WHAT SHOULD WE WORK ON NEXT? NODE-LEVEL COMPUTE ALLOCATION ACROSS OPEN SCIENTIFIC QUESTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Rapid advances in AI capabilities have enabled AI systems to contribute to research on open scientific questions. There are many open scientific questions that AI could help investigate, but the computational cost of frontier models is high. These costs motivate careful allocation of compute across open scientific questions. Two key considerations for allocation are the scientific importance of each question and its difficulty relative to the AI system's capabilities. However, research-level open questions often require prolonged investigation, leaving limited outcome-based feedback for the difficulty estimates needed to guide compute allocation. This is particularly problematic for question-level allocation, as substantial compute may be invested before sufficient feedback is available to reassess the investment. Therefore, in this paper, we propose node-level compute allocation over evolving AND/OR directed acyclic graphs (DAGs), which record intermediate research progress and dependencies among research tasks. Finer-grained feedback from these tasks guides allocation across nodes from all questions, even before the underlying questions are resolved. We illustrate this allocation process through a case study involving five research-level open questions in quantum error correction (QEC). In controlled experiments, our scheduling algorithm performed best overall and sharply reduced lock-in waste relative to model self-scheduling.

1 INTRODUCTION

As AI capabilities continue to advance, AI systems are increasingly able to contribute to research on open scientific questions (Oum, 2026; Acuaviva & Acuaviva, 2026; Zhao et al., 2026a; Chung et al., 2026). Investigating a single research-level question can require substantial compute (Chung et al., 2026; Zhao et al., 2026a), while a scientific field may contain many worthwhile questions to pursue. These questions vary in scientific importance and difficulty, with difficulty assessed relative to the AI system's capabilities. We can roughly divide these questions into four categories: important and difficult, important and relatively easy, less important and difficult, and less important and relatively easy. Under a limited compute budget, allocating more resources to one question reduces what can be spent on others. In shared-budget evaluations on mathematics and code-reasoning tasks, Fan et al. (2026) find that the tested models tend to follow presentation order and concentrate compute on earlier questions, with limited sensitivity to assigned question values. A natural objective is therefore to design a compute allocation algorithm that prioritizes scientifically important questions that are relatively easy for the system, balancing the scientific value of potential progress against the effort required to achieve it.

Feedback from research attempts can provide heuristic guidance for assessing difficulty and refining these allocation decisions. Prior work has studied compute allocation across competition-level mathematics problems under shared budgets, with evaluations on MATH-500, AIME, and Omni-MATH (Zuo & Zhu, 2025; Fan et al., 2026). Some approaches use predicted per-question compute requirements to guide allocation decisions (Zhao et al., 2026b; Wan et al., 2026). For research-level open questions, however, estimating the per-question compute required is more challenging because reference solutions are unavailable and the difficulty of each question relative to the AI system's capabilities remains uncertain. Even when an initial estimate is available, useful question-level

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

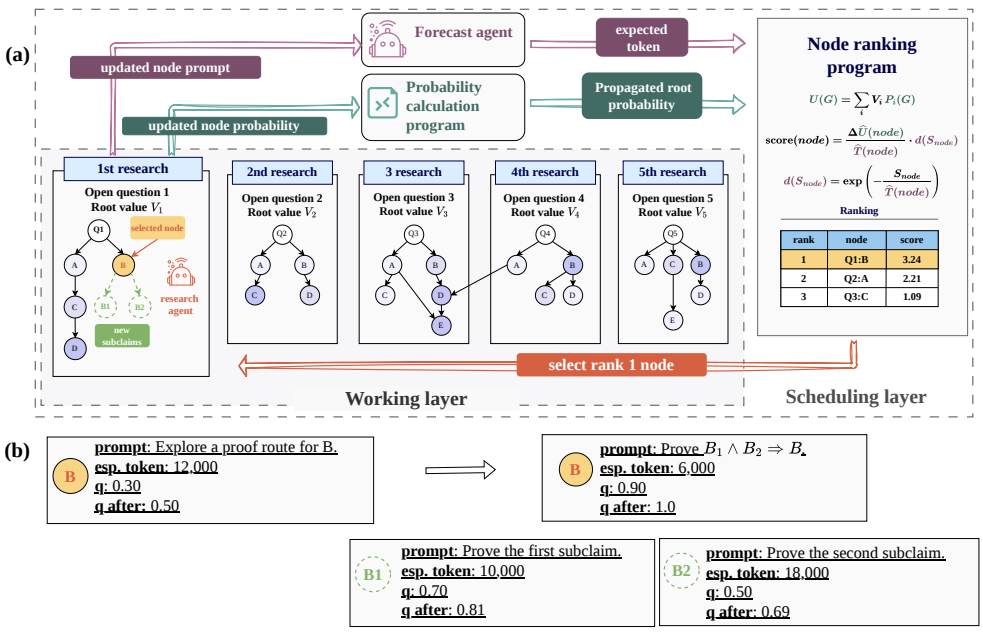

Figure 1: Node-level compute allocation across open scientific questions. **(a)** Research Agents execute selected tasks and update research DAGs in the working layer. The scheduling layer propagates node probabilities to the roots, combines predicted root-probability gains with expert-assigned scientific values, and ranks tasks. **(b)** An example of the update cycle after exploring a proof route for node B. Before execution, the selected task has a fixed prompt, a current local probability q, an expected local probability after successful execution q_{after}, and a forecast token cost. After execution, these records are refreshed for B, and corresponding records are created for the newly introduced subclaims B_1 and B_2.

outcomes may emerge only after prolonged investigation, leaving sparse feedback for updating it. Intermediate research tasks can nevertheless provide useful evidence. For example, proving a prerequisite lemma may justify allocating more compute to follow-up tasks that build on it. Conversely, a counterexample may rule out a candidate proof route, prompting compute to be redirected to tasks on other routes or questions. These challenges motivate node-level allocation, which narrows cost estimation to concrete next-step research tasks and uses feedback from those tasks to reassess further investment.

