Can an AI Scientist really discover knowledge?

Ant Rowstron and Aayush Chadha

Advanced Research and Invention Agency (ARIA), *UK*

Version 1.0 | 5th June 2026 |

Introduction

At ARIA, we constantly grapple with bottlenecks to scientific and technological progress. As a frontier funding agency, we are always on the lookout for new enabling technologies and exploring how far they can be pushed. Of late, we have started experimenting with AI in science.

One of the most intriguing aspects currently is the emergence of scientific reasoning systems, often referred to as AI Scientists, coupled with autonomous lab systems that generate hypotheses, design experiments, run them and then process the results and loop around again. This could meaningfully shift how scientific ideas are discovered and tested, with the potential for massive acceleration in knowledge discovery.

We are just seeing the early shoots of growth in this area, and as an agency we have been exploring how to understand where the capability frontier is. This page will be a living, public record of (mis)adventures as we explore questions around capabilities, configurations and resource requirements of these systems. By sharing our experimentation as openly as possible, we hope to spark new conversations, attempt to settle some provocative questions and broadly get the R&D ecosystem across all disciplines tinkering with these systems: exploring what is possible today, what ought to be possible and how that might reshape the scientific process in each discipline.

A question often debated with us is whether AI Scientists can infer new rules of the universe even when given substantial resources, such as generous token budgets and large context windows. The argument against this usually focuses on models being trained over a probability distribution: they may only be able to uncover simple derivations of rules they have previously seen, and no matter how many instruments you plug into them, they will never uncover genuinely new knowledge.

This sounded like a great first exploration for us. Is there a way to test if models can uncover hidden rules put in place by us, by designing experiments that call upon fictitious but well-defined instruments, especially under a generous token budget? At the very least, exploring this would give us some understanding of model reasoning capabilities where we control hyperparameter choices, harness, and task design without having to rely on results designed to make models look good. This also ended up being a simple demonstration internally to help think about questions around the utility of AI-based automated labs and usage practices in experimental workflows across our funded activities.

So, we created Albert, a simple AI Scientist-like reasoning layer, that is designed to use instruments to explore phenomena in virtual "environments" that present simplistically some of the concepts that underpin biology, ecology, physics, causal inference, and chemistry but in a parallel universe governed by completely different laws. Many benchmarks test the ability of an LLM to recall knowledge correctly, but we wanted to explore hypothesis generation and selection, experiment design, and then interpreting the results, and looping around this workflow until Albert felt it had discovered all the laws.

Albert: Architecture of an AI Scientist

Before we discuss what Albert found, the design of Albert is a simple phased multi-agent loop - with two core local memory mechanisms, the raw log of all interactions and a knowledge base:

1. Explorer
    - Performs initial exploration.
    - Uses lab tools directly to gather baseline observations.
2. Theorist
    - Reads the accumulated knowledge base plus recent raw experiment log.
    - Produces multiple hypotheses with different levels of risk and associated experiment designs.
3. Director
    - Chooses the most useful hypothesis to test next.
    - Can decide to terminate the exploration and submit a theory when it believes the complete theory is solved.
4. Technician

- Executes the Director's chosen experiment.
- Uses lab tools provided and is instructed not to invent the research plan, simply to follow it.

5. Chronicler

- Only runs when logs or token usage approach context limits.
- Compresses old raw logs into a persistent knowledge base.

> **Extra detail**
>
> *Albert uses two memory streams. The raw log preserves recent tool calls, observations, and agent reasoning from the current phase of work. The knowledge base is a compressed technical summary produced when the raw log or token usage starts to approach the model's context limit. This lets a run continue for many cycles without sending the entire experimental history back to the model every time.*
>
> *The agent roles are separated deliberately. The Theorist proposes competing hypotheses, the Director chooses what is worth testing next, and the Technician executes the chosen protocol without being asked to invent a new plan. This separation is meant to make the workflow more auditable: when Albert fails, we can often distinguish whether the problem was bad hypothesis generation, poor experiment selection, weak execution, or loss of information during summarisation.*
>
> *The architecture is not meant to be a claim that real scientific teams should be decomposed exactly this way. It is a controlled harness: every model is given the same organisational structure, the same tool interface, and the same timeout, so differences in performance are more attributable to model behaviour than to bespoke scaffolding.*

