**AI Agents: The Intersection of Tool Calling and Reasoning in Generative AI**

**Unpacking problem solving and tool-driven decision making in AI**

[Tula Masterman](https://medium.com/@tula.masterman?source=post_page-----ff268eece443--------------------------------)

Oct 5, 2024

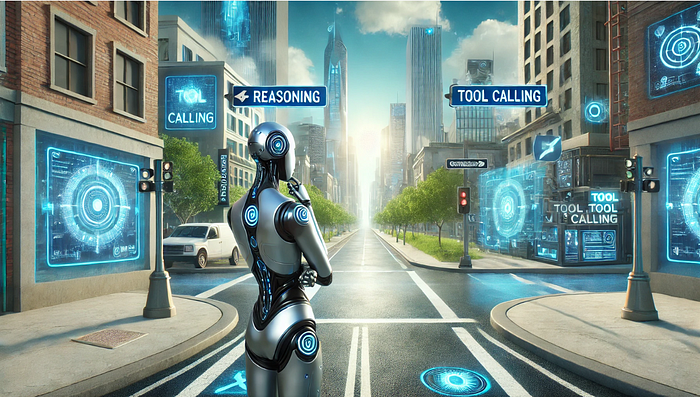


Image by Author and GPT-4o depicting an AI agent at the intersection of reasoning and tool calling

**Introduction: The Rise of Agentic AI**

Today, new libraries and low-code platforms are making it easier than ever to build AI agents, also referred to as digital workers. Tool calling is one of the primary abilities driving the “agentic” nature of Generative AI models by extending their ability beyond conversational tasks. By executing tools (functions), agents can take action on your behalf and solve complex, multi-step problems that require robust decision making and interacting with a variety of external data sources.

This article focuses on how reasoning is expressed through tool calling, explores some of the challenges of tool use, covers common ways to evaluate tool-calling ability, and provides examples of how different models and agents interact with tools.

**Expressions of Reasoning to Solve Problems**

At the core of successful agents lie two key expressions of reasoning: **reasoning through evaluation and planning** and **reasoning through tool use**.

* **Reasoning through evaluation and planning** relates to an agent’s ability to effectively breakdown a problem by iteratively planning, assessing progress, and adjusting its approach until the task is completed. Techniques like [Chain-of-Thought](https://arxiv.org/abs/2201.11903) (CoT), [ReAct](https://arxiv.org/abs/2210.03629), and [Prompt Decomposition](https://arxiv.org/abs/2210.02406) are all patterns designed to improve the model’s ability to reason strategically by breaking down tasks to solve them correctly. This type of reasoning is more macro-level, ensuring the task is completed correctly by working iteratively and taking into account the results from each stage.
* **Reasoning through tool use** relates to the agents ability to effectively interact with it’s environment, deciding which tools to call and how to structure each call. These tools enable the agent to retrieve data, execute code, call APIs, and more. The strength of this type of reasoning lies in the proper execution of tool calls rather than reflecting on the results from the call.

While both expressions of reasoning are important, they don’t always need to be combined to create powerful solutions. For example, **OpenAI’s new** **o1 model excels at reasoning through evaluation and planning** because it was trained to reason using chain of thought. This has significantly improved its ability to think through and solve complex challenges as reflected on a variety of benchmarks. For example, the o1 model has been shown to **surpass human PhD-level accuracy on the GPQA benchmark** covering physics, biology, and chemistry, and scored in the **86th-93rd percentile on Codeforces** contests. While o1’s reasoning ability could be used to generate text-based responses that suggest tools based on their descriptions, it currently lacks explicit tool calling abilities (at least for now!).

In contrast, **many models are fine-tuned specifically for reasoning through tool use** enabling them to generate function calls and interact with APIs very effectively. These models are focused on calling the right tool in the right format at the right time, but are typically not designed to evaluate their own results as thoroughly as o1 might. The [**Berkeley Function Calling Leaderboard**](https://gorilla.cs.berkeley.edu/leaderboard.html) **(BFCL) is a great resource for comparing how different models perform on function calling tasks**. It also provides an **evaluation suite to compare your own fine-tuned model** on various challenging tool calling tasks. In fact, the [latest dataset, BFCL v3](https://huggingface.co/datasets/gorilla-llm/Berkeley-Function-Calling-Leaderboard), was just released and now includes [multi-step, multi-turn function calling](https://gorilla.cs.berkeley.edu/blogs/13_bfcl_v3_multi_turn.html), further raising the bar for tool based reasoning tasks.