We use the AND/OR DAG as our research record to maintain the current state of each ongoing research project (Li et al., 2026b). Each node represents a specific claim and records its supporting evidence and review status. Claims are connected through proof routes: the AND structure captures premises that are jointly required within a route, while the OR structure represents alternative routes to the same claim. Agents update the record as they investigate claims, obtain new evidence, and revise assessments, while retaining failed claims and routes. This research record therefore preserves detailed information about intermediate nodes, providing a basis for our node-level compute allocation algorithm.

Our architecture consists of a working layer and a scheduling layer, as illustrated in Figure 1(a). In the working layer, Research Agents execute research tasks and maintain the corresponding DAGs. The scheduling layer ranks candidate tasks across all questions by considering their scientific importance and difficulty. Our unit of allocation is a concrete research task specified by a fixed prompt for a candidate node. Scientific importance is represented by the value assigned to each root question by human experts. For each candidate task, the Research Agent assesses the current local probability q and q_{after}, the local probability expected if the task produces its stated result. The probability calculation program first propagates the current local probabilities through the AND/OR DAG to obtain the root probabilities $P_i(G)$. It then replaces the corresponding q with q_{after} in a hypothetical post-task state and repeats the propagation to obtain $\widehat{P}_i^{(a)}$. The difference between the resulting utilities

gives the task's predicted utility gain. Task difficulty is approximated by its forecast token cost, estimated separately by a Forecast Agent. The ratio of predicted utility gain to forecast token cost gives the task's base allocation score. After a selected task is executed, the Research Agent reassesses q from the resulting evidence and prepares the next-step prompt and its corresponding q_{after}. The Forecast Agent then refreshes the forecast token cost, and root probabilities are recomputed for the next allocation decision (Figure 1(b)).

We study our allocation algorithm using five research-level open questions in quantum error correction (QEC): characterizing when effective distance equals circuit distance (Criterion); determining when circuit distance saturates under repeated syndrome-extraction rounds (Saturation); classifying the computational complexity of distance-preserving syndrome-extraction scheduling when the code distance is fixed (Fixed-d) and when it is part of the input (Input-d); and determining the optimal circuit distance of the Gross code (Gross). We also conduct controlled experiments on four mathematical problems with externally verifiable answers, using DeepSeek-V4-Pro and Claude Sonnet 5 as research models. The node-level scheduler performed best overall with DeepSeek-V4-Pro, while with the weaker Claude Sonnet 5 the same rule performed best at the question level; at both granularities, our scheduler sharply reduced lock-in waste relative to model self-scheduling.

2 Related Work

Compute allocation within tasks. There are several studies on how to distribute compute among subtasks, search branches, and workflow stages within individual tasks (Lin et al., 2025; Hamri & Talgam-Cohen, 2026; Li et al., 2026a). For example, GraphAHA allocates model calls among code-generation and repair actions on a search DAG (Li et al., 2026a). These approaches perform fine-grained allocation within a single task. Our framework performs node-level allocation across multiple open scientific questions, and could reduce to the same within-task setting when only one open question is considered.

Compute allocation across questions. A complementary line of work distributes a shared budget across multiple questions. Strategic Scaling formulates this problem as adaptive sampling across queries, using bandit algorithms with verifier scores or answer-agreement signals in mathematical reasoning and code generation (Zuo & Zhu, 2025). ROI-Reasoning considers sequential solve-or-skip decisions under a global token cap (Zhao et al., 2026b), while CLEAR assigns question-specific budgets through a global optimization based on predicted reasoning requirements (Wan et al., 2026). However, applying these methods directly to research-level open questions is difficult, as informative question-level feedback may remain sparse throughout prolonged investigations. Our framework instead allocates compute to concrete intermediate research tasks and uses their outcomes to reassess further investment before the underlying questions are resolved.

3 Algorithm

In this section, we introduce our node-level compute allocation algorithm, organized into a working layer and a scheduling layer as illustrated in Figure 1(a). We first describe the Research Agent and its maintenance of the research state in the working layer (Section 3.1), followed by the cost forecasting, probability calculation, and task ranking performed in the scheduling layer (Section 3.2).

3.1 Working Layer

The Research Agent executes the selected proof or exploration task and maintains the research DAG according to the research-record protocol (Li et al., 2026b). Each candidate task is associated with four allocation-related items. The fixed next-step prompt specifies the concrete task associated with a node. The current local probability q is the probability of establishing the corresponding node claim or route implication from the current research state. The expected local probability q_{after} estimates this probability if the task produces its stated expected result. The forecast token cost $\widehat{T}(a)$ estimates the cost of executing the prompt and maintaining the resulting DAG records; unlike the other three items, it is produced in the scheduling layer rather than by the Research Agent. The

prompt is stored with its node in the DAG, whereas q, q_{after}, and $\widehat{T}$ are kept in separate scheduling tables; Appendix A gives an example.

When a task is selected, the Research Agent executes the prompt associated with its target node. It then records the actual result, updates the DAG, and reassesses the current q from the resulting evidence. Next, it writes a new prompt for every node created or changed by the task. For each new prompt, the agent specifies its expected result and estimates q_{after}, the local probability expected if that result is obtained. The corresponding token-cost record remains missing or stale until the Forecast Agent supplies a new estimate. Figure 1(b) illustrates this cycle. After the exploration task for B, its current q is reassessed, its next-step prompt becomes proving $B_1 \wedge B_2 \Rightarrow B$, and a new q_{after} is estimated for that prompt. The newly introduced subclaims B_1 and B_2 are each assigned their own prompt, current q, q_{after}, and forecast token cost.