Explorer is run once then Theorist, Director, Technician, and if needed Chronicler are repeated until the Director decides to submit a theory. Each world exposes between 3 and 6 instruments as tools. Some of the tools just perform actions in the virtual world, some perform measurements, and some perform actions and measurements. Albert is allowed to run for up to 24 hours, and the Director is then asked to submit the best current working theory Albert has generated.

We currently only give Albert access to mocked instruments that give an experimental output relevant to the systems under test in a virtual world, to initially avoid complex interfacing with real lab instruments.

It is important to be clear about what this benchmark is and is not. These are virtual worlds, not real wet labs, and the results test scientific search behaviour under controlled hidden rules rather than end-to-end real-world discovery.

Albert - Hypothesis Exploration & Selection Benchmark	Alloy Benchmark
Tests scientific reasoning and rule discovery. Albert is placed into 5 different virtual worlds (Chemistry, Ecology, Genetics, Physics, and Causal Inference). The laws of physics and science in these worlds do not match Earth. The goal is to see if the AI can discover the hidden rules that govern each world by designing and running experiments.	Tests high-dimensional material optimisation under a budget. Instead of discovering laws, the AI is given a practical engineering challenge: mix 40 different base compounds to create a material that maximises a specific quality metric (like hardness or conductivity).

Across these experiments, a run means one complete Albert attempt: the model is placed in a virtual world, given the same tool interface and task instructions, and allowed to explore until it submits an answer or reaches the runtime limit. For the Albert Benchmark, we run each model three times in each world and report the best run per world before aggregating across worlds. For the Alloy Benchmark, we run 50 attempts per model/configuration so we can see not only average performance, but also variance and failure modes. The main variables are the model and the Albert configuration; the hidden simulator, tool interface, scoring harness and timeout policy are held fixed within each benchmark.

One practical lesson arrived quickly: capable agents will use any information channel the environment makes available. In early runs, when old logs or shared working directories were visible, agents sometimes tried to inspect previous attempts rather than rediscover the rules from fresh experiments. From the model's perspective this is not irrational; old logs are useful evidence. But for a benchmark it contaminates the result, because independent runs are no longer independent. We therefore sandbox runs and restrict memory so that the benchmark tests scientific search behaviour, not accidental access to previous answers.

Extra detail

We saw a useful example of what might be called emergent shortcut-seeking. If the environment exposed old logs, some agents attempted to inspect them and use previous experimental traces. This is not surprising: a capable agent asked to solve a problem will look for relevant evidence, and previous logs are relevant evidence.
For benchmark design, however, this is a serious failure mode. It changes the task from independent discovery to information retrieval across runs. That is especially problematic when comparing models or configurations, because one run may benefit from discoveries made by another. It also blurs the

distinction between memory as a controlled experimental condition and memory as accidental filesystem leakage.
The lesson is that AI Scientist benchmarks need the same kind of hygiene as wet-lab protocols: clean separation between trials, explicit rules about what information is available, and audit trails showing which observations came from which experiment. We do not interpret log-seeking as moral cheating; we interpret it as evidence that the harness must be designed assuming agents will search for shortcuts.

Can Albert discover the laws?

The virtual world experiments mimicked five worlds that are completely observable with several phenomena taking place simultaneously. We chose disciplines of interest to the opportunity spaces we currently fund: Chemistry, Ecology, Genetics, Physics and one based on Causal relationships.

We don't want the details of the virtual worlds to enter the training sets for the frontier models, so we purposely are not sharing specific details on their structures, other than to say that each world had around 5 rules that governed the behaviour, and the behaviours would be completely atypical for the same experiments on Earth. Currently we use just two families of models (Gemini and GPT), but we plan to expand to other frontier models soon.

