

Patterns

Automating quantum computing laboratory experiments with an agent-based AI framework

Highlights

- Language-model agents learn and apply laboratory knowledge
- Agents autonomously conduct complex quantum experiments
- System enables closed-loop feedback for experiment control
- Performance matches expert scientists in quantum calibration

Authors

Shuxiang Cao, Zijian Zhang, Mohammed Alghadeer, ..., Mustafa Bakr, Peter Leek, Alán Aspuru-Guzik

Correspondence

shuxiang.cao@physics.ox.ac.uk (S.C.),
zijian@cs.toronto.edu (Z.Z.),
peter.leek@physics.ox.ac.uk (P.L.),
alan@aspuru.com (A.A.-G.)

In brief

This study introduces a language-model-driven agent system that captures and applies laboratory knowledge to autonomously plan, execute, and analyze experiments. When the agent system was demonstrated on quantum computing tasks, the agents achieved performance on par with expert researchers. By enabling intelligent coordination and decision-making in complex lab settings, this approach paves the way for more flexible, scalable automation in experimental science.



Article

Automating quantum computing laboratory experiments with an agent-based AI framework

Shuxiang Cao,^{1,9,10,*} Zijian Zhang,^{2,3,10,*} Mohammed Alghadeer,¹ Simone D. Fasciati,¹ Michele Piscitelli,¹ Mustafa Bakr,¹ Peter Leek,^{1,*} and Alán Aspuru-Guzik^{2,3,4,5,6,7,8,11,*}

¹Clarendon Laboratory, Department of Physics, University of Oxford, Oxford OX1 3PU, UK

²Department of Computer Science, University of Toronto, Toronto, ON M5S 2E4, Canada

³Vector Institute for Artificial Intelligence, Toronto, ON M5G 1M1, Canada

⁴Department of Chemistry, University of Toronto, Toronto, ON M5S 3H6, Canada

⁵Department of Materials Science & Engineering, University of Toronto, Toronto, ON M5S 3E4, Canada

⁶Department of Chemical Engineering & Applied Chemistry, University of Toronto, Toronto, ON M5S 3E5, Canada

⁷Canadian Institute for Advanced Research (CIFAR), Toronto, ON M5G 1M1, Canada

⁸NVIDIA, 431 King St. W 6th, Toronto, ON M5V 1K4, Canada

⁹Present address: NVIDIA Corporation, 2788 San Tomas Expy., Santa Clara, CA 95051, USA

¹⁰These authors contributed equally

¹¹Lead contact

*Correspondence: shuxiang.cao@physics.ox.ac.uk (S.C.), zijian@cs.toronto.edu (Z.Z.), peter.leek@physics.ox.ac.uk (P.L.), alan@aspuru.com (A.A.-G.)

<https://doi.org/10.1016/j.patter.2025.101372>

THE BIGGER PICTURE Laboratories that can operate autonomously have the potential to accelerate scientific discovery, but they face a challenge: how to embed the knowledge that human researchers rely on when designing, running, and interpreting experiments. Our work addresses this challenge by introducing a language-model-driven agent framework that captures and operationalizes laboratory knowledge. This enables autonomous coordination, decision-making, and analysis. When our system was demonstrated in the complex domain of quantum computing, it autonomously performed multistep experiments and achieved outcomes comparable to those of expert researchers. This approach provides a scalable path toward more intelligent, adaptable laboratory automation, opening new opportunities for faster, more accessible scientific breakthroughs across experimental sciences.

SUMMARY

Fully automated self-driving laboratories promise high-throughput, large-scale scientific discovery by reducing repetitive labor. However, they require deep integration of laboratory knowledge, which is often unstructured, multimodal, and hard to incorporate into current AI systems. This paper introduces the “k-agents” framework, designed to support experimentalists in organizing laboratory knowledge and automating experiments with agents. The framework uses large-language-model-based agents to encapsulate laboratory knowledge, including available operations and methods for analyzing results. To automate experiments, execution agents break multistep procedures into agent-based state machines, interact with other agents to execute steps, and analyze results. These results drive state transitions, enabling closed-loop feedback control. We demonstrate the system on a superconducting quantum processor, where agents autonomously planned and executed experiments for hours, successfully producing and characterizing entangled quantum states at human-level performance. Our knowledge-based agent system opens new possibilities for managing laboratory knowledge and accelerating scientific discovery.

INTRODUCTION

Automating laboratory experiments has the potential to accelerate scientific discovery by closing the loop between experi-

mental execution, artificial intelligence (AI), and human-in-the-loop decision-making.¹ Although it is increasingly common in laboratories that experiments can be implemented through programming interfaces,^{2,3} automating experiments still requires



detailed laboratory knowledge to set the parameters of each experiment, interpret the results of the experiment, and execute complicated experiment workflows. Traditional methods of automating experiments require human experts to translate their knowledge into code for machine execution.^{4–8} However, the knowledge required for laboratory work has increased significantly over the past decades due to the growing complexity of the experimental apparatus, which now involves more intricate technical details for operation. This makes translating the required knowledge and maintaining consistency an increasingly significant human effort, which compromises the efficiency brought about by automation. In addition, the multimodal and complex nature of laboratory knowledge also makes it challenging to develop automated programs that can perform these tasks at a level comparable to that of humans.

The advent of large language models (LLMs) and multimodal LLMs has sparked new hope for more efficient experiment automation due to their universal ability to process text and image information.^{9–15} Although limited by the length of their context windows, LLMs offer the hope of efficiently understanding laboratory documents, performing analyses, generating code, and interpreting experimental images. The development of multiagent systems has further enhanced LLMs' capabilities by integrating multiple LLM-based agents to work together on a complex objective, improving both accuracy and scalability.^{16–22} The merits of LLM-based agents make them well suited to assist laboratory automation without the need for extensive human intervention. However, several challenges remain. First, much of the media, such as code and documents, that carry laboratory knowledge are proprietary and inaccessible online, meaning that they are unlikely to be included in the training data of publicly available LLMs. In addition, the amount of text that includes such specialized laboratory knowledge is often insufficient to effectively fine-tune an LLM.²³ Furthermore, laboratory knowledge is usually dynamic and frequently updated, making fine-tuning models impractical, as it is often hard to rewrite the knowledge in LLMs.^{24–26} As a result, incorporating laboratory-specific knowledge into LLMs remains difficult. Recent work on retrieval-augmented generation (RAG) offers a foundation to equip LLM-based agents with larger-scale knowledge without fine-tuning.^{27–29} However, laboratory knowledge is usually heterogeneous and multimodal, making it challenging to apply standard RAG methods directly.^{4,8,30}

Despite challenges, pioneering efforts to develop automated LLM-based agents to carry out experiments are already underway.^{31–34} For example, in Boiko et al.³¹ and Bran et al.,³² researchers equipped LLM-based agents with tools that allowed them to acquire professional knowledge from the expert-designed tool kit and even the internet. These agents were then capable of performing chemistry experiments on the basis of this knowledge. However, in general, previous work has lacked a scalable memory system and relied on keeping the conversation history when calling LLMs. This inherently restricts the agents' ability to automate long-duration tasks involving many sequential steps, as most current LLMs exhibit diminishing performance as the input length increases.^{35–37} This construction also prohibits further extensions of the system, such as supporting the handling of scientific plots.

In this work, we introduce “k-agents” (see Figure 1), a knowledge-based multiagent system designed to automate experi-

ments, particularly those requiring large-scale, multimodal laboratory knowledge and complex workflow. We developed tools for users to transfer their knowledge, creating LLM-based knowledge agents to manage laboratory knowledge without fine-tuning. These knowledge agents can help operate laboratories by holding knowledge ranging from single experiments to complex procedures and how to inspect the results of each experiment. These agents are designed to be activated selectively, allowing the system to scale efficiently as the number of agents increases. In order to fully exploit the knowledge agents, we further introduce the “execution agent,” which is responsible for coordinating the knowledge agents, gathering and filtering knowledge, generating code (scripts) to operate the laboratory, and controlling the progress of complex experiment procedures with intelligent closed-loop feedback planning. To handle multistep experiments, the execution agent decomposes complex procedures into independent experiment stages and creates an agent-based state machine. The agent-based state machine differs from a traditional state machine in that state transitions are determined by agents, rather than by rigid deterministic rules. This state-machine-based approach minimizes the experimental history that needs to be loaded into LLMs, making it feasible to conduct long-duration experiments at a human-like level of performance. The agents are created by prompting a language model on a distinct context, and the details of the prompt construction can be found in Note S2.

As a demonstration, we applied our framework to automate the calibration and characterization of single- and 2-qubit gates on our superconducting quantum processor. Superconducting qubits have become one of the most widely adopted platforms for quantum computing, with recent advancements pushing the scale to hundreds of qubits, along with active quantum error correction.^{38–41} As these systems grow in complexity, calibrating the operations of hundreds of qubits has emerged as a substantial bottleneck. Given this rapid progress, it is timely and essential to study automation solutions that can support the scalability and performance requirements of large-scale superconducting quantum devices. In addition, our framework demonstrates its ability to conduct custom experiments. Specifically, preparing the Greenberger-Horne-Zeilinger (GHZ) state requires the calibration of multiple qubits to execute the desired quantum operations. We selected this experiment to demonstrate the capabilities of our automation technique in generating an entangled quantum state and evaluating its fidelity. We propose that this framework could be adapted to other fields as a model for future research and industrial methodologies.

