benefits of *designing* AI agents, with a primary focus on Agent-Based Modeling (ABM). Research how ABMs are used across different economic fields (e.g., macroeconomics, finance, development economics) to simulate complex systems and study emergent phenomena.
(5) Investigate advanced methods for designing intelligent agents beyond simple rule-based ABMs. Explore the use of Reinforcement Learning (RL) to model adaptive agents and the application of generative agents for simulating realistic socio-economic interactions.
(6) Compare and contrast the agent-based approach with traditional economic modeling techniques (e.g., DSGE, CGE models). Synthesize the unique advantages AI agents offer, such as modeling heterogeneity, bounded rationality, and out-of-equilibrium dynamics.
(7) Critically evaluate the challenges, limitations, and ethical considerations associated with using AI agents in economic research. This includes issues of model validation, computational cost, explainability (the 'black box' problem), and potential algorithmic bias.
(8) Speculate on the future of AI agents in economics by researching emerging trends, influential working papers, and expert opinions on how these tools will shape the discipline and the skill set required for future PhD-level economists.
Analyze Results
Create Report
Ready in a few mins

[After pressing "Start Research," Gemini spent about 9 minutes consulting over a hundred sources and assembled a comprehensive report.]

AI Agents in Economic Research: A Methodological and Applied Overview

Executive Summary

This report provides a comprehensive overview of the transformative role of Artificial Intelligence (AI) agents in PhD-level economic research. It delineates the dual function of AI agents: first, as powerful research multipliers that augment and accelerate the traditional research workflow, and second, as the fundamental components of a new generation of sophisticated economic simulations. The report begins by establishing a clear definition of AI agents, distinguishing them from traditional economic agents and standard machine learning models. It then systematically explores their application in automating literature synthesis, unlocking novel datasets through advanced extraction techniques (NLP, computer vision), and enhancing the rigor of modern econometrics, particularly in causal inference. A significant portion is dedicated to the renaissance of Agent-Based Modeling (ABM), detailing how the integration of Reinforcement Learning (RL) and Large Language Models (LLMs) allows for the design of adaptive and generative agents that can simulate complex, out-of-equilibrium dynamics. Finally, the report addresses the critical methodological, ethical, and practical challenges—including the "black box" problem, model validation, and algorithmic bias—and concludes by outlining the future trajectory of the field and the evolving skill set required for the modern economist.
[remainder omitted for space reasons]

Requesting a Deep Research report from ChatGPT on the same topic leads to the following interaction:

Chat 5 (ChatGPT Deep Research): *I am an academic economist working on a paper on "AI agents for economic research" in which I would like to cover how using and designing AI agents can benefit PhD-level researchers. Give you write a 10-page overview of the topic?*

I'd be happy to help with that. Could you please clarify a few things so I can tailor the 10-page overview more effectively?
1. What is the primary objective of the paper (e.g., conceptual framework, literature review, methodological proposal, practical guide)?
2. Should the focus be more on existing tools and applications, theoretical frameworks, or forward-looking implications?
3. Do you want examples from specific economics subfields (e.g., macroeconomics, labor, IO, finance)?
4. Should the tone and structure resemble an academic paper, a white paper, or more of a policy briefing?
5. Do you want coverage of specific tools or platforms (e.g., ChatGPT, Elicit, AutoGPT, OpenAgent, etc.)?
Once I have these details, I'll begin the deep research and writing process for you.

---

1. Informing Ph.D. economists on the methodology and including a practical guide
2. all three
3. general
4. academic
5. broad coverage of the offerings of main labs plus some details on LangGraph for buildling agents

---

[After submitting the requested follow-up responses, ChatGPT spent 10 minutes performing 109 searches, whittling them down to 21 sources, and assembling a comprehensive report]

AI Agents for Economic Research

Abstract

Artificial intelligence (AI) agents powered by large language models (LLMs) are rapidly transforming how economists conduct research. This overview provides a methodological introduction and practical guide to AI agents in economic research. We first discuss the conceptual and theoretical frameworks underpinning AI agents – defining what constitutes an "AI agent," exploring how multi-agent systems and cognitive automation can augment research workflows, and addressing key considerations like replicability, domain adaptation, and evaluation. Next, we survey the current state of tools, models, and platforms enabling AI-driven research assistants. Major AI labs (OpenAI, Anthropic, Google DeepMind, Meta) offer advanced LLMs (e.g. GPT-4, Claude 2, PaLM/Bard, LLaMA 2) and frameworks that economists can leverage for literature review, coding simulations, data analysis, model diagnostics, and even drafting papers. We highlight the capabilities of these models – such as code execution, large-context processing, and open-source customization – and provide links to resources for their use. In the practical guide section, we illustrate how economists can deploy AI agents for various research tasks: scanning and summarizing literature, writing and debugging code for simulations or empirical analysis, interpreting statistical outputs and diagnosing models, and assisting in writing and editing manuscripts. Throughout, we use examples from different economics subfields to demonstrate these applications. We also present a focused section on LangGraph, an innovative framework for building customizable multi-agent research assistants. LangGraph allows researchers to design agent workflows as graphs of interconnected LLM-powered "experts" (e.g. a data collector, an analyst, a writer) with built-in memory and quality-control loops. We discuss how LangGraph can facilitate complex, collaborative agent workflows in economics research while maintaining reliability via human-in-the-loop oversight. The overview concludes with reflections on future prospects and best practices, emphasizing that AI agents – when used thoughtfully – can significantly augment economists' productivity and creativity in research.

[remainder omitted for space reasons]

Both systems fielded some useful information for this paper. However, they also illustrate several shortcomings of these systems. Sometimes they offer unvalidated niche results with great confidence—for example, I am not sure that AI agents can "enhance the rigor of modern econometrics, particularly in causal inference," as suggested by Gemini's Deep Research system above. Sometimes they exhibit overconfidence, as in the opening sentence from OpenAI's Deep Research system, which claims that "AI agents [...] are rapidly transforming how economists conduct research." And they do not always include the most recent findings, as when OpenAI refers to "advanced LLMs" and lists models that have become outdated more than a year ago.

I found Deep Research systems to be most useful in areas where a significant amount of established research already exists. For very new topics like the one on which I tested them above, they have difficulty determining which references are the most useful—a critical skill for human researchers who work at the frontier. Moreover, I found it highly useful to provide significantly more context for Deep Research queries than in regular chatbot interactions. Even though they perform research for anything from a few minutes to half an hour, the exact amount of reseach that they engage in is sometimes difficult to steer—in part because model providers need to limit their costs. For example, a query like "Perform task X for each of the 50 States of the US" is technically feasible, but for complex tasks X, it quickly reaches the internal limits that model providers have set for the use of typical Deep Research systems.

The table below lists several of the top providers of Deep Research systems, their availability, and how they differ in their features. As can also be seen from the chat examples above, Gemini Deep Research proposes a research strategy after receiving the user's prompt, asking for confirmation of the path that it intends to pursue. OpenAI Deep Research asks the user to clarify several follow-up questions about the format and direction of the search before generating the deep research report.

Table 6
Overview of Deep Research agents

System	Availability & usage limits	Features	Time
Gemini Deep Research	Free to try; better & higher allowance on paid plans	Proposes a research plan that user can confirm/ modify	5-10 min
OpenAI Deep Research	Allowance of 25/200 reports per month for Plus/Pro plan	Asks follow-up questions to optimally target response	5-30 min
Claude Deep Research	Available subject to limits for paid plans	Runs immediately; can connect with workspace	5-15 min
Perplexity Deep Research	Free to try; unlimited under paid plan	Fast option covering lots of sources	2-4 min
xAI Grok DeepSearch	Available under X premium plan	Real-time info through close integration with X platform	<1-10 min

Technically, Deep Research systems employ a multi-agent architecture to perform open-

ended investigative research tasks that would overwhelm single agents. An LLM-powered lead researcher agent serves as the orchestrator, receiving the initial research query and developing a comprehensive strategy for how to perform the research assignment. This lead agent analyzes the question's complexity, decomposes it into discrete subtasks, and spawns specialized LLM-based subagents to pursue different investigative threads in parallel. Each subagent operates with its own context window, memory, and tools – especially access to search engines and web browsers – functioning akin to an independent researcher focused on a specific aspect of the broader question. The Gemini example above describes several of the sub-problems into which the lead agent splits the problem; for each sub-problem, multiple parallel agents browse and evaluate the available information. This parallel architecture offers substantial performance advantages over a sequential approach, especially in complex research tasks in which breadth matters, for example, analyzing a question for 50 US States or for all S&P 500 IT companies. The lead agent is also able to dynamically adapt the search strategy based on intermediate findings or errors encountered.

Deep Research agents employ substantive amounts of compute. They typically use on order of magnitude more tokens than traditional chatbot interactions. None of the labs have provided detailed numbers, but estimates suggest compute cost around a dollar per report. As a result, LLM providers strictly limit the availability of Deep Research requests, indicated in the second column of Table 6, and these systems were an important reason for the high-cost subscription plans shown in Table 5. The lead researcher agent is typically a state-of-the-art LLM to ensure that the sub-agents are given the best possible instructions and the compute that they use is well-spent — just as companies like to hire capable managers to ensure that their employees perform at their best.

As LLMs continue to improve and token costs decrease, deep research systems are likely to become economically viable for an expanding range of applications. Deploying multiple agents in parallel allows for far greater complexity in what research tasks can be performed. Tasks that previously required days of human research can now be completed in minutes, with the multi-agent architecture ensuring both thoroughness and efficiency.