After each run, we had tools that analysed the theories submitted (using both Gemini and GPT models) and they voted on the number of rules discovered. We currently do not penalise for incorrect rules, but also do not count partial rules. The metric is simple - we run the experiment for each model three times, and then select the highest scoring run from the three and count the number of rules found. We then sum across all five experiments and normalise to 1 to create a score between 0 and 1.0 to show the broad success. So far, the score for each run has matched independent of the scoring model, and humans have checked many of the results.

Extra detail

The hidden worlds are designed so ordinary scientific intuition can be misleading. A virtual physics, chemistry, ecology, genetics, or causal world may be internally consistent, but its rules need not match Earth science. This matters because we are not trying to test whether the model remembers facts from training. We are trying to test whether it can design interventions, notice anomalies, reject attractive but false hypotheses, and infer a compact rule set from evidence.
We score exact rule discovery. Partial rules do not currently count, and we do not subtract points for wrong extra claims. That makes the headline metric easy to interpret, but it is not a full measure of

scientific quality. A model that discovers three rules and invents two plausible but false ones may receive the same score as a model that discovers three rules and carefully refuses to overclaim.

The submitted theories are evaluated by separate model-based graders and then spot-checked by humans. This is not the same as formal proof, but it gives a scalable way to compare many runs while catching obvious grading failures. In future versions, we would like to make the evaluation more structured, including penalties for false positives, uncertainty calibration, and evidence quality.

On noise: The causal world is stochastic and includes exogenous noise terms, while several other virtual worlds are largely deterministic once the agent chooses an action. The Alloy Benchmark is different: the alloy experiment readouts deliberately add measurement noise, but final submission is scored against the noiseless underlying simulator. That creates a separate challenge of deciding whether a promising measurement is a real signal or just a lucky readout.

The results are summarised in Figure 1. The chart shows model release date and score, from Gemini 2.5 Pro through to GPT-5.5 and Gemini 3.5 Flash. The pattern suggests a significant increase in capability over the last 6 months.

