Democratising real-world drug discovery through agentic AI

Citation for the original published paper (version of record):
He, J., Lai, H., Saigiridharan, L. et al (2026). Democratising real-world drug discovery through agentic AI. Drug Discovery Today, 31(2). http://dx.doi.org/10.1016/j.drudis.2026.104605

Democratising real-world drug discovery through agentic AI

Jiazhen He [1,6], Helen Lai [2,6], Lakshidaa Saigiridharan [1,6], Gian Marco Ghiandoni [3,*], Kinga Jenei [1], Umur Gokalp [4], Ajša Nuković [4], Ola Engkvist [1,5], Jon Paul Janet [1], Samuel Genheden [1,*]

[1] Molecular AI, Discovery Sciences, BioPharmaceuticals R&D, AstraZeneca, Pepparedsleden 1, 431 83 Mölndal, Sweden
[2] Molecular AI, Discovery Sciences R&D, AstraZeneca, The Discovery Centre (DISC), Francis Crick Avenue, Cambridge CB2 0AA, UK
[3] Augmented DMTA Platform, Data Analytics and AI, R&D IT, AstraZeneca, The Discovery Centre (DISC), Francis Crick Avenue, Cambridge CB2 0AA, UK
[4] Digital Solutions, Discovery Sciences, BioPharmaceuticals R&D, AstraZeneca, Pepparedsleden 1, 431 83 Mölndal, Sweden
[5] Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg 412 96 Gothenburg, Sweden

Agentic systems that are based on large language models (LLMs) have emerged as promising tools in the chemistry domain over the past few years. Early examples included work on CoScientist, Chemcrow, and LLM-RDF, which showcased the potential of agentic systems to assist in chemical research, in the orchestration of cheminformatics tools, and in synthetic reaction development. Despite this, the current literature lacks examples of the real-world adoption of such systems in drug discovery. We present such an example by describing our work on an agentic system called ChatInvent, which has been integrated into the discovery pipeline at AstraZeneca to aid in molecular design and synthesis planning. We discuss how the system evolved from a proof-of-concept single agent into an extensible, robust, and scalable multi-agent architecture with a graphical user interface. We emphasize the lessons learnt and the challenges that persist as we continue to work on this project, and share our perspectives on the future of agentic systems in our domain.

Keywords: agents; AI; LLMs; drug discovery; automation

Introduction

Drug discovery is a multidisciplinary effort that aims to identify drug candidates through multiple cycles of designing compounds, making them through chemical synthesis, testing them in assays, and analysing the assay results to allow prioritisation—the so-called design-make-test-analyse (DMTA) cycle.[(p1),(p2)] Computational tools can aid in all stages of DMTA, and in the past decade, such tools have been augmented with deep learning models and artificial intelligence (AI).[(p3),(p4)]

In the past two years, large language models (LLMs) have shown potential in a wide range of applications, including drug discovery.[(p5)] For example, it has been shown that LLMs are able to extract and organise chemistry literature data,[(p6),(p7)] reason on chemical structures and reactions,[(p8),(p9)] and plan chemical experiments.[(p10)] More powerful capabilities, referred to as agentic systems, can be built by coupling LLMs to external tools and databases. In principle, these hybrid architectures can plan, act and respond independently to human interventions, providing a new set of capabilities for research and development.

Agentic systems also offer the possibility of democratising access to computational tools. Leveraging many of the tools used in drug discovery requires expert knowledge and extensive

* *Corresponding authors. Ghiandoni, G.M. (gianmarco.ghiandoni@astrazeneca.com), Genheden, S. (samuel.genheden@astrazeneca.com).*
[1] These authors contributed equally.

https://doi.org/10.1016/j.drudis.2026.104605

training. For example, structure-based generative design tools may require the preprocessing of biological targets, the preparation of configuration files, and running multiple iterations of the design to adjust the parameters to desired criteria. A natural-language-operated system, such as an agentic chat application, has the potential to increase the accessibility of these expert tools significantly, reducing the gap between computational and experimental sciences. For routine tasks, the systems have the potential to save a lot of time, freeing up time to work on more challenging problems.

Chemcrow is an example of such a system, in which 18 computational tools in the chemistry domain augment OpenAI GTP-4.[p11] The authors of Chemcrow used it to autonomously plan and execute the chemical synthesis of an insect repellent. CoScientist is another example, a multi-agent system that is capable of performing chemical research such as reaction condition optimisation using a combination of web search, code execution, and experimental automation.[p12] Tippy has been presented as a multi-agent system that is able to span the entire DMTA cycle, but few details about its implementation have been provided.[p13] Similarly, FROGENT is pitched as an orchestrator of a wide range of DMTA tools, but the literature does not contain examples of how it can be used for complex workflows.[p14] Other examples include LLM-RDF for reaction development,[p15] Cactus for various cheminformatics tasks,[p16] DrugAgent for drug discovery tasks such as property prediction,[p17] SynAsk for organic synthesis,[p18] and ChatChemTS for molecular design.[p19] Although the publications that describe the aforementioned systems showcase different uses of LLM-based agentic systems in drug discovery, there is has been no published discussion on how to bring such systems into production and into the hands of scientists.

Our experience argues that it is not straightforward to build a robust, scalable, maintainable, and extendible agentic system for drug discovery, akin to Chemcrow[p11] or Coscientist,[p12] that can orchestrate in-house tools such as Reinvent[p20] or AiZynthFinder.[p21] This challenge became even more difficult when we made the system accessible to a larger group of scientists at AstraZeneca. Herein, we describe the development of ChatInvent, our chat application and agentic system. We believe that the experience shared in this paper is valuable to the community as it shows the real-world adoption of agentic systems in pharma and the lessons we have learnt from it. We also emphasise our perspective on the rapidly developing field of agentic AI.

Development of an agentic framework

Sparked by early works such as Chemcrow and CoScientist, we decided to assess the viability of LLMs in practical drug discovery settings. To this end, we developed a proof-of-concept (PoC) agentic system with the objective of evaluating whether LLMs could orchestrate sequences of operations that are typical of real-world drug design workflows within the DMTA cycle.[p2] This included interacting with chemistry tools, managing structured data, and navigating domain-specific capabilities such as internal identifiers and annotations. Unlike some of the pipelines reported in the literature that existed at the time, our use cases required more complex, conditional logic. For example, our in-house design workflows involve several steps, including the preparation of configuration files prior to execution, the conversion of data via an application programming interface (API) or database retrieval, the scoring and selection of compounds on the basis of their predicted properties using APIs, and the prioritisation of compounds on the basis of their synthetic viability.