3.3. Coding Agents

Another powerful application of AI agents are coding agents, which are terminal-based AI assistants that were originally designed for internal use by the leading AI labs to assist in their software engineering, but which have been shared publicly with users in recent months and have given rise to a new way of creating software – "vibe coding – that is, creating entire software projects based on user prompts in natural language. Vibe coding has made it possible for users with no programming experience to create software projects from beginning to end. However, the most powerful coding agents are only available via command line interface – this may feel a bit clunky to researchers who are used to sleek web-based interfaces but is a common interface for software developers. An example is given below:

Chat 6 (Claude Code): *Develop a simple econometric tool that lets users upload a CSV file with data, displays which data series are contained in it, and asks them what to regress on what using OLS. After the user has made a selection, show the regression results with the typical statistics included and display the marginal relationship between the regressor and the first variable in an appropriate graph.*

I'll help you create a simple econometric tool for OLS regression. Let me break this down into tasks and get started.
* Update Todos
* Create HTML structure with file upload and UI elements
* Implement CSV parsing and display data series
* Build variable selection interface for regression
* Implement OLS regression calculation
* Display regression results with statistics
* Create scatter plot with regression line
+ Wrote 130 lines to index.html
+ Wrote 261 lines to regression.js
The econometric tool is ready! Open index.html in a web browser to use it. The tool:
1. Accepts CSV file uploads
2. Displays data summary with variable statistics
3. Allows selection of dependent (Y) and independent (X) variables
4. Performs OLS regression with full statistics (coefficients, standard errors, t-values, p-values, R-squared)
5. Shows a scatter plot with the regression line
The regression results include all typical statistics and the tool displays the marginal relationship graphically.

I'm getting NaN for one of my regressions. Can you include a functionality that it drops missing values?
* I'll update the code to properly handle missing values by dropping them before regression.
* Read(regression.js)
+ Updated regression.js with 7 additions and 1 removal
The tool now properly handles missing values by:
1. Recognizing empty strings, 'NA', 'NaN', 'nan', and '.' as missing values during parsing
2. Showing the count of missing values for each variable in the data summary
3. Dropping rows with missing values in either variable before regression
4. Displaying how many observations were dropped in the regression results

---

The p-value does not seem to be correct.
* You're right. Let me fix the p-value calculation. The issue is with the Student's t-distribution implementation.
* Update(regression.js)
+ Updated regression.js with 47 additions and 16 removals
* Actually, let me use a better implementation for the t-distribution p-value calculation:
* Update(regression.js)
+Updated regression.js with 2 additions and 2 removals
* Update(regression.js)
+ Updated regression.js with 28 additions and 26 removals
The p-value calculation has been fixed. The tool now:
1. Properly handles missing values (empty strings, 'NA', 'NaN', 'nan', '.')
2. Shows missing value counts in the data summary
3. Drops rows with missing values before regression
4. Displays the number of dropped observations
5. Correctly calculates p-values using the Student's t-distribution

The first draft of the program took less than two minutes, and testing and debugging with the two follow-up prompts documented above took perhaps another five minutes. The resulting user interface and regression results of a sample regression are displayed in the figure below. This illustrates that creating software without special expertise has become far easier than it ever used to be before.

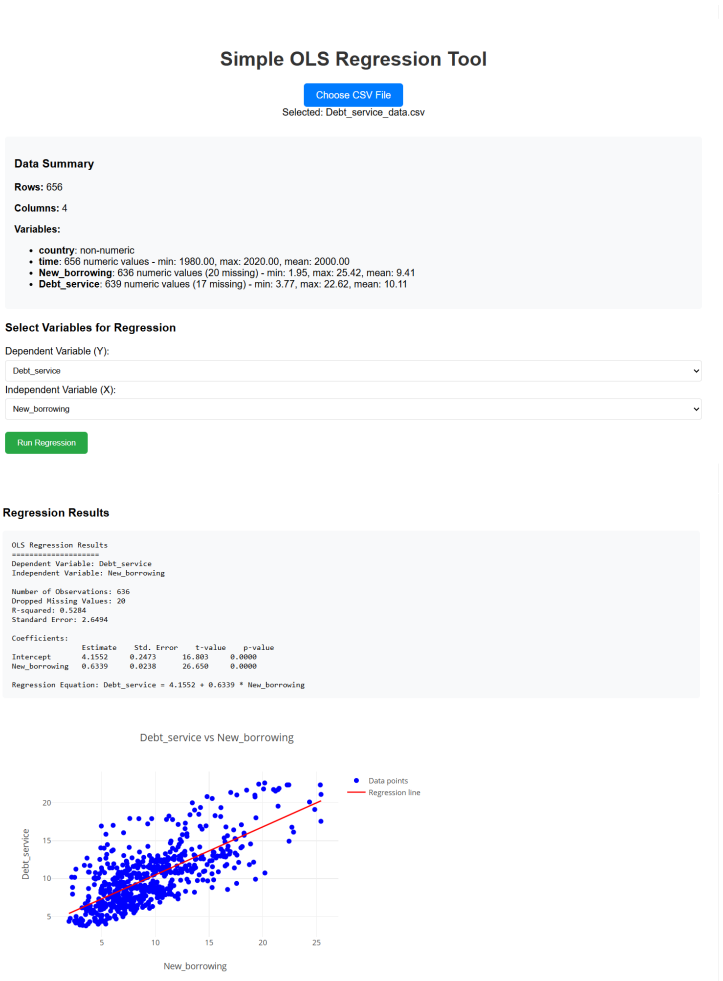


Figure 2. Claude Code Simple OLS Regression Tool

Coding agents operate on the basic principles introduced in Section 3.3.1, but with specialized capabilities for software development workflows. They combine LLM-based reasoning engines with direct access to the local file system, with code execution environments and version control systems to keep track of what changed. This architecture enables them to read and modify source code, execute commands, run tests, and manage

complete development workflows autonomously.

The three major AI labs have each released their own coding agents since the beginning of the year. Anthropic was first, releasing Claude Code as a closed system in February 2025, currently building on Claude 4/4.1. It can be accessed either via a monthly subscription to Claude Pro or Max (with certain usage limits) or on a pay-per-use basis by entering an API key. Claude Code is still considered state-of-the-art. OpenAI launched Codex CLI in April 2025, embracing an open-source approach, allowing developers to use not only OpenAI's models but also, through community forks like Open-Codex, models from Google, Anthropic, xAI, or locally-hosted open-source models. Codex is available on a pay-per-use basis. Google's Gemini CLI, the most recent entrant in June 2025, is also open-source and leverages Google's infrastructure lead to provide an exceptionally generous free tier to users, offering 60 requests per minute and 1,000 requests per day. Gemini also excels at support for PDFs and images. All three agents support core functionalities like code generation, modification and refactoring, automated debugging, unit test creation, Git integration, and multi-file editing.

Even for researchers who prefer traditional coding methods and have no interest in vibe coding, modern LLMs offer substantial value as code reviewers. These models excel at identifying bugs, security vulnerabilities, and inefficiencies in existing codebases. By simply sharing their project code with AI assistants, researchers can receive detailed feedback on potential issues ranging from memory leaks and race conditions to SQL injection vulnerabilities and algorithmic inefficiencies—catching problems that might otherwise go unnoticed until they cause unexpected failures or compromise research integrity.

For researchers who are interested in a more user-friendly interface, cloud-based AI coding environments like Replit, v0 by Vercel, and bolt.new by StackBlitz represent another approach to develop software via vibe-coding. These browser-based platforms provide instant access to AI-assisted coding without any need for local installation or cumbersome configuration. For instance, Replit, accessible via web browser at `replit.com`, combines a full development environment with integrated AI assistance, automatic hosting, and real-time collaboration features. All three platforms are particularly popular among non-technical users who want to create simple functional software without wrestling with development environment setup.

A middle ground between autonomous command-line coding agents like Claude Code and simple chatbot interfaces are AI-enhanced integrated development environments (IDEs). Tools like GitHub Copilot, Cursor, and Windsurf offer comprehensive coding support, providing intelligent suggestions, automated refactoring, and debugging assistance. These AI-powered IDEs preserve the familiar development environment that programmers expect while augmenting every aspect of the coding process with AI capabilities. For researchers accustomed to traditional development environments, these tools offer a gentle on-ramp to AI-assisted programming without requiring a wholesale shift in how they work.

Manus, an AI agent launched in March 2025 by a Singapore-based startup, demonstrated that making agents accessible could be as important as improving their capabilities. Unlike most agents at the time requiring command-line interfaces, Manus worked like a regular web application. Users could watch the agent work through a window showing its 'computer screen' as it opened browsers and navigated websites. They could replay completed tasks to understand how problems were solved, and the system continued working in the cloud after users logged off. A researcher could ask Manus to compile data from

multiple websites, close their laptop, and return later to find a completed spreadsheet. Within hours of launch, developers had created open-source versions, indicating strong demand for such user-friendly agent tools. Manus illustrates the importance of being user-friendly. By prioritizing visual feedback, replay capabilities, and eliminating technical setup, Manus lowered the barriers that had traditionally prevented non-programmers from using agents—an approach that may prove essential for more and more economists to incorporate AI agents into their research workflows.

While these agent systems offer powerful capabilities, researchers should be aware of their operational limitations. A common weakness is their brittleness to workflow interruptions—a single failed step, such as encountering a website requiring authentication, can halt an entire multi-step process. Deep Research agents may confidently synthesize information from unreliable sources alongside peer-reviewed papers without adequately distinguishing quality. Coding agents may generate syntactically correct but logically flawed code in applications that require deep domain expertise. These limitations mean that AI agents currently function best as research assistants rather than autonomous researchers—they require active human supervision to verify outputs, correct course when stuck, and ensure that economic reasoning remains sound throughout the workflow.