We extend the original research-record protocol (Li et al., 2026b) with one new rule. Under the original protocol, a registered proof route tends to lock subsequent work onto its own premises: the Research Agent writes the next-step prompts for the premises that remain open, and the scheduler ranks those prompts. New exploration nodes grow in the DAG only after all currently registered routes have been refuted. To keep the exploration of alternative routes open, our updated protocol requires each non-leaf node to carry a mandatory OR branch. Its prompt is an exploration task: it either registers a concrete new route with explicit premises or records the failed approaches and their obstruction points. The node receives its own q, q_{after}, and forecast token cost, and competes in the same candidate pool. When the registered routes have low q, the value of this OR branch rises, so the scheduling algorithm encourages the DAG to explore approaches that have not been tried before.

3.2 Scheduling Layer

3.2.1 Allocation algorithm Overview

We consider a pool of open scientific questions with roots $r_1, \ldots, r_m$ and a shared compute budget B. Their evolving research DAGs form the current state G. Our allocation unit is a concrete research task a associated with a node $v(a)$ and specified by a fixed next-step prompt. Candidate tasks from all questions compete in a common pool $\mathcal{A}(G)$ and broadly fall into two types. For a leaf node, the task seeks to prove its claim or explore possible proof routes. The proof and exploration prompt templates are provided in Appendices E.1 and E.2, respectively. For a non-leaf node, the task seeks to establish the route implication (a "gluing lemma") connecting the premises of a selected proof route to the node's claim, taking these premises as assumptions.

Human experts assign each root question a scientific value $V_i \geq 0$, representing the importance of resolving its stated claim. Let $P_i(G)$ denote the root probability computed from the current DAG and local probability assessments. We define the allocation utility as

$$U(G) = \sum_{i=1}^{m} V_i P_i(G). \tag{1}$$

For a task a, let $\widehat{\Delta U}(a; G)$ be its predicted change in this utility, $\widehat{T}(a) > 0$ its expected execution cost, and $S_a \geq 0$ the accumulated stagnation cost of its target node. The allocation score is

$$\text{score}(a; G) = \frac{\max\{0, \widehat{\Delta U}(a; G)\}}{\widehat{T}(a)} \exp\left(-\frac{S_a}{\widehat{T}(a)}\right). \tag{2}$$

The first factor compares predicted utility gain per unit of compute. The second discounts further investment after repeated expenditure without a qualifying change in the research state. Algorithm 1 in Appendix D summarizes the allocation procedure.

3.2.2 Forecast Agent

The Forecast Agent estimates the token cost of the next research task from its fixed prompt and relevant current research context. We restrict cost prediction to a single, concrete research step, avoiding the need to anticipate the entire research trajectory (Bai et al., 2026). We expect cost estimates for a single research step to be more accurate, as they draw on a clearer task specification and more detailed evidence about the current research state. We express $\widehat{T}(a)$ as the expected

number of output tokens required to execute the specified research task and perform the associated DAG maintenance and registration. This estimate is used to rank candidate tasks and does not impose a hard token limit on execution. A forecast is reused while the task prompt and relevant research context remain unchanged. A new estimate is obtained when the prompt is revised, a relevant premise changes, or new evidence changes the work required for the task.

3.2.3 PROBABILITY CALCULATION PROGRAM

The probability calculation program propagates the local estimates supplied by the Research Agent through the AND/OR DAG. At an AND node, all required premises and the corresponding route implication must hold. For example, if premises A and B have probabilities p_A and p_B, and their gluing lemma C has probability q_C, the combined probability is $p_A p_B q_C$, assuming these three components are independent. At an OR node, at least one proof route must hold: two independent routes with probabilities p_E and p_F give a combined probability of $1-(1-p_E)(1-p_F)$. Applying these rules successively through the DAG yields the root probabilities $P_i(G)$. Finally, for each candidate task a, the program evaluates a hypothetical update in which the corresponding current local probability q is replaced by q_{after}, and propagates the updated probabilities through the same DAG to obtain the expected post-task root probabilities $\widehat{P}_i^{(a)}$. The difference between the predicted and current root probabilities, weighted by the corresponding scientific values V_i, gives the task's predicted utility gain:

$$\widehat{\Delta U}(a;G) = \sum_i V_i \left[\widehat{P}_i^{(a)} - P_i(G)\right]. \tag{3}$$

The sum includes every root affected by the task. This gain is then combined with the forecast token cost and stagnation cost in the allocation score defined by Equation (2).

3.2.4 NODE RANKING PROGRAM

The ranking program evaluates Equation (2) across the full candidate pool and selects the highest-scoring eligible task. We next explain how the decay term is calculated.

Stagnation cost is associated with the target node, so $S_a = S_{v(a)}$. It accumulates only when a task remains incomplete and produces no qualifying state change; otherwise, it is reset to zero. This term discourages repeated investment in nodes on which the AI system struggles to make progress: as S_v grows, the decay factor lowers their allocation scores and reduces their priority in subsequent allocation decisions. Specifically, an incomplete task is considered stagnant only when it changes no relevant evidence level, adds no prerequisite to the root's active proof structure, and produces no evidence-supported increase in the target node's local probability q larger than a threshold τ (we use $\tau = 0.1$). A decrease in q, or an increase no larger than τ, therefore does not reset the stagnation cost. The increase is measured against the previous q rather than the predicted q_{after}: a task whose updated q falls short of q_{after} still resets the stagnation cost if it raises q by more than τ.

4 EXPERIMENTS

We use two complementary experimental settings. The first examines the scheduling trajectory over five research-level questions in QEC, where progress is represented by evolving research DAGs. The second uses four mathematical problems with externally verifiable answers to support controlled comparisons between allocation strategies.