Both types of reasoning are powerful independently, and when combined, they have the potential to create agents that can effectively breakdown complicated tasks and autonomously interact with their environment. For more examples of AI agent architectures for reasoning, planning, and tool calling [check out my team’s survey paper on ArXiv](https://arxiv.org/abs/2404.11584).

**Challenges with Tool-Calling: Navigating Complex Agent Behaviors**

Building robust and reliable agents requires overcoming many different challenges. When solving complex problems, an agent often needs to balance multiple tasks at once including planning, interacting with the right tools at the right time, formatting tool calls properly, remembering outputs from previous steps, avoiding repetitive loops, and adhering to guidance to protect the system from jailbreaks/prompt injections/etc.

**Too many demands can easily overwhelm a single agent, leading to a growing trend where what may appear to an end user as one agent, is behind the scenes a collection of many agents and prompts working together to divide and conquer completing the task**. This division allows tasks to be broken down and handled in parallel by different models and agents tailored to solve that particular piece of the puzzle.

It’s here that models with excellent tool calling capabilities come into play. While tool-calling is a powerful way to enable productive agents, it comes with its own set of challenges. Agents need to understand the available tools, select the right one from a set of potentially similar options, format the inputs accurately, call tools in the right order, and potentially integrate feedback or instructions from other agents or humans. Many models are fine-tuned specifically for tool calling, allowing them to specialize in selecting functions at the right time with high accuracy.

**Some of the key considerations when fine-tuning a model for tool calling include:**

1. **Proper Tool Selection**: The model needs to understand the relationship between available tools, make nested calls when applicable, and select the right tool in the presence of other similar tools.
2. **Handling Structural Challenges**: Although most models use JSON format for tool calling, other formats like YAML or XML can also be used. Consider whether the model needs to generalize across formats or if it should only use one. Regardless of the format, the model needs to include the appropriate parameters for each tool call, potentially using results from a previous call in subsequent ones.
3. **Ensuring Dataset Diversity and Robust Evaluations**: The dataset used should be diverse and cover the complexity of multi-step, multi-turn function calling. Proper evaluations should be performed to prevent overfitting and avoid benchmark contamination.

**Common Benchmarks to Evaluate Tool-Calling**

With the growing importance of tool use in language models, many datasets have emerged to help evaluate and improve model tool-calling capabilities. Two of the most popular benchmarks today are the Berkeley Function Calling Leaderboard and Nexus Function Calling Benchmark, both of which [Meta used to evaluate the performance of their Llama 3.1 model series](https://arxiv.org/pdf/2407.21783). A recent paper, [ToolACE](https://arxiv.org/abs/2409.00920), demonstrates how agents can be used to create a diverse dataset for fine-tuning and evaluating model tool use.

Let’s explore each of these benchmarks in more detail:

* **Berkeley Function Calling Leaderboard (**[**BFCL**](https://gorilla.cs.berkeley.edu/leaderboard.html)**):** BFCL contains 2,000 question-function-answer pairs across multiple programming languages. Today there are 3 versions of the BFCL dataset each with enhancements to better reflect real-world scenarios. For example, [BFCL-V2](https://gorilla.cs.berkeley.edu/blogs/12_bfcl_v2_live.html), released August 19th, 2024 includes user contributed samples designed to address evaluation challenges related to dataset contamination. [BFCL-V3](https://gorilla.cs.berkeley.edu/blogs/13_bfcl_v3_multi_turn.html) released September 19th, 2024 adds multi-turn, multi-step tool calling to the benchmark. This is critical for agentic applications where a model needs to make multiple tool calls over time to successfully complete a task. Instructions for e[valuating models on BFCL can be found on GitHub](https://github.com/ShishirPatil/gorilla), with the [latest dataset available on HuggingFace](https://huggingface.co/datasets/gorilla-llm/Berkeley-Function-Calling-Leaderboard), and the [current leaderboard accessible here](https://gorilla.cs.berkeley.edu/leaderboard.html). The Berkeley team has also released various versions of their Gorilla Open-Functions model fine-tuned specifically for function-calling tasks.
* **Nexus Function Calling Benchmark:** This benchmark evaluates models on zero-shot function calling and API usage across nine different tasks classified into three major categories for single, parallel, and nested tool calls. Nexusflow released NexusRaven-V2, a model designed for function-calling. The [Nexus benchmark is available on GitHub](https://github.com/nexusflowai/NexusRaven-V2/tree/master#benchmarks) and the corresponding [leaderboard is on HuggingFace](https://huggingface.co/spaces/Nexusflow/Nexus_Function_Calling_Leaderboard).
* **ToolACE:** The [ToolACE paper](https://arxiv.org/pdf/2409.00920) demonstrates a creative approach to overcoming challenges related to collecting real-world data for function-calling. The research team created an agentic pipeline to generate a synthetic dataset for tool calling consisting of over 26,000 different APIs. The dataset includes examples of single, parallel, and nested tool calls, as well as non-tool based interactions, and supports both single and multi-turn dialogs. The team released a fine-tuned version of Llama-3.1–8B-Instruct, [ToolACE-8B](https://huggingface.co/Team-ACE/ToolACE-8B), designed to handle these complex tool-calling related tasks. A [subset of the ToolACE dataset is available on HuggingFace](https://huggingface.co/datasets/Team-ACE/ToolACE).