Moreover, as AI agents become more capable of autonomous operation, they also raise important novel questions about the future of cognitive professions and the changing nature of human-AI collaboration in technical fields. Almost all information that white collar workers process and manipulate can be codified in files and thus be made available to similar agents as coding agents.[5] For researchers studying AI's economic impact, coding agents thus provide a concrete early example of AI's potential to both augment and replace human expertise, while fundamentally changing the skills and workflows required for effective software development (Anthropic, 2025*a*). Similar developments may soon affect economists.

4. Under the Hood: Building AI Agents for Research

4.1. A Self-Made Economic Data Retrieval Agent

To make the concept of AI agents concrete, let us build a simple example that demonstrates the key principles outlined in the previous section in python. The code below implements a simple agent that can answer questions about economic data by autonomously fetching and analyzing information from the Federal Reserve Economic Data (FRED) database of the St. Louis Fed. This illustrates how to implement the described architecture and helps demystify how AI agents operate. The source code is also available for download at https://www.GenAI4Econ.org. A schematic illustration of the agent workflow is given in Figure 3.

[5]For example, Anthropic employees in fields as far apart as finance, legal, marketing, and design report that they use Claude Code to automate complex workflows without writing code themselves. They describe their workflows in plain text and leave them to Claude Code to execute automatically. These applications demonstrate how AI agents are enabling non-technical professionals to execute sophisticated technical tasks without expertise in engineering (Anthropic, 2025*b*).

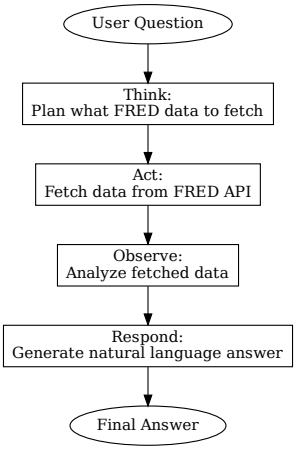


Figure 3. Architecture of Data Retrieval Agent

```
"""
econdata.py: An AI Agent using FRED - Educational Example
==========================================================
Demonstrates the core concepts of an AI agent that can:
1. Understand a question (Think)
2. Fetch economic data (Act)
3. Analyze results (Observe)
4. Generate an answer (Respond)

Setup:
1. pip install openai fredapi python-dotenv colorama
2. Get API keys from:
   - FRED: https://fred.stlouisfed.org/docs/api/api_key.html
   - OpenAI: https://platform.openai.com/api-keys
3. Create .env file with your keys in the following format:
   FRED_API_KEY=[enter key]
   OPENAI_API_KEY=[enter key]
"""

import os, json
from datetime import datetime, timedelta
from dotenv import load_dotenv
from fredapi import Fred
from openai import OpenAI
from colorama import init, Fore, Style

load_dotenv()
init()

```

```
class FREDAgent: # Define agent class

    def __init__(self): # Runs when new agent instance created
        self.fred = Fred(api_key=os.getenv('FRED_API_KEY'))
        self.llm = OpenAI(api_key=os.getenv('OPENAI_API_KEY'))

    def think(self, question): # Plan what data to fetch
        prompt = f"""What FRED series code would help answer this question?
Question: {question}

Common FRED codes: UNRATE (unemployment), FPCPITOTLZGUSA (CPI inflation),
    GDP, DFF (fed funds rate)

Return JSON: {{"explanation": "why this helps", "series_code": "
    EXACT_FRED_CODE"}}"""

        print(f"\n{Fore.CYAN}=== THINK: LLM Call ==={Style.RESET_ALL}")
        print(f"{Fore.YELLOW}Input:{Style.RESET_ALL}\n{prompt}")

        response = self.llm.chat.completions.create(
            model="gpt-4.1",
            messages=[{"role": "user", "content": prompt}],
            response_format={"type": "json_object"}
        )

        output = response.choices[0].message.content
        print(f"\n{Fore.YELLOW}Output:{Style.RESET_ALL}\n{output}")

        plan = json.loads(output)
        return plan['series_code']

    def act(self, series_code): # Fetch requested data from FRED
        print(f"\n{Fore.GREEN}=== ACT: FRED API Call ==={Style.RESET_ALL}")
        print(f"{Fore.YELLOW}Fetching:{Style.RESET_ALL} {series_code}")

        # Get series metadata for units
        info = self.fred.get_series_info(series_code)
        units = info['units']

        # Get last 2 years of data
        end = datetime.now()
        start = end - timedelta(days=730)

        print(f"{Fore.YELLOW}Period:{Style.RESET_ALL} {start.date()} to {
            end.date()}")

        data = self.fred.get_series(series_code, start, end)
        print(f"{Fore.YELLOW}Result:{Style.RESET_ALL} {len(data)} data
            points")
        print(f"Latest: {data.iloc[-1]:.2f} {units} ({data.index[-1].date()
            })")

        return data, units

    def observe(self, data, units): # Analyze fetched data
```

```
        observations = {
            "current_value": float(data.iloc[-1]),
            "current_date": data.index[-1].strftime("%Y-%m-%d"),
            "units": units
        }

        print(f"\n{Fore.MAGENTA}=== OBSERVE: Data Analysis ==={Style.RESET_ALL}")
        print(json.dumps(observations, indent=2))

        return observations

    def respond(self, question, observations): # Generate natural language response
        prompt = f"""Answer this question using the data:
Question: {question}
Data: {json.dumps(observations, indent=2)}

Provide a brief, clear answer citing specific numbers."""

        print(f"\n{Fore.CYAN}=== RESPOND: LLM Call ==={Style.RESET_ALL}")
        print(f"{Fore.YELLOW}Input:{Style.RESET_ALL}\n{prompt}")

        response = self.llm.chat.completions.create(
            model="gpt-4.1",
            messages=[{"role": "user", "content": prompt}]
        )

        output = response.choices[0].message.content
        print(f"\n{Fore.YELLOW}Output:{Style.RESET_ALL}\n{output}")

        return output

    def answer(self, question): # Define orchestrator agent
        # Think -> Act -> Observe -> Respond
        print(f"\n{Fore.BLUE}{'='*60}{Style.RESET_ALL}")
        print(f"{Fore.BLUE}Question: {question}{Style.RESET_ALL}")
        print(f"{Fore.BLUE}{'='*60}{Style.RESET_ALL}")

        try:
            series_code = self.think(question) # Think: What data?
            data, units = self.act(series_code) # Act: Fetch data
            observations = self.observe(data, units) # Observe: Analyze data
            response = self.respond(question, observations) # Respond: Generate an answer

            print(f"\n{Fore.GREEN}=== FINAL ANSWER ==={Style.RESET_ALL}")
            print(response)

            return response

        except Exception as e:
            error_msg = f"Error: {str(e)}"
            print(f"\n{Fore.RED}{error_msg}{Style.RESET_ALL}")
```

```
131             return error_msg
132
133 # Example usage
134 if __name__ == "__main__": # Execute agent with example and interactive
        prompt
135     agent = FREDAgent()
136
137     # Example query
138     agent.answer("What is the US inflation rate?")
139
140     # Interactive mode
141     print(f"\n{Fore.CYAN}Interactive mode{Style.RESET_ALL}")
142     question = input(f"\n{Fore.YELLOW}Ask an economic question: {Style.
            RESET_ALL}")
143     agent.answer(question)
```

What makes this code an "agent" rather than a simple script is its autonomous, goal-directed behavior through multiple steps. When given a question like "How is the US labor market doing?", the agent does not simply execute a predetermined sequence of commands. Instead, it exhibits the planning and tool-calling capabilities that define AI agents: it first *thinks* about what data would help answer the question, then *acts* by calling the appropriate external tool (the FRED API), *observes* by analyzing the returned data, and finally *responds* with a natural language answer. This Think-Act-Observe loop characterizes sophisticated agentic systems.

The agent demonstrates several crucial concepts from our earlier discussion. First, it combines an LLM reasoning engine (OpenAI's GPT-4.1) with external tools (FRED API), illustrating how agents extend beyond the knowledge limitations of language models alone. Second, it shows autonomous decision-making: the agent determines which economic indicator to fetch based on its understanding of the user's question, rather than requiring the user to specify technical details like FRED series codes. Third, it performs multi-step reasoning, breaking down the task of answering an economic question into discrete, manageable steps that build upon each other.

From a technical perspective, the agent's ‘answer()‘ method implements the orchestrator described in Section 3.3.1, coordinating between the reasoning engine and external tools. Each method in the class represents a distinct capability: ‘think()‘ for planning, ‘act()‘ for tool invocation, ‘observe()‘ for data analysis, and ‘respond()‘ for synthesis. This modular structure makes it easy to understand how each component contributes to the agent's overall intelligence and allows researchers to extend the agent with additional tools or more sophisticated reasoning capabilities.

Several features of the code above are worth emphasizing. First, it is fully vibe-coded – coding agents are capable of writing simple programs like this in less than a minute. The code is also structured in a clean and easily legible manner with clear documentation. Second, a significant part of the code consists of natural language instructions to the LLM. For example, lines 37 – 42 instruct the LLM to pick an appropriate FRED data series to download and specifies the format. Lines 92 – 96 instruct the LLM to interpret the data to formulate a response in natural language.

I hope that this simple example demystifies the "magic" of AI agents by showing that they are, at their core, programs that combine LLM reasoning with systematic access to

external tools. It also highlights practical considerations like API calls (each question requires two LLM calls), latency (the multi-step process takes several seconds), and the importance of prompt engineering in getting reliable results from the reasoning engine. Perhaps most importantly, it provides a foundation that researchers can build upon, adding more data sources, more sophisticated analysis capabilities, or multi-agent coordination to tackle more complex research questions.