Architectural design of agents

A critical early question was whether to develop a bespoke agent framework, or to build from existing abstraction layers, such as the popular LangChain library,[p22] or to extend Chemcrow.[p11] We initially envisioned that a 'from-scratch' implementation would deliver speed and transparency, but we recognised that this approach would involve a high maintenance overhead and slower onboarding of new functionalities. When we started our development, Chemcrow was based on an older version of LangChain that did not expose some required functionalities. We therefore built directly on LangChain, customising its class templates to better align with our needs.

We introduced a separation of tool logic and prompt metadata: each tool was implemented as a self-contained module, with prompts maintained in external, version-controlled metadata files. The separation between code and instructions allowed the tool logic to remain agnostic to the agent implementation, whether single-agent or multi-agent, and made it easier to update prompt strategies independently of the code.

We experimented with two LangChain agent types: ReAct and Structured Chat.[p23] Both agents can invoke tools but differ in how they interpret prompts and structured outputs. The ReAct agent was found to be harder to prompt, and its implementation led to technical issues when interacting with our tools; for example, the agent would fail to submit parameters in the right format or type. The Structured Chat agent was more consistent and responsive, and hence easier to control.

Single-agent prototype

The agent prototype, named LangDMTA, was implemented as a single-agent system relying on on-premises LLMs with swappable interfaces to accommodate different LLMs (such as GPT-3.5 Turbo, GPT-4 Turbo, or GPT-4o via Azure OpenAI in Foundry Models), agent types, and frameworks. Although LangDMTA was envisioned to span the full DMTA cycle, our initial development focused on molecular design.

LangDMTA integrated several domain-specific tools and platforms, including Mol2Mol from REINVENT4 (for molecule generation and scoring),[p20] AiZynthFinder,[p21] Precedent Finder,[p24] the Predictive Insight Platform (PIP) (for scoring),[p2] as well as some in-house APIs for data retrieval. We introduced a tool to convert compound synonyms to SMILES (referred to as Synonym-to-SMILES) and an LLM-based tool to manipulate CSV files as data frames (referred to as Analyzer agent). In addition, we introduced conversational memory, enabling persistent context across user interactions within a given session and thus making the dialogue between user and agent fluid. A graphical overview of the components of the system is shown in Figure 1.

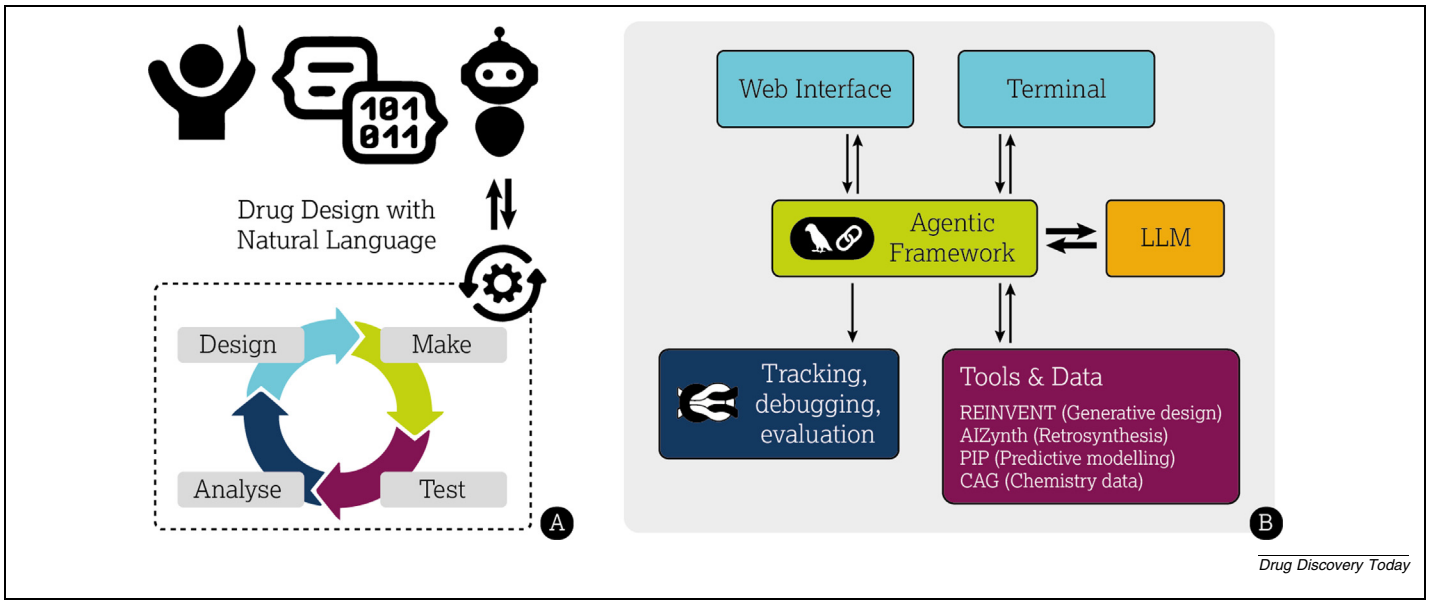

FIGURE 1

Overview of the agentic system. A. Drug design is orchestrated using natural language using a large language model (LLM)-based agentic framework. The agent autonomously plans and executes sequences that resemble looping through the design-make-test-analyse (DMTA) cycle. **B.** A high-level diagram of the components of ChatInvent. The Web interface and terminal are placed at the front end to allow interaction with the agentic framework. The framework is operated by an LLM. A set of tools and data interfaces are plugged into the agentic framework, providing access to several in-house capabilities such as Reinvent,[p20] AiZynthFinder,[p21] and PIP.[p2]

Multi-agent redesign

As we developed our framework, we realised that the single-agent implementation was not sufficiently robust and scalable. Therefore, we decided to shift to a multi-agent system, driven by both practical and architectural considerations:

- Support for more complex workflows. As workflows grow in scope and complexity, the demand for specialised tools and interfaces increases. In a single-agent framework, a monolithic agent manages all prompts and decision-making, increasing the risk of context saturation and suboptimal performance. This limitation is amplified when certain tools, such as our Analyzer agent, are inherently agent-like. A network of specialised agents better reflects the modular nature of these tools, enabling a clearer division of responsibilities.
- More efficient token consumption. In LLM-based systems, prompt tokens are a critical resource linked to latency and cost. A single-agent design often results in redundant or unnecessarily long prompts, because the agent must continuously carry and reason over a large, shared context for all tasks. By contrast, a multi-agent architecture allows each agent to operate within a specialised, minimal context, reducing token overhead while preserving task performance.
- Ease of inter-agent connectivity and integration. Within an enterprise environment such as that of AstraZeneca, a multi-agent architecture enables the easier integration of agents developed by different teams. Rather than encapsulating agents as tools within a single-agent framework, agents can be added directly to the connectivity network in a multi-agent framework. This design better supports distributed development, allowing teams to independently own, maintain, and enhance their agents while adhering to common communication standards. In addition, emerging protocols such as the model context protocol (MCP[p25]) and agent to agent (A2A[p26]) may further simplify the extension of agentic systems when incorporating new agents and tools.

We implemented the multi-agent system in LangGraph, an extension of LangChain.[p22] This architecture is centred around a supervisor agent that serves as the primary interface between the user and the agent ecosystem. Upon receiving a user query, the supervisor performs two key functions: (i) task routing, in which it identifies and dispatches the query to the most appropriate specialised agent (also referred to as a sub-agent), and (ii) task decomposition, in which it breaks down the overall request into focused task descriptions tailored to each agent's capabilities. A comparison between single and multi-agent architectures is provided in Figure 2.

By isolating task-specific instructions, individual agents receive only the information necessary for the execution of these tasks, rather than the full user query. This design choice was informed by observations made during testing: when exposed to the entire user input, agents may fail to respond appropriately, stating that they could not answer the question, despite the formulation of queries that would fit their capabilities. This behaviour was attributed to ambiguity in interpreting complex, multi-part queries. By centralising task interpretation and delegation within the supervisor, the system ensures that each sub-agent receives a clear, scoped task, and thus reliability is improved. Note that in some cases, for example when passing SMILES strings, it was beneficial to pass along the complete task description in order to avoid truncation or hallucination by the supervision agent.

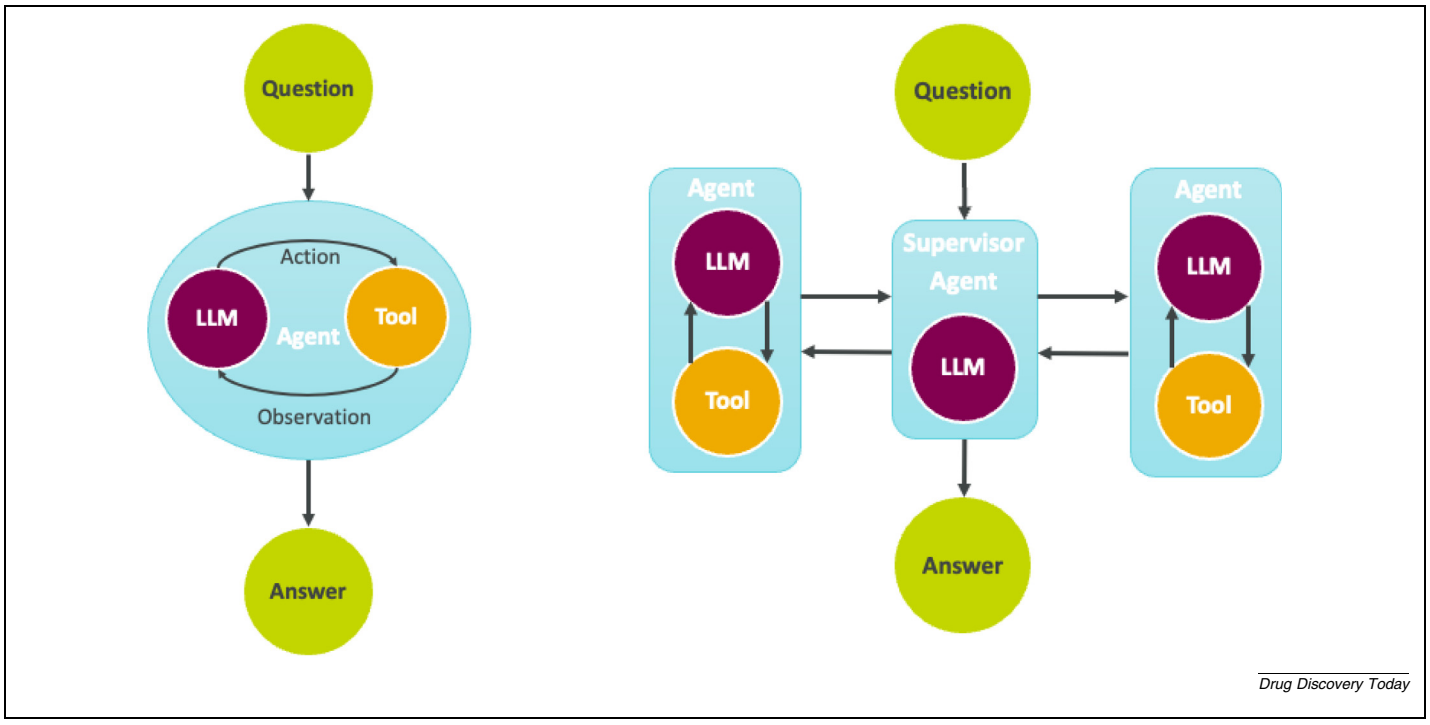

FIGURE 2

Single and multi-agent architectures. The single-agent implementation (left) shows that the query context is entirely managed within a single compartment. The multi-agent implementation (right) shows that the route from question to final answer passes through a supervisor agent that delegates work to specialised agents.

