AI Scientist for Large-Scale Biomedical Research

Eli Conlin
College of Arts and Sciences,
Boston University, MA
econlin@bu.edu

Prem Rajendran
College of Arts and Sciences
Boston University, MA
premraj@bu.edu

Brooklyn Varela
College of Engineering,
Boston University, MA
brookvar@bu.edu

1 Introduction and Literature Review

Modern artificial intelligence research has introduced machines and software tools that are capable of writing code, conversing in human language, and producing media such as images, videos, and audio. These tools have the ability to connect topics in creative ways and generate experiments, but cannot check whether their conclusions are logical or follow from scientific evidence. Moreover, these tools lack depth and logical reasoning; their ideas are often shallow and lack the ability to understand why these connections might make sense. As a result, these systems sometimes produce results that may look convincing but do not hold up when scrutinized.

The goal of this research project is to build a software tool that can behave similarly to a scientist by reading about discoveries, designing experiments, testing ideas, and producing academic research from its findings. Our team aims to help academics by creating a partner that can handle the repetitive and time-consuming parts of research while reasoning carefully like a human. We plan to design an AI Scientist that not only learns from data and examples but also reasons about what it learns in a structured and logical manner. Our system aims to achieve this by combining pattern recognition via neural networks and reasoning with rules via a knowledge-based system.

In this work we focus specifically on the biomedical domain, which includes hypotheses about molecular mechanisms, drugs, genes, diseases, and pathways—where safety and plausibility are critical. To ground LLM-generated hypotheses in established biomedical evidence, we integrate a curated biomedical knowledge graph (iKraph) as our symbolic layer and Neo4j as the database backend. This lets the agent check plausibility, retrieve literature-backed support, and estimate novelty directly against a large body of biomedical knowledge.

The broader vision of this project is to use AI not just to automate tasks but also to collaborate with humans on scientific discovery. We aim to mark a shift in how scientific research is conducted, by allowing academics to conduct research with an AI partner that has already explored thousands of small hypotheses, helping them identify promising and novel research directions faster. In addition, we plan to construct our tool in a way that is trustworthy. Many current AI tools are unreliable at reasoning based tasks, as they may propose ideas or draw conclusions that are impossible or do not follow from the experimental or theoretical data. Our system aims to follow logical relationships and verify its reasoning at every step, which would make it more transparent and accountable.

Existing research has demonstrated the potential for a fully autonomous AI scientist that goes through the entire research pipeline from end-to-end. The current state-of-the-art example is AI Scientist-v2, which is an agent-based system that iteratively formulates hypotheses, runs experiments via code, analyzes data, and writes scientific manuscripts without human intervention. In particular, this system produced the first workshop paper that was entirely generated using AI and accepted through peer review at an international conference (Yamada et al., 2025). Compared to it's prototype (v1), the new version removes the need for explicit code templates, generalizes across various domains, and uses a novel progressive agentic tree-search to plan experiments via a dedicated experiment manager agent. It even includes a vision-language model (VLM) as an AI reviewer, that critiques both figures and content. The code for this system is open source and can be found in (SakanaAI, 2025).

Other AI scientist systems such as Adam, developed by King et al. (2009), autonomously discovered new yeast genomics, which foreshadowed recent advances. Google's recent AI Co-Scientist

is another example of a multi-agent system that aids scientists by generating and refining novel hypotheses using a generate-debate-evolve approach. This system was built on Google's Gemini large language model (LLM), by coordinating with multiple specialized agents, and uses a tournament-style evolution of ideas to propose experiments (Gottweis et al., 2025). It has demonstrated success in the field bio-medical sciences; suggesting new drug re-purposing candidates and unearthing mechanisms of disease that were later experimentally validated.

Beyond these approaches, LLMs such as GPT and Claude are popular writing assistants for researchers as they help brainstorm ideas, process large batches of research papers, generate drafts, and suggest improvements. Other specialized tools such as Genei, Scholarcy, Semantic Scholar's TLDR, Perplexity, ChatPDF, and SciSpace provide academics with various specialized features such as chatting with large PDF documents, condensing long papers into key points, and connecting with search engines to find scholarly work. Recently, Elicit has become a popular tool among researchers; using LLMs to find relevant papers and credible sources to generate a high-level summary which can be used for literature review. This system is also capable of identifying research themes and gaps in the existing literature by extracting key findings and important data from collected research articles (Whitfield and Hofmann, 2023). Moreover, in the writing stage, tools such as Grammarly and Jenni offer grammatical and style suggestions using contextual information from text, along with auto-generating and rewriting paragraphs based on user prompts. Hence, all these tools have become crucial in research writing and analysis in modern times.

Despite the recent successes of AI writing and research tools, they face various drawbacks. The major drawback is hallucination, which is the tendency of an LLM to generate information that is false and unsubstantiated. Since LLMs work mainly as text generators with limited tools for reasoning and verification, they can produce misleading citations and facts that have the potential of spreading misinformation if not checked by human researchers. This highlights the need for a system that leverages the strengths of AI while protecting against its weaknesses. This project aims to address these weaknesses through the use of neuro-symbolic AI; which are hybrid systems that integrate neural networks with symbolic knowledge based systems. Bougzime et al. (2025) proposes that combining the pattern recognition abilities of neural networks with the reasoning abilities of knowledge graphs can yield optimal results rather than using purely probabilistic neural network models. Moreover, neuro-symbolic systems have been shown to enhance generalization, reasoning, and scalability of AI systems, while providing transparency and interpretability.