4.1 SCHEDULING TRAJECTORIES ON RESEARCH-LEVEL QEC QUESTIONS

4.1.1 EXPERIMENTAL SETUP

The QEC study contains five root questions: (1) Criterion: a structural criterion for when effective distance equals circuit distance; (2) Saturation: saturation of circuit distance under repeated syndrome-extraction rounds; (3) Fixed-d: the complexity of preserving effective distance when the code distance is fixed; (4) Input-d: the corresponding complexity when the distance is part of the input; and (5) Gross: the optimal circuit distance of the Gross code. Human experts assign these roots values of 100, 80, 60, 60, and 40, respectively. These values represent the relative importance

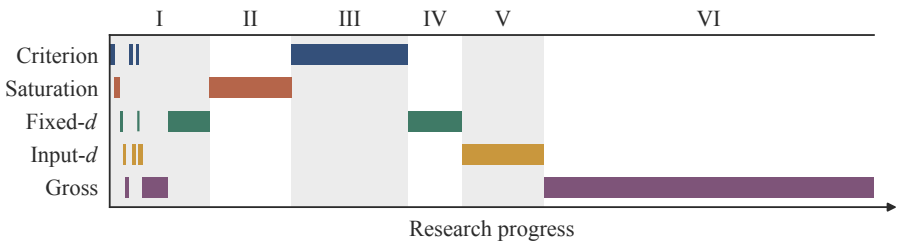

Figure 2: Scheduling trajectory over the five QEC questions.

of resolving the registered root claims and exclude estimated difficulty, predicted success, and prior outcomes.

4.1.2 Scheduling Trajectories and Analysis

Figure 2 shows the trajectory of the scheduling algorithm. Each row corresponds to one root question. Each block represents one or more consecutive tasks dispatched to that question, and its width reflects the output tokens those tasks consumed. We divide the trajectory into six phases: an initial exploration phase, followed by five phases in which the scheduler concentrates on one question at a time.

Phase I: exploration. Initially each question has only an exploration task, with similar prospects and similar forecast costs, so the scores differ mainly through the root values and the first five tasks follow the value order: Criterion, Saturation, Fixed-d, Input-d, and Gross. After this round every root probability is still small and no question holds a clear lead: the top task rarely scores more than twice the best alternative, and each new result shifts the ranking. Allocation therefore fluctuates, with small tasks spread over Criterion, Input-d, Fixed-d, and Gross and frequent switches between questions. This contrasts with the later phases, in which one question leads by a wide margin.

Phases II–VI: concentration. The later phases follow one pattern. A result raises one question's root probability well above those of the others, its candidates lead the ranking by a wide margin, and the scheduler concentrates on that question until it closes. This happens in turn for Saturation, Criterion, Fixed-d, and Input-d. Gross, the last remaining question, stays open because whether any legal schedule reaches distance twelve was never resolved. Appendix B gives the per-phase details.

4.2 Controlled Experiments on Mathematical Problems

4.2.1 Experimental Setup

The controlled study uses four mathematical problems whose answers have recently been published: Question 1 asks for the minimum order of a 3-regular penny graph (answer 16; Karabegov & Khovanova, 2026); Question 2 asks for $\lim_{n\to\infty} g(n)/n$, where $g(n)$ is the minimum size of a largest Sidon subset of an n-element weak Sidon set (answer $1/2$; Ma & Tang, 2026); Question 3 asks for the maximum number $m_{k,k+1,k+a}$ of empty convex k-gons in a set of $k+a$ points with no empty convex $(k+1)$-gon, for $a \le k/2 - 1$ (answer 2^a; Andričík et al., 2026); and Question 4 asks for

Table 1: Budget-normalized score $\bar{R}$ (Eq. (5)).

Model	Values	Node-level scheduler	Question-level scheduler	Model self-scheduling (DAG)	Model self-scheduling (no DAG)
DeepSeek-V4-Pro	Equal $(1,1,1,1)$	**1.05**	0.82	0.75	0.46
	Ascending $(1,2,3,4)$	1.56	0.59	0.47	**1.80**
	Descending $(4,3,2,1)$	**0.68**	0.49	0.20	0.13
Claude Sonnet 5	Equal $(1,1,1,1)$	0.00	**0.17**	0.00	0.09
	Ascending $(1,2,3,4)$	0.43	**0.47**	0.42	0.41
	Descending $(4,3,2,1)$	0.06	**0.28**	0.00	0.00

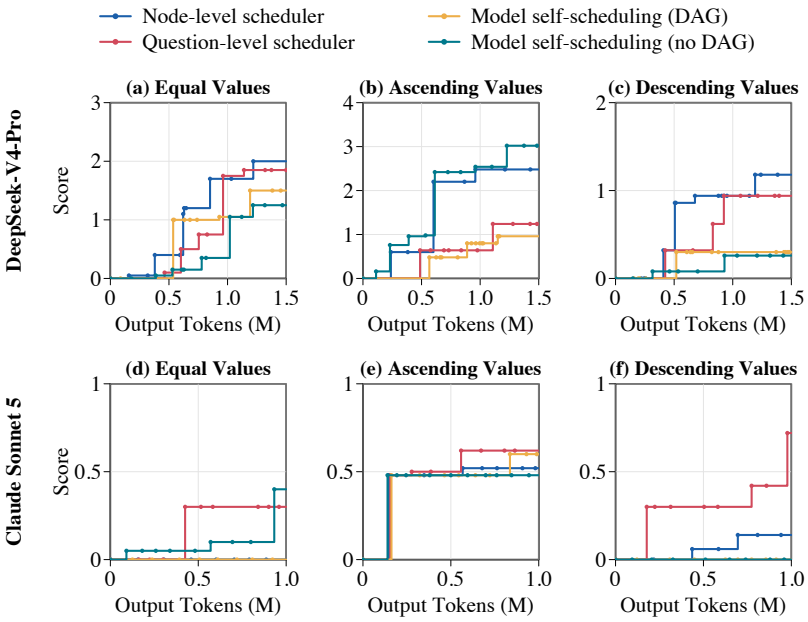

Figure 3: Score $R(t)$ against output tokens with DeepSeek-V4-Pro (top row) and Claude Sonnet 5 (bottom row) under (a, d) equal values $V = (1, 1, 1, 1)$, (b, e) ascending values $V = (1, 2, 3, 4)$, and (c, f) descending values $V = (4, 3, 2, 1)$.

the third smallest degree-growth constant of iterated line graphs (answer $11/2$; Sanghi et al., 2026). We use DeepSeek-V4-Pro and Claude Sonnet 5 as the experimental models. Each condition has an output-token budget of 1.5M with DeepSeek-V4-Pro and 1.0M with Claude Sonnet 5.