Each of these benchmarks facilitates our ability to evaluate model reasoning expressed through tool calling. These benchmarks and fine-tuned models reflect a growing trend towards developing more specialized models for specific tasks and increasing LLM capabilities by extending their ability to interact with the real-world.

**Examples of Tool-Calling in Action**

If you’re interested in exploring tool-calling in action, here are some examples to get you started organized by ease of use, ranging from simple built-in tools to using fine-tuned models, and agents with tool-calling abilities.

**Level 1 — ChatGPT**: The best place to start and see tool-calling live without needing to define any tools yourself, is through ChatGPT. Here you can use GPT-4o through the chat interface to call and execute tools for web-browsing. For example, when asked “what’s the latest AI news this week?” ChatGPT-4o will conduct a web search and return a response based on the information it finds. *Remember the new o1 model does not have tool-calling abilities yet and cannot search the web.*

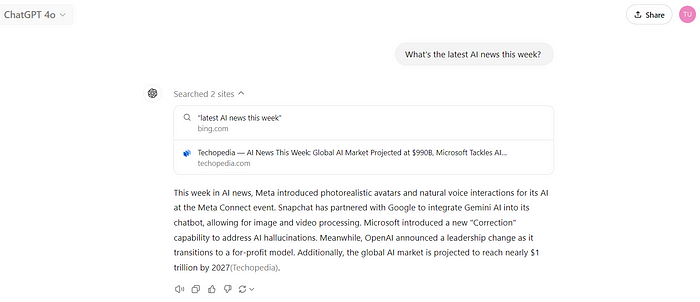


Image by author 9/30/24

While this built-in web-searching feature is convenient, most use cases will require defining custom tools that can integrate directly into your own model workflows and applications. This brings us to the next level of complexity.

**Level 2 — Using a Model with Tool Calling Abilities and Defining Custom Tools**:

This level involves using a model with tool-calling abilities to get a sense of how effectively the model selects and uses it’s tools. It’s important to note that **when a model is trained for tool-calling, it only generates the text or code for the tool call, it does not actually execute the code itself. Something external to the model needs to invoke the tool, and it’s at this point — where we’re combining generation with execution — that we transition from language model capabilities to agentic systems.**

To get a sense for how models express tool calls we can turn towards the Databricks Playground. For example, we can select the model Llama 3.1 405B and give it access to the sample tools get\_distance\_between\_locations and get\_current\_weather. When prompted with the user message “I am going on a trip from LA to New York how far are these two cities? And what’s the weather like in New York? I want to be prepared for when I get there” the model decides which tools to call and what parameters to pass so it can effectively reply to the user.



Image by author 10/2/2024 depicting using the Databricks Playground for sample tool calling

In this example, the model suggests two tool calls. Since the model cannot execute the tools, the user needs to fill in a sample result to simulate the tool output (e.g., “2500” for the distance and “68” for the weather). The model then uses these simulated outputs to reply to the user.

This approach to using the Databricks Playground allows you to observe how the model uses custom defined tools and is a great way to test your function definitions before implementing them in your tool-calling enabled applications or agents.