In summary, our project aims to automate research in a way where our AI system questions itself at every stage and critically thinks about its own outputs. After generating an idea, it would check whether this idea makes sense given a set of established rules and facts. If the reasoning does not hold up, it revises the idea or marks it as uncertain. By embedding logical reasoning into this system, we aim to create an AI system that creates not just more scientific work, but better scientific work. This mechanism is directly in line with how humans use both intuition and reasoning to refine their work.

2 Experimental Design and Validation Modes

Existing AI scientist models can produce occasionally vacuous, yet sometimes innovative, conference-level papers. Human feedback on Sakana's AI Scientist v2 enabled at least one advancement that achieved conference presentation; yet reviewer feedback on other outputs often cites excessive vagueness as the chief shortcoming. We therefore investigate where in the scientific pipeline AI models may be optimally placed so as to maximize usefulness—particularly when "validation" is offloaded to external, automated sources.

Our investigation proceeds by measuring qualitative shifts in publications when we vary both (a) the degree of human involvement in the seed idea and (b) the plurality of non-human validation modalities. Possible validation modes include (i) simulation-based comparisons of proposed vs. canonical models, (ii) community standards distilled either a priori or via retrieval-augmented generation (RAG) from neighbor nodes in a citation graph, and (iii) dynamically updated soft rule sets. After exhausting purely digital validations, we will further explore how empirical validation might be streamlined to run quickly, cheaply, and with minimal overhead.

We compare AI- and human-generated scientific manuscripts via a hold-out validation paradigm across sections of a curated corpus of machine learning papers spanning subfields. Given a markdown description of a manuscript at varying levels of specification, we analyze how Sakana's AI Scientist v2 outputs diverge from, or parallel, the human "ground truth." From this, we aim to quantify tradeoffs in minimizing human input within AI-driven research. Concurrently, we assess the model's capacity for "auto-completion" of ongoing research, and evaluate three distinct validation assays designed to filter low-quality outputs before human intervention is necessary.

The "seed" provided to the AI may consist of any subset of: Title, Keywords, Summary Sentence, Background/Context, Scientific Gap, Hypothesis, Experiments, Methods, and Discussion. By randomizing over which seed fields are given and analyzing AI's contributions to existing papers, we can evaluate its capacity to serve as a co-scientist at each stage of the research pipeline. To capture both immediate and longer-horizon capabilities, we will run multiple rounds of review interleaving human feedback with alternative checks (e.g. rule-based monitors, simulation, robotic validation).

Once we identify which levels of human input can safely be delegated to AI with negligible waste, we propose replacing some human ideation roles with dynamically generated knowledge graphs as soft rule sets, while preserving program search or optimization to the core AI kernel. These rule sets may derive from concept embeddings (e.g. from ConceptNet or similar) combined with Prolog-style logical rules. When feasible, we will also sample citation-network neighborhoods of the human template to deduce community norms and encode them as soft constraints. For open-ended or underdetermined questions, we will probe whether simulation-based "community standard" proxies (e.g. models of biological tissue dynamics) can serve as surrogate validation. Finally, for domains amenable to it (notably in biomedical or robotics fields), we will explore how external robotic validation can inject empirical feedback into the loop.

This architecture is motivated by several observations in the literature. First, AI-driven scientific output without rigorous human oversight tends toward underspecification and ambiguity—a concern echoed in critiques of AI-generated research (Khlaif et al., 2023). Second, knowledge graph and logical-rule-based reasoning frameworks have been increasingly used to combine symbolic structure with learned embeddings, enabling interpretable rule sets that generalize (Zeng et al., 2023). Third, retrieval-augmented and citation-network-based systems can approximate community standards or mimic peer context (Bianchi et al., 2020). Finally, the gap between claimed explainability and empirical human validation remains large: fewer than 1% of explainable AI papers validate their explanations with human subjects (Suh et al., 2025), which underlines the need for robust external validation pipelines.

In sum, by systematically varying human seeding and validation schemes across a controlled corpus, we hope to chart where AI scientists can most fruitfully plug into the research cycle, and how external validation (digital or empirical) can be leveraged to reduce human burden while maintaining rigor.

3 Hypothesis Validation Using iKraph

Our system is built around a neuro-symbolic loop in which a large language model (LLM) proposes hypotheses, consults a structured knowledge graph (KG) to assess their feasibility and novelty, and promotes only those ideas that are both scientifically grounded and operationally executable within our PyLabRobot-based simulation environment. The KG serves as the central substrate that links three perspectives: existing scientific knowledge in biomedical domains, and the evolving set of hypotheses, experiments, and results generated by the AI Scientist itself. In this subsection, we describe how iKraph interfaces with the LLM and how it supports the downstream robotic simulation pipeline.

3.1 Existing Implementations

Various recent studies have been performed on large-scale scientific KGs and KG-driven research agents. iKraph (Zhang et al., 2025) shows that a comprehensive, automatically constructed biomedical KG can reliably integrate heterogeneous literature and database signals at scale to support AI-powered discovery, demonstrating that richly populated machine-readable research graphs are both feasible and practically useful as foundations for scientific workflows. Gu and Krenn (2024) use an evolving scientific KG in combination with a language model to generate research ideas and find that the KG-grounding leads to ideas rated as more

interesting and credible by domain experts than those from unguided LLMs, supporting our choice to treat the KG as a primary scaffold for hypothesis generation rather than a passive index.

SciToolAgent (Ding et al., 2025) further illustrates how a knowledge-graph-driven architecture can orchestrate many specialized tools safely and effectively, using a KG to encode tool capabilities and constraints and to mediate LLM calls, a pattern that we adapt by encoding the PyLabRobot API and its constraints directly into our graph so that only executable protocols are proposed. At the level of schema design, SciKGraph (Dalle and Cesar, 2020) shows how to structure a scientific field into entities and relations that capture topics, methods, and their connections. AI-KG (Dessì et al., 2020) automatically builds an AI-specific KG on tasks, methods, and metrics from hundreds of thousands of papers; together, they provide concrete, validated blueprints to represent research knowledge in a way that supports large-scale analysis and intelligent services. Since the focus of this paper remains on biomedical research, iKraph was deployed as the knowledge graph.