The following example illustrates our simple agent in action:

Chat 7 (econdata.py agent): *How is the US labor market doing?*

=== THINK: LLM Call ===
Input:
What FRED series code would help answer this question?
Question: How is the US labor market doing?
Common FRED codes: UNRATE (unemployment), FPCPITOTLZGUSA (CPI inflation), GDP, DFF (fed funds rate)
Return JSON: {"explanation": "why this helps", "series_code": "EXACT_FRED_CODE"}
Output:
{"explanation": "The unemployment rate (UNRATE) is a key indicator of labor market health, reflecting the percentage of people actively seeking work but unable to find employment. It provides a broad gauge of labor market conditions in the US.", "series_code": "UNRATE"}
=== ACT: FRED API Call ===
Fetching: UNRATE
Period: 2023-07-29 to 2025-07-28
Result: 24 data points
Latest: 4.10 Percent (2025-06-01)
=== OBSERVE: Data Analysis ===
{
"current_value": 4.1,
"current_date": "2025-06-01",
"units": "Percent"
}
=== RESPOND: LLM Call ===
Input:
Answer this question using the data:
Question: How is the US labor market doing?
Data: {
"current_value": 4.1,
"current_date": "2025-06-01",
"units": "Percent"
}
Provide a brief, clear answer citing specific numbers.
Output:
As of June 1, 2025, the US labor market has an unemployment rate of 4.1 percent. This figure indicates a relatively healthy job market, as unemployment rates around 4 percent are generally considered low and typical of a stable economy.

4.2. From Linear Scripts to Complex LangGraph Agents

While our economic data retrieval agent demonstrates the core concepts of AI agents effectively, it represents what we might call a "first-generation" approach to agent design. The agent follows a rigid, linear path: think → act → observe → respond. This works

well for straightforward queries, but real-world research often requires more flexibility. What if the agent needs to fetch multiple data series? What if the initial data suggests a follow-up query would be valuable? What if we want the agent to backtrack when it realizes it's pursuing an unproductive path?

LangGraph is a software package available in both python and JavaScript that addresses these limitations by allowing for agents to operate as state machines with directed graphs rather than linear sequences. The fundamental difference between the simple agent in the previous subsection and LangGraph lies in how the agent's workflow is structured. Our linear implementation uses a simple orchestrator:

```
# Our linear implementation (lines 118-121 above)
def answer(self, question):
series_code = self.think(question)
data, units = self.act(series_code)
observations = self.observe(data, units)
response = self.respond(question, observations)
```

LangGraph transforms this into a graph structure with three key components:

1. State Management:. — Instead of passing data between methods via parameters and return values, LangGraph uses a centralized state object that flows through the entire graph. Each step reads from and writes to this shared state.

2. Nodes:. — Each method becomes a node function that operates on the state:

```
def think_node(state: AgentState) -> AgentState:
# Extract question from state instead of parameter
question = state["question"]
# LLM logic to determine series_code
series_code = call_llm_for_series_code(question)
# Return updated state
return {"series_code": series_code}
```

3. Constructing the Graph:. — After defining nodes, we build the graph by connecting them:

```
graph = StateGraph(AgentState)
graph.add_node("think", think_node)
graph.add_node("act", act_node)
graph.add_node("observe", observe_node)
graph.add_node("respond", respond_node)
# Define the flow
graph.add_edge("think", "act")
graph.add_edge("act", "observe")
graph.add_edge("observe", "respond")
```

However, unlike in our simple example, LangGraph can dynamically choose paths based on the state. For instance, if the agent determines multiple data series are needed, it could branch to a different analysis path without code modification.

In the appendix, we provide a complete but simplified LangGraph implementation of our economic data retrieval agent that demonstrates these architectural principles while maintaining clarity and brevity. The code blocks in the preceding points follow this example. The implementation shows how the linear flow transforms into a graph structure that provides a foundation for more sophisticated research applications.

Moreover, LangGraph offers several additional benefits for more complex agentic systems. First, every state transition is traceable, making it easier to understand and debug complex multi-step analyses. Researchers can see exactly what data the agent considered at each step and why it made specific decisions. Second, LangGraph can save the state at each node, enabling researchers to inspect the agent's reasoning process, resume interrupted analyses, or explore alternative paths from any decision point. Third, Adding new capabilities is as simple as adding new nodes and edges. For example, multiple data sources can be fetched in parallel by adding a branching path.

The key insight is that LangGraph transforms our procedural agent into a declarative graph structure. Instead of hard-coding the sequence of operations, the user can define the possible states and transitions, allowing the agent to navigate this space dynamically. This becomes particularly powerful when building more complex research agents that need to explore multiple hypotheses, handle various data sources, or adapt their strategy based on intermediate findings.

4.3. Building a Deep Research Agent Using LangGraph

Having explored the transition from linear scripts to graph-based architectures, we now turn to implementing a more sophisticated research system that embodies the multi-agent principles discussed in Section 3.3.2. The Deep Research agent presented here illustrates how LangGraph enables the construction of systems that can autonomously pursue complex research questions through parallel investigation, dynamic strategy adaptation, and multi-stage synthesis. The full source code of the agent system is available in the appendix and requires the user to sign up to both an LLM API (in our example, OpenAI) and to a search provider (here, Tavily).[6]

The architecture of our Deep Research agent mirrors the commercial systems described earlier, but is less ambitious and at a scale suitable for understanding and experimentation. Nonetheless, it is far more powerful than a simple LLM query. At its core, the system implements a lead researcher that orchestrates multiple specialized sub-agents, each pursuing different aspects of the research question in parallel. This design choice reflects that complex research questions rarely have simple, linear paths to answers. Instead, they require exploring multiple hypotheses, cross-referencing different sources, and synthesizing findings across different domains.

We start by describing the agent's set of state variables, which lies at the heart of LangGraph-based architectures. Unlike our simple economic data agent, which maintains minimal state variables between steps, the Deep Research agent tracks a rich state object

[6]A more comprehensive open-source agent "Open Deep Research" is also available from LangChain (2025). For pedagogical reason, our implementation here focuses on simplicity.

that includes the research plan, multiple subtasks, accumulated search results, analyses from different perspectives, and the evolving synthesis. This state flows through the graph, with each node reading from and contributing to the collective understanding. The formal definition of the system's state looks as follows (see lines lines 37–47 in the full code in the appendix):

```
class ResearchState(TypedDict):
    question: str
    research_plan: Dict[str, List[str]]
    subtasks: List[Dict[str, str]]
    search_results: List[Dict[str, any]]
    analysis_results: List[Dict[str, str]]
    final_report: str
```

The code below constructs the graph. It shows how LangGraph allows users to orchestrate a complex research workflow (see lines 82–108 in the full code):

```
def _build_graph(self) -> StateGraph:
    """Construct the research workflow graph"""
    workflow = StateGraph(ResearchState)

    # Add nodes
    workflow.add_node("lead_researcher", self.lead_researcher_node)
    workflow.add_node("spawn_subtasks", self.spawn_subtasks_node)
    workflow.add_node("search_agent", self.search_agent_node)
    workflow.add_node("analysis_agent", self.analysis_agent_node)
    workflow.add_node("synthesis_agent", self.synthesis_agent_node)

    # Define edges
    workflow.set_entry_point("lead_researcher")
    workflow.add_edge("lead_researcher", "spawn_subtasks")
    workflow.add_edge("spawn_subtasks", "search_agent")
    workflow.add_conditional_edges(
        "search_agent",
        self.should_continue_searching,
        {
            "continue": "search_agent",
            "analyze": "analysis_agent"
        }
    )
    workflow.add_edge("analysis_agent", "synthesis_agent")
    workflow.add_edge("synthesis_agent", END)

    return workflow.compile()
```

This graph structure explicitly defines the flow of research from strategic planning through parallel investigation to final synthesis. The conditional edges enable dynamic behavior – depending on the state, the agent can loop through multiple search iterations before proceeding to analysis, adapting to the complexity of the research question. In the code snippet above, the conditional edges are implemented through the `should_continue_searching` method. They allow the agentic system to dynamically decide whether to continue

gathering information or proceed to analysis based on the current state. This flexibility proves essential when dealing with research questions: simple queries might require only a few searches, while nuanced topics might demand extensive investigation across multiple subtasks.

In the analysis phase, agents examine the findings in the context of the defined research directions and questions. They can identify patterns, contradictions, and gaps – the kind of higher-order thinking that distinguishes research from mere information retrieval. The parallel execution of these analysis agents means that multiple perspectives on the data develop simultaneously, potentially delivering insights that sequential processing might miss. Finally, the `synthesis_agent` node performs perhaps the most challenging task: integrating all the diverse findings into a coherent narrative. This node must balance comprehensiveness with clarity and condense all the retrieved findings into a clear narrative. Using a highly capable model here ensures that the final report maintains the quality expected from a deep research system.

An architectural feature that makes the system more powerful is its extensive use of parallel execution. The implementation leverages Python's `ThreadPoolExecutor` to enable parallel operation of search and analysis. For the former, multiple search queries can execute simultaneously within each subtask. When the search_agent node activates, it spawns multiple search threads by creating "future" tickets that are appended to a list of all open tickets (search_futures). The function "future.result()" in the penultimate line waits until all parallel tasks have been executed before delivering the results.

```
# Perform parallel searches using ThreadPoolExecutor
search_futures = []
for query in subtask['search_queries']:
    future = self.executor.submit(self._perform_search, query)
    search_futures.append(future)

# Collect results from all parallel searches
for future in search_futures:
    results = future.result()
    state['search_results'].extend(results)
```

This parallelism dramatically reduces search time – if each search takes 2 seconds and a subtask has 3 queries, parallel execution saves 4 seconds per subtask. A similar form of parallel analysis occurs in the analysis layer. Our implementation processes subtasks in batches of three to minimize graph recursions while maintaining parallelism within each batch. Commercial systems likely employ dozens of search agents investigating different research directions simultaneously.