Knowledge agents

We define “knowledge agents” as AI agents whose performance is measured by their ability to receive and transfer knowledge. For instance, an agent qualifies as a strong knowledge agent if it accepts knowledge from natural language inputs and responds correctly to related queries using natural language. Developing more advanced knowledge agents offers significant advantages. First, a more general ability to accept knowledge reduces the effort needed for humans to translate knowledge into computer-friendly formats. Furthermore, the ability to transfer knowledge facilitates more rigorous testing of agents⁴² and avoids

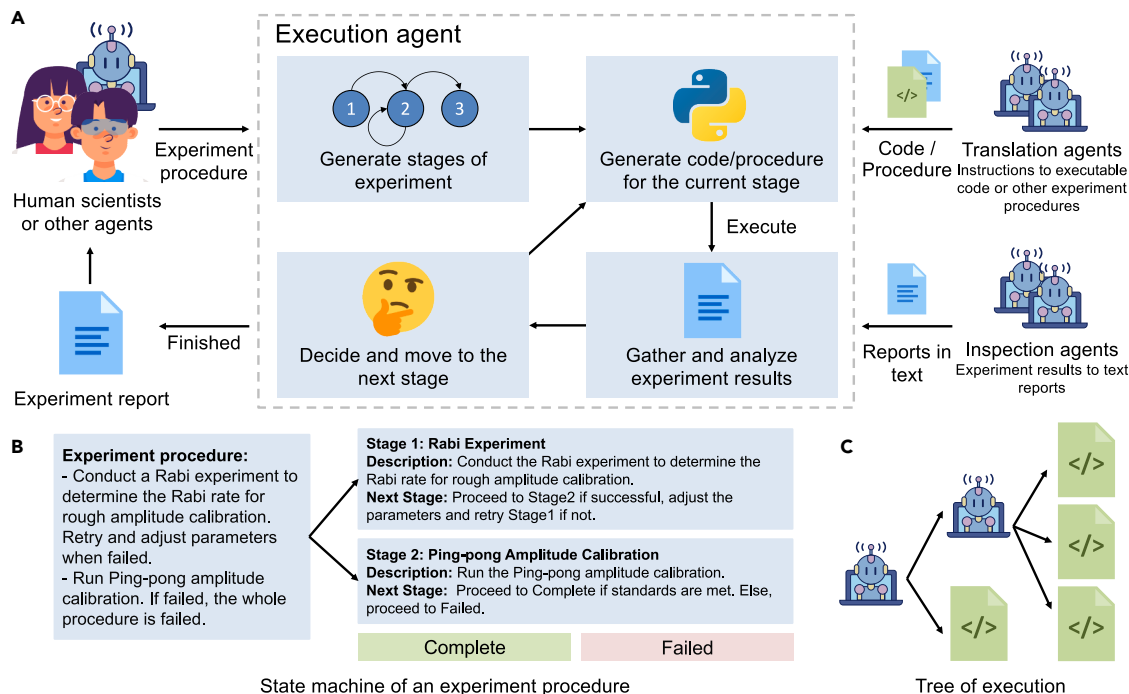


Figure 1. Overview of the “k-agents” framework and agent-based state machine architecture

(A) Overview of the k-agents framework. Given a procedure in natural language, the execution agent first decomposes it into an agent-based state machine, which contains experiment stages. Each stage holds an independent experiment instruction to be translated with the translation agents. The transition between stages is driven by the generated transition rules and the reports from the inspection agents.

(B) Agent-based state machine of instructions. The figure demonstrates how an experiment procedure can be decomposed into an agent-based state machine. The decomposition divides experiment procedures into experiment stages that hold single instructions along with a transition rule for deciding the next stage. (C) Tree of translation. The translating of instructions can be viewed as an expansion of a tree, in which the translation results are represented as the child nodes. An instruction can be translated as a simple experiment (code node) or an experiment procedure (agent node) that needs further execution and translation with another execution agent.

treating the system as a black box, improving the general trustworthiness of AI systems.

Prior works^{31–34} have limitations in scalability, which constrain their ability to manage complex tool sets and finish tasks that require many steps. These systems typically construct a single prompt that embeds both task instructions and the entire tool set. However, as shown in Yao et al.,⁴³ Wang et al.,⁴⁴ and Barres et al.,⁴⁵ the performance of these systems can degrade significantly as the length of input to the LLM, including the list of tools and history of action, increases. For example, in Barres et al.,⁴⁵ the authors demonstrate that the performance of LLM agents can decrease to nearly zero when more than seven actions are required for the task in their setup.

To address some of these issues, works such as Boiko et al.³¹ introduce modular architectures that allow the developer to manually divide tools into components. Additionally, they rely on RAG^{31,32} for data access, which often screens and retrieves documents based on relevance rather than a deeper understanding of their structure.

To resolve the above issue, we introduce the k-agents framework, which provides tools to implement the aforementioned bidirectional knowledge transferability and the scalability toward a large number of tools and long-horizon tasks. In k-agents, we implement user-friendly interfaces that enable users to encapsulate knowledge within different LLM-based agents. These agents

can then transfer their knowledge to the users and other agents in the format of natural language with LLMs. Our framework has a scalable memory architecture with an agent-based retrieval mechanism. This architecture enables good performance while enabling the required context length and token count to grow more slowly as the memory size grows. This enables agents to dynamically interact with a large, indexed memory of procedural knowledge and existing experiments. During retrieval, the agents conduct the reflection and selection processes to improve the accuracy and contextual relevance of retrieved information. This design allows our framework to support more complex and long-duration tasks.

Code translation agent

The first kind of knowledge agent in k-agents is the “code translation agent,” which is designed to accept and apply the knowledge of the available experiment interfaces that can be called by code. Here, the term “experiment” stands for not only a direct experiment on an instrument but any general action that is available in the laboratory. It can also be a call to a tool or even another agent. Given instructions in natural language, these agents are responsible for translating natural language instructions into the corresponding code. To help the users of k-agents construct code translation agents, we provide an abstract Python class that allows users to define the set of experiments available in the laboratory explicitly. The users can then define

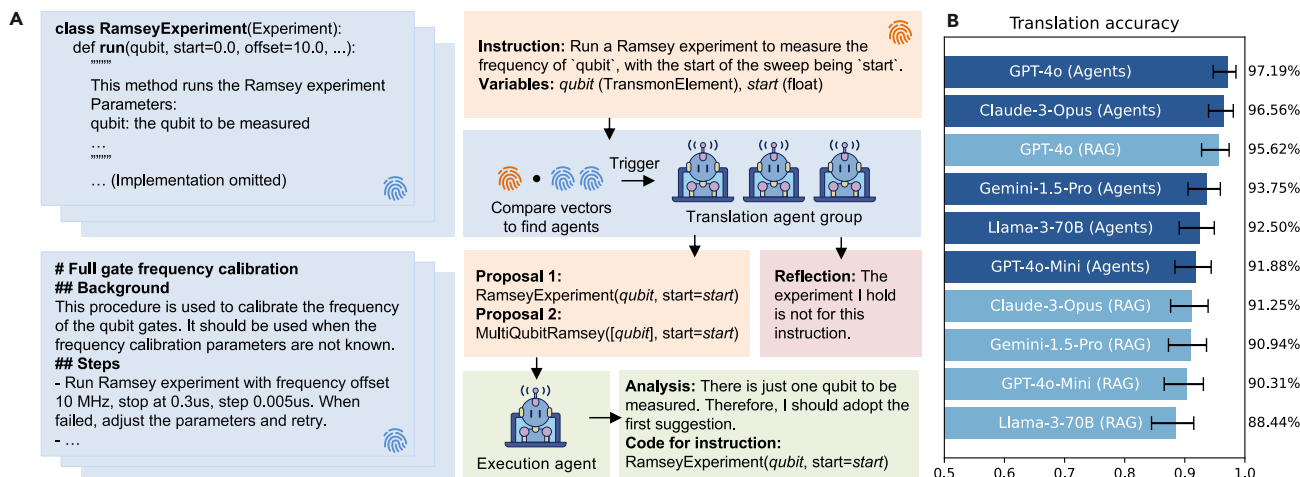


Figure 2. Translation agents architecture and performance benchmarking

(A) Translation agents. Translation agents are responsible for translating an incoming instruction into executable code. The agents will be activated when their characterizing vectors (blue fingerprints) have a significant overlap with the instruction’s vector (orange fingerprint). The activated agents will try to translate the instruction based on their knowledge. If the translation agent deems the translation as valid, the result will be output to the execution agent for a final selection and execution.

(B) Accuracy of instruction translation. We prepare a test set containing 80 instructions from eight experiments to be translated. We compare k-agents (marked as Agents), which uses Algorithm 1, with standard RAG methods (marked as RAG), which directly load the signature of the experiment class into the prompt. In our setup with 17 code translation agents, we found that k-agents demonstrates better performance than standard RAG methods in choosing the correct experiment class. Further, k-agents can support heterogeneous translation agents working together, which is hard to achieve with standard RAG methods. The error bars are estimated using the Wilson score confidence interval for binomial proportions. Details of this benchmark can be found in Note S3.

all possible experiment interfaces, ensuring that the AI system can access and utilize them. As illustrated in Figure 2A, human experimentalists are expected to document background knowledge, required parameters, and implementation code for each experiment in the run method of the subclass representing the experiment. After indexing all available experiment classes, k-agents will construct a code translation agent for each of the classes.

Procedure translation agent

Experiment procedures in laboratories can involve complex workflows. These workflows may require experimentalists to determine a sequence of experiments based on the results of the experiments executed. Traditionally, this knowledge has been maintained either through unstructured documentation or by relying on the memory of the experimentalists themselves. To address this, as shown in Figure 2A, k-agents introduces a standardized format to store examples of how to implement instructions using multistep experiment procedures. These examples are then used to produce another kind of knowledge agent that we call the “procedure translation agent,” which is capable of translating an instruction into its corresponding procedure based on stored examples. Similar to code translation agents, procedure translation agents also output code. The code will call an execution agent to execute the procedure, which we will introduce later.