Each specialised agent within the system is designed to manage a distinct set of tasks. 'Design' focuses on molecular design and scoring using Reinvent[p20] and PIP.[p2] 'Synthesis' handles synthesis-related tasks with AiZynthFinder[p21] and Precedent Finder.[p24] The 'Utility' agent supports various molecular preprocessing tasks, including SMILES validation and the conversion of compound names or internal codes to their corresponding SMILES representations. Finally, the agent-tool Analyzer carries out data manipulation.

Common issues

To illustrate the types of challenge that we have encountered during development, we summarise ten representative errors and their solutions in Table 1. These errors are listed in the chronological order in which they were resolved. Several issues stemmed from prompt engineering: many were fixed by imposing stricter rules in agent or tool prompts, often by emphasising key constraints using capital letters or markdown formatting. Some issues disappeared entirely when upgrading to newer LLMs, highlighting the importance of evaluating new models as they become available. Other common issues could be solved by changing how messages are flowing in the multi-agent architecture and the information contained in such messages. Other problems were addressed by adjusting message flow within the multi-agent architecture, including the information carried by each message. This underscores the need for continued research on multi-agent system design. Despite these improvements, occasional agent misbehaviour remains a challenge when scaling to larger systems.[p27]

Testing and evaluation of agent performance

The answers generated by the LLMs that are integrated within agentic systems must be consistent and accurate.[p28] A challenge with LLMs is their tendency to hallucinate, which, in an agentic system, may lead to incorrect input into tools or tool selection, and made-up answers. Agentic systems present several key challenges for automated testing when compared to traditional testing frameworks.[p25] The execution of agentic workflows is inherently non-deterministic because of the use of natural language for their operation. Slight variations in the phrasing of the input prompt can have a drastic effect on the execution. The output is also returned in natural language and can vary between repeats, which can lead to inconsistent decisions. To ensure robustness and evaluate the reliability of our system, we integrated it with the testing framework LangFuse.[p29]

LangFuse allows the tracking of the agent execution, enabling step-by-step diagnosis, debugging, and programmatic evaluation of outputs from the system. LangFuse also exposes the option of having an 'LLM as a judge' for reviewing the agent execution. In this case, an external LLM can evaluate the output of an agentic system to determine whether or not it is correct.

We implemented a suite of automated tests consisting of typical questions prompted to an agent around a new or existing feature. The auto-tests reduce the risk of breaking existing capabilities when the tool logic or prompts are changed. The questions address two task categories: (1) tool-specific tasks, with each question involving at least one tool and aimed at verifying whether the corresponding agent can be correctly invoked and utilised; and (2) workflow tasks, which require the coordination

TABLE 1

Common issues encountered during agent development.

Error description	Solution
The agent attempted to solve chemistry problems (e.g., structure transformations or substructure detection) directly instead of invoking the appropriate tools.	Imposed strict prompt rules requiring that all chemistry operations be executed exclusively through the designated tools.
When building a single ReAct agent, the tools could not accept lists or dictionaries as input parameters. At the time, only the ReAct agent framework was supported by the GPT-3.5-Turbo model.	Switched to the structured chat agent framework using GPT-4 Turbo.
The agent intermittently produced "Could not parse LLM output" errors based on the latest chain-of-thought step, similar to known LangChain issues.	These errors disappeared after upgrading to GPT-4o.
In the multi-agent setting, when a query combined SMILES conversion with generation or synthesis tasks, the utility agent failed to isolate the SMILES component and instead returned control to the supervisor.	Modified the workflow so that the utility agent no longer consumes the full user query. The supervisor now provides a curated task description containing only SMILES-related operations.
The agent occasionally altered or truncated file paths returned by tools.	Enforced stricter file-path handling rules through prompt engineering.
The supervisor returned minimal final answers (e.g., stating that results were stored in a CSV at a URL without providing the file or link).	Revised the supervisor's prompts and expanded the inter-agent message schema to include file paths and returned artefacts explicitly.
Supervisor agent would truncate long SMILES when providing them as input to sub-agents.	Adopted a new message-passing strategy: some sub-agents can receive the raw user query and outputs from other agents rather than a supervisor-generated task summary.
The agent sometimes failed to supply required parameters (often dictionaries), even when explicitly requested in the tool prompt.	Added default empty values (e.g., an empty dictionary), allowing the agent to self-correct when parameters are omitted.
The agent sometimes got confused when new features and tools were added, and attempted to assume or interpret inputs to the tools.	Strengthened prompt instructions, emphasising key rules using uppercase or bold formatting to ensure strict adherence.

TABLE 2

Example questions from the test suite.[a]

Example question	Tools used
What is the SMILES string of ibuprofen?	Synonym-to-SMILES
Can you generate similar molecules to [SMILES]?	Reinvent
How many hydrogen bond donors and acceptors does [SMILES] have?	Scoring
What is the ROCS similarity of [SMILES] to the native ligand in [FILEPATH]?	Scoring
I would like to know the MW and number of rotatable bonds within the molecule: [SMILES], [SMILES]	Scoring
Filter the molecules in the file smiles_test_filter.csv based on the criteria of MW between 550 and 650, hydrogen bond donors (HBD) maximum 4	Analyzer
How many compounds have MW between 500 and 600, HBD < 3 in smiles_test_filter.csv?	Analyzer
Plot the correlation of different properties in the file [FILEPATH]	Analyzer
How can I make each of these molecules: [SMILES], [SMILES]	AiZynthFinder
What are similar reactions to Cl.c1ccccc1 ≫ Clc1ccccc1?	Precedent Finder
Can you generate molecules similar to Gleevec but with lower logD?	Synonym-to-SMILES, Reinvent, Scoring
Generate molecules similar to [SMILES], while make sure they have a similar shape to the native ligand in structure [PDB ID]. Show me the ones with similarity greater than 0.85	Reinvent, Scoring, Analyzer
Can you generate molecules similar to [COMPOUND ID] with lower hydrogen bond acceptor (HBA) number, higher MW, logD between 1 and 3, and number of rotatable bonds ≤9?	Synonym-to-SMILES, Reinvent, Scoring, Analyzer
Can you give me molecules similar to [COMPOUND ID] that do not contain morpholine?	Synonym-to-SMILES, Reinvent, Scoring, Analyzer
Can you give me some molecules similar to [SMILES] and show me the synthesis routes for the top 5 similar ones?	Reinvent, AiZynthFinder

[a] SMILES indicates a SMILES string, FILEPATH is a path to a file on disc, PDB ID is a crystal structure ID on the Protein Data Bank, COMPOUND ID an internal identifier for a compound. These placeholders are replaced with real values in the test suites.

of multiple agents or tools, thereby testing the system's abilities to route tasks appropriately and to integrate results across tools. At the time of writing this article, our set suite included 33 questions, although it is constantly expanding. Example questions are shown in Table 2 to illustrate the level of complexity of the prompts and tool calls. Furthermore, we introduced mock tools to return precomputed results in order to avoid dependence on integrations when testing the core system. Mock tool outputs are deterministic and reduce the execution time of the auto-tests.