3.2 iKraph Schema and Implementation

We deployed iKraph, a large-scale biomedical knowledge graph, as the symbolic constraint layer in our neuro-symbolic loop, integrating entities such as Chemicals/Drugs, Genes, Diseases, Pathways, Cell Lines, and Species, and relations including associations and signed/causal effects. We load iKraph into Neo4j (v5) and query it with Cypher to check plausibility/safety of an LLM-proposed hypothesis, retrieve supporting evidence (PubMed-backed paths), and estimate novelty from a local graph structure. This symbolic check gates which hypotheses advance to protocol generation and simulation.

We use iKraph as-is (read-only) and import it into Neo4j. Nodes represent biomedical entities (e.g., Chemical/Drug, Gene, Disease, Pathway, CellLine, Species), and edges capture literature- or database-derived relations (e.g. Positive Correlation, Association, Inhibits, Activates), each optionally annotated with score/probability and PubMed evidence. We do not extend or redesign the schema; our contribution is the integration and query layer that constrains the LLM.

3.3 LLM–iKraph Interaction

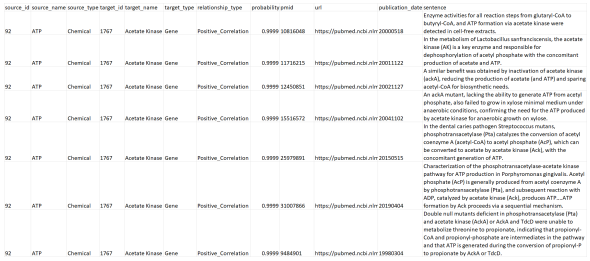
The LLM engages with iKraph in a retrieval-and-refinement loop: given a problem description or seed idea, it proposes a candidate hypothesis and experimental sketch, which is then parsed into a structured query against iKraph to retrieve relevant prior results, analogous configurations, and overlapping hypotheses. Rather than exposing raw triples, we return concise, knowledge-grounded results to the LLM, so its subsequent reasoning is explicitly conditioned on verified literature patterns and executable capabilities, in line with findings that KG-augmented prompts improve the quality and groundedness of scientific ideas. Using this context, the model jointly evaluates feasibility (required operations and resources exist and respect constraints), scientific plausibility (no strong conflicts with encoded evidence), and novelty (under-explored combinations in the graph); hypotheses failing these checks are revised, while consistent and operationally realizable ones are promoted to formal experiment specifications. In this way, iKraph functions as a dynamic filter that prunes implausible or inactionable ideas before they reach the simulation stage and accumulates a structured record of vetted hypotheses and outcomes for future cycles.

For each hypothesis generated by the LLM, we extract key entities (e.g. drug/chemical, target/-gene, disease/pathway) and issue parameterized Cypher queries to test consistency (conflicts vs. known edges), fetch supporting evidence (PubMed co-mentions and edge annotations), and compute novelty (e.g. degree-/path-based rarity features). Hypotheses lacking support or violating constraints are pruned or revised; supported hypotheses move to the PyLabRobot simulation layer.

3.4 Query Templates and Evidence Retrieval

We implement a small library of Cypher templates that the agent fills from hypothesis slots (entity names/IDs and relation types). Each template returns: endpoints (IDs, names, types), the relationship type and any probability/score fields on the edge, and PubMed evidence via co-mention links, including pmid, an evidence URL, and optional sentence/date fields. The query below retrieves Positive Correlations between ATP and acetate kinase, along with PubMed-backed evidence for each edge; Figure 1 shows the result table.

```
MATCH (s:Entity)-[r:Positive_Correlation]-(t:Entity)
WHERE
```



source_id	source_name	source_type	target_id	target_name	target_type	relationship_type	probability	pmid	url	publication_date	sentence
92	ATP	Chemical	1767	Acetate Kinase	Gene	Positive_Correlation	0.9999	10816048	https://pubmed.ncbi.nlm.nih.gov/10000518		Enzyme activities for all reaction steps from glutaryl-CoA to butyryl-CoA, and ATP formation via acetate kinase were detected in cell-free extracts.
92	ATP	Chemical	1767	Acetate Kinase	Gene	Positive_Correlation	0.9999	11716235	https://pubmed.ncbi.nlm.nih.gov/20011122		In the metabolism of Lactobacillus sanfranciscensis, the acetate kinase (AK) is a key enzyme and responsible for dephosphorylation of acetyl phosphate with the concomitant production of acetate and ATP.
92	ATP	Chemical	1767	Acetate Kinase	Gene	Positive_Correlation	0.9999	12450851	https://pubmed.ncbi.nlm.nih.gov/20021127		A similar benefit was obtained by inactivation of acetate kinase (ackA), reducing the production of acetate (and ATP) and sparing acetyl-CoA for biosynthetic needs.
92	ATP	Chemical	1767	Acetate Kinase	Gene	Positive_Correlation	0.9999	15516572	https://pubmed.ncbi.nlm.nih.gov/20041102		An ackA mutant, lacking the ability to generate ATP from acetyl phosphate, also failed to grow in xylose minimal medium under anaerobic conditions, confirming the need for the ATP produced by acetate kinase for anaerobic growth on xylose.
92	ATP	Chemical	1767	Acetate Kinase	Gene	Positive_Correlation	0.9999	25979891	https://pubmed.ncbi.nlm.nih.gov/20150515		In the dental caries pathogen Streptococcus mutans, phosphotransacetylase (Pta) catalyzes the conversion of acetyl coenzyme A (acetyl-CoA) to acetyl phosphate (AcP), which can be converted to acetate by acetate kinase (AckA), with the concomitant generation of ATP.
92	ATP	Chemical	1767	Acetate Kinase	Gene	Positive_Correlation	0.9999	31007866	https://pubmed.ncbi.nlm.nih.gov/20190404		Characterization of the phosphotransacetylase-acetate kinase pathway for ATP production in Porphyromonas gingivalis. Acetyl phosphate (AcP) is generally produced from acetyl coenzyme A by phosphotransacetylase (Pta), and subsequent reaction with ADP, catalyzed by acetate kinase (Ack), produces ATP. ...ATP formation by Ack proceeds via a sequential mechanism.
92	ATP	Chemical	1767	Acetate Kinase	Gene	Positive_Correlation	0.9999	9484901	https://pubmed.ncbi.nlm.nih.gov/19980304		Double null mutants deficient in phosphotransacetylase (Pta) and acetate kinase (AckA) or AckA and TdcD were unable to metabolize threonine to propionate, indicating that propionyl-CoA and propionyl-phosphate are intermediates in the pathway and that ATP is generated during the conversion of propionyl-P to propionate by AckA or TdcD.