Our implementation also demonstrates an important optimization strategy: using different models for different roles. The lead researcher, responsible for strategic planning and final synthesis, uses a more capable model (GPT-4.1), whereas the analysis agents, which perform a more straightforward task on a well-defined input, can use a smaller, faster model (GPT-4.1-mini). This tiered approach can significantly reduce costs while maintaining quality where it matters most – in strategic planning and final synthesis. The following is the output the agent generated in response to the default question in the code:

Chat 8 (deepresearch.py): *What are the labor market effects of transformative AI expected to be?*

[performed approximately 1 minute of analysis]
Lead Researcher: Developing research strategy...
Research strategy developed with 3 directions
Creating research subtasks...
Created 5 focused research subtasks
Will execute ~15 searches across all subtasks
Search Agent: Processing subtasks 1-3/5
Batch complete: collected 9 search results
Search Agent: Processing subtasks 4-5/5
Batch complete: collected 6 search results
Analysis Agents: Analyzing search results...
Completed analysis of 5 subtasks
Synthesis Agent: Creating final report...
Research report completed!

The Labor Market Effects of Transformative AI: A Comprehensive Research Synthesis

Executive Summary

Transformative artificial intelligence (AI) is rapidly reshaping global labor markets, with profound implications for occupational structures, wage dynamics, and workforce adaptation. This report synthesizes recent empirical research and expert analyses to assess the expected labor market effects of transformative AI. Key findings include:

- **Occupational Displacement and Creation:** Routine, repetitive, and some creative jobs are most susceptible to automation, while new roles are emerging in AI development, oversight, and augmentation. By
2030, up to 30% of U.S. jobs may be automated, with 60% experiencing significant task modification [National University, 2024].
- **Wage and Inequality Dynamics:** AI adoption is driving wage polarization, benefiting high-skilled workers and exacerbating income inequality. Urban and tech-centric regions are seeing job growth, while
rural and low-skill sectors face stagnation or decline [Veritone, 2025; Brookings, 2024].
- **Labor Market Adaptation and Policy Response:** Reskilling programs and social safety nets are essential but have mixed effectiveness. Success depends on targeted, well-funded, and experiential approaches, as well as robust social protections [Brookings, 2024; HBR, 2023].
- **Synthesis:** The net effect of transformative AI is a dual labor market: expanding opportunities for high-skill workers while increasing displacement risks for low-skill and routine occupations. Without proactive policy and workforce development, inequality is likely to worsen.

Recommendations include prioritizing targeted reskilling, strengthening social safety nets, fostering human-AI collaboration, and investing in longitudinal research to monitor labor market transitions.
---

1. Occupational Displacement and Creation

1.1. Jobs Most at Risk

[report truncated for space reasons]

Our self-made deep research system successfully compiled a report complete with real citations, at a total cost of approximately 1 cent in OpenAI tokens and 15 Tavily search queries. The contrast with commercial Deep Research systems is instructive as the analysis is clearly less sophisticated. While Google's Gemini Deep Research might spawn dozens of parallel sub-agents processing hundreds of sources in total, our implementation

typically handles 5-6 subtasks with controlled parallelism – sufficient for understanding the architecture while remaining computationally tractable. The core principles remain identical: decomposition of complex questions, parallel investigation, and systematic synthesis. Another reason for the superior performance of commercial Deep Research systems is that they have been optimized via reinforcement learning on millions of attempts to learn when to pick which tool and when to optimally continue the research process versus when to stop research in a particular direction.

Our implementation also hints at natural extensions. We could use more sophisticated (and expensive) LLMs as orchestrator. Additional parallel processing could be added at the subtask level, allowing multiple research directions to be explored simultaneously. Specialized analysis nodes could run in parallel for different content types – statistical analysis for quantitative data, sentiment analysis for policy documents, and citation network analysis for academic literature could all process the same search results concurrently.

Perhaps most importantly, our simple Deep Research agent demonstrates that sophisticated AI research systems are not black boxes accessible only to major tech companies. With roughly 300 lines of well-structured Python code – much of it natural language prompts – researchers can build systems that embody the same architectural principles as cutting-edge commercial offerings. The explicit parallelism in the implementation shows how multi-agent systems achieve their impressive performance: not through any single breakthrough, but through the coordinated effort of multiple specialized agents working in concert.

As with our simpler examples, this code was largely vibe-coded with assistance from AI coding agents, underscoring a recursive truth: AI agents are now capable of building AI agents. This capability spiral, where each generation of tools enables the creation of still more sophisticated successors, suggests that the pace of advancement in AI-assisted research will likely accelerate rather than plateau.

4.4. Open Protocols for Agent Interoperability

As AI agents are becoming increasingly sophisticated and specialized, they need standardized communication protocols. Just as the internet required the HyperText Transfer Protocol (HTTP) standard to enable communication between different systems, the emerging ecosystem of AI agents requires protocols that enable seamless interaction between agents built on different frameworks, hosted by different providers, and specialized for different tasks. Without such standards, the challenge is exponential: every system uses different data formats, authentication methods, and error handling approaches, meaning N agents connecting to M tools requires N×M custom integrations. With standardized protocols, this reduces to just N+M connections. Two recent open protocols, the Model Context Protocol (MCP) and the Agent2Agent (A2A) Protocol, address complementary aspects of this challenge and have significant implications for how researchers can build and deploy AI systems.

The Model Context Protocol (MCP): Connecting Agents to Resources, Tools and Data

The Model Context Protocol was introduced by Anthropic in November 2024 (Anthropic, 2024) and has been adopted by all the major AI labs since. It has emerged as the de-facto standard to build secure, two-way connections between AI agents and the resources, tools,

and data from another party. Consider the challenge faced by a researcher employing an AI assistant to analyze economic data. Without MCP, connecting the assistant to each data source – FRED, other institutional databases, private research repositories – requires custom integration code. Every new data source requires its own type of implementation (as we encountered in our FRED example above), making fully connected systems difficult to scale. MCP solves this by providing a universal connection standard, much like USB-C provides a universal physical connection for devices.

The protocol operates on a client-server architecture where MCP servers offer resources, tools, and data sources, while MCP clients (typically AI agents or applications) can connect to these servers to access their capabilities. For example, an economic institution like a central bank or an individual researcher compiling a new data source can create an MCP server to make their data accessible to any AI system or agent without requiring users to develop dataset-specific knowledge or API details. The MCP server can act as a standardized gateway (Model Context Protocol Working Group, 2025).

At a technical level, creating an MCP server requires building a standardized gateway to the data that is offered, similar to creating a basic website: one defines which datasets to provide (e.g., economic statistics), what analytical tools to provide (e.g., time series analysis or data filtering), and what access permissions to implement. The MCP framework handles the complex communication protocols automatically, so developers can focus on specifying what data to share rather than how to share it. IT departments can implement a basic MCP server in a matter of days, as it primarily involves wrapping existing data access methods in the standard MCP format – much like putting a universal adapter on existing electrical equipment rather than rewiring the entire system.

Once an MCP server is operational, researchers can connect to it through two primary modes. For local use (e.g., within an institution or on an individual computer), users simply add the server configuration to their AI application (like Claude Code or Desktop), similar to adding a printer to their computer – the AI assistant then gains direct access to query datasets, run analyses, and retrieve results from the MCP server through natural language commands. For public access, institutions can deploy their MCP servers on the internet with appropriate authentication, allowing external researchers to connect after obtaining credentials, thereby providing automated access to valuable research data while maintaining security and usage controls.

At the time of writing – less than a year after the protocol was established – close to 10,000 MCP servers were available for AI agents to connect to. For economists, general-use MCP servers provide AI agents automated access to the user's file system, email systems (e.g., Gmail or Outlook), local databases, GitHub, or apps like Slack. There are also economics-specific MCPs, for example third-party wrappers that provide access to FRED or IMF data available on the website pulsemcp.com that users can connect to so that their AI agents or chatbots can automatically access the relevant data. I expect that the institutions behind these economic databases will soon offer their own MCP servers. The rapidly expanding set of MCP servers will continue to expand the availability of high-quality economic resources to AI agents to enable them to manage larger parts of the research lifecycle.

The Agent2Agent Protocol: Enabling Multi-Agent Collaboration

While MCP focuses on connecting agents to data, the Agent2Agent (A2A) protocol, launched by Google with several dozen industry partners in April 2025, enables AI agents to communicate with each other, securely exchange information, and coordinate actions on top of various enterprise platforms or applications. The protocol was transferred to the Linux Foundation to make it more independent in June 2025. It addresses a different but equally important challenge, how specialized agents can work together on complex tasks.

The A2A protocol enables several key capabilities. For each participating agent, Agent Cards detail the agent's capabilities and connection information, allowing other agents to understand what tasks it can perform and how to transact with it. The protocol supports everything from quick queries between agents to long-running collaborative research projects. A tangible example of a (fictitious) Agent Card is the following:

```
{
    "agent_id": "panel_regression_agent",
    "name": "Panel Regression Specialist",
    "endpoint": "https://www.genai4econ.org/agents/panel",
    "defaultInputModes": ["csv", "xlsx", "stata_dta"],
    "defaultOutputModes": ["regression_table", "latex", "json"],
    "capabilities": [
        {
            "skill": "fixed_effects",
            "description": "Run fixed effects regression with entity and
                time FE"
        },
        {
            "skill": "random_effects",
            "description": "Perform random effects estimation with Hausman
                test"
        }
    ]
}
```

These Agent Cards can be posted in public registries, under well-known URLs, or in pre-configured allow-lists of an orchestrator agent so that other agents can discover the capabilities that are on offer.