Outside of the Databricks Playground, we can observe and evaluate how effectively different models available on platforms like HuggingFace use tools through code directly. For example, we can load different models like Llama 3.2–3B-Instruct, ToolACE-8B, NexusRaven-V2–13B, and more from HuggingFace, give them the same system prompt, tools, and user message then observe and compare the tool calls each model returns. This is a great way to understand how well different models reason about using custom-defined tools and can help you determine which tool-calling models are best suited for your applications.

Here is an example demonstrating a tool call generated by Llama-3.2–3B-Instruct based on the following tool definitions and user message, the same steps could be followed for other models to compare generated tool calls.

import torch  
from transformers import pipeline  
  
function\_definitions = """[  
 {  
 "name": "search\_google",  
 "description": "Performs a Google search for a given query and returns the top results.",  
 "parameters": {  
 "type": "dict",  
 "required": [  
 "query"  
 ],  
 "properties": {  
 "query": {  
 "type": "string",  
 "description": "The search query to be used for the Google search."  
 },  
 "num\_results": {  
 "type": "integer",  
 "description": "The number of search results to return.",  
 "default": 10  
 }  
 }  
 }  
 },  
 {  
 "name": "send\_email",  
 "description": "Sends an email to a specified recipient.",  
 "parameters": {  
 "type": "dict",  
 "required": [  
 "recipient\_email",  
 "subject",  
 "message"  
 ],  
 "properties": {  
 "recipient\_email": {  
 "type": "string",  
 "description": "The email address of the recipient."  
 },  
 "subject": {  
 "type": "string",  
 "description": "The subject of the email."  
 },  
 "message": {  
 "type": "string",  
 "description": "The body of the email."  
 }  
 }  
 }  
 }  
]  
"""  
  
# This is the suggested system prompt from Meta  
system\_prompt = """You are an expert in composing functions. You are given a question and a set of possible functions.   
Based on the question, you will need to make one or more function/tool calls to achieve the purpose.   
If none of the function can be used, point it out. If the given question lacks the parameters required by the function,  
also point it out. You should only return the function call in tools call sections.  
  
If you decide to invoke any of the function(s), you MUST put it in the format of [func\_name1(params\_name1=params\_value1, params\_name2=params\_value2...), func\_name2(params)]\n  
You SHOULD NOT include any other text in the response.  
  
Here is a list of functions in JSON format that you can invoke.\n\n{functions}\n""".format(functions=function\_definitions)



Image by author sample output demonstrating generated tool call from Llama 3.2–3B-Instruct

From here we can move to Level 3 where we’re defining Agents that execute the tool-calls generated by the language model.

**Level 3 Agents (invoking/executing LLM tool-calls)**: Agents often express reasoning both through planning and execution as well as tool calling making them an increasingly important aspect of AI based applications. Using libraries like LangGraph, AutoGen, Semantic Kernel, or LlamaIndex, you can quickly create an agent using models like GPT-4o or Llama 3.1–405B which support both conversations with the user and tool execution.

Check out these guides for some exciting examples of agents in action:

* LangGraph: [Local RAG Agent with Llama 3](https://langchain-ai.github.io/langgraph/tutorials/rag/langgraph_adaptive_rag_local/)
* AutoGen: [Solve Tasks Requiring Web Info](https://github.com/microsoft/autogen/blob/main/notebook/agentchat_web_info.ipynb)
* Semantic Kernel: [Getting Started with Agents in Semantic Kernel](https://github.com/microsoft/semantic-kernel/blob/main/dotnet/samples/GettingStartedWithAgents/README.md)
* LlamaIndex: [Agent Usage Pattern Documentation](https://docs.llamaindex.ai/en/stable/module_guides/deploying/agents/usage_pattern/)

**Conclusion:**

The future of agentic systems will be driven by models with strong reasoning abilities enabling them to effectively interact with their environment. As the field evolves, I expect we will continue to see a proliferation of smaller, specialized models focused on specific tasks like tool-calling and planning.

It’s important to consider the current limitations of model sizes when building agents. For example, according to the [Llama 3.1 model card](https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_1), the Llama 3.1–8B model is not reliable for tasks that involve both maintaining a conversation and calling tools. Instead, larger models with 70B+ parameters should be used for these types of tasks. This alongside other emerging research for fine-tuning small language models suggests that smaller models may serve best as specialized tool-callers while larger models may be better for more advanced reasoning. By combining these abilities, we can build increasingly effective agents that provide a seamless user experience and allow people to leverage these reasoning abilities in both professional and personal endeavors.

