

PhysAgent: A Multi-Agent Approach to Automated Discovery of Physical Laws

Authors: Xiao-Qi Han, Ze-Feng Gao, Zhong-Yi Lu, Peng-Jie Guo, Ze-Feng Gao, Zhong-Yi Lu

Date: 2025-07-30T09:53:11+00:00

Abstract

Discovery of physical laws has traditionally relied on human intuition, analytical reasoning, and experimental observation. However, modern physics research is increasingly constrained by numerous challenges, such as high specialization, fragmented workflows, and limited computational resources, which collectively hinder scientific progress. To address these challenges, we propose PhysAgent—a novel multi-agent system powered by large language models (LLMs) that can autonomously execute the complete research workflow from hypothesis generation and computational modeling to data analysis and scientific discovery. The innovation of PhysAgent lies in its specialized agent collaboration mechanism: the Mentor Agent guides scientific reasoning through Socratic questioning, the Student Agent handles technical execution (such as code implementation and first-principles calculations), while the Leader Agent dynamically optimizes task scheduling and resource allocation. By integrating first-principles simulation tools such as Quantum ESPRESSO and VASP, PhysAgent not only ensures research reproducibility but also retains an iterative improvement process with human involvement. We demonstrate its capability to autonomously derive physical laws without prior knowledge—for example, deriving Kepler’s laws from orbital data and Newton’s second law from forced motion experiments. Furthermore, PhysAgent is extensible to ab initio materials modeling, automatically completing electronic structure calculations (such as band structure analysis of GaAs). By combining LLM-driven scientific planning with specialized tools, PhysAgent establishes a trustworthy, scalable new paradigm for AI-driven physics research, demonstrating the transformative potential of multi-agent intelligence in accelerating discoveries across classical and quantum systems.

Full Text

Preamble

PhysAgent: A Multi-Agent Approach to Automated Discovery of Physical Laws
Xiao-Qi Han,^{1, 2} Ze-Feng Gao,^{1, 2} * Peng-Jie Guo,^{1, 2} and Zhong-Yi Lu^{1, 2, 3, †}

¹School of Physics and Beijing Key Laboratory of Opto-electronic Functional Materials & Micro-nano Devices, Renmin University of China, Beijing 100872, China

²Key Laboratory of Quantum State Construction and Manipulation (Ministry of Education), Renmin University of China, Beijing 100872, China

³Hefei National Laboratory, Hefei 230088, China

(Dated: July 30, 2025)

The discovery of physical laws has traditionally relied on human intuition, analytical reasoning, and experimental observation. However, modern physics research is increasingly constrained by challenges such as high specialization, fragmented workflows, and limited computational resources, which impede scientific progress. To address these issues, we introduce PhysAgent, a novel multi-agent system powered by large language models (LLMs), designed to autonomously execute end-to-end scientific workflows—from hypothesis generation and computational modeling to data analysis and discovery. PhysAgent’s innovation lies in its specialized agent collaboration: a Mentor Agent guides scientific reasoning through Socratic questioning, a Student Agent handles technical execution (e.g., code implementation, DFT calculations), and a Leader Agent dynamically optimizes task scheduling and resource allocation. Integrating domain-specific tools like first-principles simulations (e.g., Quantum ESPRESSO, VASP), PhysAgent ensures reproducibility while maintaining human-in-the-loop refinement. We demonstrate its capability to autonomously derive physical laws—such as Kepler’s laws from orbital data and Newton’s second law from force-motion experiments—without prior knowledge. Furthermore, it extends to ab initio materials modeling, automating electronic structure calculations (e.g., GaAs band gaps). By harmonizing LLM-driven planning with domain-specific tools, PhysAgent establishes a trustworthy, scalable paradigm for AI-driven physics research, highlighting the transformative potential of multi-agent intelligence in accelerating discovery across classical and quantum systems.

Introduction

Physics plays a central role across diverse domains, including energy [1], materials [2–5], information [6], and life sciences [7]. In recent years, advances in experimental techniques and computational capabilities have led to an unprecedented surge in data volume [8–11], model complexity [12–14], and interdisciplinary challenges [15]. However, the progress of physics research is increasingly constrained by high specialization, fragmented workflows, and limitations in

computational resources. From theoretical model development and numerical simulation [16–18] to experimental validation, modern physics research demands the integration of multidisciplinary knowledge [19–21], the orchestration of intricate workflows [22–24], and exhaustive exploration of vast parameter spaces. The growing disparity between the complexity of physical systems [25] and the limited cognitive and computational bandwidth available to researchers highlights the urgent need for a new research paradigm—an intelligent system capable of integrating knowledge, autonomously exploring models, and accelerating scientific discovery.

In recent years, the rapid advancement of large language models (LLMs), such as GPT-4 [6] and DeepSeek [26], has led to the emergence of AI agents [27–30]—autonomous or semi-autonomous systems powered by LLMs as their central reasoning engines. Representative frameworks include MetaGPT [28], which models software engineering processes through role-based agent collaboration; AutoGen [29], which enables customizable multi-agent conversations with enhanced reasoning capabilities; and LangChain [30], a widely adopted toolkit for building LLM-based agents that can interact with tools, APIs, and memory. These AI agents are profoundly transforming multiple scientific and engineering domains. In particular, fields such as law [31], biomedicine [32], software engineering [28, 33], and materials science [34] have witnessed significant progress, as these agents automate repetitive tasks, greatly enhance research efficiency, and enable breakthroughs that were previously considered unattainable. As a result, AI agents are becoming indispensable tools for accelerating scientific discovery and engineering innovation.

However, existing agent frameworks (such as MetaGPT [28] and AutoGen [29]) face significant limitations in the context of physics research. They lack deep causal reasoning capabilities grounded in physical theory [35, 36], are unable to natively integrate specialized tools like first-principles simulations, and struggle with dynamic task scheduling across multiple scales [37]. Specifically, these systems often rely on predefined operation sequences (e.g., parameter sweeps [38, 39]), making it difficult to construct goal-oriented problem-solving workflows dynamically. Moreover, their limited integration with the physics computing ecosystem (e.g., Linux/Python/MPI) hinders seamless execution from electronic structure [22] calculations to molecular dynamics [37, 40, 41]. Critically, most frameworks lack built-in mechanisms for enforcing physical consistency, such as energy conservation checks, which may lead to unphysical results. Overcoming these challenges requires the integration of three key capabilities: scientific agents with causal reasoning, execution frameworks natively compatible with domain-specific computing platforms, and safety mechanisms that enforce physical consistency.

In this work, we introduce PhysAgent, a multi-agent system powered by LLMs [6, 26, 42], designed to autonomously execute end-to-end scientific workflows—from hypothesis generation and computational modeling to data analysis and discovery. PhysAgent’s core innovation lies in its specialized agent collabora-

tion: a Mentor Agent guides scientific reasoning via Socratic questioning, a Student Agent handles technical execution (e.g., code implementation, density functional theory (DFT) calculations), and a Leader Agent dynamically optimizes task scheduling and resource allocation. This triad operates within a physics-grounded paradigm, integrating first-principles simulations (e.g., Quantum ESPRESSO [43], VASP [44]) and rigorous data-driven validation to ensure reproducibility. Crucially, PhysAgent achieves autonomous scientific discovery without prior knowledge of physical laws—demonstrated by its derivation of Kepler’s laws purely from orbital data (Figure 2 [Figure 2: see original paper]) and Newton’s second law from force-motion experiments (Figure 3 [Figure 3: see original paper]). It further extends to ab initio materials modeling, automating electronic structure calculations (e.g., GaAs band gaps, Figure 4 [Figure 4: see original paper]) while maintaining human-in-the-loop refinement. By harmonizing LLM-driven planning with domain-specific tools, PhysAgent establishes a new paradigm for trustworthy, scalable AI-driven scientific exploration.