Inspection agent

A major burden for experimentalists has been the need to wait for experimental results and decide on the next step based on them. In many scenarios, experimentalists must analyze figures to assess the success of an experiment and determine the next steps. In the k-agents framework, we introduce inspection

agents that have the knowledge needed to evaluate the results of each experiment. These agents will be called after the execution of each experiment to analyze its outcomes. We provide an interface to inject knowledge for each function that produces experiment figures. This feature is implemented as a Python decorator. In the decorator, the users can add instructions for how to analyze the figure. Besides, as illustrated in Figure 3A, our interface allows users to add example figures to help others understand the new figures. Based on the knowledge from the interfaces, we construct inspection agents equipped with multi-modal LLMs to inspect and analyze new experiment figures and output text-based reports based on their knowledge.

Execution agent

We introduce the execution agent to coordinate knowledge agents and manage complex experiment procedures. This agent takes experiment procedures written in natural language as input and driven by a text-only LLM. Since the selection of the next experiment in a procedure could depend on the results of previous experiments, we design the execution agent to first decompose the procedure into a “state machine,” in which each state represents a distinct experiment stage of the procedure (see Figure 1C). Each stage of the experiment is assigned a single instruction that contains only one experiment or subprocedure. As an agent-based state machine, there is also a transition rule at each stage in natural language that determines how to select the next stage based on the result of the experiment.

By constructing the agent-based state machine, the execution agent transforms the task of executing the procedure into executing the agent-based state machine. The execution is illustrated in Figure 1A. Starting from the initial stage, at each stage,

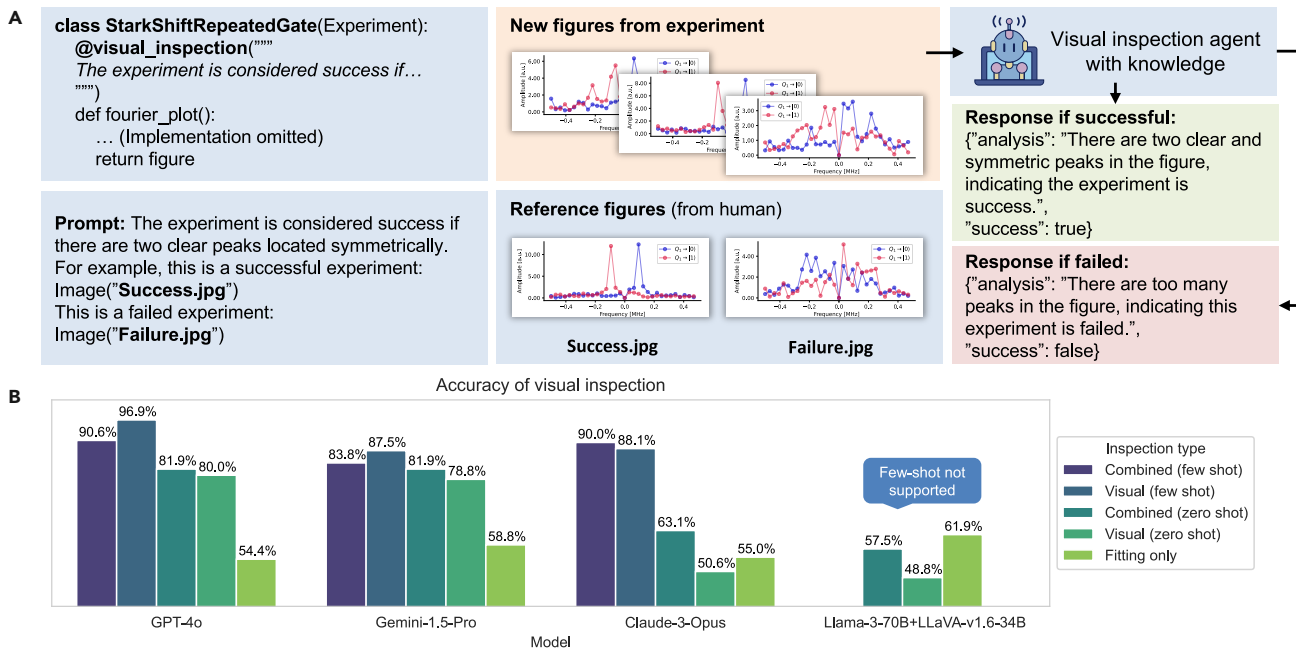


Figure 3. Visual inspection agents and performance benchmarking

(A) Visual inspection agent. Our framework introduces visual inspection agents by attaching decorators to the member methods of the experiment class that produce figures. The decorator accepts a prompt as input, and users can add example figures to the prompt. Whenever the experiment is carried out, the execution agent can call the inspection agents to generate a text-based report based on the outcome figure and their knowledge (prompts).

(B) Performance benchmarking of visual inspection. We compare the performance of the visual inspection method in determining whether a certain experiment is successful or not. We added three types of inspection agents. Visual agents with a text description of success criteria (zero-shot), visual agents with text descriptions and example figures (few-shot), and fitting agents who provide reports based on fitting results. We test the accuracy of inspection on each type of agent. We also combine the results of fitting agents and visual agents in the “combined” setup. We found that providing example figures significantly improves accuracy. Additionally, we found that reports synthesized from multiple inspection agents can give better accuracy in some cases.

the execution first translates the instruction at the stage into executable code with the assistance of translation agents. To increase the accuracy and efficiency of the translation, the execution agent will activate only translation agents related to the context. The relevance score will be calculated based on the embedding similarity between the context and the description of the translation agents. The execution will select a reasonable response from the translation agents and execute the translated code. After executing the code, the agent gathers a natural language report from inspection agents and, based on the report and the transition rules, determines the next stage to transition to. If the reports suggest updating the parameters in the new stage, the execution agent will also pass the update to it. This state-machine architecture enables the execution agent to focus on executing each stage of the procedure efficiently without needing to monitor the entire experimental history when deciding on the next experiment.

RESULTS

In this section, we demonstrate how the k-agents framework can be applied to the calibration and operation of a superconducting quantum processor. Superconducting circuits provide a popular physical platform for building quantum information processors. These circuits are fabricated on a chip using nanofabrication techniques. The circuit components operate under quantum me-

chanical principles at millikelvin temperatures, forming non-linear quantum resonators, which can be excited similar to atoms using microwave signals. Information can be stored in the quantum states of these resonators, which serve as qubits on the processor.^{46–49} To control these qubits, external electronics connected to the chip generate precise microwave signals. These signals must be carefully calibrated to accurately set the operational parameters, which are crucial for executing logical quantum gates. Typically, the parameters include the shape of the pulse (the duration and amplitude at a certain time t) and the initial phase and the frequency of the pulse. For a 2-qubit gate, multiple pulses may be required to send at the same time to generate the entanglement.^{48,50,51}

A significant challenge in superconducting circuit research is identifying these optimal operation parameters. Although automated scripts can determine some parameters, they still require customization by scientists based on different hardware setups. Moreover, programmatically validating the correctness of the results of these scripts is usually difficult to implement. Scientists must continuously monitor experimental status and adjust the calibration scripts to optimize operational parameters. Moreover, these parameters drift over time and occasionally require recalibration. This process is time consuming and repetitive, often not yielding direct results for physical research, yet it is essential for conducting superconducting circuit research. This calibration process is complex and labor intensive, leading to

significant challenges for scaling quantum computing systems. These difficulties are not limited to the superconducting circuit platform, but also extend to ion traps,^{52–54} spin qubits,^{55–58} and Rydberg atom systems.^{59–61} Moreover, the techniques developed for these tasks have broader implications beyond quantum computing, which has great potential to accelerate scientific and technological research in general.

The above challenges fit the targeting scenario of the k-agents framework. We adapt our existing control software stack to the k-agents framework, which enables monitoring experimental results, recommending further parameter adjustment, and executing automated calibrations and experiments. To demonstrate its capabilities and assess effectiveness, we conducted three experiments. Before moving on to the hardware demonstration, we first benchmarked the performance of k-agents in the context of superconducting qubits. We adopted the LeeQ software,⁶² originally used to control superconducting quantum processors in the lab, to work with the k-agents system and provided it with additional multimodal information to determine the success or failure of the experiments (see [Note S4](#) for more detail). First, we benchmarked the translation process using different LLM models, comparing the typical RAG method with our approach. Our results show that the GPT-4o model has the best performance and archives an accuracy of 97%, which we consider sufficient for practical application (see [Figure 2B](#)). In practice, when code generation fails, producing unusable code or syntax errors, we prompt the agent to retry, further improving the success rate. We also evaluated the inspection agents to determine if they could accurately identify the success or failure of experiments. We found that the best performance was achieved when using inspection agents constructed with example figures (see [Figure 3B](#)).

Following the successful benchmarking results, we designed the experiments to further evaluate the performance of k-agents on real hardware. First, we demonstrate fully automated recalibration of single-qubit parameters. Second, we showcase the automated discovery and calibration of 2-qubit gates on our platform. Third, we highlight the AI agent’s ability to generate quantum states based on natural language descriptions, including the creation of a GHZ state across 3 qubits, utilizing previous calibrations.