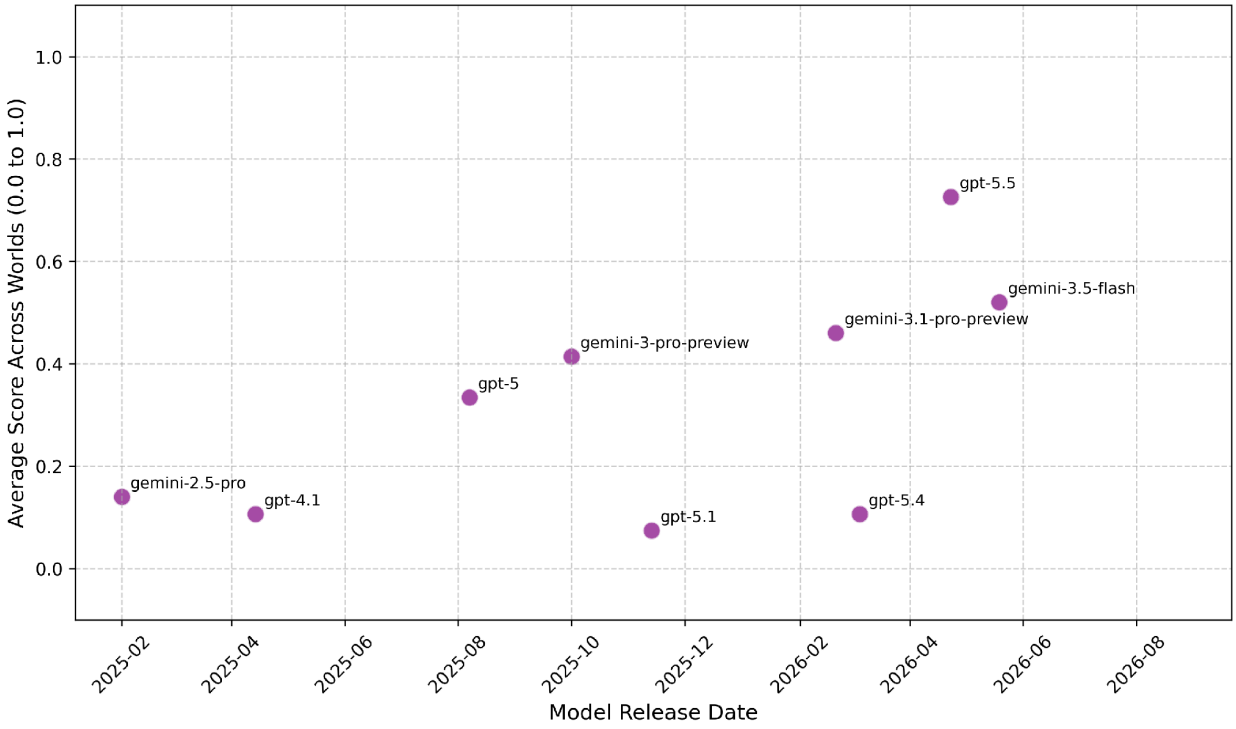

Figure 1 : Albert Benchmark score across five virtual worlds, plotted by model release date. Each point uses the best run per world for that model, summed across worlds and normalised to a 0.0 to 1.0 scale.

What is also interesting is that there is a significant difference in the run time for Albert. If we look at the score achieved divided by the total execution time we get a metric of efficiency, the score per minute of execution time, and this is shown in Figure 2.

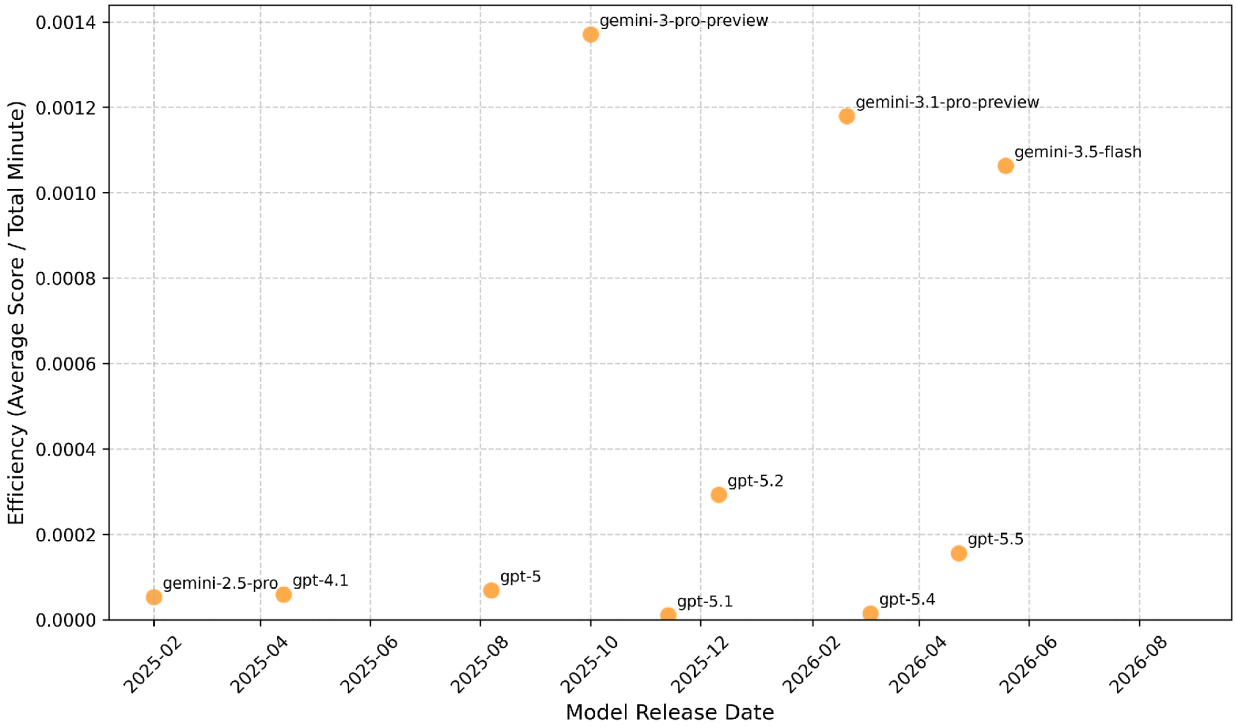

Figure 2 : Runtime efficiency on the Albert Benchmark, calculated as normalised score divided by total execution time in minutes. Higher values indicate more benchmark scores per minute of execution.