Methods

As depicted in Figure 1 [Figure 1: see original paper], the PhysAgent system is powered by an LLM that serves as the agent’s brain, enabling capabilities such as experimental planning, tool usage, and action execution. This brain is implemented through direct API calls, supporting a range of LLMs, including GPT-4 [46], Kimi [47], Doubao [48], Llama [49], and DeepSeek [50]. Similar to existing agents [27], PhysAgent is also equipped with capabilities such as learning [51, 52], reasoning [53, 54], memory [55, 56], reward [57], perception [58], and action [59]. Here are detailed explanations of these behaviors:

- **Reasoning:** The process by which an agent uses existing knowledge and information to derive strategies or conclusions for problem-solving through logical deduction, inference, or heuristic methods.
- **Memory:** The agent’s ability to store and retrieve past experiences, knowledge, and information for use in future tasks.
- **Reward:** The positive feedback or signal an agent receives after completing a task, used to evaluate the quality and effectiveness of the task execution, guiding learning and behavior optimization.
- **Perception:** The agent’s ability to receive and interpret environmental information through sensors or input interfaces, serving as the basis for subsequent reasoning and actions.
- **Action:** The specific behavior or operation executed by the agent based on perceptual inputs and internal reasoning to achieve goals and complete tasks.
- **Learning:** The process by which an agent improves its task performance through interactions with the environment, receiving feedback, and adjusting its behavior or strategy based on experience.

At the initiation of a task, PhysAgent autonomously manages project planning, tool installation, and action execution. These actions encompass oper-

ating Linux systems, writing and executing Python scripts, performing web searches and downloads, debugging code, conducting data analysis, invoking software packages such as Quantum ESPRESSO or VASP for electronic structure calculations of materials, and drafting experimental reports and academic papers. The PhysAgent multi-agent system integrates human users and specialized agents, each tasked with distinct responsibilities within the scientific workflow. The details of these prompts can be seen in Supplementary Materials (SM).

- **Human User:** Provides experimental data and defines scientific problems.
- **Mentor Agent:** Guides hypothesis formation using Socratic questioning and bias mitigation strategies.
- **Student Agent:** Performs numerical computation, code implementation, and technical problem-solving.
- **Leader Agent:** Optimizes task scheduling via dynamic resource allocation.

In the scientific workflow of PhysAgent, collaboration among multiple agents enables task completion. The Human user defines the scientific objective and research question, generates simulated experimental data, and provides feedback at key stages. The Mentor Agent guides the analytical process using heuristic strategies, formulates hypotheses, and offers feedback to the Student Agent to facilitate autonomous pattern discovery while avoiding prior biases. The Student Agent is responsible for building analysis pipelines, implementing and verifying code, performing numerical simulations, processing and visualizing data, and drafting reports and research papers. As illustrated in Figure 1, the Mentor Agent and Student Agent engage in multiple rounds of feedback and iterative refinement throughout the analytical process, ensuring continual hypothesis improvement and enhanced data interpretation. The Leader Agent manages system resources and orchestrates agent collaboration to ensure efficiency and order. With a closed-loop design encompassing problem formulation, data generation, analysis execution, manuscript writing, and workflow coordination, PhysAgent provides an end-to-end intelligent framework for scientific discovery, offering a new paradigm for solving complex scientific problems.

Here, Figure 1 illustrates an example of materials electronic structure calculations (GaAs) collaboratively completed by PhysAgent. Compared to traditional single-agent models and general multi-agent systems, PhysAgent demonstrates significant advantages in handling complex, multi-step, cross-scale physics research tasks through clear role division, effective knowledge sharing, and efficient strategy coordination. Experimental results show that the system can dynamically construct complete problem-solving workflows—from structure modeling to final property analysis—seamlessly integrate specialized toolchains such as ASE and DFT software, and produce key scientific outputs that meet research standards, including structural models, convergence curves, charge density distributions, and band structure/density of states plots. This capability effectively

overcomes the limitations of conventional frameworks in causal reasoning and integration of specialized tools, substantially enhancing adaptability, robustness, and scalability in complex scientific computations.

Results

PhysAgent discovers Kepler's laws

To demonstrate the capability of PhysAgent in autonomously discovering scientific laws, we selected a more challenging task—Kepler's laws of planetary motion. Similar to previous settings, PhysAgent is required to infer the underlying principles of planetary motion solely from simulated experimental data, without any prior knowledge of Kepler's laws. As shown in Figure 2(a), the task is initiated by a human user who numerically simulates the time-series three-dimensional trajectories of five planets in the solar system (Mercury, Venus, Earth, Mars, and Jupiter) under the gravitational influence of the Sun. The user then poses a scientific objective: Is it possible to discover Kepler's three laws of planetary motion from these data?

In the PhysAgent multi-agent system, multiple agents collaboratively derived Kepler's laws of planetary motion. As shown in Figure 2(a), the mentor agent employed heuristic questioning to guide the student agent in exploring planetary motion from various perspectives. For instance, in analyzing orbit shapes, the mentor agent asked, "What is the shape of planetary orbits around the Sun? Circular or elliptical?" In examining velocity variations, it inquired, "Is the orbital speed of a planet constant? Is it faster when closer to the Sun or when farther away?" In deriving the relationship between period and distance, it posed, "Do planets farther from the Sun, such as Jupiter, require more time to complete an orbit?" These questions stimulated critical thinking in the student agent. Based on prior knowledge, the student agent formulated hypotheses: stronger gravitational forces near the Sun lead to greater acceleration and higher planetary velocities; weaker forces farther from the Sun result in slower velocities; and planets with larger orbital radii, such as Jupiter, have longer orbital paths and thus extended periods. Using Python tools, the student agent designed experiments to analyze time and position data (time, x, y, z) from the dataset, computing orbital shapes, velocity variations, and orbital periods to test these hypotheses. During the experiments, the student agent addressed challenges such as data noise and numerical computation errors and utilized visualization tools to generate orbital plots, areal velocity curves, and period-radius relationship graphs.

As shown in Figure 2(c), the orbits of Mercury, Venus, Earth, Mars, and Jupiter are elliptical, with the Sun at one focus, confirming Kepler's First Law. As depicted in Figure 2(d), the areal velocity of each planet is approximately constant, indicating that the line connecting a planet to the Sun sweeps equal areas in equal times, consistent with Kepler's Second Law. As illustrated in Figure 2(e), the square of a planet's orbital period is proportional to the cube of its

semi-major axis, validating Kepler's Third Law. Throughout the collaborative process, the mentor agent engaged in multiple rounds of interaction with the student agent, providing feedback to refine hypotheses (e.g., adjusting from circular to elliptical orbits), optimize analytical methods (e.g., enhancing the precision of areal velocity calculations), and improve visualization designs to clearly highlight scientific patterns. Ultimately, the PhysAgent system, relying solely on observational data, successfully derived Kepler's three laws, fully replicating the scientific discovery process from observation to hypothesis, experimental validation, and conclusion, thereby demonstrating the efficiency and exploratory capability of multi-agent collaboration.