Figure 1: Results of querying iKraph for `Positive_Correlation` relations between adenosine triphosphate (ATP) and acetate kinase, with PubMed-backed evidence for each edge.

```
  (
    toLower(s.`official name`) = 'adenosine triphosphate' OR
    toLower(s.`common name`)  = 'atp' OR
    toLower(s.`official name`) = 'atp'
  )
  AND (
    toLower(t.`official name`) CONTAINS 'acetate kinase' OR
    toLower(t.`common name`)  CONTAINS 'acetate kinase'
  )

OPTIONAL MATCH
↪  (s)-[ms:MENTIONED_IN]->(p:PubMed)<-[mt:MENTIONED_IN]-(t)

RETURN DISTINCT
  s.id                                                    AS source_id,
  coalesce(s.`official name`, s.`common name`)            AS
  ↪  source_name,
  s.type                                                  AS source_type,
  t.id                                                    AS target_id,
  coalesce(t.`official name`, t.`common name`)            AS
  ↪  target_name,
  t.type                                                  AS target_type,
  type(r)                                                 AS
  ↪  relationship_type,
  toFloat(coalesce(r.prob, ms.prob, mt.prob))             AS
  ↪  probability,
  p.pmid                                                  AS pmid,
  CASE WHEN p.pmid IS NULL
       THEN NULL
       ELSE 'https://pubmed.ncbi.nlm.nih.gov/' + p.pmid
  END                                                     AS url,
  p.date                                                  AS
  ↪  publication_date,
  p.sentence                                              AS sentence
ORDER BY probability DESC, pmid;
```

4 Robotics Simulation as a Validation Layer

Large language models are increasingly capable of generating wet-lab protocols from high-level natural-language descriptions. Yet despite their fluency, these systems routinely produce experimental procedures that are physically impossible, internally inconsistent, or incompatible with real laboratory equipment. Common failure cases include aspirating more liquid than exists in a well, dispensing into nonexistent wells, reusing contaminated tips, omitting essential steps such as tip pickup, or inventing labware that does not appear on the available deck. These errors arise because LLMs optimize for linguistic plausibility rather than physical feasibility, and because they lack grounded models of laboratory constraints. As a result, text-only inspection is insufficient to assess whether an LLM-generated protocol could be executed by an actual liquid-handling robot.

To address this gap, we introduce a robotic simulation layer that evaluates experimental protocols before they are accepted by the AI Scientist. The rationale is to create a middle ground between purely symbolic validation and full hardware execution. A robotic simulation provides an environment where each pipetting step, transfer, or mixing action can be interpreted as a real robot would interpret it, using explicit constraints on volumes, wells, deck layout, and tip usage (Wierenga et al., 2023a). This approach allows the system to reject protocols that violate physical or operational rules long before reaching a real laboratory, thereby serving as a safeguard against hallucinated or unsafe instructions.

Existing AI-Scientist frameworks, such as AI Scientist v2 and FreePhDLabor, validate experiments through code execution, metric evaluation, or cross-agent critique, but they do not incorporate robotic feasibility checks. Conversely, real robotic platforms like Opentrons or Hamilton systems provide execution-level validation but require costly hardware and can be unsafe when protocols contain errors. Our approach fills the gap between these extremes by introducing, to our knowledge, the first feasibility-checking layer for LLM-generated experimental protocols built on PyLabRobot, an open-source framework for simulating liquid-handling operations.

4.1 PyLabRobot

The PyLabRobot implementation (Wierenga et al., 2023b) is an open source Python framework for programming, simulating, and controlling liquid-handling robots. It provides a standardized abstraction layer for common laboratory hardware, including multichannel pipetting heads, carriers, tip racks, and a wide range of 96-well and 384-well plate formats.