Practical Implications for Economic Research

While economic research is just beginning to explore these protocols, industry has already deployed them at scale. MCP powers thousands of integrations including GitHub for code management, Slack for team communications, and Jira for project tracking—enabling AI agents to autonomously manage software development workflows. A2A has seen rapid adoption in customer service, where specialized agents handle billing, technical support, and scheduling in parallel, and in enterprise automation, orchestrating document processing, compliance checking, and approval workflows across departments. These deployments demonstrate that the infrastructure for sophisticated multi-agent research systems already exists.

The described protocols also create powerful possibilities for economic research workflows. Consider a research project investigating the employment effects of AI adoption. A lead research agent may connect to multiple data sources through MCP servers – BLS employment statistics, company financial reports, patent databases, or academic paper repositories. Each connection uses the same standardized protocol, eliminating integration complexity. The lead agent can then delegate subtasks to specialized agents, e.g., data cleaning agents, financial analysis agents, econometric agents, visualization agents, etc. Despite being built by different creators on different systems, agents can communicate through A2A, sharing results and coordinating their efforts.

Both protocols are also usable with LangGraph. MCP enjoys native support through the `langchain-mcp-adapters` package, which seamlessly converts MCP tools into LangGraph-compatible formats. A2A integration currently requires wrapping LangGraph agents with A2A server code, but the process is well-documented with official examples.

Beyond building and connecting agents, rigorous evaluation systems are essential for production deployment. MLflow 3, released in June 2025, exemplifies the emerging infrastructure for agent monitoring, offering specialized functionality for tracking agent performance, comparing different prompting strategies, and evaluating multi-step workflows. The platform enables researchers to systematically assess agent reliability across tasks, measure drift in performance over time, and identify failure modes before they impact research outcomes.

Future Directions

These protocols represent early steps toward a more interconnected automated AI ecosystem. As they mature, I expect several developments in research: Common and reusable agent patterns may emerge for literature review, data analysis, and paper writing. Agents at different institutions could collaborate while respecting data privacy and access controls, enabling new forms of distributed research. These protocols may also enable markets for specialized agents in research and other domains.

5. Conclusions

This paper has documented the evolution from LLM-based chatbots to AI agents—autonomous systems that plan, use tools, and execute multi-step research tasks. The examples throughout demonstrate that we have crossed a critical threshold: economists can now build sophisticated research assistants without technical expertise, agents can autonomously conduct literature reviews across hundreds of sources, and "vibe coding" enables the creation of complex analytical tools through natural language alone. These are not future possibilities but present realities, as the working code examples in this paper attest.

The trajectory from reactive chatbots to proactive agents reveals something profound about the nature of research automation. When AI systems can decompose complex questions, gather information autonomously, and synthesize findings without constant human oversight, we are witnessing the early stages of what may become comprehensive research automation. The Deep Research agents that today compile existing knowledge may tomorrow generate novel hypotheses and test them. The coding agents that currently implement our specifications may soon design their own research methodologies.

However, humans are not yet obsolete in research. While current LLM-based AI systems excel at synthesis and implementation, they have yet to consistently demonstrate genuine creativity. The spark of insight that identifies a truly novel research question, the intuition that connects disparate phenomena, the judgment that recognizes profound implications—these remain at present human contributions. AI agents are superb research assistants but not yet independent researchers.

The gap, however, may be narrowing faster than we anticipate. The capability spiral documented in this paper—where AI helps build better AI, which enables still more sophisticated systems—suggests an accelerating pace of development. If human-level AI emerges across all cognitive domains, as many AI pioneers predict, the automation of economic research becomes not a question of if but when. The multi-agent systems that today require human orchestration may tomorrow coordinate themselves, pursuing research agendas we can barely imagine.

For economists, this presents both immediate opportunities and long-term challenges. In the near term, AI agents offer unprecedented leverage: junior researchers can accomplish what once required entire teams, established scholars can pursue more ambitious agendas, and technical barriers that once excluded many from computational work are dissolving. The democratization of research capabilities may lead to a golden age of discovery, as more minds can test more hypotheses with fewer constraints.

Yet this rapid integration of AI into research workflows also brings unexpected vulnerabilities. Gans (2025) documents how authors can manipulate AI-powered peer review systems through prompt injection – a phenomenon that reveals both the speed of AI adoption in academia and the emergence of new forms of strategic behavior. While his analysis suggests such manipulation might paradoxically improve welfare by preventing over-reliance on automated reviews, it underscores a broader point: as AI agents become more central to research processes, we must anticipate and address novel forms of gaming and manipulation that were impossible in purely human systems.[7]

More fundamentally, we must prepare for a transformation of our discipline. As research generation becomes increasingly automated, the economist's role will shift from producing analysis to defining values, interpreting implications, and ensuring that economic insights serve human flourishing. The questions that will matter most—what problems deserve investigation, how should we evaluate trade-offs, what constitutes progress—will remain irreducibly human even as the technical machinery of research becomes artificial.

The practical implications are clear. First, economists should embrace these tools now, not merely to boost productivity but to understand their capabilities and limitations. Building your own agents, as this paper demonstrates, demystifies the technology and reveals both its power and its boundaries. Second, we must invest in the uniquely human aspects of economic thinking—ethical reasoning, creative problem formulation, and wisdom about social welfare—that will remain essential even in an AI-saturated future.

Finally, economists have a special responsibility. We understand incentives, market dynamics, and resource allocation. We grasp the subtleties of human behavior and social coordination. These insights will be crucial for the transition to an AI-augmented research

[7]Beyond their immediate research applications, AI agents may fundamentally reshape economic markets and institutions as they evolve from tools to economic actors in their own right (see, e.g., Hadfield and Koh, 2025; Rusak et al., 2025).

ecosystem. By engaging deeply with AI agents today—building them, using them, and thinking carefully about their implications—we can help shape a future where artificial intelligence amplifies rather than replaces human economic wisdom.

The competition period with AI that Kasparov experienced for chess in the 1990s may be beginning for economic research. Unlike chess, however, the stakes extend far beyond academic prestige. How we integrate AI agents into economic research will influence how society understands and addresses its most pressing challenges. The tools documented in this paper are not just productivity enhancers but harbingers of a transformed discipline. Our task is to guide that transformation wisely, ensuring that as our artificial colleagues grow more capable, our human contribution grows more thoughtful, more ethical, and more focused on what truly matters for human well-being.

Appendix A: Additional Examples

A1. Data Retrieval Agent Using LangChain

The following LangGraph implementation transforms the class-based agent from 4.4.1 into a graph of interconnected functions. The code begins by defining ‘AgentState‘, a TypedDict that replaces our method parameters and return values with a single state object containing all data needed throughout the analysis. Each of our original methods becomes a node function that receives this state, performs its operation, and returns only the fields it needs to update. For instance, ‘think_node‘ extracts the question from state and returns just the series_code, while ‘act_node‘ takes that series_code and adds the fetched data values. This pattern makes data flow explicit and traceable.

The graph construction in ‘create_fred_agent()‘ illustrates the declarative nature of LangGraph. Rather than calling methods sequentially, we define nodes and edges that specify possible execution paths. The ‘add_edge‘ calls create a linear flow for this simple example, but this structure easily extends to support branching, loops, and parallel execution. When we invoke the compiled graph with an initial question, LangGraph automatically manages the state updates as execution flows through each node, ultimately producing our final response. This separation of logic (nodes) from flow (edges) is what enables LangGraph agents to handle complex, adaptive research workflows that would be cumbersome to implement in traditional procedural code.

```
"""
econdata_langgraph.py: FRED Agent with LangGraph
================================================
A concise LangGraph implementation demonstrating graph-based agent architecture
for economic data retrieval. Maintains the same flow as econdata.py but with
explicit state management and graph structure.

Setup:
1. pip install openai fredapi python-dotenv colorama langgraph grandalf "langsmith==0.3.45"
2. Use an .env file with FRED_API_KEY and OPENAI_API_KEY
"""

import os, json
```

```
from datetime import datetime, timedelta
from typing import TypedDict, Optional
from dotenv import load_dotenv
from fredapi import Fred
from openai import OpenAI
from langgraph.graph import StateGraph, END

load_dotenv()

# State definition - data that flows through the graph
class AgentState(TypedDict):
    question: str
    series_code: Optional[str]
    data_value: Optional[float]
    data_date: Optional[str]
    units: Optional[str]
    response: Optional[str]
    error: Optional[str]

# Initialize clients
fred = Fred(api_key=os.getenv('FRED_API_KEY'))
llm = OpenAI(api_key=os.getenv('OPENAI_API_KEY'))

# Node functions - each represents a step in our analysis
def think_node(state):
    """Determine which FRED series to fetch based on the question."""
    prompt = f"""What FRED series code would help answer this question?
Question: {state['question']}

Common FRED codes: UNRATE (unemployment), FPCPITOTLZGUSA (CPI inflation),
    GDP, DFF (fed funds rate)

Return JSON: {{"series_code": "CODE"}}"""

    response = llm.chat.completions.create(
        model="gpt-4.1",
        messages=[{"role": "user", "content": prompt}],
        response_format={"type": "json_object"}
    )

    result = json.loads(response.choices[0].message.content)
    return {"series_code": result['series_code']}

def act_node(state):
    """Fetch data from FRED API."""
    try:
        # Get series info and recent data
        info = fred.get_series_info(state['series_code'])
        data = fred.get_series(
            state['series_code'],
            observation_start=datetime.now() - timedelta(days=365)
        )

        return {
            "data_value": float(data.iloc[-1]),
```

```
            "data_date": data.index[-1].strftime("%Y-%m-%d"),
            "units": info['units']
        }
    except Exception as e:
        return {"error": str(e)}

def respond_node(state):
    """Generate natural language response from the data."""
    if state.get('error'):
        return {"response": f"Error: {state['error']}"}

    prompt = f"""Answer this question using the data:
    Question: {state['question']}
    Data: {state['data_value']} {state['units']} as of {state['data_date']}
    """

    response = llm.chat.completions.create(
        model="gpt-4.1",
        messages=[{"role": "user", "content": prompt}]
    )

    return {"response": response.choices[0].message.content}

# Build the graph
def create_fred_agent():
    """Construct the agent graph."""
    graph = StateGraph(AgentState)

    # Add nodes
    graph.add_node("think", think_node)
    graph.add_node("act", act_node)
    graph.add_node("respond", respond_node)

    # Define flow
    graph.add_edge("think", "act")
    graph.add_edge("act", "respond")
    graph.add_edge("respond", END)

    # Set entry point
    graph.set_entry_point("think")

    return graph.compile()

# Example usage
if __name__ == "__main__":
    agent = create_fred_agent()
    question = "How is the US labor market doing?"
    result = agent.invoke({"question": question})

    print(f"\nQuestion: {question}")
    print(f"Answer: {result['response']}")

    # Show graph structure
    print("\nGraph Structure:")
    print(agent.get_graph().draw_ascii())
```