Scoring. Each problem is scored by a fixed list of milestones derived from the structure of its reference solution; the full rubric is given in Appendix C. A milestone is credited at the first settled task after which the condition's own artifacts rigorously establish it, and the per-question score $r_i(t) \in [0, 1]$ is the sum of the credited milestone points. The question scores are then weighted by the root values V_i:

$$R(t) = \frac{4}{\sum_i V_i} \sum_{i=1}^{4} V_i \, r_i(t) \in [0, 4]. \tag{4}$$

Because the timing of progress matters for allocation, we also report the budget-normalized area under the score curve,

$$\bar{R} = \frac{1}{B} \int_0^B R(t) \, dt, \tag{5}$$

where B is the output-token budget of the run; $\bar{R}$ equals the average score held over the budget.

We compare four allocation conditions. (1) *Node-level scheduler*: the deterministic scheduler ranks all eligible node-level tasks using the method in Section 3, while the Research Agent maintains the DAG and local probability records and the Forecast Agent supplies the forecast token costs. In our runs, we observed that the Forecast Agent's forecast token costs often deviated substantially from the realized costs, which may reflect the limited calibration of the underlying model for this kind of prediction. We therefore added a calibration mechanism in this experiment: the measured output tokens of previously settled research tasks are provided to the Forecast Agent as few-shot examples when it estimates the cost of a new task.(2) *Question-level scheduler*: each question is treated as a single candidate "super-node" with one next-step prompt, q, q_{after}, $\widehat{T}$, and S, and the same ranking formula is applied without an internal DAG or within-question probability propagation. (3) *Model self-scheduling with a DAG*: the model itself selects the next eligible research node from the current DAG and writes the instruction for that research attempt; it receives the same scientific DAG records but no algorithmic scores, and it does not maintain q, $\widehat{T}$, S, or persistent next-step prompts.

(4) *Model self-scheduling without a DAG*: the model selects a question and writes a research instruction, based only on the problem statements and the unstructured research notes accumulated from previous attempts.

4.2.2 RESULTS

Figure 3 shows the score trajectories for all three value settings, and Table 1 reports the budget-normalized score $\bar{R}$ of each condition. From these results, we highlight three findings.

First, our allocation rule is robust to how scientific value is aligned with difficulty, and its best granularity depends on the research model. Question 4 is the most tractable of the four problems, so the three value settings cover three regimes: equal values, the most tractable question receiving the highest weight (ascending), and the most tractable question receiving the lowest weight (descending). With DeepSeek-V4-Pro, as Table 1 shows, the node-level scheduler ranks first under equal values and descending values, and second under ascending values. Clearly, the node-level scheduler performs best overall in this set of experiments. As a comparison, we repeat the experiment with Claude Sonnet 5, a weaker research model on these problems: its scores are substantially lower than those of DeepSeek-V4-Pro (Figure 3). Here the node-level scheduler loses its advantage, but the same allocation rule applied at question granularity (the question-level scheduler) ranks first in all three settings. That may be because with weaker model, node-level tasks tend to stall on numerical search and enumeration instead of establishing the node's claim: none of its node-level tasks completes the assigned node, and its budget is spread over many partially explored nodes. Allocating across whole questions keeps successive attempts on the same line of work, which is more robust for a weaker model. Since research-level open questions are typically attacked with frontier models, node-level allocation is the preferred granularity in practice.

Second, maintaining a DAG aggravates lock-in, and our scheduling algorithm counteracts it. To quantify whether a condition keeps investing in a question without making progress, we define the *lock-in waste* as the fraction of research output tokens spent on tasks that are at least the third consecutive task on the same question without any new rubric milestone on that question, averaged over the three value settings. Table 2 arranges the four conditions by whether a DAG is maintained and whether allocation is performed by the model or by our scheduling algorithm. The first row of Table 2 allocates at the node level over a DAG, and the second row allocates at the question level without one. For both models and in both columns, the first row shows more lock-in than the second: with DeepSeek-V4-Pro, 39% versus 13% under model self-scheduling and 5% versus 0% under algorithmic scheduling; with Claude Sonnet 5, 51% versus 40% and 4% versus 2%. That may be because a DAG always exposes a concrete next premise on the active route, which invites the scheduler to keep extending that route even when it produces no progress. Our scheduling algorithm counteracts this effect: with DeepSeek-V4-Pro, it reduces the waste from 39% to 5% with a DAG and from 13% to 0% without one; with Claude Sonnet 5, from 51% to 4% and from 40% to 2%, because the stagnation cost and the re-estimation of q and $\widehat{T}$ after every task lower the priority of questions on which repeated work has stalled.

Table 2: Average lock-in waste over the three value settings.

	DeepSeek-V4-Pro		Claude Sonnet 5	
	Model	Algorithm	Model	Algorithm
With DAG	39%	5%	51%	4%
Without DAG	13%	0%	40%	2%

Third, when the most valuable question is difficult, only the scheduling algorithms avoid overinvesting in it. Under descending values, the most valuable question, Question 1, is one of the more difficult problems, and no condition obtains substantial credit on it. The two model self-scheduling conditions nevertheless concentrate their budgets on the most valuable questions (Figure 4(c)): without a DAG, the model spends ten of its eleven tasks and 88% of its research output tokens on Question 1, and with a DAG, it moves to Question 2 after four tasks on Question 1 and then spends 82% of its research output tokens there. Neither condition ever attempts Questions 3 or 4, and both curves remain nearly flat after their first gains (Figure 3(c)). With Claude Sonnet 5, the concentration is even stronger: both model self-scheduling conditions spend all of their tasks within the budget on Question 1 and obtain no credit (Figure 4(f)). Our scheduling algorithm avoids this failure through two mechanisms: the accumulation of the stagnation cost S and the upward

