CAN CURRENT LANGUAGE MODELS CLOSE THE DISCOVERY TO APPLICATION LOOP?

Zhou Ziheng[1,*], **Huacong Tang**[1,*] **Jinyuan Zhang**[1], **Haowei Lin**[2], **Bangcheng Yang**[1], **Qian Long**[3], **Fang Sun**[1], **Yizhou Sun**[1], **Yitao Liang**[2], **Yingnian Wu**[1], **Demetri Terzopoulos**[1], **Xiaofeng Gao**[3]
[1]University of California, Los Angeles (UCLA), Los Angeles, CA, USA
[2]Peking University, Beijing, China
[3]Amazon
[*]Equal contribution
josephziheng@ucla.edu, hctang@g.ucla.edu

ABSTRACT

The capacity to discover causal regularities and apply them to build functional systems—the discovery-to-application loop—is a hallmark of general intelligence. However, evaluating this capacity has been hindered by the vast complexity gap between scientific discovery and real-world engineering. We introduce SciCrafter, a Minecraft-based benchmark that operationalizes this loop through parameterized redstone circuit tasks. Agents must ignite lamps in specified patterns (e.g., simultaneously or in timed sequences), where scaling target parameters exponentially increases construction complexity and required knowledge—forcing genuine discovery rather than reliance on memorized solutions. Evaluating frontier models including GPT-5.2, Gemini-3-Pro, and Claude-Opus-4.5, we find that all plateau at approximately 25% success rate. To diagnose these failures, we decompose the loop into four capacities—knowledge gap identification, experimental discovery, knowledge consolidation, and knowledge application—and design targeted interventions to isolate each gap. Our analysis reveals that knowledge application bottlenecks mid-to-low tier models, while knowledge gap identification becomes the primary limitation for frontier models. We hope these findings shed light on current capacity gaps and inform future research on AI systems that can navigate the discovery-to-application loop. Project page: https://scicrafter-bench.github.io/.

1 INTRODUCTION

In human society, the interplay between discovery (e.g., science) and application (e.g., engineering) forms a self-evolving loop that drives technological advancement. Scientific investigations uncover causal models of how the world works (Pearl, 2009; Newell & Simon, 1976; Gupta et al., 2024), providing knowledge for engineering to build complex systems (Simon, 1996), while engineering challenges in turn spark new scientific inquiries. This capacity to navigate the discovery-to-application loop is a hallmark of general intelligence (Lake et al., 2017; Tenenbaum et al., 2011; Gopnik & Wellman, 2012; Goodman et al., 2008). Indeed, anthropological evidence even suggests these intertwined processes played a pivotal role in the evolution of human cognition itself (Stout, 2011; Malafouris, 2021; Vaesen, 2012; Lombard & Högberg, 2021).

A key barrier to evaluating the discovery-to-application loop is the "complexity gap"—the vast disparity in timescale and resources between identifying a scientific principle and engineering a real-world application. Modern science-to-technology pipelines span years, and physical experiments require sophisticated robotic manipulation that remains an unsolved challenge.

Minecraft offers an ideal surrogate that bridges this gap: as a persistent 3D world with rich spatial, causal, and temporal dynamics, it supports open-ended construction of complex systems—up to fully functional CPUs.[1] While basic mechanics are widely documented, precise low-level behaviors

[1]https://minecraft.fandom.com/wiki/Tutorials/Redstone_computers
This work is unrelated to the author's specific role at Amazon.