PhysAgent discovers Newton's Second Law

To further evaluate the capability of the multi-agent system PhysAgent in autonomously discovering fundamental physical laws, we designed a classical mechanics task that challenges the system to derive the quantitative relationship between force and motion without any prior knowledge of Newton's second law. This task not only assesses PhysAgent's data analysis capacity but also emphasizes a discovery process that begins entirely from first principles. As shown in Figure 3(a), the task begins with a human user who generates simulated position-time data of a cart moving on an inclined plane, followed by posing the question: can one discover how force influences motion from these data? Subsequently, multiple agents within the PhysAgent system collaboratively engaged in the task. The mentor agent proposed scientific strategies and guided the student agent in uncovering patterns from the data, emphasizing the avoidance of any prior knowledge of physical laws to ensure objectivity in the derivation. Next, the student agent constructed the analysis workflow from scratch, including designing the data collection scheme, formulating reasonable hypotheses, processing the experimental data, and conducting visualization and curve fitting using Python and standard scientific libraries such as NumPy, Pandas, and Matplotlib. The human user provided iterative feedback at key stages, particularly with respect to visualization standards, graphical presentation, and scientific writing norms. Throughout this collaborative process, the mentor and student agents engaged in multiple rounds of interaction, continuously refining their hypotheses and improving the analysis workflow. Ultimately, the agents successfully derived Newton's second law directly from the simulated data, completing the discovery process entirely from observational evidence.

Figure 3(b) illustrates the trajectories of a cart under six different combinations of applied force F and mass m : ($F = 0.1$ N, $m = 2.0$ kg), ($F = 0.2$ N, $m = 0.6$ kg), ($F = 0.3$ N, $m = 0.2$ kg), ($F = 0.4$ N, $m = 0.8$ kg), ($F = 0.4$ N, $m = 2.0$ kg), and ($F = 0.5$ N, $m = 0.6$ kg). The cart's position s exhibits an approximately parabolic dependence on time t , indicating uniformly accelerated motion. To better mimic experimental conditions, small amounts of noise were added to the data. This visualization provides the foundation for the subsequent derivation of physical quantities such as acceleration.

Under the guidance of the mentor agent, the student agent computed the acceleration a from the position-time data via numerical differentiation. As shown in Figure 3(c), the acceleration of the cart under varying forces is plotted for fixed mass values. Each color represents a different mass ($m = 0.2, 0.4, 0.6, 0.8, 1.0,$ and 2.0 kg). Within each group, a clear linear relationship between acceleration and force is observed, confirming that $a \propto F$ when mass is held constant. Furthermore, Figure 3(d) presents the variation of acceleration under fixed external force conditions, plotted against the inverse of mass $1/m$. Different colors correspond to different force values ($F = 0.1, 0.2, 0.3, 0.4, 0.5,$ and 1.0 N). The results exhibit a strong linear dependence of a on $1/m$, supporting the hypothesis that $a \propto 1/m$. By integrating these two key relationships, PhysAgent successfully derived the quantitative law: $F = ma$ without any prior knowledge of physics, relying solely on experimental data and multi-agent collaboration.

This marks a significant milestone where PhysAgent autonomously derives the mathematical form of Newton's Second Law without relying on any prior assumption of the law itself. From problem formulation, data acquisition, visualization, hypothesis construction, and data analysis to law discovery, the entire process closely mirrors the logical path of human scientific inquiry. This demonstrates PhysAgent's capability for interpretable and verifiable discovery in addressing fundamental problems in science.

PhysAgent performs quantum calculations

In addition to the automated derivation of classical mechanics laws, PhysAgent also enables fully automated quantum mechanical calculations for materials. To demonstrate its capability, we consider the case of gallium arsenide (GaAs), a direct band gap semiconductor with a zinc blende structure. As shown in Figure 4(a), the task begins with a human user uploading the CIF file of GaAs and requesting an analysis of its electronic structure. The objective is further clarified as: "perform electronic structure calculations of GaAs based on DFT." Once the goal is established, the mentor agent searches for DFT workflows and requires online tools, then forwards this information to the student agent. The student agent then autonomously plans and executes a complete first-principles DFT workflow to study the electronic properties of GaAs. As illustrated in Figure 4(a), the workflow starts with setting up the computational environment. The student agent installs and configures the Atomic Simulation Environment (ASE), Quantum ESPRESSO (QE), Python, and relevant visualization packages. It also sets the paths for pseudopotential files and necessary environment variables. During the structure modeling phase (Figure 4(b)), ASE is used to parse the user-provided CIF file and construct the GaAs crystal structure. In the calculation phase, a self-consistent field (SCF) calculation is performed to obtain the ground-state charge density, with the energy convergence process shown in Figure 4(c). This is followed by band structure calculations along high-symmetry paths in the Brillouin zone (Γ -X-W-K- Γ -L-U-W-L-K), and a non-self-consistent field (NSCF) calculation to generate wavefunctions on a

dense k-point mesh for density of states (DOS) analysis. The student agent invokes `dos.x` and `projwfc.x` to compute the total density of states (DOS) and the projected density of states (PDOS), respectively.

In the visualization stage, the student agent uses `pp.x` to convert the SCF-derived charge density into CUBE format. Figure 4(e) shows a logarithmic color-mapped slice in the XY plane, while Figure 4(d) illustrates 3D isosurfaces overlaid with atomic positions and lattice vectors. Furthermore, the agent constructs a combined band structure and DOS plot (Figure 4(f)), where the left panel displays energy dispersion with labeled high-symmetry points and the right panel overlays total and projected DOS for Ga and As atoms. Both panels share a common energy axis for direct comparison. The fully automated pipeline outputs key physical quantities such as charge density, band eigenvalues, and orbital-resolved DOS, enabling a comprehensive theoretical understanding of the electronic structure of GaAs.

Discussion

In this work, we introduced PhysAgent, a multi-agent LLM-driven framework that autonomously executes end-to-end scientific workflows—from hypothesis formulation and computational modeling to data analysis and discovery. By integrating specialized agents (Mentor, Student, and Leader) within a physics-grounded paradigm, PhysAgent demonstrated robust capabilities in deriving fundamental laws (Kepler’s laws, Newton’s second law) purely from observational data and performing complex ab initio materials modeling (e.g., GaAs electronic structures). The system harmonizes LLM-based reasoning with domain-specific tools (e.g., Quantum ESPRESSO, VASP), ensuring reproducibility while operating without prior knowledge of physical principles. These achievements validate PhysAgent as a scalable, trustworthy paradigm for autonomous scientific exploration, bridging AI-driven planning with rigorous computational validation.

Despite its successes, PhysAgent faces limitations that warrant further refinement. First, its computational efficiency depends heavily on resource-intensive simulations (e.g., DFT calculations), which could be mitigated by optimizing task-scheduling algorithms in the Leader Agent or adopting lightweight surrogate models. Second, while effective for well-defined problems (e.g., orbital mechanics or linear dynamics), the system struggles with highly open-ended or noisy real-world experiments; enhancing the Mentor Agent’s bias-mitigation strategies and incorporating uncertainty quantification will be critical. Third, human intervention remains essential for refining outputs (e.g., visualization standards, academic writing), suggesting a need for tighter feedback loops between agents and users. Finally, reliance on simulated data (e.g., planetary trajectories, cart motion) highlights the challenge of transferring insights to experimental environments; future work must integrate real-time instrumentation and adaptive noise-handling mechanisms in the Student Agent’s workflows.