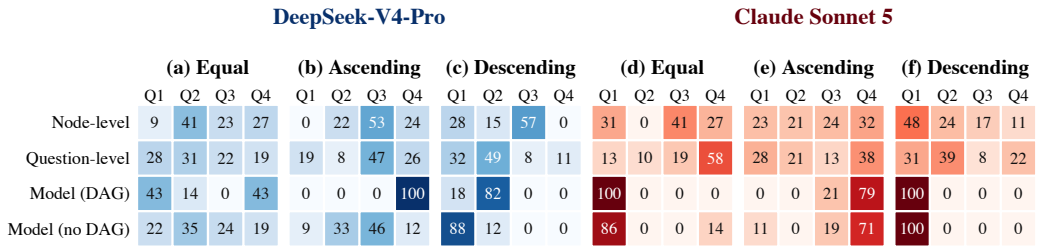

Figure 4: Research-token allocation by question with DeepSeek-V4-Pro (a–c) and Claude Sonnet 5 (d–f) under equal (a, d), ascending (b, e), and descending (c, f) values.

re-estimation of $\widehat{T}$. After an initial exploration of Question 1, the node-level scheduler dispatched two follow-up tasks on its subclaims; both produced no qualifying progress, so their stagnation costs grew and the scheduler moved to Question 3 with the next task. The question-level scheduler shows the same mechanisms at the question scale: after two attempts on Question 1 without progress, the accumulated stagnation cost S and the upward re-estimation of $\widehat{T}$ removed Question 1 from further consideration. The node-level scheduler therefore distributes its research across Questions 1–3, obtains credit on Questions 2 and 3, and reaches the highest final score in this setting. With Claude Sonnet 5, the question-level scheduler likewise leaves Question 1 after two attempts without progress, obtains credit on Questions 2 and 4, and ranks first in this setting.

5 Conclusion

We introduced a node-level compute allocation algorithm for distributing a shared research budget across multiple open scientific questions. Rather than committing a large budget to an entire question, the algorithm repeatedly selects a concrete next-step research task from a global candidate pool. It combines human-assigned root values, probabilities propagated through evolving AND/OR DAGs, predicted task effects, forecast token costs, and stagnation costs to rank these tasks. The research DAGs preserve intermediate progress and provide new evidence for subsequent allocation decisions before the root questions are resolved. We study this framework through scheduling trajectories over five research-level QEC questions and controlled comparisons on four mathematical problems. In these comparisons, the node-level scheduler performed best overall with DeepSeek-V4-Pro, the question-level variant performed best with Claude Sonnet 5, and both sharply reduced lock-in waste relative to model self-scheduling. Since research-level open questions are typically attacked with frontier models, node-level allocation is the preferred granularity in practice.

6 Discussion

The allocation decisions depend on model-generated estimates of the current and post-task local probabilities, q and q_{after}, as well as the forecast token cost $\widehat{T}$. Inaccurate estimates can change the ordering of otherwise comparable tasks. Future work should test whether few-shot examples improve either estimate, and determine whether post-training a dedicated token-cost predictor can provide more reliable cost forecasts.

The scheduler is also sensitive to how the same research state is decomposed into nodes and proof routes. Suppose two questions contain proof routes of comparable difficulty, but one route is represented as a single coarse node while the other is divided into several jointly required premises. The propagated probability of the latter contains more multiplicative factors and will generally be lower, even if the finer decomposition does not make the underlying research problem more difficult. This difference can reduce its predicted root-level gain and allocation score, causing the scheduler to favor the more coarsely represented question solely because of how the research records were constructed. Fair comparison across questions therefore requires their DAGs to be maintained at comparable levels of granularity, with consistent criteria for registering claims, prerequisites, and proof routes.

AI USE STATEMENT

The research questions, conceptual framework, scheduling method, experimental design, and mathematical claims advanced by this paper were developed by the authors. Generative AI systems assisted with executing parts of the research workflow, including mathematical derivations and proof development, scheduled research tasks, literature retrieval, code implementation, and manuscript drafting. The authors reviewed and verified all AI-assisted content and take responsibility for the final paper.

REFERENCES

Antonio Acuaviva and Pablo Acuaviva. Mathematical discovery in the wild: AI-guided proofs in Banach space theory. *arXiv preprint arXiv:2607.17388*, July 2026. URL https://arxiv.org/abs/2607.17388v1.

Martin Andričík, Alica Dományová, Adam Džavoronok, Aleksa Džuklevski, and Matouš Šafránek. Many holes but no large one: maximizing k-holes while forbidding $(k+1)$-holes. *arXiv preprint arXiv:2606.08762*, 2026.

Longju Bai, Zhemin Huang, Xingyao Wang, Jiao Sun, Rada Mihalcea, Erik Brynjolfsson, Alex Pentland, and Jiaxin Pei. How do AI agents spend your money? analyzing and predicting token consumption in agentic coding tasks. *arXiv preprint arXiv:2604.22750*, 2026. URL https://arxiv.org/abs/2604.22750v2.

Stephen Chung, Wenyu Du, and William J. Wesley. Autonomous mathematical discovery in an open-world multi-agent environment. *arXiv preprint arXiv:2608.23691*, August 2026. URL https://arxiv.org/abs/2608.23691v1.

Chenrui Fan, Yize Cheng, Ming Li, Yongyuan Liang, Tianyi Zhou, and Soheil Feizi. Thinking hard, not smart: Reasoning models fail to ration test-time compute across questions. *arXiv preprint arXiv:2608.07968*, August 2026. URL https://arxiv.org/abs/2608.07968v1.

May Hamri and Inbal Talgam-Cohen. ZEBRA: Zero-shot budgeted resource allocation for LLM orchestration. *arXiv preprint arXiv:2605.20485*, May 2026. URL https://arxiv.org/abs/2605.20485v1.

Alexander Karabegov and Tanya Khovanova. Minimal 3-regular penny graph. *arXiv preprint arXiv:2602.01287*, 2026.