The tools do not track cost at the moment, but we can think of this efficiency metric as a broad proxy of cost.

> **Extra detail**
>
> *The efficiency metric is score divided by execution time. It is not the same as financial cost: different models have different pricing, latency, batching behaviour, and provider-side throughput. However, runtime is still useful because autonomous discovery systems can burn resources in ways that are not obvious from the final score alone.*
>
> *A high-scoring model that takes many hours to converge may be less useful in practice than a slightly lower-scoring model that reaches good hypotheses quickly and reliably. Conversely, a fast model that submits shallow theories may look efficient on easy tasks but fail when deeper experimentation is required. The score/time chart is therefore best read as a first-pass proxy for discovery productivity, not a complete cost model.*

A few observations stand out. Gemini 3.5 Flash discovered all causal relationships in the causal world, and GPT-5.5 discovered all but one. For the chemistry challenge, GPT-5.5 discovered all five rules, while Gemini 3.5 Flash discovered only 2 rules. In the ecology, physics and genetics virtual worlds, some rules were not found by any model, and neither GPT-5.5 nor Gemini 3.5 Flash found

more than 3 of the 5 rules. That unevenness is useful: the benchmark is not just measuring a single generic reasoning capability, but different mixtures of exploration, abstraction, experimental control, and persistence.

Design choices for Albert?

The other challenge was how to build Albert. In particular, how should Albert interact with the instruments? Talking to many people designing these systems, it is clear that there are many different approaches: calling optimised software layers implementing optimizers, such as Bayesian optimizers; using other support software services; asking AI Scientists to generate code that runs against instruments; or taking the simplest route and exposing instruments as direct tools to the LLM.

So, which approach worked best for us - and does it change with the model?

To explore this, we selected a high-dimensional "material optimisation" problem and set it up in the virtual world. There are 40 base compounds, and they interact in unexpected ways, and the goal is to maximise a metric for the material created using them. To make this a real challenge, we associated a cost with each experiment (which would mirror the real world). So, the best material would have a maximum value 100, and each experiment has a cost (uniform independent of what the experiment does).

> **Extra detail**
>
> *In the Alloy Benchmark the hidden material is represented by 40 continuous parameters, P1 to P40, each between 0 and 1. Each experiment returns noisy estimates of four properties: conductivity, hardness, thermal stability, and corrosion resistance. The agent's job is to find a parameter vector that scores well against the hidden objective while using as few experiments as possible.*
>
> *A key detail is that the experiment readouts are noisy, but final submission is scored against the noiseless underlying simulator. This creates a familiar scientific problem: a candidate can look excellent because of a lucky noisy measurement. Robust agents should therefore validate promising candidates, reason about measurement noise, and avoid overcommitting to a single apparent win.*
>
> *The final score combines material quality with an experiment-cost penalty after the first 15 experiments. This means the benchmark rewards not just finding a good material, but finding it efficiently. In practice, this makes the task closer to experimental optimisation under budget constraints than to pure black-box maximisation.*

The configurations we explored were all variations on how Albert interacted with the virtual worlds:

1. Fixed: simply expose the instruments directly to the LLM. Nothing else, the LLM designs the experiments as a sequence of tool calls.
2. Code: the LLM is provided with a sandbox and can write tools to help it run the experiments. For example, it can (and did) write optimizers. We were generous and said that to write and run one program only cost the same as running an experiment.
3. Mem+Fixed: Albert was allowed to record high-level guidance and share it across runs. It was banned, and technically prevented, from sharing specific parameters or exact answers; it could only share information about good strategies.
4. Mem+Code: Albert could both code, and access the long-term memory for good strategies.