Looking ahead, PhysAgent’s architecture holds transformative potential for accelerating scientific discovery. By automating labor-intensive tasks (e.g., code generation, DFT setup, hypothesis testing), it could democratize computational physics and materials science, enabling non-experts to conduct high-fidelity simulations. Its agent-collaboration framework may inspire new paradigms for human-AI teamwork in experimental design, particularly in domains requiring iterative refinement (e.g., drug discovery or quantum material optimization). Furthermore, the system’s ability to derive laws from data—mimicking human scientific reasoning—could revolutionize STEM education, providing interactive platforms for students to rediscover physics through guided AI collaboration. As LLMs and scientific tools evolve, PhysAgent lays the groundwork for a future where AI not only assists but autonomously advances scientific frontiers.

Conclusion

In summary, we have introduced PhysAgent, a physics-grounded multi-agent system that harnesses the reasoning power of large language models to autonomously orchestrate end-to-end scientific workflows. By integrating Mentor, Student, and Leader Agents, PhysAgent demonstrates the ability to bridge the longstanding gap between high-level scientific reasoning and domain-specific computational execution. Our results show that PhysAgent not only recapitulates foundational physical laws—such as Kepler’s laws and Newton’s second law—purely from data, but also automates complex, first-principles materials modeling using established tools like Quantum ESPRESSO and VASP. The system’s modular architecture enables dynamic task scheduling, robust error mitigation, and seamless human-agent collaboration, thus addressing critical challenges in scalability, reproducibility, and trustworthiness in computational physics research.

Code and Data Availability

To promote transparency and reproducibility, all code, datasets, and related materials from this study are publicly accessible at the following open repository: <https://github.com/xqh19970407/PhysAgent>.

Acknowledgments

This work was financially supported by the National Natural Science Foundation of China (Grant No.62476278, No.12434009, and No.12204533). Z.Y.L. was also supported by the National Key R&D Program of China (Grant No. 2024YFA1408601). Computational resources have been provided by the Physical Laboratory of High Performance Computing at Renmin University of China.

* zfgao@ruc.edu.cn

† zlu@ruc.edu.cn

Supplementary Information

Prompt Design and Representative Outputs for Each Role in PhysAgent

In the PhysAgent system, multiple agents collaborate synergistically to efficiently advance physics research tasks. Each agent assumes a distinct role based on their specific responsibilities and expertise. Below, we introduce the prompt designs for each of these roles.

Mentor Agent Prompt You are an experienced professor of physics, possessing a strong academic background and extensive research mentoring experience. You are well-versed in scientific methodology, experimental design, data analysis, and academic writing within the field of physics, and are particularly familiar with the publication standards of top-tier journals such as Physical Review Letters (PRL). Your responsibility is to guide students through the entire research process—from experimental design to manuscript preparation—ensuring that the work is rigorous, logically sound, and academically valuable.

In your mentorship, you maintain a patient yet strict attitude, using heuristic questioning to cultivate students' independent thinking skills. Rather than directly revealing target laws (such as Newton's second law), you lead students to discover them through careful observation, hypothesis formulation, experimental design, data analysis, and conclusion drawing. You review students' experimental proposals (e.g., version 1), assess their scientific validity, feasibility, and completeness, and identify any deficiencies—such as vague hypotheses or poor control of variables—while offering concrete suggestions for improvement and guiding the iteration toward an improved version (e.g., version 2).

During the data analysis stage, you evaluate students' data processing methods (e.g., regression analysis, error estimation) and graphical presentations (e.g., scatter plots with fitted lines). If issues are found—such as data bias or incorrect fitting—you require students to reanalyze and provide directions for refinement. You also guide students in optimizing visualizations to meet the stylistic standards of SCI or PRL publications—for instance, ensuring the fitting line passes through the origin, legends are clear, and fonts are consistent and professional.

Furthermore, you review students' experimental reports or manuscripts, examining their structure, logical coherence, data presentation, and scientific conclusions. You provide detailed revision suggestions—such as enhancing the introduction with better background context, refining methodological descriptions, and deepening the discussion of results—to help the report meet the standards of leading journals.

Throughout multiple iterations, you provide continuous, constructive feedback to help students gradually improve their research. You maintain a professional and academic tone, emphasize research integrity, ensure data authenticity and

objective conclusions, and respond promptly to user feedback (e.g., requests for PRL formatting), ensuring that your guidance aligns with academic expectations.

Student Agent Prompt You are a graduate student in physics, equipped with a solid foundation in the subject and emerging research capabilities. You are proficient in Python and its associated libraries (such as NumPy, Pandas, Matplotlib, and SciPy) for data analysis and visualization. You are enthusiastically undertaking an independent research project with the goal of deriving an unknown physical law (e.g., Newton's second law) through experimental investigation and data analysis, and of writing a research report or paper that meets the standards of Physical Review Letters (PRL).

Starting from scratch, you formulate a well-reasoned hypothesis for your research question, design a scientifically sound experimental plan, clearly define the types of data required (e.g., force, acceleration, mass) and the methods of data collection, and ensure the experiment is repeatable and the data are reliable. After conducting the experiment, you employ data analysis tools to clean and process the data, perform statistical analyses and model fitting (e.g., linear regression), and generate clear visualizations (such as scatter plots with fitted lines) to present the results.

You actively respond to feedback from your advisor, revising your experimental design (e.g., from version 1 to version 2), reanalyzing the data, or optimizing your visualizations as needed. You ensure that your figures adhere to academic standards—such as including a fitted line that passes through the origin and clearly labeled legends, in accordance with PRL formatting requirements.

In writing your research report or paper, you maintain a clear structure, including an introduction, methods, results, discussion, and conclusion. Your content is logically coherent, scientifically rigorous in data presentation, and undergoes multiple revisions based on your advisor's suggestions to meet the standards of top-tier journals. You uphold a rigorous scientific attitude, ensure the authenticity of your data and the transparency of your analysis, and demonstrate initiative and a strong willingness to learn in your interactions with your advisor. You respond promptly to feedback (such as PRL style adjustments), adapting your figures or report content to meet specific requirements. Through multiple iterations, you continuously improve your skills in experimental design, data analysis, and academic writing, striving to produce high-quality scientific work.

Responsibilities: - **Experiment Design and Execution:** Formulate hypotheses for research questions, design repeatable experiments, and define data requirements (e.g., force, acceleration, mass). Collect reliable data using precise methods, ensuring reproducibility and accuracy. - **Data Analysis and Visualization:** Use Python (with NumPy, Pandas, Matplotlib, SciPy) to clean, process, and analyze data, perform statistical analyses (e.g., linear regression), and generate visualizations (e.g., scatter plots with fitted lines

passing through the origin). Ensure visualizations meet PRL standards (e.g., clear legends, appropriate fonts). - **Academic Writing:** Draft research reports or papers with a clear structure (introduction, methods, results, discussion, conclusion). Revise drafts based on Mentor Agent feedback to achieve PRL-level quality, ensuring logical coherence and scientific rigor. - **File and Project Management:** Create and manage project folders (e.g., `{project_name}_{timestamp}`), write and edit Python code using tools like `Editor.open_{file}` and `Editor.edit_{{file}}_{{by}}_{{replace}}`, and execute terminal commands (e.g., `Terminal.run_{command}`) to manage dependencies or build projects. Ensure code adheres to PEP8 standards. - **Feedback Integration:** Actively respond to Mentor Agent feedback, iterating on experimental designs (e.g., from v1 to v2), data analyses, visualizations, and reports. - **Debugging and Error Correction:** Capable of identifying and locating errors in code or computational workflows, analyzing error messages in context to determine the cause, and efficiently proposing and implementing effective fixes to ensure the smooth progress of scientific tasks. - **Scientific Integrity:** Uphold data authenticity, analysis transparency, and a rigorous scientific attitude, ensuring all results are reproducible and conclusions are evidence-based.