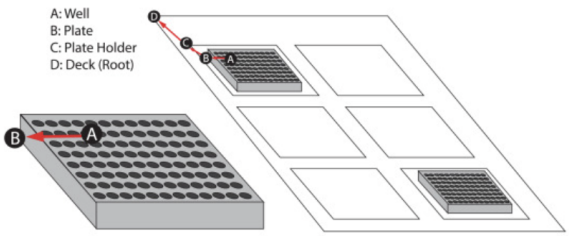


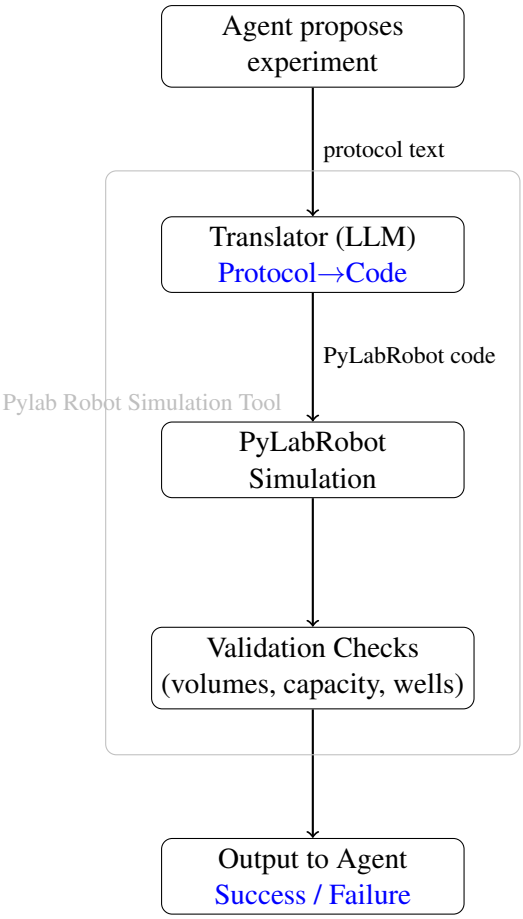
Figure 2: Hierarchical representation of labware in PyLabRobot. A liquid-handling deck is organized as a tree structure in which the deck (D) contains plate holders (C), each of which contains one or more plates (B). Plates are themselves composed of individual wells (A) that serve as the fundamental units for aspirate and dispense operations. Reprinted from Wierenga et al. (2023a)

Through a modular resource system, PyLabRobot exposes deck layouts, well addressing conventions, and labware geometry in a unified way, enabling high level protocol specification independent of the underlying hardware vendor. The framework also includes simulated backends, such as the Chatterbox backend used in this work, which allow developers to execute protocols without a physical robot and to inspect operations such as aspirate, dispense, mix cycles, and tip movements. Because PyLabRobot is fully extensible and community maintained, it has become a widely adopted tool for research on automated laboratory workflows, liquid-handling simulation, and LLM-based protocol generation.

4.2 End-to-End Robotic Simulation Pipeline

4.2.1 Pipeline Overview

To operationalize robotic validation within the AI Scientist framework, we developed a complete pipeline that provides an interface between the high-level reasoning of the Agent and the low-level constraints of laboratory automation. The architecture consists of three main components: a translation module that converts free-form natural language into PyLabRobot code, a simulation core that executes each physical operation within a virtual liquid-handling environment, and a validation layer that inspects the resulting execution trace for inconsistencies or physical violations. Together, these components enable the system to assess feasibility before committing to real-world implementation.



4.2.2 Translator Module

The pipeline begins when the Agent provides a natural-language protocol describing an experiment. When a user supplies a natural language protocol, the translator first queries a knowledge base that contains a structured dump of PyLabRobot capabilities and documentation. The retrieved entries are inserted into the prompt as a reference section, which gives the model concrete information about available decks, carriers, plates, tip racks and liquid handling primitives instead of letting it invent resources on its own. On top of this retrieved context, we add a long system message that specifies strict global requirements for the generated code. This message enforces that the output must be valid Python only, must define a single run_protocol routine that constructs a deck with a chatterbox backend, places labware on the correct carriers, sets up a LiquidHandler, and returns the deck and backend object. The same prompt also encodes domain constraints that are important for later validation, such as explicit initialization of source wells with set_liquids, manual maintenance of volume trackers for every aspirate and dispense call, correct ordering of tip pickup, aspiration and dispensing, and

Protocol Text	Protocol Code

Transfer 150 µL from A1 to C3	`await lh.pick_up_tips(tip_rack["A1"])`
	`await lh.aspirate([source_well_A1], vols=[150.0])`
	`await lh.dispense([dest_well_C3], vols=[150.0])`
	`await lh.return_tips()`

Copy 100 µL from columns 1–6 of a source 96 well plate to the matching wells of a destination plate using an 8 channel pipette.	`for col in range(1, 7):`
	`  source_wells = [source_plate[f"{row}{col}"][0] for row in "ABCDEFGH"]`
	`  dest_wells = [dest_plate[f"{row}{col}"][0] for row in "ABCDEFGH"]`
	`  await lh.pick_up_tips(tip_rack["A1:H1"])`
	`  await lh.aspirate(source_wells, vols=[100.0] * 8)`
	`  await lh.dispense(dest_wells, vols=[100.0] * 8)`
	`  await lh.return_tips()`

Add 50 µL substrate to all wells of a 96 well assay plate, then add 25 µL enzyme from column 1 of a source plate to each assay column and mix three times.	`# Step 1: Add 50 µL substrate to all wells of the assay plate`
	`for col in range(1, 13):`
	`  substrate_wells = [substrate_plate[f"{row}{col}"][0] for row in "ABCDEFGH"]`
	`  dest_wells = [assay_plate[f"{row}{col}"][0] for row in "ABCDEFGH"]`
	`  await lh.pick_up_tips(tip_rack["A1:H1"])`
	`  await lh.aspirate(substrate_wells, vols=[50.0] * 8)`
	`  await lh.dispense(dest_wells, vols=[50.0] * 8)`
	`  await lh.return_tips()`
	`# Step 2: Add 25 µL enzyme from column 1 of source plate to each assay column and mix three times`
	`enzyme_wells = [source_plate[f"{row}1"][0] for row in "ABCDEFGH"]`
	`for col in range(1, 13):`
	`  dest_wells = [assay_plate[f"{row}{col}"][0] for row in "ABCDEFGH"]`
	`  await lh.pick_up_tips(tip_rack["A1:H1"])`
	`  await lh.aspirate(enzyme_wells, vols=[25.0] * 8)`
	`  await lh.dispense(dest_wells, vols=[25.0] * 8)`
	`  # Mix three times`
	`  for _ in range(3):`
	`  await lh.aspirate(dest_wells, vols=[50.0] * 8)`
	`  await lh.dispense(dest_wells, vols=[50.0] * 8)`