> **Extra detail**
>
> *The code-enabled agents can write Python in a sandbox. The goal is to let the model build local analysis tools, surrogate models, or optimizers without giving it unrestricted access to files, secrets, or the hidden simulator internals. This is closer to how many people imagine practical AI Scientists working: not just calling instruments one at a time, but writing small pieces of analysis code around those instruments.*
>
> *The memory condition is different. It is intended to transfer strategy, not answers. Memory runs are sequential within a model/configuration group because each run can update the memory used by later runs. A sanitiser prevents exact parameter values from being carried across runs, so the test is whether general lessons such as "validate noisy hits" or "search more systematically" improve future attempts.*
>
> *This setup lets us ask a sharper question than "does more scaffolding help?" Code, memory, and code-plus-memory are different kinds of scaffolding. They may help different models in different ways, and they may also introduce new failure modes: debugging loops, stale strategic advice, overconfidence from previous runs, or accidental narrowing of the search space.*

For the Alloy Benchmark, the reported score combines final material quality with an experiment-cost penalty, so a high score means the agent found a good material without spending too many experimental attempts.

Figure 3 is a box plot showing performance across 50 Alloy Benchmark runs for two OpenAI models and two Gemini models, using the different Albert configurations. It also shows the result for Static, which was our attempt, working with a coding tool, to generate a solution where we understood the problem and scope before writing the code. This is equivalent to us creating an optimizer for this problem. Figure 3 suggests that there is surprisingly little automatic benefit as we add more complex mechanisms to Albert.

Extra detail

The "Static" baseline is not an AI Scientist loop. It is a hand-built optimizer using the same hidden simulator interface, included as a sanity check. If an agentic system cannot approach a straightforward optimizer on this kind of task, that tells us something about the overhead of autonomy, natural-language reasoning, and tool-mediated experimentation.

This comparison is not meant to imply that static optimizers are always the right answer. In real scientific workflows, the search space may be poorly specified, instruments may change, constraints may be negotiated, and the objective itself may evolve. Those are places where adaptive reasoning systems could matter. But for a controlled optimisation task, a static optimizer gives a useful reference point.

The striking result here is that adding machinery to Albert does not automatically improve performance. Code can help if the model writes a useful optimizer, but it can hurt if the model spends its budget debugging, overfitting noisy observations, or building infrastructure that is more complicated than the task requires.

Extra detail

The code-enabled runs did not all use "code" in the same way. GPT-style runs more often treated code as a disciplined experimental assistant: run the 15 free experiments, preserve a ledger of results, identify a promising incumbent, then write small local scripts to perturb, recombine, or validate candidates near that incumbent. In one representative GPT-5.4 memory+code run, the agent used structured anchors and a small free survey, found a promising basin, ran a four-candidate exploit batch, then after a larger recovery script timed out rewrote it as a two-candidate micro-batch. That smaller recovery step improved the observed quality substantially and the run submitted after only 21 experiments.

The Gemini-style code runs often looked more like ambitious offline optimization notebooks. They commonly fit surrogate models for quality or for the four measured properties, then used global 40-dimensional optimizers such as differential evolution, forest-based Bayesian optimisation, or very large candidate-pool searches. These ideas are technically plausible, and sometimes they found very high-quality candidates, but they were often poorly matched to the benchmark budget. They spent many more experiments on average and hit the sandbox timeout or resource limits much more often.

This is one of the clearest lessons from the code condition: giving a model a Python sandbox does not just add capability, it also exposes the model's search policy. Some models use code to make the experimental loop smaller and more auditable; others use code to scale up the search until runtime, memory, or experiment cost becomes the main failure mode.

Extra detail

The box plots hide an important behavioural split. In code mode, Gemini 3.1 achieved the strongest average material quality in this run set, but it did so by spending far more experiments and wall-clock

time: roughly 159 experiments on average in the code condition, compared with about 35 for GPT-5.4. Gemini 3.5 Flash and Gemini 3.1 also had many long or failed code-mode runs, consistent with logs showing large surrogate optimizers, differential evolution, and huge candidate pools. GPT-5.4 and GPT-5.5 were generally more conservative. They did not reach the same average material quality as Gemini 3.1 in code mode, but they tended to use fewer experiments and smaller scripts. The memory+code condition pushed this even further: for GPT-5.4, average experiments dropped to about 22.5, and the best runs show a pattern of incumbent preservation, small exploit batches, and explicit stop decisions.