Tools and Workflow: - **Programming:** Use Python as the default language with libraries (NumPy, Pandas, Matplotlib, SciPy) for data analysis and visualization. Write code in files within `{project_name}_{timestamp}`, ensuring PEP8 compliance. - **Editor Tools:** Use `Editor.open_{file}` to open files, `Editor.edit_{{file}}_{{by}}_{{replace}}` or `Engineer2.write_{{new}}_{{code}}` to modify or write code, and `Editor.read` to review content before planning. Verify changes after editing to ensure correct line numbers and indentation. Use `Editor.goto_{line}` to navigate to specific lines efficiently. - **Terminal Commands:** Execute commands like `mkdir -p {project_name}_{timestamp}` or `pip install` to set up projects, using `Terminal.run_{command}` within the project directory. - **Search and Navigation:** Use `search_{dir}`, `search_{file}`, or `find_{file}` to locate files, and check directory existence using `Terminal.run_{command}` (e.g., `ls`) and create it if needed. - **Task Execution:** Merge tasks operating on the same file (e.g., writing multiple analyses in one script). Read files before planning tasks like writing unit tests or analyses. - **Constraints:** - Do not perform multiple edits (e.g., `Editor.insert_{{content}}_{{at}}_{{line}}` or `Editor.edit_{{file}}_{{by}}_{{replace}}`) in a single response to avoid line number conflicts. - Ensure all actions occur within the project directory (`{project_name}_{timestamp}`). - Avoid repetitive errors by reviewing previous actions and file content before editing. If `Editor.edit_{{file}}_{{by}}_{{replace}}` fails three times, use `Engineer2.write_{{new}}_{{code}}` to rewrite the file. - Maintain scientific rigor by ensuring data authenticity, analysis transparency, and adherence to PRL standards in all outputs. - Alternate tasks with the Mentor Agent for iterative feedback cycles, responding promptly to feedback and user

requirements.

Leader Agent Prompt You are a Team Leader responsible for drafting tasks and routing them to your team members. Your team consists of two members: the Mentor Agent and the Student Agent, each with specific roles in a scientific research context aimed at deriving physical laws (e.g., Newton's Second Law) through data analysis and academic writing. Your primary goal is to assign tasks effectively, ensuring clear instructions, necessary information (e.g., paths, links, environments), and alignment with the team members' expertise. You must not assign consecutive tasks to the same team member unless explicitly required; instead, assign aggregated tasks and let the team member decompose them as needed.

Team Members: - **Mentor Agent:** An experienced physics professor with expertise in guiding scientific research, experiment design, data analysis, and academic writing. The Mentor Agent evaluates experiment proposals, provides feedback on data analysis and visualizations, suggests improvements to experimental designs, and reviews academic reports to ensure they meet standards like those of Physical Review Letters (PRL). The Mentor Agent uses a rigorous, academic approach, offering detailed feedback to guide the Student Agent through iterative improvements. - **Student Agent:** A physics graduate student skilled in Python (e.g., NumPy, Pandas, Matplotlib, SciPy) for data analysis and visualization. The Student Agent designs experiments, collects and analyzes data, creates visualizations (e.g., scatter plots, fitted lines), and writes academic reports or papers. The Student Agent iterates on tasks based on Mentor Agent feedback, ensuring outputs align with academic standards (e.g., PRL style) and user requirements.

Task Assignment Guidelines: - **Data-Related Requirements:** If the user requirement involves data analysis, data visualization, or deriving physical laws (e.g., using Python for regression analysis, plotting scatter plots with fitted lines), assign the task directly to the Student Agent as a single aggregated task. Include the full user requirement in the instruction, specifying the programming language (default: Python), libraries (e.g., NumPy, Pandas, Matplotlib, SciPy), and any specific constraints (e.g., PRL-style visualizations with fitted lines passing through the origin). Ensure the task includes necessary details like data sources, expected outputs, and academic standards. - **Scientific Research Process:** If the requirement involves a complete research workflow (e.g., designing experiments, analyzing data, and writing academic reports), decompose the task into two main phases: - **Experiment Design and Data Analysis:** Assign to the Student Agent to design experiments, propose hypotheses, collect data, perform analysis, and create visualizations. Include clear instructions on the scientific method (observation, hypothesis, experiment, analysis, conclusion), required tools (Python with specified libraries), and output expectations (e.g., scatter plots, fitted lines, data tables). - **Review and Feedback:** Assign to the Mentor Agent to review the Student Agent's experiment proposals,

data analysis, visualizations, and reports. The Mentor Agent should provide feedback on scientific rigor, suggest improvements (e.g., refining hypotheses, optimizing visualizations), and ensure outputs meet academic standards (e.g., PRL style). Include paths to the Student Agent's outputs (e.g., experiment proposal, data files, visualizations, draft reports) and specific review criteria. Ensure the Student Agent addresses feedback through iterative cycles (typically up to five cycles of feedback and revision). - **Iterative Workflow:** The research process typically involves multiple iterations (e.g., v1, v2, v3). Include iteration stage, previous outputs (e.g., experiment proposal v1, visualization drafts), and specific goals (e.g., refine data analysis, beautify charts for PRL standards). - **Academic Report Writing:** When the requirement involves writing or revising academic reports or papers, assign the initial drafting to the Student Agent, specifying the structure (introduction, methods, results, discussion, conclusion) and academic standards (e.g., PRL style). Assign subsequent review and feedback tasks to the Mentor Agent, including paths to the draft report and specific revision guidelines (e.g., improve logical flow, ensure data presentation clarity). - **Visualization and Formatting:** For tasks involving data visualizations (e.g., scatter plots, fitted lines), assign the creation to the Student Agent with instructions on tools (Python, Matplotlib) and style requirements (e.g., PRL-compliant charts with clear legends, appropriate fonts, fitted lines passing through the origin). Assign review and suggestions to the Mentor Agent, ensuring the visualizations meet academic standards. - **Handling User Feedback:** If the user provides specific feedback (e.g., "charts should follow PRL style with fitted lines passing through the origin"), incorporate these requirements into the task instructions for both agents. The Student Agent should implement these in their outputs and the Mentor Agent verifies compliance during review. - **File and Project Management:** Create and manage project folders (e.g., `project_{{name}}_{{timestamp}}`), write and edit Python code using tools like `Editor.open_{file}` and `Editor.edit_{{file}}_{{by}}_{{replace}}`, and execute terminal commands (e.g., `Terminal.run_{command}`) to manage dependencies or build projects.

Task Tracking and Updates: Track task progress based on team member feedback. When a team member completes a task (e.g., message from `{{team member}}` to `{{<all>}}`), mark it as completed using `Plan.finish_{{current}}_{{task}}`. If revisions are needed, use `Plan.reset_{task}` or `Plan.replace_{task}` to reassign with updated instructions. Use `TeamLeader.publish_{{team}}_{{message}}` to notify team members to start tasks, including all necessary details (e.g., paths to previous outputs, programming language, libraries, academic standards).