Xitao Li, Haijun Wang, Gege Yuan, Qiyuan Wu, Jiali Wei, Ming Fan, and Xiaofei Xie. GraphAHA: Graph-based adaptive search with heterogeneous actions for test-time code generation. *arXiv preprint arXiv:2609.12757*, September 2026a. URL https://arxiv.org/abs/2609.12757v1.

Yuqing Li, Yu Gan, Zeguan Wu, Tianao Wu, and Junyu Liu. A claim-centered research record for AI-assisted theoretical research: Examples in quantum query complexity. Research manuscript, 2026b. URL https://github.com/YuqingLiii/Claim-Centered-Research-Graph/blob/main/paper/claim-centered-research-record.pdf.

Junhong Lin, Xinyue Zeng, Jie Zhu, Song Wang, Julian Shun, Jun Wu, and Dawei Zhou. Plan and budget: Effective and efficient test-time scaling on reasoning large language models. *arXiv preprint arXiv:2505.16122*, 2025. URL https://arxiv.org/abs/2505.16122v3.

Jie Ma and Quanyu Tang. Largest sidon subsets in weak sidon sets. *arXiv preprint arXiv:2602.23282*, 2026.

Sang-il Oum. A proof of the cycle double cover conjecture by OpenAI: An exposition. *arXiv preprint arXiv:2607.16356*, July 2026. URL https://arxiv.org/abs/2607.16356v1.

Aryan Sanghi, Anubhav Dhar, and Sudeshna Kolay. On euler paths and the maximum degree growth of iterated higher order line graphs. *arXiv preprint arXiv:2602.09585*, 2026.

Xu Wan, Speed Zhu, Jianwei Cai, Guang Chen, XiMing Huang, Wiggin Zhou, and Mingyang Sun. The shadow price of reasoning: Economic perspective on optimal budget allocation for LLMs. *arXiv preprint arXiv:2606.03092*, June 2026. URL https://arxiv.org/abs/2606.03092v2.

Mingming Zhao, Jiqian Dong, Kangping Xu, Zadid Hasan, Chengrui Fan, Shan Jiang, Shuai Mao, Ting Lingya, Linyi Zou, Tailin Zhou, Yun Hin Chan, Wenkai Zhang, Zhanhong Zhou, Guowei Huang, Hongliang Li, Wenjing Cun, Zhitang Chen, Mingxuan Yuan, and Yanhui Geng. ScienceFlow: A long-horizon agent for ML research, scientific discovery and beyond. *arXiv preprint arXiv:2608.14354*, August 2026a. URL https://arxiv.org/abs/2608.14354v1.

Muyang Zhao, Qi Qi, and Hao Sun. ROI-reasoning: Rational optimization for inference via precomputation meta-cognition. *arXiv preprint arXiv:2601.03822*, January 2026b. URL https://arxiv.org/abs/2601.03822v1.

Bowen Zuo and Yinglun Zhu. Strategic scaling of test-time compute: A bandit learning approach. *arXiv preprint arXiv:2506.12721*, 2025. URL https://arxiv.org/abs/2506.12721v2.

A SCHEDULING RECORDS

Figure 5 shows how the four allocation-related items of Section 3.1 are stored. The prompt is recorded in `graph.nodes[node_id].scheduling.next_prompt` and specifies the concrete next task associated with that node. The current q is stored in the separate `q_records` table; a node claim is located by its node identifier, whereas a route implication is located by its route identifier. In Figure 5, `node::B`, `node::B1`, and `node::B2` identify the current q records associated with nodes B, B_1, and B_2, respectively. The corresponding q_{after} is stored in `action_q_records`, indexed by the task's action identifier, and the forecast token cost is stored in `t_records`, indexed by the same action identifier. The corresponding tasks are identified by `a_B`, `a_B1`, and `a_B2` in both tables. For example, `a_B` locates both $q_{\text{after}} = 1.00$ and the forecast token cost $\widehat{T} = 6{,}000$ for the next-step prompt associated with B.

Graph data structure

```
graph.nodes["B"]
canonical
  claim: Claim B
  assessment: unresolved
  routes: proof-route-1
scheduling
  transition_id: "a_B"
  next_prompt:
    Prove B_1 ∧ B_2 ⇒ B.
  expected_result:
    A proof or documented obstruction.
```

Separate scheduling tables

q_records	
node::B	$q = 0.90$
node::B1	$q = 0.70$
node::B2	$q = 0.50$

action_q_records	
a_B	$q_{\text{after}} = 1.00$
a_B1	$q_{\text{after}} = 0.81$
a_B2	$q_{\text{after}} = 0.69$

t_records	
a_B	$\widehat{T} = 6{,}000$
a_B1	$\widehat{T} = 10{,}000$
a_B2	$\widehat{T} = 18{,}000$

Figure 5: Example of the graph data structure and scheduling tables for node B. The left panel shows the graph data structure, including the next-step prompt and expected result associated with B. The right panel shows example entries from `q_records`, `action_q_records`, and `t_records`.

B SCHEDULING DETAILS

Phase II: Saturation. Saturation receives only its initial exploration task during Phase I, but that task registers a route of a few obligations that the forecasters judge likely. Its root probability (0.029) becomes the largest among the open questions, its six best candidates occupy the top of the ranking, and the leader scores 3.5 times the best alternative. The scheduler concentrates on Saturation until the question closes.

Phase III: Criterion. With Saturation closed, Criterion combines the largest value (100) with a proof route that needs only a few more proof tasks. Its eight tasks lead the ranking, the first by a factor of 7.7; after an exploration task on a prerequisite, the scheduler stays on Criterion until it closes.

Phase IV: Fixed-d. Once Criterion closes, the three open questions have similar root probabilities (about 0.002), and the top candidates from Fixed-d and Input-d score almost equally. A Fixed-d proof task is dispatched, and its result establishes four of the five premises of the hardness route at once. Because the root probability multiplies over these premises, it jumps from 0.002 to 0.24, and the task on the one remaining premise now scores about 100 times any task from another question. The scheduler completes that premise and closes Fixed-d.