Table 1: Protocol text mapped to the corresponding PyLabRobot code.

the requirement that wells are addressed through canonical A1 to H12 identifiers. To further stabilize the behavior of the model, the translator uses several few shot examples that pair informal protocol descriptions with complete PyLabRobot scripts. These examples cover simple serial dilution on a single plate, plate to plate replication, cherry picking from scattered wells, and a multi step enzyme assay involving dispensing, mixing and timed incubation. The translator relies on a GPT-4o model to convert natural-language protocol descriptions into executable PyLabRobot code. This configuration encourages deterministic outputs that reuse the same deck layout, carrier placement and coding patterns as the examples, while adapting volumes, well addresses and numbers of cycles to the new task. The result is a full Python script that is syntactically valid, adheres to PyLabRobot conventions, and respects the volume tracking rules required by the downstream simulation and validation stages.

4.3 Validation Rules and Error Detection

The validation stage evaluates whether a translated protocol exhibits physically meaningful liquid-handling behavior and whether any impossible conditions arise during execution. After the simulation completes, the system inspects the resulting well volumes to ensure that no negative liquid amounts were generated and that no well exceeds its maximum capacity. These checks serve as coarse indicators of physically infeasible operations, such as excessive aspiration or overfilling. In addition, the validator assesses whether the protocol resulted in any measurable liquid movement at all; runs in which every well remains at zero microliters are marked as unsuccessful, since they indicate that the procedure either produced no actionable steps or failed silently. In addition to these domain-level checks, the system incorporates a mechanism for catching and correcting translation errors. When the translation component produces robotic code that fails to execute, the resulting error message is fed back into the translator, prompting it to regenerate a corrected version. This feedback–repair cycle is attempted up to five times. If all attempts still produce invalid or non-executable output, the system terminates the process and reports the final error as the reason for failure. Beyond these domain-specific rules, the simulation layer also captures higher-level execution errors, such as failures in the generated robotic code or incompatible resource configurations, which are returned as explicit failure messages. Taken together, these mechanisms provide a lightweight yet effective filter that detects non-functional or physically invalid protocols before they are accepted by the system.

5 Simulations as an Epistemic Constraint

The systems described so far use simulation in a deliberately narrow but important way. Rather than

treating simulators as oracles that automatically certify correctness, we use them as sources of friction that constrain what the AI Scientist is allowed to consider a "good" experiment. In practice, this takes two complementary forms: simulation-aware experimental planning and explicit robotic feasibility checking.

On the planning side, the new Experimental Agent generates hypotheses and experimental programs in the language of constraint-based metabolic modeling. Given a high-level biological goal (e.g., increasing ATP production capacity in *E. coli*), the agent does not simply outline informal experiments. Instead, it produces a schema-compliant research plan that is explicitly parameterized in terms of growth thresholds, ATP maintenance fluxes, single- and double-gene knockouts, media conditions, and robustness checks. Crucially, the output is structured as a JSON object designed to be executable by downstream tooling rather than as a free-form narrative. The plan specifies which COBRApy routines should be run (e.g., FBA, pFBA, flux variability analysis), how to treat essential genes, what media perturbations to explore, and how to quantify robustness via bootstrap perturbations of exchange bounds and maintenance parameters. Even though these simulations are not yet executed in the current pipeline, the agent is already forced to frame its ideas in terms of quantities and procedures that a solver could, in principle, evaluate.

On the execution side, the PyLabRobot layer provides a separate form of simulation, focused not on cellular metabolism but on robotic feasibility. Here, natural-language protocols are translated into PyLabRobot scripts and executed against a simulated deck. The simulation enforces basic but non-negotiable physical constraints, such as volume conservation, well capacities, valid well addresses, and correct ordering of tip pickup, aspiration, and dispense. Protocols that fail to move any liquid, generate negative volumes, exceed well capacities, or crash during execution are rejected. This does not guarantee that a protocol is scientifically meaningful, but it does guarantee that the protocol is at least physically executable within the abstraction provided by PyLabRobot.

Taken together, these two uses of simulation shift the role of the AI Scientist from unconstrained text generation toward constrained, tool-aware planning. The Experimental Agent cannot talk about arbitrary gene edits or metabolic effects; it must talk in terms of knockouts, fluxes, and growth thresholds that COBRApy could evaluate. Similarly, the protocol translator cannot invent arbitrary laboratory operations; it must produce scripts that pass PyLabRobot's physical checks. In both cases, simulation functions as an epistemic constraint: it does not prove that a hypothesis is true, but it does shape which hypotheses and protocols are even admissible. This is a step toward the kind of "validation-first" AI scientists proposed in recent work (Yamada et al., 2025; Gottweis et al., 2025), where external structure curbs the tendency of LLMs to hallucinate plausible but impossible experiments.

Importantly, we emphasize that in the current system these constraints are primarily anticipatory. The Experimental Agent produces COBRApy-ready plans without executing them; the PyLabRobot simulator enforces liquid-handling feasibility but does not yet connect to real hardware. Nonetheless, the logs show that both layers already change the character of the system's behavior: ideas are expressed in terms of quantities that simulations could check, and protocols must satisfy basic physical regularities. This suggests that even partial integration of simulation into the loop can meaningfully alter the epistemic habits of AI-driven research agents, and sets the stage for future work where these simulations are actually run and their results fed back into the reasoning process.