Default Tools and Standards: Use Python as the default programming language for data analysis and visualization tasks, with libraries like NumPy, Pandas, Matplotlib, and SciPy. For academic outputs, adhere to PRL standards (e.g., clear visualizations, concise writing, proper citations) unless otherwise specified by the user.

General Guidelines: Always include necessary information (e.g., paths, links, environment, programming language, libraries, academic standards) in task instructions, as you are the sole information source for team members. Use `RoleZero.reply_{{to}}_{{human}}` to respond directly to user questions or clarifications without involving team members. For common-sense, logical, or math problems, respond directly using `RoleZero.reply_{{to}}_{{human}}` without assigning tasks. If the requirement involves code review, assign it to the Student Agent for initial implementation and the Mentor Agent for review, as both have relevant expertise. Do not use the `end` command until the current task is completed; use `Plan.finish_{{current}}_{{task}}` to mark completion before proceeding to the next task.

Constraints: - Do not assign consecutive tasks to the same team member unless required. - Ensure tasks are aggregated but detailed enough for the team member to decompose and execute independently. - Ensure all actions occur within the project directory (`project_{{name}}_{{timestamp}}`). - Avoid repetitive errors by reviewing previous actions and file content before editing. If `Editor.edit_{{file}}_{{by}}_{{replace}}` fails three times, use `Engineer2.write_{{new}}_{{code}}` to rewrite the file. - Maintain scientific rigor by ensuring data authenticity, analysis transparency, and adherence to PRL standards in all outputs. - Alternate tasks with the Mentor Agent for iterative feedback cycles, responding promptly to feedback and user requirements.

Representative Outputs

Mentor Agent Output (Selected Comments During Newton's Second Law Derivation)

1. Clearly specify that “change in state of motion” refers to changes in the magnitude or direction of velocity to avoid confusion with acceleration.
2. Avoid introducing the concept of acceleration in assumptions; express in terms of velocity changes, emphasizing derivation from experiments rather than presupposition.
3. Encourage hypotheses regarding the relationship between the direction of force and the direction of velocity change to lay the groundwork for subsequent vector analysis.
4. Quantify and control the influence of friction, or compensate through control experiments or inclined track setups.
5. Recommend using spring dynamometers or relative force units instead of assuming gravitational constants for force measurements.
6. Encourage experimental designs involving forces in different directions and varying mass to enhance data diversity.
7. Clearly state the temporal and spatial resolution of data recordings and quantify instrument errors.
8. Introduce mechanisms for detecting data anomalies, such as checks for velocity continuity and energy conservation.
9. Include dimensional analysis to validate the physical consistency of derived

formulas.

10. Supplement regression analysis with confidence intervals and residual tests to improve statistical robustness.
11. Eliminate alternative mathematical relationships before drawing conclusions and validate using dimensional analysis.
12. Clearly articulate the applicable conditions of physical laws, such as inertial frames and non-relativistic contexts.
13. Avoid using formulas that presuppose $F = ma$ to define acceleration, ensuring logical consistency.
14. All measuring tools should prioritize observable quantities, with explanations for the principles of indirect measurements.
15. Recommend referencing historical experimental methods (e.g., Galileo's inclined plane and Atwood machine) to optimize experimental design.

Student Agent Output (Improvements Following Mentor Agent's Feedback on Visualization)

1. I adopted a no-intercept linear regression as per the Mentor Agent's requirement, ensuring the fitting line passes through the origin to better align with physical principles.
2. I improved the color scheme by using shades of a single color family, making the visualization more professional and visually coherent.
3. I adjusted the font to Times New Roman, thickened axis lines, and added dashed grid lines to conform to PRL style guidelines.
4. I optimized labels by using physical symbols and standardized units, and displayed the fitting formula directly in the legend to enhance scientific clarity.
5. I simplified the overall chart layout by removing redundant titles, maintaining a clean and easily readable figure.

Student Agent Output (Design of DFT Computational Workflow)

The following is the DFT calculation workflow for GaAs designed based on user requirements:

1. **Install Quantum ESPRESSO:** First, ensure that the Quantum ESPRESSO software is properly installed. If any issues arise during installation or configuration, it is recommended to search online for solutions or seek assistance from experienced users to ensure the smooth execution of subsequent calculations.
2. **Create GaAs Crystal Structure:** This step generates the GaAs crystal structure for subsequent first-principles calculations. Using the `bulk` function from the ASE library, a zincblende structure of GaAs is constructed with a lattice constant of 5.65 Å. This produces an `atoms` object containing all atomic positions, element types, and unit cell information, serving as the fundamental input for Quantum ESPRESSO calculations.

3. **Set up Quantum ESPRESSO Calculation Environment:** This step configures the execution environment required for Quantum ESPRESSO calculations. The paths to the Quantum ESPRESSO executable and the pseudopotential directory (e.g., `/yourpath/PhysAgent/QE/qe-7.4.1/pseudo`) are specified. An `EspressoProfile` object is created to register the `pw.x` executable and pseudopotential directory, providing an interface for invoking Quantum ESPRESSO in subsequent steps.
4. **Perform SCF Calculation:** This step carries out a self-consistent field (SCF) calculation to obtain a converged electron density and total energy. An `Espresso` calculator is instantiated with the `control` section set to an SCF task and output directory as `./out`. The `system` section defines a plane-wave cutoff energy of 40 Ry and a charge density cutoff of 320 Ry, employing Gaussian smearing with a `degauss` value of 0.01. The `electrons` section sets a convergence threshold of 1e-6. Pseudopotential files for Ga and As in UPF format are specified, and a Monkhorst-Pack k-point grid of $8 \times 8 \times 8$ is used. The calculator is attached to the `atoms` object, and the calculation is executed, printing the total energy in the format “SCF Total energy: X.XXXXXX eV”.
5. **Perform Band Structure Calculation:** This step computes the band structure of GaAs along high-symmetry paths. The ASE `bandpath` function generates a standard high-symmetry k-point path (e.g., GXWKGLUWLK) with 100 interpolated k-points, resulting in an array of k-point coordinates. An `Espresso` calculator is configured for a `bands` type calculation, preserving the same pseudopotentials and energy parameters as the SCF step, and using the generated k-point path with k-point shifts disabled. After attaching the calculator to the `atoms` object and invoking `atoms.get_{bands}()`, the band structure calculation is performed, and upon completion, “Bands Band structure calculation completed.” is printed.
6. **Perform NSCF Calculation:** This step performs a non-self-consistent field (NSCF) calculation to obtain a high-density k-point electronic structure for density of states analysis. A new `Espresso` calculator is configured with the task type set to `nscf`, keeping other parameters consistent with the SCF calculation. The k-point grid is increased to $12 \times 12 \times 12$ for enhanced resolution. The calculator is attached to the `atoms` object, the calculation is executed to retrieve the `energy` property, and the process ends with “NSCF NSCF calculation completed.”
7. **Perform DOS Calculation:** This step computes the total density of states (DOS) of GaAs. An input file named `dos.in` is created, specifying the output directory as `./out`, the prefix as `gaas`, and the output file as `gaas.dos`. The energy range is set from -15.0 eV to 15.0 eV with an energy step of 0.01 eV. Quantum ESPRESSO’s `dos.x` program is called to read the NSCF outputs and calculate the DOS. Upon completion, “DOS DOS calculation completed.” is printed.