Phase V: Input-d. The same pattern repeats: one construction settles three premises of Input-d (root probability $0.003 \to 0.43$), and the task targeting its last lemma ranks first until the root closes.

Phase VI: Gross. Only Gross tasks remain. Whether any legal schedule reaches distance twelve was never resolved, and the question stays open.

Table 3: Milestone rubric. Points within a question sum to 1.

Question	Milestone	Points
1	Explicit 16-vertex 3-regular penny graph with verifiable coordinates	0.40
	Exclusion of $n = 4$ and $n = 6$	0.05
	Exclusion of $n = 8$	0.05
	Exclusion of $n = 10$	0.10
	Exclusion of $n = 12$	0.10
	Exclusion of $n = 14$	0.20
	Answer 16 with the construction and all exclusions	0.10
2	Sidon $\Leftrightarrow$ 3-AP-free inside a weak Sidon set	0.05
	Valid lower bound $c > 0$	0.10
	Lower bound $1/2$	0.30
	Verified nontrivial upper construction (ratio below 1)	0.20
	Upper construction with ratio $1/2 + o(1)$	0.25
	Answer $1/2$ with the lower bound $1/2$ and the ratio-$(1/2 + o(1))$ construction	0.10
3	Construction proving $m_{k,k+1,k+a} \geq 2^a$	0.40
	Upper bound for $a = 1$	0.10
	Valid reduction of the general upper bound to an explicitly stated lemma	0.10
	General upper bound $m_{k,k+1,k+a} \leq 2^a$	0.30
	Answer 2^a with the construction and the general upper bound	0.10
4	Verified catalogue of graphs attaining 3, 4, and $11/2$	0.30
	Basic lemmas (subgraph monotonicity and a stabilization criterion)	0.10
	Exclusion of maximum degree ≥ 4 below $11/2$	0.10
	Exclusion of graphs with cycles below $11/2$	0.10
	Complete classification of subcubic trees	0.30
	Answer $11/2$ with all of the above	0.10

C Milestone Rubric

Each problem is scored against milestones derived from the structure of its published reference solution (Table 3). A milestone is credited at the first settled task after which the condition's own artifacts rigorously establish it: an argument that checks line by line, or a certificate or computation that the grader re-verifies. A false claim or a proof with a genuine gap earns nothing, and a later repair is credited at the repairing task. Stating the reference answer without the corresponding arguments earns no credit. Independent grading agents applied the same rubric to all twelve runs; identical artifacts that appear in two conditions through task replay receive the same credit. No reference answer, rubric, or grading output is shown to any model.

D Allocation Procedure

Algorithm 1 summarizes the allocation procedure of Section 3.2.

E Research Task Prompt Templates

E.1 Proof Template

For a leaf node, `{prompt}` asks the agent to prove the node's claim. For a non-leaf node, it asks the agent to establish the route implication associated with the selected proof route, treating the premises as hypotheses without re-proving them; establishing the implication alone does not establish the parent claim while its premises remain unresolved. The direction-specific rule for proof tasks is:

> A prove-direction attempt that does not complete the selected node or exact target route must expose the complete proof decomposition rather than merely recording one generic failure route. Register every mathematically distinct route and every child node needed to represent the attempt: separate the components actually established, the components refuted or shown to fail, and each still-OPEN obligation; connect each component to every route that depends on it with correct AND/OR semantics; and give every OPEN child an actionable repair task. The number of routes and children is determined by the mathematics

Algorithm 1 Node-level compute allocation across open questions

Require: Research state G, root values $\{V_i\}$, budget B, declared accounting and stopping rules
Initialize node stagnation counters and a cumulative cost ledger
while the budget and stopping rules permit another allocation **do**
  Prepare candidate tasks $\mathcal{A}(G)$ with fixed prompts
  if $\mathcal{A}(G) = \varnothing$ **then**
    break
  end if
  Obtain or refresh q and q_{after} for every candidate task
  Obtain or reuse valid forecast token costs $\widehat{T}(a)$ for every candidate
  Complete missing or stale assessments before ranking
  Propagate current and predicted probabilities; compute U and $\widehat{\Delta U}$
  Form the eligible set $\mathcal{A}_+ = \{a \in \mathcal{A}(G) : \widehat{\Delta U}(a; G) > 0\}$
  if $\mathcal{A}_+ = \varnothing$ **then**
    break
  end if
  Select $a^\star \in \arg\max_{a \in \mathcal{A}_+} \text{score}(a; G)$
  Execute the fixed task $a^\star$ to its declared stopping point
  Record its outcome and charged cost; update G and the cumulative ledger
  Update stagnation counters and next-step prompts; invalidate affected estimates
end while
return Research state G and the allocation and cost records

and is never preset. A failed or partial proof may not leave the unresolved target as the same lone node.

E.2 Exploration Template

For an exploration task, `{prompt}` specifies the proposed investigation of the target claim using the currently available evidence, for example candidate proof routes with explicit premises, an obstruction, or a decisive construction or lemma to pursue next. The direction-specific rule for exploration tasks is:

> Every explore-direction attempt must end by registering the complete set of substantive routes corresponding to what the attempt actually learned, including unsuccessful and partial routes. Create every separately executable load-bearing claim as its own node; connect each node through every route that uses it; record mathematically distinct alternatives, requirements, reductions, and obstructions separately; preserve the correct AND/OR meaning; and leave an actionable next task for every still-OPEN premise. A completed exploration with a real blueprint makes the owner CONJECTURED. A partial exploration is not exempt: register the attempted routes, the exact point at which each failed or remains open, every component that was established, and every unresolved premise. If no viable solution route was found, register the investigated failed or obstruction routes and their evidence instead of leaving a blank undivided root. Never collapse several obligations into one cosmetic node, force several routes into one, invent dependencies, or omit an obligation.