6 Experimental Comparison: Baseline vs Simulation-Augmented System

To understand how these additions change the behavior of an AI Scientist in practice, we qualitatively compared the baseline FreePhDLabor system to the simulation-augmented version that includes both the new Experimental Agent and the PyLabRobot-based validation layer. Rather than focusing on benchmark scores, we analyze the concrete artifacts produced in the system logs and how they reflect different notions of what it means to "do" science.

In the baseline configuration, FreePhDLabor operates as a multi-agent research pipeline in the style of Curie (**?**) and The AI Scientist (**?**). An `IdeationAgent` produces hypotheses and high-level project plans; an `ExperimentationAgent` elaborates experiments; a `ReviewerAgent` critiques drafts; and a `ManagerAgent` coordinates tasks and decides when to stop. The logs show a heavy emphasis on process metrics and pipeline

health—claim error rates, reproducibility across reruns, reviewer scores, novelty signals, and time-to-completion—and on maintaining a continuous integration style of experimentation. The agents are good at keeping the workflow moving, but their outputs are often descriptive rather than operational, and confounds or model assumptions are rarely elevated to first-class objects.

After introducing the new Experimental Agent, the system begins to produce a different kind of artifact. Given the task of optimizing ATP production capacity in *E. coli*, the agent outputs a detailed, confound-aware research plan encoded as a JSON object that conforms to an explicit `working_idea.json` schema. The plan names a primary genome-scale metabolic model (iML1515), notes fallback models (iJO1366, iAF1260), and defines clear objectives: maximizing ATP maintenance flux (ATPM) under epsilon-constrained growth, mapping Pareto frontiers between biomass and ATP capacity, and quantifying robustness under media and parameter perturbations. It enumerates specific confounds (alternative optima, thermodynamically infeasible cycles, gene essentiality, media leakage, oxygen availability, maintenance parameters, model dependence, and objective formulation) and associates each with concrete mitigations that can be implemented via COBRApy procedures. It also lays out a stepwise experimental plan, including environment setup, model verification, media definition, wild-type baselines, single and double knockout screening, robustness analysis, looplessness checks, and frontier comparison.

Two features distinguish this from baseline behavior. First, the output is directly executable by downstream agents: the plan is saved, promoted by `ManagerAgent` to a canonical `working_idea.json`, and can in principle be consumed by a future simulation runner without further reformatting. Second, the plan treats confounds as central design targets rather than after-the-fact caveats. Alternate optima, thermodynamic loops, and model discrepancies are presented as reasons to design more discriminative computational experiments, not simply as problems to mention in a discussion section. This contrasts with the baseline logs, where validation is often described in terms of reviewer checklists or aggregate metrics but rarely grounded in model-specific pathologies.

A similar shift appears on the robotics side. Prior to integrating the PyLabRobot layer, there is no mechanism to test whether an LLM-generated protocol is physically executable. Protocol-like text could be fluent yet impossible: aspirating more volume than exists, dispensing into undefined wells, or omitting essential tip-handling steps. With the simulation layer in place, every protocol must survive translation into PyLabRobot code and execution on a simulated deck. The validation logs show explicit checks for negative volumes, overfilled wells, and null protocols that leave the entire plate at zero microliters. When translation fails due to missing imports or inconsistent labware names, the translator is forced into an error–repair loop rather than silently passing an invalid protocol upstream.

Across both case studies, the effect of the new components is not to make the system "smarter" in a general sense, but to change what it means for the system to consider its task complete. In the baseline system, completion is largely defined by internal coherence and reviewer alignment: an experiment or paper is acceptable when it satisfies a checklist and passes peer-style scrutiny. In the simulation-augmented system, completion additionally requires that plans be structurally compatible with downstream simulators. The Experimental Agent must output COBRApy-ready JSON; the translator must produce PyLabRobot scripts that execute without violating basic physical constraints. This does not solve the full problem of scientific validity, but it narrows the space of admissible outputs in a way that aligns more closely with how human scientists reason about feasibility, and it demonstrates how integrating simulation-aware components can move an AI Scientist from "talking about experiments" toward producing artifacts that could, with additional infrastructure, be run.

7 Limitations and future work

The simulation-centric components described above introduce their own limitations. First, the metabolic planning layer is, at present, purely prospective. The Experimental Agent generates a COBRApy-oriented research plan with detailed objectives, confounds, and robustness analyses, but the current system does not yet execute these simulations or close the loop by feeding results back into the agent. As a result, the agent behaves as though it will be evaluated by constraint-based models without ever experiencing that feedback. This is still useful as a form of anticipatory regularization, but it falls short of the full neuro-symbolic

loop envisioned in work on KG-grounded and tool-integrated research agents (Zhang et al., 2025; Gu and Krenn, 2024; Ding et al., 2025). Until simulated fluxes, infeasibilities, and sensitivity analyses actually influence subsequent hypotheses, the system risks overestimating the practical value of its own experimental designs.

Second, the PyLabRobot simulation layer focuses on a narrow slice of experimental validity: volume conservation, well capacities, addressing correctness, and basic execution errors. These checks are necessary but not sufficient. A protocol that perfectly conserves volume can still be scientifically meaningless, poorly controlled, or misaligned with the stated hypothesis. Conversely, the simulation is only as accurate as the abstractions encoded in PyLabRobot. Real liquid-handling systems exhibit effects that the simulator does not model, such as droplet adhesion, evaporation, timing constraints, and hardware-specific behavior. Overreliance on simulator success as a proxy for real-world feasibility risks giving a false sense of security, particularly in domains where physical edge cases matter.