Our experimental platform is a 16-qubit superconducting quantum processor configured in a four-by-four square lattice connectivity layout. The characterization details of this device have been reported in [Note S6](#). The qubits employed are standard fixed-frequency transmon qubits with coaxial geometry.^{63,64} For the demonstration of our framework, we selected a subset of three adjacent qubits. In superconducting qubit systems, 2-qubit gates usually have to be implemented between adjacent qubits, and the crosstalk and noise are more significant. Our demonstration using an adjacent 3-qubit subset is consistent with current hardware constraints and design practices. Although it involves only a few qubits, this setup reflects the typical conditions of academic labs, where small-scale experiments suffice for exploring quantum phenomena and improving qubit performance. For further details about the hardware setup, please refer to [Note S6](#).

In the first experiment, we demonstrate automated calibration and benchmarking of a single qubit with k-agents. To facilitate

the calibration process, we provided the agent with a document detailing the general procedure for single-qubit calibration in natural language. At the start of the experiment, we manually adjusted the initial parameters to deviate from their optimal values. Based on the document, the execution agent first decomposed the procedure into two distinct stages: calibration and benchmarking. The calibration stage was further broken down into multiple steps, in which the agent sequentially adjusted the qubit’s frequency, amplitude, and DRAG (derivative removal by adiabatic gate) parameters. If a failure occurred during the procedure, the step would be retried. After several unsuccessful attempts, it would revert to the previous stage to try again. This process is driven by the agent to emulate the typical behavior of a human scientist who implements the calibration process. We observed that the agent was able to perform transition correctly between stages based on the experiment result and found the acceptable parameters. At the benchmarking stage, the agent successfully performed a randomized benchmarking on the single qubit and validated its fidelity (see [Figure 4](#)).

Following the single-qubit calibration experiment, we demonstrated the automated discovery of 2-qubit gate parameters. Specifically, we focused on the siZZle (Stark-induced ZZ by level excursions; ZZ is an interaction in which two qubits influence each other’s states) gate,^{65–68} which can be used to create entanglement between two fixed-frequency superconducting qubits. By driving both qubits off resonance at the same frequency simultaneously, the ZZ term in the qubits’ Hamiltonian is altered compared to when the qubits are undriven. Leveraging this effect, we constructed a pulse sequence with calibrated amplitude and frequency to adjust the ZZ interaction strength, followed by fine-tuning the pulse duration to achieve precise qubit entanglement. The most challenging part of the process was the search for optimal driving amplitude and frequency, while the pulse duration calibration was relatively straightforward, accomplished programmatically by measuring the ZZ interaction rate during the drive. In this experiment, the AI agent autonomously identified a set of working parameters for the siZZle gate.

For a human scientist, searching these parameters typically requires starting with empirical knowledge and measuring the ZZ interaction strength at specific driving frequencies and amplitude. The search process then proceeds by adjusting the frequency and amplitude based on the experimental outcomes. When an experiment succeeds, we often increase the pulse amplitude to enhance the ZZ interaction rate or adjust the driving frequency closer to the qubit’s transition frequency. Conversely, when an experiment fails, we either decrease the amplitude or move the frequency away from the transition frequency. There are three regions that need to be explored: below both qubits’ transition frequencies, between the two, and above both qubits’ transition frequencies. We inject this empirical knowledge of parameter selection into two LLM-based agents that are accessible to the experiment history for what parameters have been tried. By wrapping these agents into two special experiments, the execution agent can call them to get the next frequency and amplitude to try. The parameter searching can then be formulated as an alternate calling to these special experiments and the experiments for testing the proposed parameters.

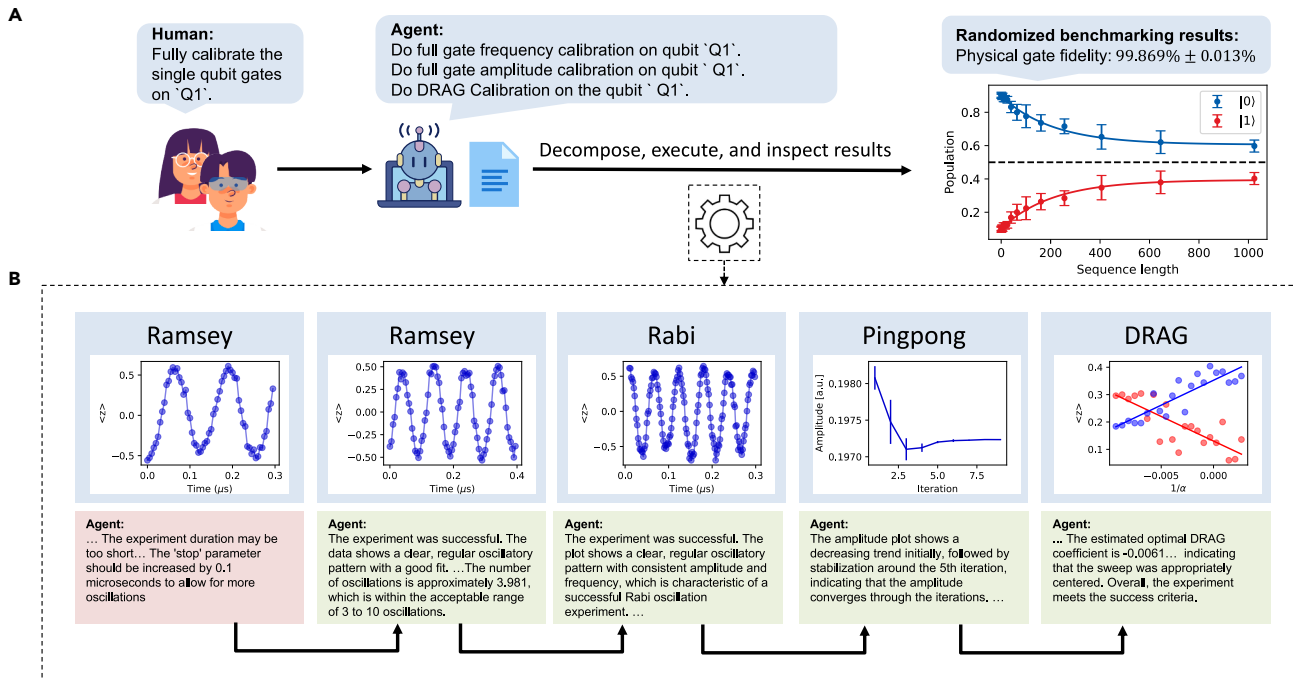


Figure 4. Automated single-qubit parameter recalibration driven by k-agents

(A) The human scientist instructs the k-agents framework for calibrating the single-qubit gate parameters. Based on the reference documents provided in natural language, k-agents successfully breaks the calibration into four steps, followed by the randomized benchmarking experiment to characterize the single-qubit gate fidelity.

(B) In this example, k-agents finds that the first Ramsey experiment had not collected enough oscillations to estimate the frequency. Therefore, it repeats this experiment by increasing the experiment time. The following experiments all pass the success criteria of the k-agents. In the end, k-agents implements the randomized benchmarking and reports the gate fidelity.

For each parameter pair (frequency and amplitude), we measured the ZZ interaction rate by performing Hamiltonian tomography on the ZZ term using two methods: continuous-time tomography and repeated-gate tomography. The latter method, which accounts for the effects of the pulse's rising and falling edges, yielded more accurate ZZ interaction measurements per pulse.

The most challenging aspect was determining the success of each experiment. To address this, we plotted the Fourier transform of the Hamiltonian tomography results and let an inspection agent decide whether the experiment was successful, where the agent is equipped with a few-shot visual knowledge about the success criteria. Furthermore, we monitored the state of the control qubit, plotting its expectation value of $\langle Z \rangle$, and tasked another inspection agent with ensuring that it remained stable during the driving process, avoiding any unwanted excitation.

We ran the experiment for 3 h, limiting the execution agent to carrying out 100 experiments to search for parameters and test a maximum of 20 frequencies. By the end of the experiment (see Figure 5), the agent successfully identified an optimal set of parameters (frequency 4,726 MHz and amplitude 0.3049).

Finally, we demonstrate the ability of the AI agent to carry out experiments based on natural language instructions. After successfully calibrating the gates, we instructed the execution agent to perform state tomography for a GHZ state. Additionally, we requested the agent to perform process tomography for the 2-qubit gate parameters it had calibrated. The agent success-

fully implemented this experiment and reported the state fidelity (see Figure 6).

DISCUSSION

In this study, we introduce the k-agents framework for creating laboratory-related knowledge agents and demonstrate its capability to automate experiments. We applied the system in a real-world setting, successfully demonstrating the ability of k-agents by performing single-qubit and 2-qubit gate calibrations as well as generating and characterizing a GHZ state on a quantum processor based on human instructions. The k-agents succeeded in orchestrating the experiments, interpreting the multimodal experimental results, and guiding the execution of the experiment with closed-loop feedback. These achievements suggest that our system is a valuable tool for research groups working with superconducting quantum processors, with the potential for broader applications in research automation across other fields. Additionally, we also applied two benchmark experiments on k-agents. We found that our agent-based translation of natural language instructions outperforms standard RAG approaches in terms of accuracy. By our benchmark of the inspection agents, we showed that current multimodal LLMs have the ability to analyze experiment figures based on human instructions, especially when provided with example figures.