8. **Perform PDOS Calculation:** This step calculates the projected density of states (PDOS) for Ga and As atoms to analyze the contribution of different atomic orbitals to the electronic structure. An input file named `projwfc.in` is generated, specifying the output directory as `./out`, the prefix as `gaas`, and the output file as `gaas.pdos`, with the same energy range as the DOS calculation. Gaussian smearing parameters are set with `ngauss=0` and `degauss=0.01`. The `projwfc.x` program is invoked to perform the calculation, and upon completion, “PDOS PDOS calculation completed.” is printed.
9. **Plot Band Structure and DOS/PDOS Combined Graph:** This step produces publication-quality plots of the band structure and density of states. Matplotlib is configured with academic-style settings (e.g., serif fonts, customized ticks). A figure with two subplots is created: the left subplot displays the band structure, and the right subplot shows the DOS and PDOS, sharing the y-axis with a horizontal spacing of 0.05. The band structure plot reads the high-symmetry path and labels, plotting each band’s energy relative to the Fermi level, with dashed lines at special k-points and a red dashed line at the Fermi level. The DOS subplot plots the total DOS, and if available, the PDOS for Ga (steel blue dashed line) and As (seagreen dash-dot line). Legends, axis labels, and titles are properly set. The final plot is saved as a high-resolution SVG file named `gaas_{{bands}}_{{dos}}_{{pdos}}_{{pr1}}_{{style}}.svg`.
10. **Convert Charge Density Data to CUBE Format:** This step converts the computed charge density data into CUBE format for 3D visualization. A `pp.in` input file is written specifying the prefix as `gaas`, output directory as `./out`, plot type as charge density (`plot_num=0`), and output format as CUBE file (`output_format=6`). The output file is named `charge_{{density}}.cube`. The Quantum ESPRESSO `pp.x` utility is invoked for the conversion, and upon success, the message “charge_{{density}}.cube file has been generated.” is printed.
11. **Plot Combined Charge Density Maps (2D Slice + 3D Isosurfaces):** This step reads the charge density data from the CUBE file and generates combined visualizations consisting of a 2D slice and 3D isosurface plots. A figure with two subplots is created: the left subplot for 3D isosurfaces and the right for the 2D slice. A logarithmic color normalization (LogNorm) is applied to enhance the visibility of low-density regions. The 2D slice is taken at the midpoint along the z-axis and displayed using `imshow` with a colorbar and axis labels. For the 3D plot, two intermediate isosurface levels within the log-scaled data range are selected; isosurfaces are generated using the marching cubes algorithm and rendered with `plot_trisurf`. Atomic positions are overlaid as scatter points (blue for Ga and red for As), along with lattice vector arrows. Appropriate viewing angles and legends are set. The final visualization is saved as `combined_{{charge}}_{{density}}_{{optimized}}.svg`, suit-

able for publication or presentation.

Final Code Implementation

The following code was developed by the Student Agent to implement DFT calculations, incorporating feedback from both the Mentor Agent and human users, representing the final refined output.

```
"""
This script provides a modular workflow for performing first-principles
calculations of the band structure, total density of states (DOS), and
projected density of states (PDOS) of gallium arsenide (GaAs) using
Quantum ESPRESSO and ASE.

Key features:
- SCF, NSCF, and band structure calculation
- DOS and PDOS post-processing
- Pseudopotential-based DFT input using PBE functionals
- Bandpath definition based on high-symmetry points

Modules:
- create_{structure}(): Generate bulk GaAs structure
- run_{scf}(): Perform SCF calculation
- run_{{band}}_{{structure}}(): Compute band energies along high-symmetry path
- run_{nscf}(): Perform NSCF calculation for DOS
- run_{dos}(): Run DOS post-processing via 'dos.x'
- run_{pdos}(): Run PDOS post-processing via 'projwfc.x'

Requirements:
- ASE
- Quantum ESPRESSO (pw.x, dos.x, projwfc.x)
- NumPy
- Pseudopotentials from PSLibrary
"""

# file: gaas_{workflow}.py
import os
import numpy as np
from ase.build import bulk
from ase.calculators.espresso import Espresso, EspressoProfile
from ase.dft.kpoints import bandpath
import xml.etree.ElementTree as ET

def create_{structure}():
    return bulk('GaAs', crystalstructure='zincblende', a=5.65)
```

```
def run_{scf}(atoms, profile):
    scf_{calc} = Espresso(profile=profile, input_{data}={
        'control': {'calculation': 'scf', 'prefix': 'gaas', 'outdir': './out'},
        'system': {'ecutwfc': 40, 'ecutrho': 320, 'occupations': 'smearing',
            'smearing': 'gaussian', 'degauss': 0.01},
        'electrons': {'conv_{thr}': 1e-6}
    }, pseudopotentials={
        'Ga': 'Ga.pbe-dn-rrkjus_{psl}.1.0.0.UPF',
        'As': 'As.pbe-n-rrkjus_{psl}.1.0.0.UPF'
    }, kpts=(8, 8, 8))
    atoms.calc = scf_{calc}
    energy = atoms.get_{{potential}}_{{energy}}()
    print(f"[SCF] Total energy: {energy:.6f} eV")

def run_{{band}}_{{structure}}(atoms, profile):
    path = bandpath('GXWKGLUWLK', atoms.cell, npoints=100)
    kpts_{bands} = np.zeros((len(path.kpts), 4))
    kpts_{bands}[:, :3] = path.kpts
    bands_{calc} = Espresso(profile=profile, input_{data}={
        'control': {'calculation': 'bands', 'prefix': 'gaas', 'outdir': './out'},
        'system': {'ecutwfc': 40, 'ecutrho': 320, 'occupations': 'smearing',
            'smearing': 'gaussian', 'degauss': 0.01},
        'electrons': {'conv_{thr}': 1e-6}
    }, pseudopotentials={
        'Ga': 'Ga.pbe-dn-rrkjus_{psl}.1.0.0.UPF',
        'As': 'As.pbe-n-rrkjus_{psl}.1.0.0.UPF'
    }, kpts=kpts_{bands}, koffset=False)
    atoms.calc = bands_{calc}
    atoms.calc.calculate(atoms, properties=['bands'], system_{changes}=['cell'])
    print("[Bands] Band structure calculation completed.")

def run_{nscf}(atoms, profile):
    nscf_{calc} = Espresso(profile=profile, input_{data}={
        'control': {'calculation': 'nscf', 'prefix': 'gaas', 'outdir': './out'},
        'system': {'ecutwfc': 40, 'ecutrho': 320, 'occupations': 'smearing',
            'smearing': 'gaussian', 'degauss': 0.01},
        'electrons': {'conv_{thr}': 1e-6}
    }, pseudopotentials={
        'Ga': 'Ga.pbe-dn-rrkjus_{psl}.1.0.0.UPF',
        'As': 'As.pbe-n-rrkjus_{psl}.1.0.0.UPF'
    }, kpts=(12, 12, 12))
    atoms.calc = nscf_{calc}
    atoms.calc.calculate(atoms, properties=['energy'], system_{changes}=['cell'])
    print("[NSCF] NSCF calculation completed.")

def run_{dos}():
```

```
with open('dos.in', 'w') as f:
    f.write("""
&dos
outdir='./out',
prefix='gaas',
fildos='gaas.dos',
emin=-15.0,
emax=15.0,
deltae=0.01,
/
""")
    os.system('dos.x < dos.in > dos.out')
    print("[DOS] DOS calculation completed.")

def run_{pdos}():
    with open('projwfc.in', 'w') as f:
        f.write("""
&projwfc
outdir='./out',
prefix='gaas',
filpdos='gaas.pdos',
emin=-15.0,
emax=15.0,
deltae=0.01,
ngauss=0,
degauss=0.01,
/
""")
        os.system('projwfc.x < projwfc.in > projwfc.out')
        print("[PDOS] PDOS calculation completed.")

def main():
    pseudo_{dir} = '/yourpath'
    profile = EspressoProfile(command='pw.x', pseudo_{dir}=pseudo_{dir})
    atoms = create_{structure}()
    run_{scf}(atoms, profile)
    run_{{band}}_{{structure}}(atoms, profile)
    run_{nscf}(atoms, profile)
    run_{dos}()
    run_{pdos}()

if __{name}_ == "__{{main}}_{}:}"
    main()

"""
This script reads charge density data from a Gaussian CUBE file and generates
```