Our test suite was designed to cover common user prompts to the agent but is not exhaustive. It does not cover all of the possible ways to call the tools, nor does it cover different rephrases of the same prompts. Furthermore, these test questions are not a replacement for the more typical unit and integration tests.

Comparing single-agent and multi-agent systems

To benchmark our single- and multi-agent architectures, we employed a test suite of 33 questions. Each question was submit-

ted three times to account for the inherent stochasticity in LLMs. We evaluated the quality of agent responses from different angles. First, we examined basic performance metrics, including response time, token consumption, and the number of errors encountered during execution, including both API-related issues and tool invocation failures. This evaluation was conducted for both tool-specific tasks and workflow tasks. To isolate and accurately assess the agent's speed in routing tasks and summarising answers, we employed mock tools with simulated responses, thereby avoiding confounding effects from the variable runtime of external tools or transient errors and service unavailability. Second, we evaluated the sequence of tool calls executed by each agent. For each workflow question, we defined a set of 'complete tool call' sequences that represent the ideal execution path, indicating that the question had been fully addressed. In addition, we identified 'partial tool call' sequences, which only partially satisfy the user query but still produce a valid answer while requiring minimal follow-up from the user. It should be noted that both 'complete' and 'partial' sequences were considered successful in our validation. Any tool call sequences that did not fall into these categories were classified as incorrect, indicating an inadequate or erroneous response.

Figure 3 summarises the benchmark results. On the left side, the radar plot reports averaged performance metrics across all test questions. The multi-agent system was more efficient in both response time and token usage but showed a higher rate of tool-call errors when compared to the single-agent system. Many of these errors stemmed from difficulties that the LLM had in interpreting (Pydantic-generated) dictionary input schemas, an issue that remains under investigation. Despite this, the multi-agent system consistently self-corrected and produced the correct final answers, resulting in a higher proportion of complete executions overall. This is shown in the pie charts on the right side of Figure 3, which display the results from our second methodology of assessment, the tool-call sequence correctness. The multi-agent system exhibits a larger proportion of complete tool call sequences, a lower proportion of partial tool call sequences, and a slightly higher proportion of incorrect tool call sequences when compared to the single-agent system. Although the higher error rate is often mitigated through self-correction, refining the prompts and system design remains a key priority.

Chat interface for agentic drug discovery

The single- and multi-agent systems can be accessed directly through a command-line interface or within a Python environment. Programmatic access is not, however, suitable for all scientists and does not, on its own, support the broader adoption of AI platforms. To address this, we developed ChatInvent, a web-based application that offers a graphical and natural language interface between users and the underlying system.

Application overview

Upon entering ChatInvent, users are welcomed with a graphical user interface (GUI) that displays three tabs containing example prompts, available features, and system limitations (see Figure 4). Users can ask questions through a chat box, with the possibility to upload input files or use a molecular canvas to draw molecules. Molecules created in the canvas can be copied as SMILES strings and incorporated into queries. When a result is generated, the agent returns a detailed conversational explanation. If the response includes molecules, these are displayed as images. Users can hover over the images and copy their SMILES strings or can load the structures in the interactive canvas for inspection or modification. The system also supports returning downloadable files such as CSVs when the task involves bulk data generation, scoring, or analysis.

Except for certain chemistry-specific capabilities, the interface closely resembles other LLM-driven chat systems such as ChatGPT. However, we have integrated additional GUI elements that are designed to increase transparency and user confidence:

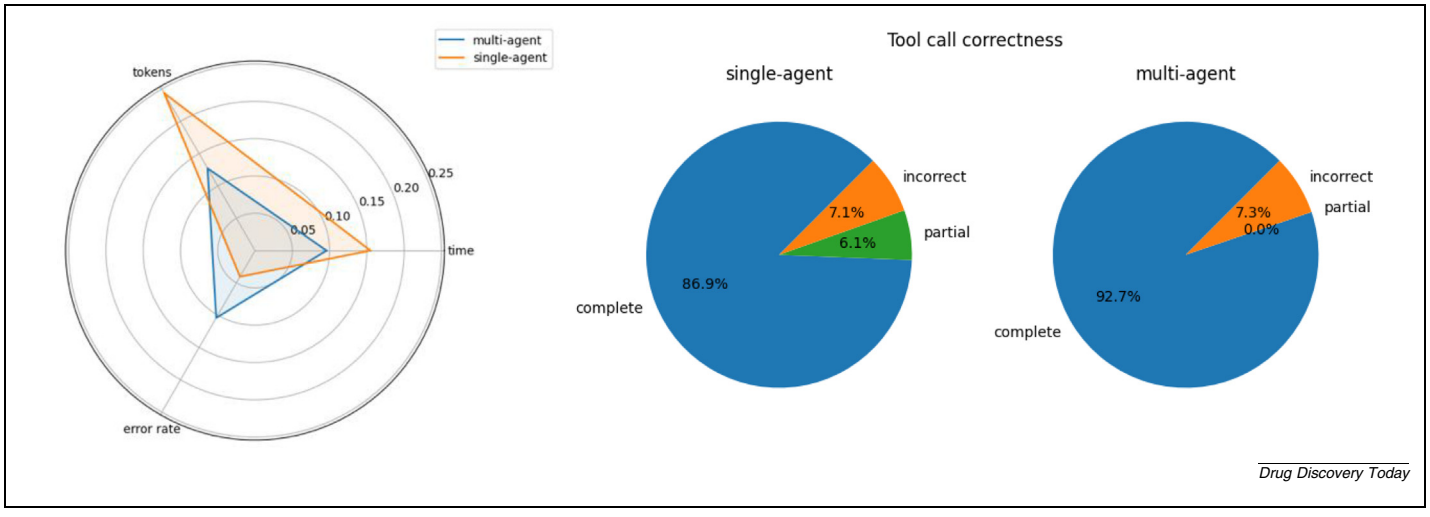

FIGURE 3

Benchmarking the performance of single- and multi-agent systems. On the left, a radar plot measuring average tokens, time, and error rate during the validation of the agents. All metrics were normalised between 0 and 1. On the right, two pie charts report the ratio of complete, partial, and incorrect calls made by single- and multi-agent systems.