A2. Deep Research Agent

The following implementation demonstrates a sophisticated multi-agent research system that embodies the principles discussed in Section 3.3.2. The architecture is discussed in Section 4.4.3. Unlike the simple data retrieval agent above, this system orchestrates multiple specialized agents to pursue complex research questions through parallel investigation and synthesis. The code showcases how LangGraph enables the construction of research systems that autonomously decompose questions into subtasks, execute dozens of searches in parallel, analyze findings from multiple perspectives, and synthesize comprehensive reports. At approximately 370 lines of code, it illustrates that powerful AI research systems are accessible to researchers and developers, not just major tech companies.

The implementation centers around a rich ResearchState object that tracks the research plan, subtasks, search results, analyses, and the evolving synthesis as it flows through the graph. The graph structure, defined in `_build_graph()`, creates a workflow from strategic planning through parallel investigation to final synthesis. Key architectural features include the use of `ThreadPoolExecutor` for parallel search execution (up to 10 concurrent operations), batch processing of subtasks to minimize graph recursions while maximizing parallelism, and a tiered approach to model selection, using GPT-4.1 for strategic planning and synthesis while employing GPT-4.1-mini for the more straightforward analysis tasks. The conditional edges in the graph enable dynamic behavior, allowing the system to adapt its search depth based on the complexity of findings. This design demonstrates how multi-agent systems achieve their performance not through any single breakthrough, but through the coordinated effort of specialized agents working in concert.

Text

```
"""
deep_research_agent.py: Deep Research Agent for Economic Research using LangGraph
================================================================================

A multi-agent system that performs comprehensive research on economic topics through:
1. Lead Researcher - Develops research strategy and synthesizes findings
2. Search Agents - Execute parallel searches across multiple sources
3. Analysis Agents - Analyze results from different perspectives
4. Synthesis Agent - Integrates all findings into a comprehensive report

Setup:
1. pip install langchain-openai langgraph tavily-python python-dotenv
2. Get API keys from:
   - OpenAI: https://platform.openai.com/api-keys
   - Tavily: https://app.tavily.com/
3. Create .env file with your keys in the following format:
   OPENAI_API_KEY=[enter key]
   TAVILY_API_KEY=[enter key]
"""

```

```
import os
from typing import Dict, List, TypedDict, Annotated, Literal
import json
import asyncio
from concurrent.futures import ThreadPoolExecutor
from dotenv import load_dotenv

from langchain_openai import ChatOpenAI
from langchain_core.messages import HumanMessage, SystemMessage
from langgraph.graph import StateGraph, END
from langgraph.graph.message import add_messages
from tavily import TavilyClient

# Load environment variables
load_dotenv()


class ResearchState(TypedDict):
    """State that flows through the research graph"""
    question: str
    research_plan: Dict[str, List[str]]
    subtasks: List[Dict[str, str]]
    search_results: List[Dict[str, any]]
    analysis_results: List[Dict[str, str]]
    final_report: str
    messages: Annotated[List, add_messages]
    current_subtask: int
    max_iterations: int


class DeepResearchAgent:
    """Multi-agent system for deep economic research"""

    def __init__(self, openai_api_key: str = None, tavily_api_key: str =
        None):
        # Use provided keys or fall back to environment variables
        openai_key = openai_api_key or os.getenv('OPENAI_API_KEY')
        tavily_key = tavily_api_key or os.getenv('TAVILY_API_KEY')

        if not openai_key:
            raise ValueError("OpenAI API key not found. Please set
                OPENAI_API_KEY in .env file")
        if not tavily_key:
            raise ValueError("Tavily API key not found. Please set
                TAVILY_API_KEY in .env file")

        # Initialize LLMs
        self.lead_llm = ChatOpenAI(
            model="gpt-4.1",
            temperature=0.1,
            api_key=openai_key
        )
        self.analysis_llm = ChatOpenAI(
            model="gpt-4.1-mini",
            temperature=0.1,
```

```
72              api_key=openai_key
73          )
74
75          # Initialize tools
76          self.tavily = TavilyClient(api_key=tavily_key)
77          self.executor = ThreadPoolExecutor(max_workers=10)  # Increased for
                more parallelism
78
79          # Build the research graph
80          self.graph = self._build_graph()
81
82      def _build_graph(self) -> StateGraph:
83          """Construct the research workflow graph"""
84          workflow = StateGraph(ResearchState)
85
86          # Add nodes
87          workflow.add_node("lead_researcher", self.lead_researcher_node)
88          workflow.add_node("spawn_subtasks", self.spawn_subtasks_node)
89          workflow.add_node("search_agent", self.search_agent_node)
90          workflow.add_node("analysis_agent", self.analysis_agent_node)
91          workflow.add_node("synthesis_agent", self.synthesis_agent_node)
92
93          # Define workflow
94          workflow.set_entry_point("lead_researcher")
95          workflow.add_edge("lead_researcher", "spawn_subtasks")
96          workflow.add_edge("spawn_subtasks", "search_agent")
97          workflow.add_conditional_edges(
98              "search_agent",
99              self.should_continue_searching,
100             {
101                 "continue": "search_agent",
102                 "analyze": "analysis_agent"
103             }
104         )
105         workflow.add_edge("analysis_agent", "synthesis_agent")
106         workflow.add_edge("synthesis_agent", END)
107
108         return workflow.compile()
109
110     def lead_researcher_node(self, state: ResearchState) -> ResearchState:
111         """Lead researcher develops the overall research strategy"""
112         print("Lead Researcher: Developing research strategy...")
113
114         prompt = f"""Develop a research strategy for: {state['question']}
115
116         Create 3-4 research directions and key questions to investigate.
117         Focus on quality over quantity.
118
119         Return JSON:
120         {{
121             "research_directions": ["direction1", "direction2", "direction3
                "],
122             "key_questions": ["question1", "question2", "question3"],
123             "data_requirements": ["data1", "data2", "data3"]
124         }}
```

```
        """

        response = self.lead_llm.invoke([SystemMessage(content=prompt)])
        research_plan = json.loads(response.content)

        state['research_plan'] = research_plan
        state['messages'].append(HumanMessage(content=f"Research plan: {
            research_plan}"))

        print(f"Research strategy developed with {len(research_plan['
            research_directions'])} directions")

        return state

    def spawn_subtasks_node(self, state: ResearchState) -> ResearchState:
        """Create specific subtasks based on the research plan"""
        print("Creating research subtasks...")

        subtasks = []
        directions = state['research_plan']['research_directions']
        questions = state['research_plan']['key_questions']

        # Diagonal pairing: each direction with corresponding question
        for i, direction in enumerate(directions):
            if i < len(questions):
                subtask = {
                    "direction": direction,
                    "question": questions[i],
                    "status": "pending",
                    "search_queries": self._generate_search_queries(
                        direction, questions[i])
                }
                subtasks.append(subtask)

        # Add cross-cutting combinations for depth
        if len(directions) >= 2 and len(questions) >= 2:
            subtasks.extend([
                {
                    "direction": directions[0],
                    "question": questions[1],
                    "status": "pending",
                    "search_queries": self._generate_search_queries(
                        directions[0], questions[1])
                },
                {
                    "direction": directions[1],
                    "question": questions[0],
                    "status": "pending",
                    "search_queries": self._generate_search_queries(
                        directions[1], questions[0])
                }
            ])

        state['subtasks'] = subtasks
        state['current_subtask'] = 0
```

```
175         state['search_results'] = []
176         state['analysis_results'] = []
177 
178         print(f"Created {len(subtasks)} focused research subtasks")
179         total_searches = sum(len(subtask['search_queries']) for subtask in
                subtasks)
180         print(f"  Will execute ~{total_searches} searches across all
                subtasks")
181 
182         return state
183 
184     def _generate_search_queries(self, direction: str, question: str) ->
            List[str]:
185         """Generate specific search queries for a subtask"""
186         prompt = f"""Generate 2-3 search queries for:
187         Direction: {direction}
188         Question: {question}
189 
190         Focus on recent studies, data, and expert analyses.
191         Return JSON: {{"queries": ["query1", "query2", "query3"]}}
192         """
193 
194         response = self.lead_llm.invoke([SystemMessage(content=prompt)])
195         return json.loads(response.content)['queries'][:3]
196 
197     def search_agent_node(self, state: ResearchState) -> ResearchState:
198         """Execute parallel searches for current batch of subtasks"""
199         current_idx = state['current_subtask']
200         batch_size = 3  # Process 3 subtasks per graph iteration (not a
                parallelism limit)
201         end_idx = min(current_idx + batch_size, len(state['subtasks']))
202 
203         if current_idx < len(state['subtasks']):
204             print(f"Search Agent: Processing subtasks {current_idx + 1}-{
                    end_idx}/{len(state['subtasks'])}")
205 
206             # Execute all searches for this batch in parallel
207             all_search_futures = []
208             for idx in range(current_idx, end_idx):
209                 subtask = state['subtasks'][idx]
210                 for query in subtask['search_queries']:
211                     future = self.executor.submit(self._perform_search,
                            query)
212                     all_search_futures.append(future)
213 
214             # Collect results
215             for future in all_search_futures:
216                 state['search_results'].extend(future.result())
217 
218             # Update progress
219             for idx in range(current_idx, end_idx):
220                 state['subtasks'][idx]['status'] = 'searched'
221             state['current_subtask'] = end_idx
222 
223             print(f"  Batch complete: collected {len(all_search_futures)}
```

```
                search results")
224
225         return state
226
227     def _perform_search(self, query: str) -> List[Dict]:
228         """Perform a single search using Tavily API"""
229         try:
230             results = self.tavily.search(
231                 query=query,
232                 search_depth="basic",  # Faster for pedagogical example
233                 max_results=3,  # Reduced for focused results
234                 include_domains=None,  # Search all domains
235                 include_answer=True,  # Include AI-generated answer
236                 include_raw_content=False,  # We want processed content
237                 include_images=False
238             )
239             search_results = [{
240                 "query": query,
241                 "url": r.get("url", ""),
242                 "title": r.get("title", ""),
243                 "content": r.get("content", ""),  # Full content, no
                        truncation
244                 "score": r.get("score", 0)
245             } for r in results.get("results", [])]
246
247             # Add Tavily's AI-generated answer if available
248             if results.get("answer"):
249                 search_results.insert(0, {
250                     "query": query,
251                     "url": "Tavily AI Summary",
252                     "title": "AI-Generated Summary",
253                     "content": results["answer"],
254                     "score": 1.0
255                 })
256
257             return search_results
258         except Exception as e:
259             print(f"Search error for '{query}': {e}")
260             return []
261
262     def should_continue_searching(self, state: ResearchState) -> Literal["
        continue", "analyze"]:
263         """Decide whether to continue searching or move to analysis"""
264         if state['current_subtask'] < len(state['subtasks']):
265             return "continue"
266         return "analyze"
267
268     def analysis_agent_node(self, state: ResearchState) -> ResearchState:
269         """Analyze search results for each subtask in parallel"""
270         print("Analysis Agents: Analyzing search results...")
271
272         analysis_futures = []
273
274         for subtask in state['subtasks']:
275             if subtask['status'] == 'searched':
```

```
                relevant_results = [r for r in state['search_results']
                                    if r['query'] in subtask['search_queries']]

                future = self.executor.submit(
                    self._analyze_subtask_results,
                    subtask,
                    relevant_results
                )
                analysis_futures.append((subtask, future))

        # Collect analysis results
        for subtask, future in analysis_futures:
            state['analysis_results'].append({
                "subtask": subtask,
                "analysis": future.result()
            })

        print(f"Completed analysis of {len(state['analysis_results'])} subtasks")

        return state

    def _analyze_subtask_results(self, subtask: Dict, results: List[Dict]) -> str:
        """Analyze results for a specific subtask"""
        prompt = f"""Analyze these search results:

        Direction: {subtask['direction']}
        Question: {subtask['question']}
        Results: {json.dumps(results, indent=2)}

        Provide a comprehensive analysis (300-500 words) covering:
        - Key findings and evidence
        - Data and statistics
        - Different perspectives
        - Practical implications

        Cite sources as [Title, URL].
        """

        response = self.analysis_llm.invoke([SystemMessage(content=prompt)])
        return response.content

    def synthesis_agent_node(self, state: ResearchState) -> ResearchState:
        """Synthesize all analyses into a comprehensive report"""
        print("Synthesis Agent: Creating final report...")

        prompt = f"""Synthesize the research findings into a comprehensive report.

        Question: {state['question']}

        Research Plan: {json.dumps(state['research_plan'], indent=2)}
```

```
326
327         Analysis Results: {json.dumps(state['analysis_results'], indent=2)}
328
329         Write a well-structured research report (3-5 pages) that includes:
330         - Executive summary with key findings
331         - Analysis of each research direction
332         - Synthesis of insights across findings
333         - Recommendations and future research areas
334
335         Use citations in [Source, Year] format. Make it suitable for
                executive audiences.
336         """
337
338         response = self.lead_llm.invoke([SystemMessage(content=prompt)])
339         state['final_report'] = response.content
340
341         print("Research report completed!")
342
343         return state
344
345     async def research(self, question: str) -> str:
346         """Execute the deep research process"""
347         initial_state = {
348             "question": question,
349             "research_plan": {},
350             "subtasks": [],
351             "search_results": [],
352             "analysis_results": [],
353             "final_report": "",
354             "messages": [],
355             "current_subtask": 0,
356             "max_iterations": 10
357         }
358
359         # Run the graph with increased recursion limit
360         config = {"recursion_limit": 50}
361         final_state = await self.graph.ainvoke(initial_state, config=config
                )
362
363         return final_state['final_report']
364
365     def shutdown(self):
366         """Clean up resources"""
367         self.executor.shutdown(wait=True)
368
369
370 # Example usage
371 if __name__ == "__main__":
372     # Initialize and run research
373     agent = DeepResearchAgent()
374     question = "What are the labor market effects of transformative AI
            expected to be?"
375
376     print(f"Starting deep research on: {question}")
377     print("-" * 80)
```

```
    report = asyncio.run(agent.research(question))

    print("\n" + "="*80)
    print("RESEARCH REPORT")
    print("="*80)
    print(report)

    # Clean up resources
    agent.shutdown()
```