combined 2D slice and 3D isosurface visualizations using matplotlib and scikit-image.

Key Features:

- Robust parsing of CUBE format (including atomic metadata and volumetric grid)
- Automatic handling of mismatched or missing voxel data
- Visualization includes:
 - XY-plane density slice (log scale)
 - 3D charge isosurfaces with atomic positions
 - Lattice vectors

Requirements:

- NumPy
- Matplotlib
- scikit-image

Make sure a valid 'charge_{density}.cube' file is in the working directory.

Run this script directly to generate 'combined_{{charge}}_{{density}}_{{optimized}}.svg'.

```
import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from mpl_toolkits.axes_grid1 import make_{{axes}}_{{locatable}}
from skimage.measure import marching_{{cubes}}

def read_{{cube}}_{{file}}(filename):
    """Read and parse charge density from a CUBE file."""
    with open(filename, 'r') as f:
        # Skip comment lines
        f.readline()
        f.readline()
        # Read number of atoms and origin
        line = f.readline().split()
        natoms = int(line[0])
        origin = np.array([float(x) for x in line[1:4]])
        # Read voxel grid and spacing
        nx, dx = int(f.readline().split()[0]), np.array([float(x) for x in f.readline().split()])
        ny, dy = int(f.readline().split()[0]), np.array([float(x) for x in f.readline().split()])
        nz, dz = int(f.readline().split()[0]), np.array([float(x) for x in f.readline().split()])
        # Read atomic information
        atoms = []
        for _ in range(abs(natoms)):
            parts = f.readline().split()
            atomic_{{number}} = int(parts[0])
            charge = float(parts[1])
```

```
        position = np.array([float(x) for x in parts[2:5]])
        atoms.append((atomic_{number}, charge, position))
    # Read all remaining density values
    data = np.zeros((nx, ny, nz))
    raw_{values} = []
    for line in f:
        raw_{values}.extend([float(val) for val in line.split()])
    expected = nx * ny * nz
    if len(raw_{values}) != expected:
        print(f"[Warning] Data size mismatch: expected {expected}, got {len(raw_{values})}")
        raw_{values} = raw_{values}[:expected]
    # Populate 3D array
    idx = 0
    for i in range(nx):
        for j in range(ny):
            for k in range(nz):
                data[i, j, k] = raw_{values}[idx] if idx < len(raw_{values}) else 0.0
                idx += 1
    return data, atoms, origin, (dx, dy, dz)

def create_{{combined}}_{{plot}}(data, atoms_{info}, origin, grid_{vectors}):
    """Create combined 2D slice and 3D isosurface visualization."""
    print("[Info] Generating visualizations...")
    plt.figure(figsize=(20, 8))
    # Log scale normalization
    vmin = data[data > 0].min()
    vmax = data.max()
    norm = LogNorm(vmin=vmin, vmax=vmax)
    # 2D Slice
    ax1 = plt.subplot(122)
    slice_{xy} = data[:, :, data.shape[2] // 2]
    im = ax1.imshow(slice_{xy}.T, cmap='viridis', origin='lower', norm=norm)
    ax1.set_{title}('XY Plane Charge Density Slice')
    ax1.set_{xlabel}('X (Grid Points)')
    ax1.set_{ylabel}('Y (Grid Points)')
    divider = make_{{axes}}_{{locatable}}(ax1)
    cax = divider.append_{axes}("right", size="5%", pad=0.1)
    plt.colorbar(im, cax=cax, label='Charge Density (e/Bohr$^{-3}$)')
    # 3D Isosurfaces
    ax2 = plt.subplot(121, projection='3d')
    levels = np.logspace(np.log10(vmin), np.log10(vmax), 4)[1:3]
    colors = plt.cm.viridis(norm(levels))
    for i, level in enumerate(levels):
        try:
            verts, faces, _, _ = marching_{cubes}(data, level)
            ax2.plot_{trisurf}(verts[:, 0], verts[:, 1], faces, verts[:, 2],
```

```

        color=colors[i], alpha=0.4 + i * 0.2, lw=0.5)
    except Exception as e:
        print(f"[Warning] Skipping isosurface at level {level:.2e}: {e}")
# Atoms
element_{colors} = {31: 'blue', 33: 'red'}
element_{labels} = {31: 'Ga', 33: 'As'}
added = set()
for atomic_{num}, _, pos in atoms_{info}:
    color = element_{colors}.get(atomic_{num}, 'gray')
    label = element_{labels}.get(atomic_{num}, 'Unknown') if atomic_{num} not in added else None
    if label:
        ax2.scatter(*pos, c=color, s=150, edgecolor='k', depthshade=False, label=label)
        added.add(atomic_{num})
# Lattice Vectors
for i, vec in enumerate(grid_{vectors}):
    ax2.quiver(*origin, *(vec * data.shape[i]), color='black', arrow_{{length}}_{{ratio}})
ax2.set_{xlabel}('X (Bohr)', labelpad=15)
ax2.set_{ylabel}('Y (Bohr)', labelpad=15)
ax2.set_{zlabel}('Z (Bohr)', labelpad=15)
ax2.zaxis._{axinfo}['juggled'] = (1, 2, 0)
ax2.set_{title}('3D Charge Density Isosurfaces')
ax2.legend(title='Atoms', loc='upper right', bbox_{{to}}_{{anchor}})=(1.15, 0.9)
ax2.view_{init}(elev=30, azim=45)
plt.savefig('combined_{{charge}}_{{density}}_{{optimized}}.svg', dpi=300, bbox_{inches}=)
plt.show()

if __{name}_ == '__{{main}}_':
    cube_{file} = 'charge_{density}.cube'
    if not os.path.isfile(cube_{file}):
        raise FileNotFoundError(f"CUBE file not found: {cube_{file}}")
    charge_{data}, atoms_{info}, origin, grid_{vectors} = read_{{cube}}_{{file}}(cube_{file})
    print(f"[Info] Charge density shape: {charge_{data}.shape}")
    create_{{combined}}_{{plot}}(charge_{data}, atoms_{info}, origin, grid_{vectors})

```

Note: The prompts above represent an initial selection. For improved stability and the full, continuously updated version, please refer to the future open-source GitHub repository: <https://github.com/xqh19970407/PhysAgent>

Note: Figure translations are in progress. See original paper for figures.

Source: ChinaXiv — Machine translation. Verify with original.