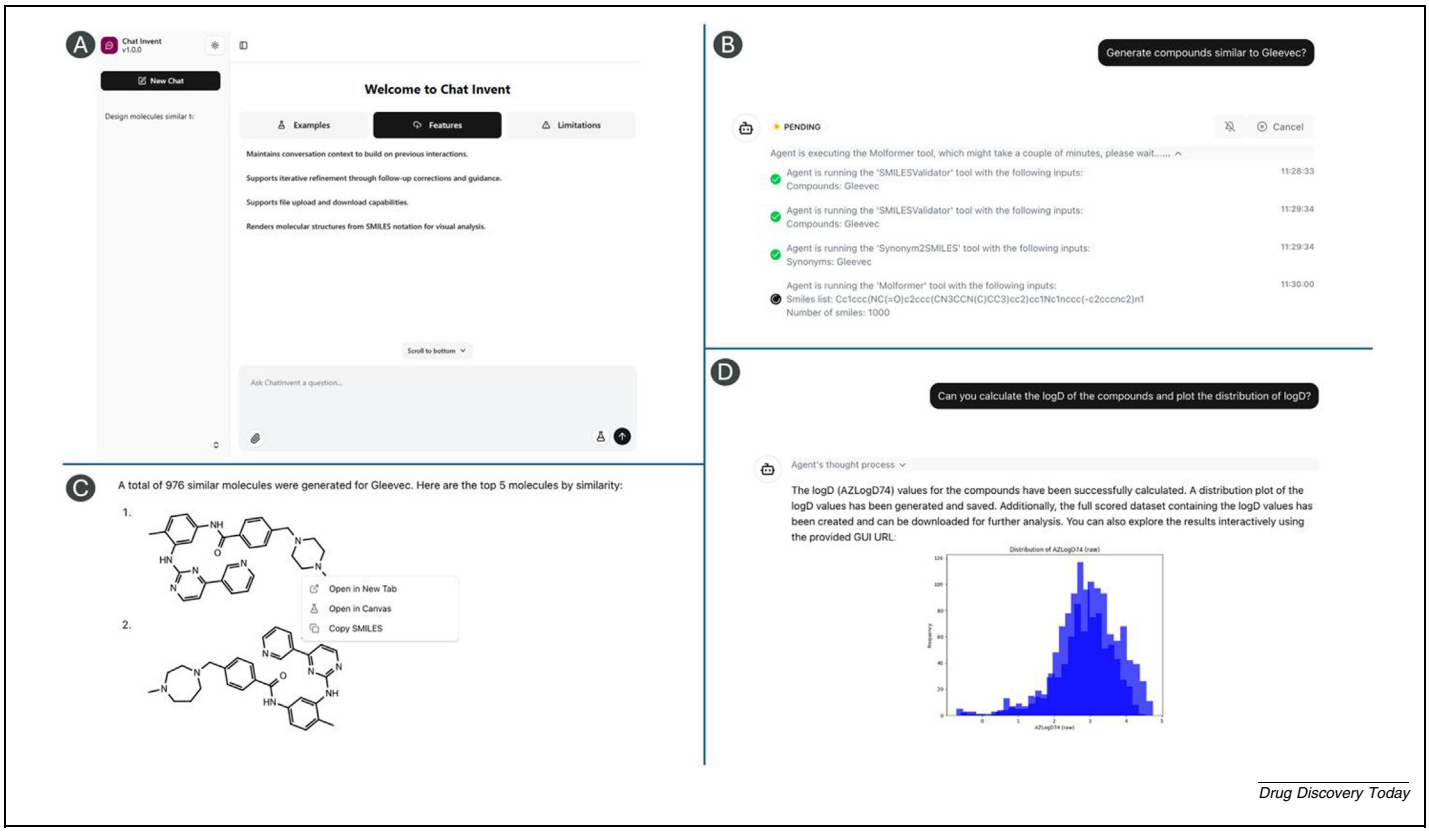

FIGURE 4

ChatInvent, a web-based application that offers a graphical and natural language interface between users and the underlying system. A. The ChatInvent landing page. The column on the left lists previous conversations with the agents and provides a button to start a new chat. The main container includes three tabs with examples, features, and system limitations. At the bottom, typed queries and attachments can be entered into a chat box. The flask on the bottom right opens up a panel with a canvas for drawing compound structures interactively. **B.** An agent thinking about how to generate molecules similar to Gleevec. **C.** The output of the agent having completed the prompt in panel B. **D.** The output of the agent for a follow-up question, showing the distribution of logD of the generated compounds.

- When a user submits a query, the agent's reasoning process is displayed in real time. This includes the tools selected, the inputs passed to them, and the execution sequence. Users can follow the workflow step by step, making it easier to judge whether the agent is handling the query appropriately.
- The chat interface implements a feedback mechanism: users can like, dislike, or leave specific comments on messages from the agent. Feedback is associated with specific messages and sessions, allowing developers to monitor usage trends and improve the system. Feedback is also logged with LangFuse, which supports evaluation and debugging by surfacing detailed session-level traces, including tool usage and agent behaviour.

A typical use of ChatInvent could involve the generation of analogues of a query molecule, and their prioritisation using different scoring techniques. An illustration of the GUI for such a task is shown in Figure 4. Such a task already involves a certain degree of expertise and experience from the user, for example, familiarity with synonym searching in databases and with running and post-processing design and scoring tools. ChatInvent removes these barriers while still exposing the agents' reasoning and operations to the user.