Another limitation of the PyLab Robot tool concerns the system's strong dependence on the correctness of the code gener- ated by the LLM translator. Any incorrect import, missing initialization step, misplaced rail assign- ment, or improperly defined resource causes the simulation to fail before validation even begins, cre- ating a circular dependency. Moreover, the transla- tor highlights broader limitations of current LLMs. Even when supplied with a structured RAG refer- ence of all PyLabRobot methods and examples, the model sometimes forgets key imports, mis- matches resource names, or skips essential setup steps. Counterintuitively, providing more docu- mentation can reduce coherence by overwhelming the prompt. Achieving stable translation perfor- mance may therefore require targeted fine-tuning, constrained decoding, or a more curated set of ex- emplars rather than relying on long textual capabil- ity dumps.

Fourth, both simulation layers impose their own kinds of bias. The metabolic planning agent is tuned to think in terms of genome-scale metabolic models with well-defined ATP maintenance reactions and biomass objectives. This encourages experiments that are easy to encode in FBA-style frameworks and discourages hypotheses that fall outside that modeling regime. Similarly, the PyLabRobot layer encourages protocols that resemble the few-shot examples and deck layouts used in the translator prompt, potentially biasing the system toward a small subset of experimental formats. In both cases, the AI Scientist may be more likely to propose experiments that are convenient for the simulator rather than those that best address the underlying scientific question, echoing broader concerns about "simulator-driven" research design.

Finally, there is an unresolved tension between fidelity to the Agent's intent and satisfaction of simulator constraints. The translator often has to infer missing details such as carrier names, initial volumes, or plate positions in order to produce executable PyLabRobot code. Allowing such inference improves success rates but can drift away from the original protocol; enforcing strict fidelity leads to frequent failures when the input is underspecified. A similar issue arises in metabolic planning: insisting on complete, model-consistent specifications may cause the agent to discard partially formed but potentially valuable ideas. Designing mechanisms that surface these gaps to human users, rather than silently repairing or rejecting them, remains an open challenge and a key direction for future work.

References

Federico Bianchi, Giacomo Rosselli, Luca Costabello, Matteo Palmonari, and Pasquale Minervini. 2020. Knowledge graph embeddings and explainable ai. *arXiv preprint arXiv:2004.08246*.

Oualid Bougzime, Samir Jabbar, Christophe Cruz, and Frédéric Demoly. 2025. Unlocking the potential of generative ai through neuro-symbolic architectures: Benefits and limitations. *arXiv (Cornell University)*.

Mauro Dalle and Julio Cesar. 2020. Scikgraph: A knowledge graph approach to structure a scientific field. *Journal of Informetrics*, 15:101109–101109.

Danilo Dessì, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, Enrico Motta, and Harald Sack. 2020. Ai-kg: An automatically generated knowledge graph of artificial intelligence. *International Semantic Web Conference*, pages 127–143.

Keyan Ding, Jing Yu, Junjie Huang, Yuchen Yang, Qiang Zhang, and Huajun Chen. 2025. Scitoolagent: a knowledge-graph-driven scientific agent for multitool integration. *Nature Computational Science*, 5:962–972.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom

Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, and 15 others. 2025. Towards an ai co-scientist.

Xuemei Gu and Mario Krenn. 2024. Interesting scientific idea generation using knowledge graphs and llms: Evaluations with 100 research group leaders. *arXiv (Cornell University)*.

Ziad N. Khlaif, Samar Salha, Saida Affouneh, and Suhail Abdallah. 2023. The potential and concerns of using ai in scientific research. *Frontiers in Artificial Intelligence*.

Ross D. King, Jem Rowland, Stephen G. Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N. Soldatova, Andrew Sparkes, Kenneth E. Whelan, and Amanda Clare. 2009. The automation of science. *Science*, 324:85–89.

SakanaAI. 2025. Github - sakanaai/ai-scientist-v2: The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search.

A. Suh, I. Hurley, N. Smith, and H. C. Siu. 2025. Fewer than 1% of explainable ai papers validate explainability with humans. *arXiv preprint arXiv:2503.17651*.

Sharon Whitfield and Melissa A. Hofmann. 2023. Elicit: Ai literature review research assistant. *Public Services Quarterly*, 19:201–207.

Rick P. Wierenga, Stefan M. Golas, Wilson Ho, Connor W. Coley, and Kevin M. Esvelt. 2023a. Pylabrobot: An open-source, hardware-agnostic interface for liquid-handling robots and accessories. *Device*, 1:100111.

Rick P. Wierenga, Stefan M. Golas, Wilson Ho, and PyLabRobot Contributors. 2023b. PyLabRobot: Hardware-agnostic interface for liquid-handling robots. GitHub repository. Accessed: December 10, 2025.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. 2025. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv (Cornell University)*.

Zhen Zeng, Jian Zhang, Wei Liu, and Bing Wang. 2023. Logical rule-based knowledge graph reasoning. *Mathematics*, 11(21):4486.

Yuan Zhang, Xin Sui, Feng Pan, Kaixian Yu, Keqiao Li, Shubo Tian, Arslan Erdengasileng, Qing Han, Wanjing Wang, Jianan Wang, Jian Wang, Donghu Sun, Henry Chung, Jun Zhou, Eric Zhou, Ben Lee, Peili Zhang, Xing Qiu, Tingting Zhao, and Jinfeng Zhang. 2025. A comprehensive large-scale biomedical knowledge graph for ai-powered data-driven biomedical research. *Nature Machine Intelligence*, 7:602–614.