While methods such as rule-based systems, reinforcement learning, and Bayesian optimization have been used for

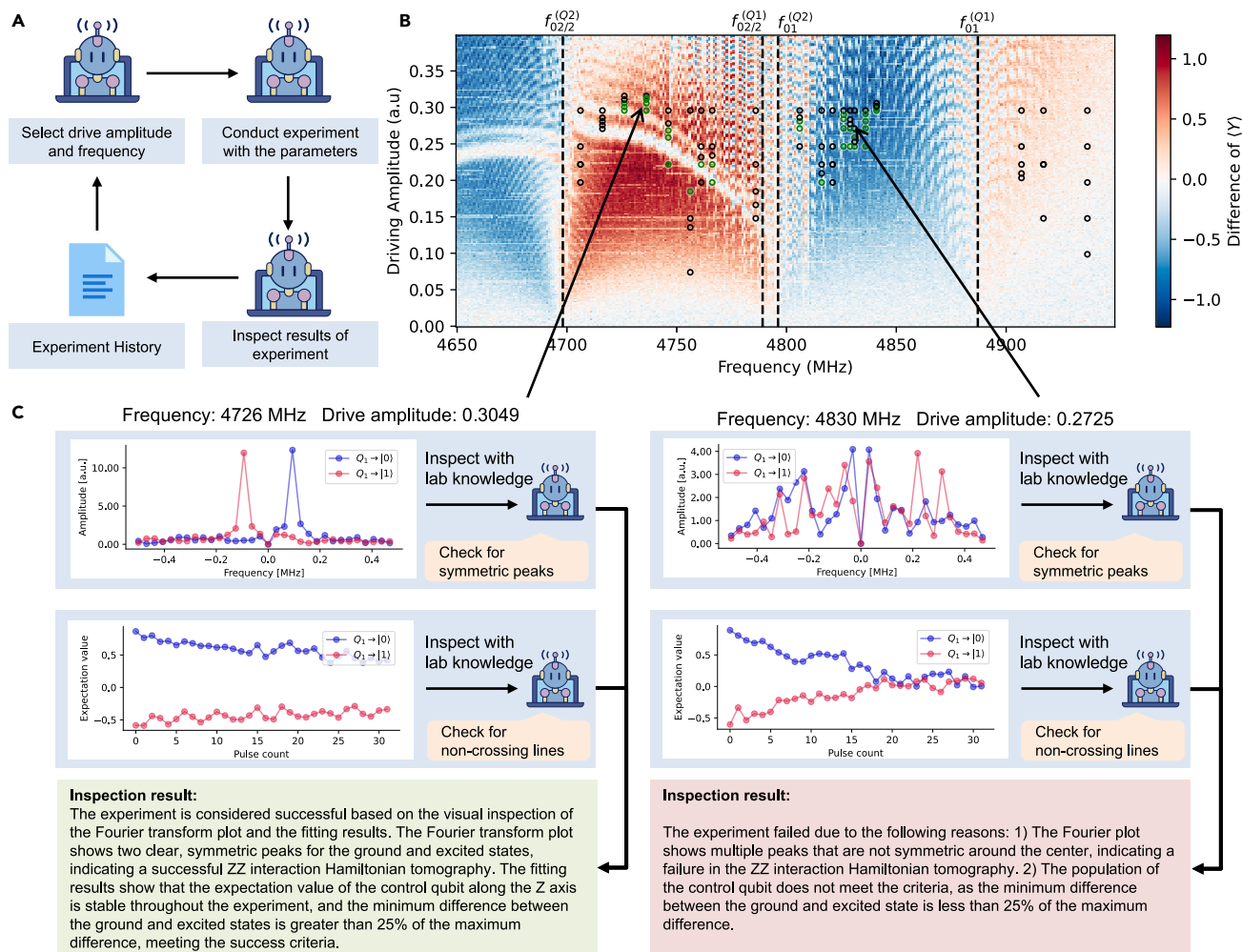


Figure 5. Two-qubit gate parameter discovery driven by k-agents

(A) The workflow of the discovery process, where k-agents continuously reads the history of previous experiments and suggests the next parameter set likely to succeed.

(B) The set of discovered parameters and the corresponding parameter search space. The background indicates the ZZ interaction strengths visualized from data collected during an overnight experiment of which the agent has no prior knowledge. The green circles denote locations where the agent identified an acceptable parameter set, while the black regions indicate failed attempts. For more details on the search for siZZle gate parameters, refer to [Note S5](#).

(C) Examples of figures observed by the agent, along with its response, illustrating a typical success case and a failure case, respectively.

calibrating quantum devices, our approach is fundamentally different. First, the k-agents framework is designed to automate entire experimental procedures, rather than focusing on the tuning of specific parameters. Optimization methods like reinforcement learning and Bayesian optimization typically require extensive manual tuning and carefully crafted reward functions and are limited to narrow, predefined tasks. In contrast, our framework provides a more general, interpretable, and adaptable solution, particularly well suited to dynamic and evolving quantum laboratory environments. Moreover, unlike traditional rule-based systems that depend on rigid, hand-coded logic, the k-agents framework uses multimodal prompts (natural language and images) and LLMs to enable more flexible and efficient automation.

One of the strengths of the k-agents framework is its approach to knowledge representation. It requires no more effort than maintaining standard laboratory documentation and fits naturally

into routine protocol management. While we acknowledge that adopting any new framework involves some onboarding and training, the setup effort is comparable to preparing laboratory manuals or protocols for new PhD students or junior staff. Once standard documents already exist—which is common practice in most laboratories—our agents can be configured with minimal additional work.

While our framework shares conceptual similarities with prior agent systems like ChemCrow³² and Voyager,⁶⁹ it introduces several important distinctions that enhance agent-based AI for laboratory automation. First, our framework is specifically tailored for quantum computing laboratories, where experiment automation involves breaking down long-horizon workflows and dynamically adjusting parameterized experiments by analyzing previous results. In contrast, ChemCrow targets open-ended challenges in general chemistry, focusing on relatively straightforward,

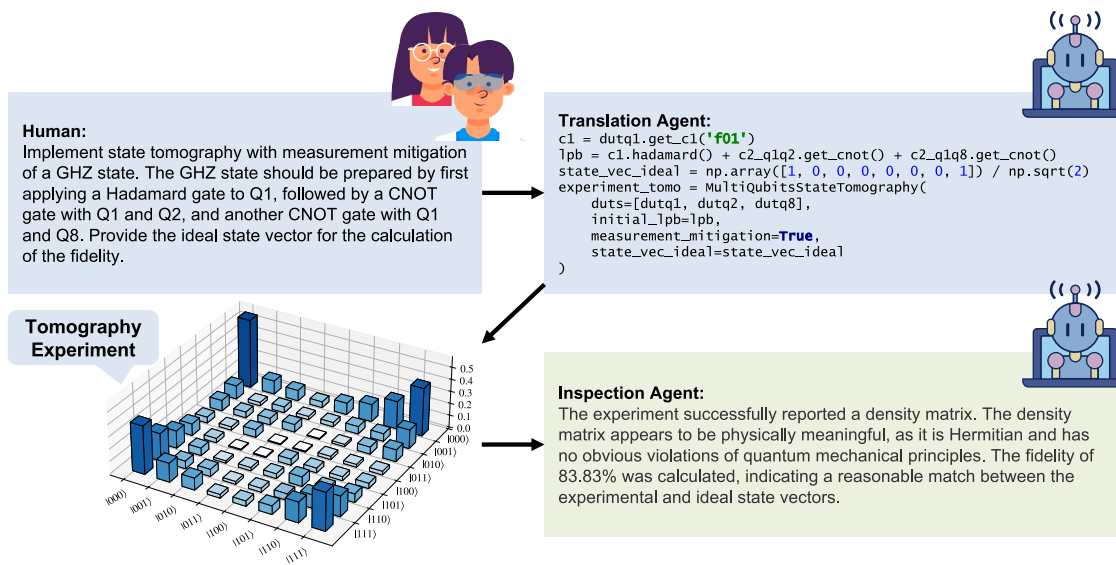


Figure 6. Automation of the GHZ state generation and tomography with k-agents

First, the instructions are provided by human scientists. Then, the code is generated by the translation agent to implement the experiment. The density matrix plot of the generated GHZ state is constructed from the experiment result. Finally, a report in natural language is generated by an inspection agent.

short-horizon tasks that require less iterative reasoning and fewer reexecutions. Second, unlike Voyager, which utilizes large internet-scale corpora about Minecraft that has already trained into LLMs, our system operates effectively without depending on such extensive external data, making it better suited to laboratory environments. Additionally, Voyager develops agent skills through simulated environments—a resource not always available in laboratory settings—whereas our k-agents framework emphasizes practical application by building on existing laboratory code bases. These distinctions reflect our commitment to improving the practicality and scalability of agent-based AI for real-world scientific experimentation.

However, the k-agents framework has some potential weaknesses that make full automation challenging. First, our method relies on clean and well-formatted code bases and procedure documents. However, such structured knowledge may not always exist, and transforming existing knowledge could take considerable human effort. Future research on structuralizing unstructured documents and code bases is still needed to completely automate knowledge translation. A possible direction of research could involve enabling the framework to generate structured code and procedures based on unstructured project files and human inputs. Second, using LLM, the execution of k-agents exhibits higher latency than that of traditional automation methods that do not require LLM-based planning and translation of knowledge at runtime. We note that this problem can be well mitigated by caching the response of LLMs and using the same response whenever the input is the same.

In addition, our current implementation of execution agents within the agent-based state machine does not support fine-grained, interruptible execution; it allows only for termination of the entire program in response to external signals or emergency conditions. However, we acknowledge that in quantum systems,

it would be beneficial to provide an interactive mechanism for human scientists to monitor and intervene in the experiment’s progress. Addressing this limitation is a priority for future development, and we plan to explore the integration of real-time interrupt mechanisms, hardware-level hooks, and human-in-the-loop safety protocols.