Discussion and lessons learned

Publications on agentic systems such as Chemcrow and CoScientist have showcased the capabilities of agentic systems but are confined within their own research scope by the prompts tested, tools integrated, and frameworks. While inspired by these efforts, we deliberately made different design choices. A critical factor for us was designing our solution to be accessible and scalable, ensuring usability across the scientific community in AstraZeneca. Another factor was designing a solution that could be maintained and upgraded once productionised. Drawing upon our experience with ChatInvent, our discussion here focuses on aspects that we recommend as considerations when building an agentic system for drug discovery.

Futureproofing integrations through MCP

We built ChatInvent on mature software platforms such as Reinvent, AiZynthFinder, and PIP, which expose robust APIs that agents can consume. This allowed us to focus on the development of the agentic framework and the web application. However, these integrations rely on bespoke client code in the tools, which, despite the maturity of our platform interfaces, creates a certain degree of coupling between the agentic system and underlying services. A more sustainable approach would be to

expose functionalities through Model Context Protocol (MCP) servers, which provide a standardised interface for connecting LLMs to external tools. Although MCP was not adopted during our initial development, we see clear long-term benefits in transitioning to MCP. At the time of the review of this manuscript, an RDKit MCP server is being tested in the ChatInvent development environment. Upgrading existing and future capabilities to MCP would reduce maintenance overhead, improve interoperability, and make it easier for other teams to extend ChatInvent.

Agent recall in extended conversations

The integration of LangFuse into ChatInvent has been important in allowing us to monitor and understand the behaviour of the agent. The agentic system currently struggles with long conversations, in which the human operator is iteratively refining data generated by the agent, and in which memory from different sub-agents need to be passed around by the supervisor. We are investigating novel mechanisms to aid in the selection of the relevant piece of memory, something that is enabled by the multi-agent architecture of our system. As highlighted by the common errors listed in Table 1, there is still a great need for research into multi-agent architectures and how the information flows within the system.

Upgrading frameworks and LLMs

The first generation of agentic systems in chemistry was based on frameworks or LLMs that have now been surpassed by new tools and models. The field is moving fast, making it challenging to maintain compatibility between components and LLMs. For example, replacing LLM endpoints or making slight changes to a tool-calling mechanism can lead to markedly different behaviours. During the development of ChatInvent, we upgraded the LLM on two occasions, first from GPT-3.5 to GPT-4, then to GTP-4o, and now we are preparing a transition to GPT-5. Each change in LLM required at least one week of prompt tuning and comprehensive evaluation. In addition, framework interfaces are also still subject to breaking changes. Our system is built on LangChain and LangGraph, which have stabilised enough that their latest releases have not caused major disruptions; nevertheless, future upgrades may require caution to avoid disruption in deployments. The automated test suite with typical user question have enabled us to ensure some level of reproducibility when upgrading LLM or LangChain.

We also envisage that the future production-grade architectures, including that of ChatInvent, could be replaced with serverless solutions such as AWS Bedrock. Switching to a serverless architecture would allow granular versioned control of agents via configuration files, and would permit prompt developers to focus on the business logic of tools and integrations. Alternatively, a hybrid architecture that combines in-house agents with managed services can address diverse requirements while allowing rapid upgrades of components needing new capabilities, such as improved context management. Hybrid architectures, however, incur increased complexity; consequently, enforcing standardized protocols, such as A2A (for agents) and MCP (for tools), becomes essential to ensure sustainable development. For research and experimentation, we may continue to use bespoke and locally deployed agentic frameworks, but for production systems such as ChatInvent, these are factors to consider carefully.

Long-term operational challenges

The rapid changes in the field of agents and LLMs are a double-edged sword. On one hand, the technology is advancing at an incredible pace, resulting in new and better solutions that may deserve to be adopted as replacements for their predecessors. On the other hand, such a lack of convergence reflects the scarce maturity of LLM operations (LLMOps) compared to other operational fields. We argue that the full potential of agents in production systems will be leveraged once these technologies begin to stabilise, making their operation easier to maintain in the long term.

Change management and governance

The wide adoption of agentic systems such as ChatInvent in drug discovery will require some changes at the organisational level. Change management is required because performing drug discovery using natural language involves moving into a new paradigm that differs from the way in which computer-aided drug discovery has been carried out for the past 50 years. As a complement, robust governance needs to be established to prevent the accidental misuse of agents, and to ensure that outputs are validated, documented, and reviewable.

We anticipate that agentic systems will impact operations in drug discovery by broadening access to expert tools such as generative and predictive AI for molecule design, which may result in more efficiency in pharmaceutical R&D. The development of ChatInvent can be seen as an early milestone prompting future opportunities from the integration of LLMs and agents in the drug discovery workflow.

CRediT authorship contribution statement

Jiazhen He: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Helen Lai:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Lakshidaa Saigiridharan:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Data curation, Conceptualization, Formal analysis. **Gian Marco Ghiandoni:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Kinga Jenei:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Investigation, Data curation. **Umur Gokalp:** Software. **Ajša Nuković:** Project administration. **Ola Engkvist:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization. **Jon Paul Janet:** Writing – review & editing, Writing – original draft, Supervision, Resources, Project administration, Methodology, Funding acquisition, Conceptualization. **Samuel Genheden:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Project administration,

Investigation, Funding acquisition, Formal analysis, Conceptualization.

Acknowledgements

We thank Gavin Edwards and Michael Ughetto from the Biological Insights Knowledge Graph team at AstraZeneca for their early insights on implementing the agent framework. We thank Prakash Rathi for his feedback on the manuscript. We thank Daniel Alvarez for his later contribution to ChatInvent.

Competing interests

JH, HL, LS, GMG, KJ, OE, JPJ, and SG are AstraZeneca employees and shareholders. No competing interests are declared.

Funding

The Open Access licensing of this manuscript was enabled by BioPharma R&D at AstraZeneca. The authors have applied a Creative Commons Attribution (CC BY) licence to any author accepted manuscript version.

Data availability statement

The data resulting from the benchmarking are available on request.

Data availability

Data will be made available on request.