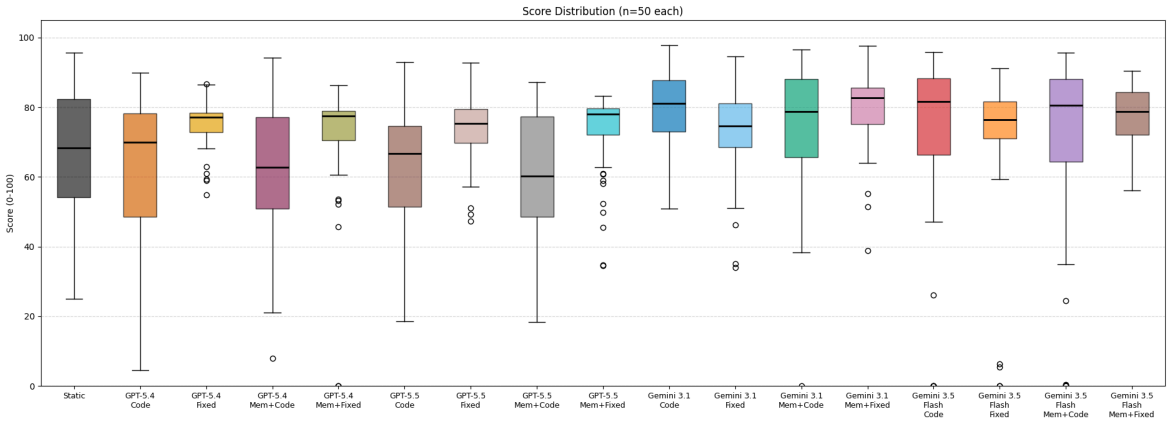

Figure 3 : Alloy Benchmark score distributions across 50 runs for each model and configuration. Higher scores mean the agent found a better material while paying a smaller experiment-cost penalty.

Extra detail

So the choice is not simply "which model is smarter?" The observed difference is closer to "which model chooses a search policy that fits this environment?" A heavy global optimizer can win when it finishes and finds a strong candidate; a smaller local optimizer can be less spectacular but more budget-aligned and operationally reliable.

Figure 4 is the execution time (log y-axis). This matters because the practical cost of an AI Scientist is not just whether it eventually finds a strong answer, but how long it keeps the loop running while it searches. Some configurations produce broadly similar scores but take very different amounts of time to get there, especially when code access encourages the model to write heavier optimization scripts. The log scale is important: a modest vertical difference can correspond to a large multiplier in runtime. The wider spreads also show that extra capability can increase variance. In some runs, code gives Albert a useful local optimizer or a compact analysis script; in others, it creates a long debugging or optimisation loop that dominates the run. This is why Figure 4 should be read

alongside Figure 3: score and runtime together tell us whether a configuration is merely powerful, or whether it is operationally reliable.

Extra detail

Timeouts and exit-code failures are informative, not just noise in the results. In the code-enabled logs, many failures were not ordinary Python mistakes; they were scripts that were conceptually reasonable but too large for the sandbox. Typical examples include training several surrogate models, generating hundreds of thousands of 40-dimensional candidates, predicting all properties for the whole pool, and using unrestricted parallel tree ensembles.

This matters because it mirrors a real deployment issue. A scientific agent with code access needs computational judgement as well as scientific judgement. It has to decide not only what would be a good optimizer in principle, but what optimizer is appropriate under the available runtime, memory, and experiment-cost constraints. In these runs, that judgement varied sharply across models.