We monitor the cost of the k-agents system. During the parameter search for the 2-qubit gate, we observed that the LLM used 1,373,207 input tokens (including image tokens) and 168,039 output tokens in 3 h, costing less than US\$5.00. We believe that there is potential for further optimization to reduce these costs. Regarding time efficiency, despite network delays, LLMs currently take longer than humans to perform inspections. The inspection agent utilizes multimodal LLMs and requires a few seconds to evaluate each experimental result, with an inference time comparable to that of the standard LLM agents without the multimodal ability. However, LLMs tend to be more efficient than humans in generating code. Therefore, overall efficiency is comparable.

The use of superconducting quantum processors simplifies some challenges, such as safety control and error tolerance. In our system, the risk of the agent executing code that causes significant damage or unacceptable costs is very low. If it fails, we can simply allow it to try again. However, this may not be the case in other applications. Prior research^{1,70,71} has focused heavily on safety control, and this is an area we plan to explore further in the future.

METHODS

Details on the knowledge agents developed

Definition of knowledge agents

The concept of knowledge agents is defined based on the work proposed by Zhang et al.,¹⁵ in which an agent’s learning ability is

defined as its ability to internalize knowledge given related inputs. A knowledge agent can then be interpreted as an agent with the ability to learn (i.e., acquire knowledge) and to apply the learned knowledge to respond to other agents (i.e., transfer knowledge). A bare LLM can be regarded as a knowledge agent as it is able to absorb knowledge from its training data. However, it is difficult for LLMs to internalize laboratory documents and codes as their knowledge because of the current pitfalls in model editing.²⁴ Therefore, we regard bare LLMs as weak knowledge agents compared to the knowledge agents we proposed, which are good at accepting knowledge from laboratory documentation.

Translation agents

In k-agents, we included two types of translation agents: “code translation agent,” which translates instructions into code, and “procedure translation agent,” which translates instructions into experiment procedures that may contain multiple institutions. Each translation agent is designed to process only instructions that are very close to the description of an existing experiment. However, by recursively translating the instructions, the whole system can achieve higher generalizability.

Initialization. Each code translation agent is directly constructed from a class of experiments. Similarly, procedure translation agents are also constructed from the structured documents of procedures mentioned above. The document translation agent will try to translate the instructions that resemble the title of the experiment item into code that invokes an execution agent to execute the corresponding procedure.

Activation. Many translation agents might exist based on the complexity of the experimental system. However, not all translation agents are needed to translate a certain instruction. Therefore, we designed the system to use embedding similarity and to activate only the agents related to the instruction to be translated. When generating the agent, we use LLMs to generate a series of natural language instructions $\{I\}$ that the agent should translate. We further calculate the embeddings $\{\vec{E}\}$ of these instructions. When there is an incoming instruction with embedding \vec{E}_i , the score S of a translation agent is calculated by the maximal inner product between E_i and the vectors in $\{\vec{E}\}$:

$$S = \max_{\vec{E}_i \in \{\vec{E}\}} \vec{E}_i \cdot \vec{E}_i^\dagger. \quad (\text{Equation 1})$$

Code translation. When activated, the code translation agent attempts to translate the instruction into executable code. The Python class signature for the experiment and the instruction will be passed to an LLM to generate a code translation based on the class. We employ two strategies to improve the translation accuracy throughout this process. First, we implement the chain-of-thought (CoT)⁷² strategy to improve performance. Before generating code, the LLM is first asked to provide a paragraph of analysis on how to approach the translation. This helps guide the LLM in retrieving key points from the context and making useful inferences. Next, we adopt a self-reflection strategy to improve the accuracy. The agent will determine whether the instruction should be translated using the experiment class based on the previous analysis. This step selects out cases where the class is not suitable for the instruction, helping to avoid hallucinations. If the agent con-

cludes that the instruction should be translated into code using the class, it proceeds to generate a candidate code snippet. This candidate code is then sent to the execution agent for comparison with other potential candidates.

Procedure translation. Our procedure translation agents are designed similarly to code translation agents. If an agent deems an incoming instruction matches the procedure it holds, it will try to rewrite the procedure title to suit the instruction. A code snippet that calls an execution agent to execute the rewritten title will then be generated and sent to the execution agent as a code candidate.

Generalizability. In both types of translations, the agents generate new code and new instructions based on the input instruction and therefore generalize the knowledge they have. However, this generalizability is well controlled, as translations that deviate too much from the agent’s knowledge will be decided as improper by the strategy we described. This limited generalizability mitigates the negative effect of the LLM’s knowledge from the public corpus that is not suitable for a private experimental setup.

Details on execution agents

In k-agents, the execution of experiments is controlled by the execution agents. The execution process involves decomposing the incoming procedure into experiment stages, translating the instruction at each stage into code by translation agents, executing the generated code, and analyzing the experiment results to determine the next step.

Instruction decomposition

When handling experiment procedures, we ask the execution agent to decompose the instructions into experimental stages and make an agent-based state machine. Each stage in the agent-based state machine contains a single step of the instructions that includes only one experiment or subprocedure to be translated by the translation agents. In addition, an experiment procedure might have a complex control flow. For example, selecting the next instruction to execute might depend on the result of the previous instructions. Therefore, at each stage, we also attach a prompt on how to select the next stage given the result of this stage.

The stage generation is done roughly in two calls to an LLM. First, we employ the LLM to extract a list of independent experiments from the procedure. This step removes all the sentences controlling the procedure’s progression and generates the instructions at each stage. Then, we add indices to the stages, such as stage 1 and stage 2, so their indices can be used to refer to them when generating the transition rules. We also add two special stages, COMPLETE and FAILED, to the list of stages. Finally, we provide the LLM with the list of stages and the original procedure, asking it to attach the transition rule for each experiment.

Translation candidate selection

When executing a stage, the execution agent activates the translation agents and gathers candidate code from them. Then, the execution agent synthesizes the code to execute on the basis of the candidates it receives. Specifically, all the candidate code will be input into an LLM with a prompt that asks the LLM to generate an analysis and the final code for the translation. This synthesis process has two merits. First, there might be

Algorithm 1. Translation with self-reflection

- (1) Set the group of translation agents $\{A\}$, the number N_k for the number of activated agents, and the number N_{\max} for the maximal number of activated agents.
- (2) Input the instruction I to be translated.
- (3) Calculate the score of each agent in $\{A\}$ by Equation 1.
- (4) For each agent A whose score is ranked top N_k in $\{A\}$, do the following:
 - (a) Using LLM, check whether the instruction can be translated by the knowledge held by A . If so, generate the code C that implements the instruction. Else, output nothing.
 - (b) Add C to the set of code candidates.
- (5) If the set of code candidates $\{C\}$ is empty, increase the number of N_k by 2 and do the following: if $N_k < N_{\max}$, go back to step 2. Else, the algorithm fails.
- (6) If $\{C\}$ is non-empty, add the instruction and the set of code candidates $\{C\}$ to the context of LLM and generate the code C^* for I .
- (7) Output C^* as the result of translation.

two experiments with similar descriptions. The translation agents for them are likely to generate candidate code together. By providing them as candidates, we make it possible for LLM to analyze which solution is more suitable. Further, as the activating mechanism might not be perfect, there might not be any answer from the translation agents because the desired translation agent is not activated. In this case, we design the execution agent to try to activate more translation agents until a proper candidate appears. We summarize the translation process in Algorithm 1. When applying the algorithm in Section Results, we set $N_k = 3$ and $N_{\max} = 9$.

Execution

Starting from the first stage, the execution agent executes the instruction in the current stage by executing the translated code with the help of the translation agents, as described in the previous section. After execution, the inspection agent generates a report of the experiment in natural language. Based on the report, the execution agent chooses the next state to transition to. Finally, after reaching the COMPLETE or the FAILED state, the execution agent will generate a report of the entire execution. We summarize the execution process in Algorithm 2.

We note that the execution agent also maintains a variable table. At the end of each experiment, variables might be injected from the experiment for the use of other experiments. The injection of variables can be programmed by the users when they implement the class for the experiment.

Quantum processor hardware

The quantum processor used in this experiment consists of 16 coaxmon qubits arranged in a square lattice configuration. Coaxmon qubits, a variant of transmon qubits with coaxial geometry and off-chip wiring, have demonstrated high coherence, low crosstalk, and stable higher excited states.^{63,64,73,74}

To enable the execution agents to conduct experiments, we integrated the k-agents framework with the LeeQ framework,⁶² customized software used for controlling superconducting quantum processors in the lab. The LeeQ framework, which has been independently used by human scientists, manages the composition, compilation, and optimization of quantum gates into executable instructions. These processed microwave sequences are then sent to the QubiC system, an electronic system that generates and processes microwave signals and connects to the quantum processor.

In practice, human scientists write code that predefines experiments and reuse it in different scenarios by modifying the arguments of these experiments. Typically, scientists manually adjust the parameters of these experiments until they achieve their desired outcome. We adapted these predefined experiments to be compatible with and indexable by k-agents. In total, the LeeQ framework includes more than 40 built-in experiments, of which we selected 17 for indexing by the k-agents. For further details on the experiments, please refer to Note S4.