**Agentes de IA: La Intersección entre la Llamada a Herramientas y el Razonamiento en la IA Generativa**  
*Desentrañando la resolución de problemas y la toma de decisiones impulsada por herramientas en la IA*  
*Tula Masterman*  
*5 de octubre de 2024*

**Introducción: El Auge de la IA Agente**  
Hoy en día, nuevas bibliotecas y plataformas de bajo código facilitan más que nunca la creación de agentes de IA, también llamados trabajadores digitales. La llamada a herramientas es una de las habilidades principales que impulsa la naturaleza “agente” de los modelos de IA generativa, extendiendo su capacidad más allá de las tareas de conversación. Al ejecutar herramientas (funciones), los agentes pueden actuar en tu nombre y resolver problemas complejos de varios pasos que requieren toma de decisiones robusta e interacción con diversas fuentes de datos externas.

Este artículo explora cómo el razonamiento se expresa a través de la llamada a herramientas, analiza algunos de los desafíos asociados con el uso de herramientas, revisa formas comunes de evaluar la capacidad de llamada a herramientas y brinda ejemplos de cómo diferentes modelos y agentes interactúan con herramientas.

**Expresiones de Razonamiento para Resolver Problemas**  
En el núcleo de los agentes exitosos se encuentran dos expresiones clave de razonamiento: razonamiento mediante evaluación y planificación, y razonamiento a través del uso de herramientas.

* *Razonamiento mediante evaluación y planificación* se refiere a la capacidad de un agente para descomponer efectivamente un problema mediante la planificación iterativa, evaluación de progreso y ajuste de enfoque hasta completar la tarea. Técnicas como *Chain-of-Thought* (CoT), *ReAct* y *Descomposición de Prompts* son patrones diseñados para mejorar la capacidad del modelo de razonar estratégicamente descomponiendo tareas para resolverlas correctamente. Este tipo de razonamiento es a nivel macro, asegurando que la tarea se complete correctamente trabajando de manera iterativa y considerando los resultados de cada etapa.
* *Razonamiento mediante uso de herramientas* se refiere a la habilidad del agente para interactuar efectivamente con su entorno, decidiendo qué herramientas llamar y cómo estructurar cada llamada. Estas herramientas permiten al agente recuperar datos, ejecutar código, llamar a APIs y más. La fortaleza de este tipo de razonamiento radica en la ejecución correcta de las llamadas a herramientas en lugar de reflexionar sobre los resultados de la llamada.

Ambas expresiones de razonamiento son importantes, aunque no siempre es necesario combinarlas para crear soluciones potentes. Por ejemplo, el nuevo modelo o1 de OpenAI destaca en razonamiento mediante evaluación y planificación, gracias a que fue entrenado para razonar utilizando *chain of thought*. Esto ha mejorado significativamente su capacidad para pensar y resolver desafíos complejos, como lo reflejan diversas pruebas de referencia. Sin embargo, actualmente carece de habilidades explícitas de llamada a herramientas (¡al menos por ahora!).

En contraste, muchos modelos están afinados específicamente para el razonamiento mediante uso de herramientas, lo que les permite generar llamadas a funciones e interactuar con APIs de manera muy efectiva. Estos modelos se centran en llamar a la herramienta adecuada en el formato correcto en el momento adecuado, aunque normalmente no están diseñados para evaluar sus propios resultados tan minuciosamente como podría hacerlo el modelo o1.

**Desafíos con la Llamada a Herramientas: Navegando Comportamientos Complejos de Agentes**  
Construir agentes robustos y confiables requiere superar varios desafíos. Al resolver problemas complejos, un agente a menudo necesita equilibrar múltiples tareas a la vez, incluyendo planificación, interacción con las herramientas adecuadas en el momento adecuado, formateo correcto de las llamadas a herramientas, recordación de salidas de pasos previos, evitar bucles repetitivos y adherirse a directrices para proteger el sistema de inyecciones de prompt o de “jailbreaks”.

Es aquí donde los modelos con excelentes capacidades de llamada a herramientas entran en juego. Aunque la llamada a herramientas es una forma poderosa de habilitar agentes productivos, viene con su propio conjunto de desafíos. Los agentes deben comprender las herramientas disponibles, seleccionar la correcta entre un conjunto de opciones potencialmente similares, formatear los inputs de manera precisa, llamar a herramientas en el orden correcto y, potencialmente, integrar comentarios o instrucciones de otros agentes o humanos.