REFERENCES

Anthropic. 2024. "Introducing the Model Context Protocol." *Anthropic Blog*. https://www.anthropic.com/news/model-context-protocol. Published: Nov 25, 2024; Accessed: Jun 30, 2025.

Anthropic. 2025*a*. "Anthropic Economic Index: AI's Impact on Software Development." *Anthropic Blog*. https://www.anthropic.com/research/impact-software-development. Published: Apr 28, 2025; Accessed: Jul 27, 2025.

Anthropic. 2025*b*. "How Anthropic Teams Use Claude Code." *Anthropic Blog*. https://www.anthropic.com/news/how-anthropic-teams-use-claude-code. Published: Jul 24, 2025; Accessed: Jul 27, 2025.

Castelvecchi, Davide. 2025. "DeepMind and OpenAI Achieve Gold-Medal Level at the International Mathematical Olympiad." *Nature (news)*. https://www.nature.com/articles/d41586-025-02343-x. Published: 24 July 2025; Accessed: 26 August 2025.

Chiang, Wei-Lin, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. "Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference." https://doi.org/10.48550/arXiv.2403.04132.

Chiou, Lyndie, and Clara Moskowitz. 2025. "At Secret Math Meeting, Researchers Struggle to Outsmart AI." *Scientific American*. https://www.scientificamerican.com/article/inside-the-secret-meeting-where-mathematicians-struggled-to-outsmart-ai/. Published: June 6, 2025; Accessed: June 30, 2025.

Gans, Joshua. 2025. "Can Author Manipulation of AI Referees be Welfare Improving?" *NBER Working Paper*, w34082. https://www.nber.org/papers/w34082.

Hadfield, Gillian K., and Andrew Koh. 2025. "An Economy of AI Agents." *NBER Working Paper*. Forthcoming.

Hendrycks, Dan. 2025. *Introduction to AI Safety, Ethics, and Society*. Boca Raton, FL:Routledge. https://www.routledge.com/Introduction-to-AI-Safety-Ethics-and-Society/Hendrycks/p/book/9781032869926.

Kahneman, Daniel. 2011. *Thinking, fast and slow*. New York:Farrar, Straus and Giroux.

Korinek, Anton. 2023. "Generative AI for Economic Research: Use Cases and Implications for Economists." *Journal of Economic Literature*, 61(4): 1281–1317.

Korinek, Anton. 2024. "LLMs learn to collaborate and reason: December 2024 Update of Generative AI for Economic Research: Use Cases and Implications for Economists." *Journal of Economic Literature 61(4)*. https://www.aeaweb.org/articles?id=10.1257/jel.20231736.

Kwa, Thomas, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. 2025. "Measuring AI Ability to Complete Long Tasks." *arXiv:2503.14499*. https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/.

LangChain. 2025. "Open Deep Research." *LangChain Blog*. https://blog.langchain.com/open-deep-research/. Accessed Jun 30, 2025.

Model Context Protocol Working Group. 2025. "Model Context Protocol: Specification." *Version 2025-06-18*. https://modelcontextprotocol.io/specification/2025-06-18. Published: Jun 18, 2025; Accessed: Jun 30, 2025.

OpenAI. 2025. "Introducing Deep Research." *OpenAI Blog*. https://openai.com/index/introducing-deep-research/. Published Feb 2, 2025; Accessed Jun 30, 2025.

Palazzolo, Stephanie, and Cory Weinberg. 2025. "OpenAI Plots Charging $20,000 a Month for PhD-Level Agents." *The Information*. https://www.theinformation.com/articles/openai-plots-charging-20-000-a-month-for-phd-level-agents. Published: Mar 5, 2025; Accessed: Jun 30, 2025.

Rein, David, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. "GPQA: A Graduate-Level Google-Proof Q&A Benchmark." *arXiv:2311.12022*. https://arxiv.org/abs/2311.12022.

Rusak, Gili, Peyman Shahidi, Ben Manning, Andrey Fradkin, and John Horton. 2025. "AI Agents as Economic Agents." *NBER Working Paper*. Forthcoming.

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. "ReAct: Synergizing Reasoning and Acting in Language Models." *arXiv:2210.03629*. https://doi.org/10.48550/arXiv.2210.03629.