Algorithm 2. Execution agent workflow

- (1) Input the experiment procedure P .
- (2) Using LLM, decompose P into experiment stages (states) $\{S\}$ and transition rules $\{T\}$ at each stage. The decomposition also specifies an initial state S_0 as the current state S^* . Two final states, COMPLETE and FAILED, are also in the $S\{\}$.
- (3) At the current experiment stage S^* , do the following:
 - (a) Translate the instruction contained in S^* into code with Algorithm 1.
 - (b) Execute the translated code.
 - (c) Activate available inspection agents in the executed experiment and summarize the results from them with LLM, giving a summarized report R . Add S^* and R to the experiment history.
 - (d) Using LLM, decide the next stage to transition to, based on the transition rule T^* of S^* and the report R .
 - (e) If the next stage is COMPLETE or FAILED, add it to the experiment history and break the loop (i.e., go to step 4). Else, set S^* to be the next stage and go to step 3(a).
- (4) Using LLM, summarize the experiment history and produce a report R for P .

RESOURCE AVAILABILITY

Lead contact

Requests for further information and resources should be directed to and will be fulfilled by the lead contact, Alán Aspuru-Guzik (alan@aspuru.com).

Materials availability

This study did not generate new unique reagents.

Data and code availability

The source codes for k-agents can be found in Zhang.⁷⁵ The codes for the experiments and benchmarks can be found in Cao et al.⁶²

ACKNOWLEDGMENTS

The authors thank Yuning Zhang for insightful discussions. This project is supported by Schmidt Sciences, LLC. S.C. acknowledges support from Schmidt Science. P.L. acknowledges support from the EPSRC (EP/T001062/1, EP/N015118/1, and EP/M013243/1). M.B. acknowledges support from an EPSRC QT Fellowship grant (EP/W027992/1). A.A.-G. thanks Anders G. Frøseth for his generous support. A.A.-G. also acknowledges the generous support of Natural Resources Canada and the Canada 150 Research Chairs program. This research is part of the University of Toronto's Acceleration Consortium, which receives funding from the Canada First Research Excellence Fund (CFREF).

AUTHOR CONTRIBUTIONS

S.C.: conceptualization, investigation, methodology, software, validation, writing – original draft, and writing – review & editing; Z.Z.: conceptualization, methodology, software, validation, writing – original draft, and writing – review & editing; M.A.: investigation and writing – review & editing; S.D.F.: investigation and writing – review & editing; M.P.: investigation; M.B.: resources; P.L.: resources; A.A.-G.: supervision and project administration.

DECLARATION OF INTERESTS

A.A.-G. is a founder of Kebotix, Inc., a company specializing in closed-loop molecular discovery, and IntrepidLabs, Inc., a company using self-driving laboratories for pharmaceuticals. P.L. is the founder and chief science officer of Oxford Quantum Circuits Limited, a company developing superconducting circuit quantum computers.

DECLARATION OF GENERATIVE AI AND AI-ASSISTED TECHNOLOGIES IN THE WRITING PROCESS

During the preparation of this work, the authors used ChatGPT in order to check for grammar errors and typos and improve the clarity of the writing. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at <https://doi.org/10.1016/j.patter.2025.101372>.

Received: March 5, 2025
Revised: June 5, 2025
Accepted: August 27, 2025
Published: September 23, 2025

REFERENCES

- Tom, G., Schmid, S.P., Baird, S.G., Cao, Y., Darvish, K., Hao, H., Lo, S., Pablo-García, S., Rajaonson, E.M., Skreta, M., et al. (2024). Self-driving laboratories for chemistry and materials science. *Chem. Rev.* **124**, 9633–9732.
- Sim, M., Vakili, M.G., Strieth-Kalthoff, F., Hao, H., Hickman, R.J., Miret, S., Pablo-García, S., and Aspuru-Guzik, A. (2024). Chemos 2.0: An orchestration architecture for chemical self-driving laboratories. *Matter* **7**, 2959–2977.
- Seifrid, M., Strieth-Kalthoff, F., Haddadnia, M., Wu, T.C., Alca, E., Bodo, L., Arellano-Rubach, S., Yoshikawa, N., Skreta, M., Keunen, R., and Aspuru-Guzik, A. (2024). Chempy: an open-source python interface for chemspeed robotic chemistry and materials platforms. *Digital Discovery* **3**, 1319–1326.
- Stein, H.S., and Gregoire, J.M. (2019). Progress and prospects for accelerating materials science with automated and autonomous workflows. *Chem. Sci.* **10**, 9640–9649.
- Shi, Y., Prieto, P.L., Zepel, T., Grunert, S., and Hein, J.E. (2021). Automated experimentation powers data science in chemistry. *Acc. Chem. Res.* **54**, 546–555.
- Xie, Y., Sattari, K., Zhang, C., and Lin, J. (2023). Toward autonomous laboratories: Convergence of artificial intelligence and experimental automation. *Prog. Mater. Sci.* **132**, 101043.
- Yu, X.J., Chi, X., Wee, A.T.S., Rusydi, A., and Breese, M.B.H. (2019). A scripting LabVIEW based program for experiment automation in synchrotron radiation applications. *Rev. Sci. Instrum.* **90**, 103902.
- Seifrid, M., Pollice, R., Aguilar-Granda, A., Morgan Chan, Z., Hotta, K., Ser, C.T., Vestfrid, J., Wu, T.C., and Aspuru-Guzik, A. (2022). Autonomous chemical experiments: Challenges and perspectives on establishing a self-driving lab. *Acc. Chem. Res.* **55**, 2454–2466.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **33**, 1877–1901.
- OpenAI (2023). Gpt-4 technical report. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2303.08774>.
- Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A.M., Hauth, A., et al. (2023). Gemini: a family of highly capable multimodal models. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2312.11805>.
- Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2407.21783>.
- Liu, H., Li, C., Wu, Q., and Lee, Y.J. (2024). Visual instruction tuning. *Advances in Neural Information Processing Systems* **36**.
- Liu, H., Li, C., Li, Y., and Lee, Y.J. (2024). Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 26296–26306.
- Zhang, Z., Aronowitz, S., and Aspuru-Guzik, A. (2024). A theory of understanding for artificial intelligence: composability, catalysts, and learning. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2408.08463>.
- Li, G., Hammoud, H., Itani, H., Khizbullin, D., and Ghanem, B. (2023). Camel: Communicative agents for “mind” exploration of large language model society. *Adv. Neural Inf. Process. Syst.* **36**, 51991–52008.
- Chan, C.M., Chen, W., Su, Y., Yu, J., Xue, W., Zhang, S., Fu, J., and Liu, Z. (2023). Chateval: Towards better llm-based evaluators through multi-agent debate. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2308.07201>.
- Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., and Wang, C. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation framework. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2308.08155>.
- Talebirad, Y., and Nadiri, A. (2023). Multi-agent collaboration: Harnessing the power of intelligent llm agents. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2306.03314>.
- Zhang, Y., Yang, S., Bai, C., Wu, F., Li, X., Li, X., and Wang, Z. (2024). Towards efficient llm grounding for embodied multi-agent collaboration. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2405.14314>.

21. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., and Zhang, X. (2024). Large language model based multi-agents: A survey of progress and challenges. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2402.01680>.
22. Li, X., Wang, S., Zeng, S., Wu, Y., and Yang, Y. (2024). A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth* 7, 9.
23. Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2001.08361>.
24. Li, Z., Zhang, N., Yao, Y., Wang, M., Chen, X., and Chen, H. (2024). Unveiling the pitfalls of knowledge editing for large language models. In *The Twelfth International Conference on Learning Representations* <https://openreview.net/forum?id=fNktD3ib16>.
25. Hoelscher-Obermaier, J., Persson, J., Kran, E., Konstas, I., and Barez, F. (2023). Detecting edit failures in large language models: An improved specificity benchmark. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2305.17553>.
26. Zhang, N., Yao, Y., Tian, B., Wang, P., Deng, S., Wang, M., Xi, Z., Mao, S., Zhang, J., Ni, Y., et al. (2024). A comprehensive study of knowledge editing for large language models. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2401.01286>.
27. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Adv. Neural Inf. Process. Syst.* 33, 9459–9474.
28. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., and Wang, H. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2312.10997>.
29. Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.t. (2020). Dense passage retrieval for open-domain question answering. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2004.04906>.
30. Franklin, A., and Perovic, S. (1998). Experiment in physics. <https://plato.stanford.edu/entries/physics-experiment/#pagetoprighit>.
31. Boiko, D.A., MacKnight, R., Kline, B., and Gomes, G. (2023). Autonomous chemical research with large language models. *Nature* 624, 570–578.
32. Bran, A., M., Cox, S., Schilter, O., Baldassari, C., White, A.D., and Schwaller, P. (2024). Augmenting large language models with chemistry tools. *Nat. Mach. Intell.* 1, 11.
33. Darvish, K., Skreta, M., Zhao, Y., Yoshikawa, N., Som, S., Bogdanovic, M., Cao, Y., Hao, H., Xu, H., Aspuru-Guzik, A., et al. (2024). Organa: A robotic assistant for automated chemistry experimentation and characterization. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2401.06949>.
34. Yoshikawa, N., Skreta, M., Darvish, K., Arellano-Rubach, S., Ji, Z., Bjørn Kristensen, L., Li, A.Z., Zhao, Y., Xu, H., Kuramshin, A., et al. (2023). Large language models for chemistry robotics. *Auton. Robots* 47, 1057–1086.
35. Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2023). Lost in the middle: How language models use long contexts. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2307.03172>.
36. Li, T., Zhang, G., Do, Q.D., Yue, X., and Chen, W. (2024). Long-context llms struggle with long in-context learning. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2404.02060>.
37. Li, D., Shao, R., Xie, A., Sheng, Y., Zheng, L., Gonzalez, J., Stoica, I., Ma, X., and Zhang, H. (2023). How long can context length of open-source llms truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.
38. Acharya, R., Aghababae-Beni, L., Aleiner, I., Andersen, T.I., Ansmann, M., Arute, F., Arya, K., Asfaw, A., Astrakhantsev, N., Atalaya, J., et al. (2024). Quantum error correction below the surface code threshold. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2409.15065>.
39. Reichardt, B.W., Aasen, D., Chao, R., Chernoguzov, A., van Dam, W., Gaebler, J.P., Gresh, D., Lucchetti, D., Mills, M., Moses, S.A., et al. (2024). Demonstration of quantum computation and error correction with a tesseract code. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2409.04628>.
40. Putterman, H., Noh, K., Hann, C.T., MacCabe, G.S., Aghaeimeibodi, S., Patel, R.N., Lee, M., Jones, W.M., Moradinejad, H., Rodriguez, R., et al. (2024). Hardware-efficient quantum error correction using concatenated bosonic qubits. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2409.13025>.
41. Brock, B.L., Singh, S., Eickbusch, A., Sivak, V.V., Ding, A.Z., Frunzio, L., Girvin, S.M., and Devoret, M.H. (2024). Quantum error correction of qubits beyond break-even. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2409.15065>.
42. Krenn, M., Pollice, R., Guo, S.Y., Aldeghi, M., Cervera-Lierta, A., Friederich, P., dos Passos Gomes, G., Häse, F., Jinich, A., Nigam, A., et al. (2022). On scientific understanding with artificial intelligence. *Nat. Rev. Phys.* 4, 761–769.
43. Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. (2024). τ -bench: A benchmark for tool-agent-user interaction in real-world domains. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2406.12045>.
44. Wang, H., Wang, R., Xue, B., Xia, H., Cao, J., Liu, Z., Pan, J.Z., and Wong, K.F. (2024). Appbench: Planning of multiple apis from various apps for complex user instruction. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2410.19743>.
45. Barres, V., Dong, H., Ray, S., Si, X., and Narasimhan, K. (2025). τ 2-bench: Evaluating conversational agents in a dual-control environment. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2506.07982>.
46. Blais, A., Huang, R.S., Wallraff, A., Girvin, S.M., and Schoelkopf, R.J. (2004). Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation. *Phys. Rev.* 69, 062320.
47. Acharya, R., Aleiner, I., Allen, R., Andersen, T.I., Ansmann, M., Arute, F., Arya, K., Asfaw, A., Atalaya, J., Babbush, R., et al. (2023). Suppressing quantum errors by scaling a surface code logical qubit. *Nature* 614, 676–681.
48. Krantz, P., Kjaergaard, M., Yan, F., Orlando, T.P., Gustavsson, S., and Oliver, W.D. (2019). A quantum engineer's guide to superconducting qubits. *Appl. Phys. Rev.* 6, 021318.
49. Kjaergaard, M., Schwartz, M.E., Braumüller, J., Krantz, P., Wang, J.I.J., Gustavsson, S., and Oliver, W.D. (2020). Superconducting qubits: Current state of play. *Annu. Rev. Condens. Matter Phys.* 11, 369–395.
50. Sheldon, S., Magesan, E., Chow, J.M., and Gambetta, J.M. (2016). Procedure for systematically tuning up cross-talk in the cross-resonance gate. *Phys. Rev.* 93, 060302.
51. Wei, K.X., Lauer, I., Pritchett, E., Shanks, W., McKay, D.C., and Javadi-Abhari, A. (2024). Native two-qubit gates in fixed-coupling, fixed-frequency transmons beyond cross-resonance interaction. *PRX Quantum* 5, 020338.
52. Maksymov, A., Niroula, P., and Nam, Y. (2021). Optimal calibration of gates in trapped-ion quantum computers. *Quantum Sci. Technol.* 6, 034009.
53. Romaszko, Z.D., Hong, S., Siegele, M., Puddy, R.K., Lebrun-Gallagher, F. R., Weidt, S., and Hensinger, W.K. (2020). Engineering of microfabricated ion traps and integration of advanced on-chip features. *Nat. Rev. Phys.* 2, 285–299.
54. Bruzewicz, C.D., Chiaverini, J., McConnell, R., and Sage, J.M. (2019). Trapped-ion quantum computing: Progress and challenges. *Appl. Phys. Rev.* 6, 021314.
55. Botzem, T., Shulman, M.D., Foletti, S., Harvey, S.P., Dial, O.E., Bethke, P., Cerfontaine, P., McNeil, R.P.G., Mahalu, D., Umansky, V., et al. (2018). Tuning methods for semiconductor spin qubits. *Phys. Rev. Appl.* 10, 054026.
56. Burkard, G., Ladd, T.D., Pan, A., Nichol, J.M., and Petta, J.R. (2023). Semiconductor spin qubits. *Rev. Mod. Phys.* 95, 025003.

57. Chatterjee, A., Stevenson, P., De Franceschi, S., Morello, A., de Leon, N. P., and Kuemmeth, F. (2021). Semiconductor qubits in practice. *Nat. Rev. Phys.* *3*, 157–177.
58. Stano, P., and Loss, D. (2022). Review of performance metrics of spin qubits in gated semiconducting nanostructures. *Nat. Rev. Phys.* *4*, 672–688.
59. Wu, X., Liang, X., Tian, Y., Yang, F., Chen, C., Liu, Y.C., Tey, M.K., and You, L. (2021). A concise review of rydberg atom based quantum computation and quantum simulation. *Chin. Phys. B* *30*, 020305.
60. Adams, C.S., Pritchard, J.D., and Shaffer, J.P. (2019). Rydberg atom quantum technologies. *J. Phys. B Atom. Mol. Opt. Phys.* *53*, 012002.
61. Cong, I., Levine, H., Keesling, A., Bluvstein, D., Wang, S.T., and Lukin, M. D. (2022). Hardware-efficient, fault-tolerant quantum computation with rydberg atoms. *Phys. Rev. X* *12*, 021049.
62. Cao, S., and Zhang, Z.; MoGhadeer (2025). Shuxiangcao/leeq: Patterns paper release (Zenodo). <https://doi.org/10.5281/zenodo.16622693>.
63. Spring, P.A., Cao, S., Tsunoda, T., Campanaro, G., Fasciati, S., Wills, J., Bakr, M., Chidambaram, V., Shteynas, B., Carpenter, L., et al. (2022). High coherence and low cross-talk in a tileable 3d integrated superconducting circuit architecture. *Sci. Adv.* *8*, eabl6698.
64. Rahamim, J., Behrle, T., Peterer, M.J., Patterson, A., Spring, P.A., Tsunoda, T., Manenti, R., Tancredi, G., and Leek, P.J. (2017). Double-sided coaxial circuit qed with out-of-plane wiring. *Appl. Phys. Lett.* *110*, 222602.
65. Mitchell, B.K., Naik, R.K., Morvan, A., Hashim, A., Kreikebaum, J.M., Marinelli, B., Lavrijsen, W., Nowrouzi, K., Santiago, D.I., and Siddiqi, I. (2021). Hardware-efficient microwave-activated tunable coupling between superconducting qubits. *Phys. Rev. Lett.* *127*, 200502.
66. Xiong, H., Ficheux, Q., Somoroff, A., Nguyen, L.B., Dogan, E., Rosenstock, D., Wang, C., Nesterov, K.N., Vavilov, M.G., and Manucharyan, V.E. (2022). Arbitrary controlled-phase gate on fluxonium qubits using differential ac stark shifts. *Phys. Rev. Res.* *4*, 023040.
67. Wei, K., Magesan, E., Lauer, I., Srinivasan, S., Bogorin, D., Carnevale, S., Keefe, G., Kim, Y., Klaus, D., Landers, W., et al. (2021). Quantum crosstalk cancellation for fast entangling gates and improved multi-qubit performance. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2106.00675>.
68. Wei, K.X., Magesan, E., Lauer, I., Srinivasan, S., Bogorin, D.F., Carnevale, S., Keefe, G.A., Kim, Y., Klaus, D., Landers, W., et al. (2022). Hamiltonian engineering with multicolor drives for fast entangling gates and quantum crosstalk cancellation. *Phys. Rev. Lett.* *129*, 060501.
69. Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023). Voyager: An open-ended embodied agent with large language models. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2305.16291>.
70. Wattoo, Z.S., Vitis, P., Zhu, R., Depner, N., Zhang, I., Hein, J., Gujarati, A., and Seltzer, M. (2024). Rabbit, a robot arm bug intervention tool for self-driving labs. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (IEEE)*, pp. 353–361.
71. Leong, S.X., Griesbach, C.E., Zhang, R., Darvish, K., Zhao, Y., Mandal, A., Zou, Y., Hao, H., Bernales, V., and Aspuru-Guzik, A. (2024). Steering towards safe self-driving laboratories. Preprint at Chem. <https://doi.org/10.26434/chemrxiv-2024-2qx28>.
72. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q.V., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* *35*, 24824–24837.
73. Cao, S., Lall, D., Bakr, M., Campanaro, G., Fasciati, S.D., Wills, J., Chidambaram, V., Shteynas, B., Rungger, I., and Leek, P.J. (2024). Efficient characterization of qudit logical gates with gate set tomography using an error-free virtual gate model. *Phys. Rev. Lett.* *133*, 120802.
74. Cao, S., Bakr, M., Campanaro, G., Fasciati, S.D., Wills, J., Lall, D., Shteynas, B., Chidambaram, V., Rungger, I., and Leek, P. (2024b). Emulating two qubits with a four-level transmon qudit for variational quantum algorithms. *Quantum Sci. Technol.* *9*, 035003.
75. Zhang, Z. (2025). Evolver/k_agents: Patterns Paper Release (Zenodo). <https://doi.org/10.5281/zenodo.16622431>.