**Consideraciones Clave para Afinar un Modelo en Llamada a Herramientas**

1. *Selección de Herramienta Adecuada*: El modelo debe comprender la relación entre las herramientas disponibles, hacer llamadas anidadas cuando sea aplicable y seleccionar la herramienta correcta en presencia de herramientas similares.
2. *Manejo de Desafíos Estructurales*: Aunque la mayoría de los modelos utilizan el formato JSON para la llamada a herramientas, también pueden usarse otros formatos como YAML o XML. El modelo debe incluir los parámetros apropiados para cada llamada, usando posiblemente resultados de una llamada anterior en llamadas subsecuentes.
3. *Asegurar Diversidad en el Conjunto de Datos y Evaluaciones Robustas*: El conjunto de datos utilizado debe ser diverso y cubrir la complejidad de la llamada a funciones de varios pasos y turnos. Las evaluaciones deben realizarse para evitar el sobreajuste y evitar la contaminación de benchmarks.

**Benchmarks Comunes para Evaluar la Llamada a Herramientas**  
Con la creciente importancia del uso de herramientas en modelos de lenguaje, han surgido varios conjuntos de datos para ayudar a evaluar y mejorar las capacidades de llamada a herramientas de los modelos. Dos de los benchmarks más populares hoy en día son el *Berkeley Function Calling Leaderboard* (BFCL) y el *Nexus Function Calling Benchmark*. Un artículo reciente, *ToolACE*, demuestra cómo los agentes pueden usarse para crear un conjunto de datos diverso para la afinación y evaluación de modelos en el uso de herramientas.

* *Berkeley Function Calling Leaderboard (BFCL)*: Incluye pares de pregunta-función-respuesta en múltiples lenguajes de programación. Hoy en día existen tres versiones del conjunto de datos BFCL, cada una con mejoras para reflejar mejor escenarios del mundo real.
* *Nexus Function Calling Benchmark*: Evalúa modelos en llamadas a funciones y uso de API en nueve tareas diferentes clasificadas en tres categorías principales: llamadas a herramientas individuales, paralelas y anidadas.
* *ToolACE*: El artículo demuestra un enfoque creativo para superar desafíos relacionados con la recolección de datos reales para llamada a funciones, utilizando una tubería agente para generar un conjunto de datos sintético para llamada a herramientas que incluye más de 26,000 APIs.

**Ejemplos de Llamada a Herramientas en Acción**  
A continuación se presentan algunos ejemplos organizados por nivel de complejidad, desde herramientas incorporadas hasta el uso de modelos afinados y agentes con habilidades de llamada a herramientas:

* *Nivel 1 — ChatGPT*: El mejor lugar para comenzar y ver la llamada a herramientas en vivo, sin necesidad de definir herramientas tú mismo, es a través de ChatGPT. Puedes usar GPT-4o en la interfaz de chat para llamar y ejecutar herramientas para navegar por la web.
* *Nivel 2 — Uso de un Modelo con Habilidades de Llamada a Herramientas y Definición de Herramientas Personalizadas*: Este nivel implica usar un modelo con habilidades de llamada a herramientas para evaluar cuán efectivamente selecciona y usa sus herramientas. Un ejemplo es el Playground de Databricks, donde se puede observar cómo el modelo Llama 3.1–405B usa herramientas definidas en la plataforma.
* *Nivel 3 — Agentes que Ejecutan Llamadas a Herramientas de Modelos de Lenguaje*: Los agentes a menudo expresan razonamiento tanto a través de la planificación y ejecución como mediante llamada a herramientas, lo que los convierte en un aspecto cada vez más importante en las aplicaciones basadas en IA.

**Conclusión**  
El futuro de los sistemas agentes estará impulsado por modelos con fuertes habilidades de razonamiento que les permitan interactuar efectivamente con su entorno. A medida que el campo evoluciona, es probable que veamos una proliferación de modelos especializados más pequeños, enfocados en tareas específicas como la llamada a herramientas y la planificación. Al combinar estas habilidades, podemos construir agentes cada vez más efectivos que proporcionen una experiencia de usuario fluida y permitan aprovechar estas habilidades de razonamiento tanto en entornos profesionales como personales.