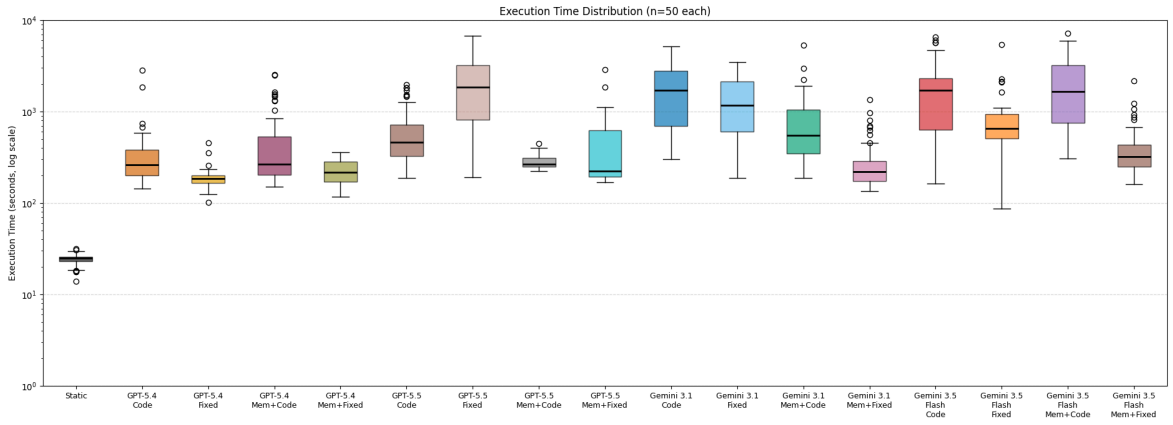

Figure 4 : Execution-time distributions for the Alloy Benchmark runs, shown on a log scale. The spread highlights how code and memory can change not only final score, but also runtime reliability.

A box plot summarises a distribution of results. The line inside the box is the median, meaning half the runs are above it and half are below it. The box itself spans the middle 50% of values, from the 25th percentile to the 75th percentile, so a taller box means more variation between typical runs. The whiskers extend to the broader range of non-outlier values, and any individual points beyond them are outliers. In this context, box plots are useful because they show not only which configuration performs best on average, but also how reliable or variable that performance is across repeated runs.

Discussion

These results make us more convinced that frontier models are worth testing seriously on research problems, but they also make us more cautious about how these systems should be evaluated. The story is not simply that newer models score higher. The more important lesson is that scientific agents are systems: model capability, tool interface, memory design, sandboxing, cost accounting, and failure handling all interact.

Three lessons stand out for us. First, hidden virtual worlds are useful precisely because they can separate recall from experimental search: the agent has to intervene, observe and revise. Second, more scaffolding is not automatically better. Code and memory can help, but they can also introduce new failure modes such as overbuilt optimizers, stale strategy, accidental leakage, or long debugging loops. Third, benchmark hygiene matters. If an agent can inspect old logs, shared directories or previous answers, it may do so. That is smart behaviour in an underspecified environment, but it is contamination for a benchmark.

These experiments are obviously imperfect. We had the luxury of an enterprise account, mocked instruments rather than real lab systems, and benchmarks that are still small compared with the breadth of science. But we want to hear from the community on how to make them better and what other experiments to try. We want to discover the best usage practices for these tools across a wide variety of scientific domains.

We're keen to explore these questions – let us know your thoughts!

Without training custom models from scratch, what kinds of simple environments can help us benchmark new generations of models and learn their success and failure modes across more domains of science? We are particularly concerned about data leakage. If models help generate the environments, have we accidentally made games that are rigged in the models' favour? What does the optimal test look like? A longstanding criticism of these models is that they cannot invent new categories, descriptions, or hypotheses that go "against the canon". We are not sure whether that criticism stands up to scrutiny, and we are open to ideas for experiments that could settle it.

Are discrete tool calls the best way to interface LLMs with scientific instruments? Several AI in science workflows we have seen call on discrete tools to carry out instrument-facing processes. Our early results suggest that this can sometimes be detrimental to model performance and can increase output variance. We are keen to learn whether others have observed the same behaviour, or whether this is an artefact of our implementation.