References

1. Plowright AT, Johnstone C, Kihlberg J, Pettersson JA, Robb GR, Thompson RA. Hypothesis driven drug design: improving quality and effectiveness of the design-make-test-analyse cycle. *Drug Discov Today*. 2012;2012:56–62. https://doi.org/10.1016/j.drudis.2011.09.012.
2. Ghiandoni GM, Evertsson E, Riley DJ, Tyrchan C, Rathi PC. Augmenting DMTA using predictive AI modelling at AstraZeneca. *Drug Discov Today*. 2024;29, 103945. https://doi.org/10.1016/j.drudis.2024.103945.
3. Ferreira FJ, Carneiro AS. AI-driven drug discovery: a comprehensive review. *ACS Omega*. 2025;10:23889–23903. https://doi.org/10.1021/acsomega.5c00549.
4. Hasselgren C, Oprea TI. Artificial intelligence for drug discovery: are we there yet? *Annu Rev Pharmacol Toxicol*. 2024;64:527–550. https://doi.org/10.1146/annurev-pharmtox-040323-040828.
5. Ramos MC, Collison CJ, White AD. A review of large language models and autonomous agents in chemistry. *Chem Sci*. 2024;16:2514–2572. https://doi.org/10.1039/d4sc03921a.
6. Zheng Z, Zhang O, Borgs C, Chayes JT, Yaghi OM. ChatGPT Chemistry Assistant for text mining and prediction of MOF synthesis. *J Am Chem Soc*. 2023;145:18048–18062. https://doi.org/10.1021/jacs.3c05819.
7. Vangala S et al. Suitability of large language models for extraction of high-quality chemical reaction dataset from patent literature. *J Cheminform*. 2024;16:131. https://doi.org/10.1186/s13321-024-00928-8.
8. Mirza A et al. A framework for evaluating the chemical knowledge and reasoning abilities of large language models against the expertise of chemists. *Nat Chem*. 2025;17:1027–1034. https://doi.org/10.1038/s41557-025-01815-x.
9. Runcie NT, Deane CM, Imrie F. Assessing the chemical intelligence of large language models. *arXiv*. 20252505.07735. https://doi.org/10.48550/arXiv.2505.07735.
10. Yoshikawa N et al. Large language models for chemistry robotics. *Auton Robot*. 2023;47:1057–1086. https://doi.org/10.1007/s10514-023-10136-2.
11. Bran AM, Cox S, Schilter O, Baldassari C, White AD, Schwaller P. Augmenting large language models with chemistry tools. *Nat Mach Intell*. 2024;6:525–535. https://doi.org/10.1038/s42256-024-00832-8.
12. Boiko DA, MacKnight R, Kline B, Gomes G. Autonomous chemical research with large language models. *Nature*. 2023;624:570–578. https://doi.org/10.1038/s41586-023-06792-0.
13. Fehlis Y et al. Accelerating drug discovery through agentic AI: a multi-agent approach to laboratory automation in the DMTA cycle. *arXiv*. 20252507.09023. https://doi.org/10.48550/arXiv.2507.09023.
14. Pan Q, Xu D, Yao JX, Ma L, Zhu Z, Ji J. FROGENT: an end-to-end full-process drug design agent. *arXiv*. 20252508.10760. https://doi.org/10.48550/arXiv.2508.10760.
15. Ruan Y et al. An automatic end-to-end chemical synthesis development platform powered by large language models. *Nature Commun*. 2024;15:10160. https://doi.org/10.1038/s41467-024-54457-x.
16. McNaughton AD, Ramalaxmi G, Kruel A, Knutson CR, Varikoti RA, Kumar N. CACTUS: chemistry agent connecting tool usage to science. *ACS Omega*. 2024;9:46563–46573. https://doi.org/10.1021/acsomega.4c08408.
17. Liu S et al. DrugAgent: automating AI-aided drug discovery programming through LLM multi-agent collaboration. *arXiv*. 20242411.15692. https://doi.org/10.48550/arXiv.2411.15692.
18. Zhang C et al. SynAsk: unleashing the power of large language models in organic synthesis. *Chem Sci*. 2024;16:43–56. https://doi.org/10.1039/d4sc04757e.
19. Ishida S, Sato T, Honma T, Terayama K. Large language models open new way of AI-assisted molecule design for chemists. *J Cheminform*. 2025;17:36. https://doi.org/10.1186/s13321-025-00984-8.
20. Loeffler HH et al. Reinvent 4: modern AI–driven generative molecule design. *J Cheminform*. 2024;2024:20. https://doi.org/10.1186/s13321-024-00812-5.
21. Saigiridharan L, Hassen AK, Lai H, Torren-Peraire P, Engkvist O, Genheden S. AiZynthFinder 4.0: developments based on learnings from 3 years of industrial application. *J Cheminform*. 2024;16:57. https://doi.org/10.1186/s13321-024-00860-x.
22. LangChain. www.langchain.com/. Accessed August 2025.
23. Yao S et al. ReAct: synergizing reasoning and acting in language models. *arXiv*. 20222210.03629. https://doi.org/10.48550/arXiv.2210.03629.
24. Bauer CA, Kogej T, Genheden S, Norrby P. Precedent Finder: locating pareto-optimal reactions. *J Chem Inf Model*. 2025;65:9378–9382. https://doi.org/10.1021/acs.jcim.5c01797.
25. Model Context Protocol. *What is the Model Context Protocol (MCP)?* https://modelcontextprotocol.io/docs/getting-started/intro. Accessed August 2025.
26. A2A Protocol. *Agent2Agent Protocol*. https://a2aprotocol.ai/. Accessed August 2025.
27. Barbi O, Yoran O, Geva M. Preventing rogue agents improves multi-agent collaboration. *arXiv*. 20252502.05986. https://doi.org/10.48550/arXiv.2502.05986.
28. Xiao M, Xiao Y, Ji S, Cai H, Xue L, Zhang P. Assessing the robustness of LLM-based NLP software via automated testing. *arXiv*. 20242412.21016. https://doi.org/10.48550/arXiv.2412.21016.
29. Langfuse. *Open Source LLM Engineering Platform*. https://langfuse.com/. Accessed August 2025.