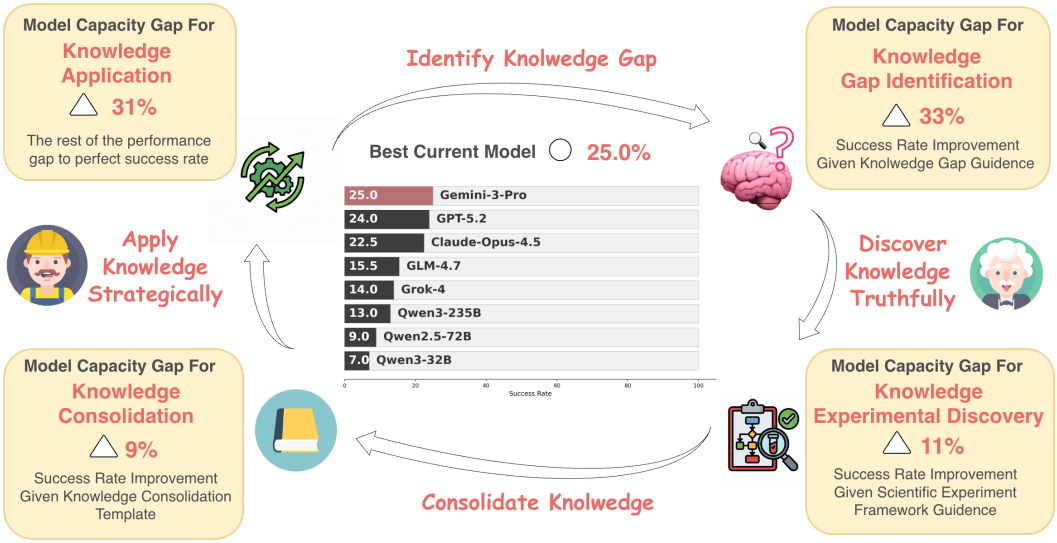

Figure 1: **Decomposing performance gaps in the Discovery-to-Application loop for the SCICRAFTERTask, illustrated with Gemini 3 Pro. Middle:** State-of-the-art models have substantial gaps; Gemini 3 Pro reaches only 25.0% success. We highlight four key bottlenecks: **Top Right (Identification):** Oracle guidance on *what* to discover (not the knowledge itself) boosts success by 33% (to 58.0%), revealing target identification as a major failure mode. **Bottom Right (Discovery):** Adding a 'scientist' sub-agent with experiments provides an extra 11% gain (to 69.0%), underscoring the value of systematic discovery. **Bottom Left (Consolidation):** Structured templates for distilling findings outperform unconstrained summaries, which often miss key information. **Top Left (Application):** The remaining 31% gap reflects limits in knowledge application. See Table 1 for details.

are edition-dependent and poorly captured by LLMs, requiring agents to discover hidden dynamics through experimentation. Crucially, its rules can be programmatically modified across tasks, preventing memorization and forcing genuine discovery; and by abstracting physical manipulation into discrete block-based interactions, it isolates the cognitive challenge of scientific inquiry and engineering design from robotic limitations.

This setting allows us to investigate two fundamental questions: **(1) Can current AI agents autonomously navigate the full discovery-to-application loop? (2) If not, which cognitive capacities constitute the primary bottlenecks?** To address these questions, we introduce SCICRAFTER, a task suite that operationalizes and evaluates these capacities with scalable complexity.

The task schema is intentionally simple: ignite N lamps in specified patterns (e.g., simultaneously, or following a delay sequence $[t_1, t_2, \ldots, t_n]$) within a fixed area (see Figure 2). This design ensures evaluation remains invariant across difficulty levels—complexity scales by adjusting target parameters alone—while construction complexity and required knowledge grow exponentially. For instance, synchronizing four lamps may succeed with basic wiring, but scaling to eight requires discovering the "nested hub" pattern. Scaling further from 32 to 64 lamps demands understanding signal degradation and how to use repeaters to circumvent it. We design five task families, each with five difficulty levels that introduce progressively larger knowledge gaps (see Sec. 3 for details).

To assess how modern AI navigates this discovery-to-application loop, we evaluated a suite of state-of-the-art models, ranging from frontier reasoning models like GPT-5.2, Gemini-3-Pro and Claude-Opus-4.5 to popular small open-source models like Qwen3-32B. We equip them with a state of art coding agent framework so that the results can best reflect the performance of the current state-of-art autonomous agents. Our results reveal a sobering reality: current agents consistently fail to close the loop, with performance plateauing around a mere 25% success rate (Figure 1).

To pinpoint the cognitive bottlenecks responsible for this failure, we decompose the discovery-to-application loop into four distinct capacities: (1) **Knowledge Gap Identification**—formulating targeted research questions; (2) **Experimental Discovery**—designing and executing rigorous experiments; (3) **Knowledge Consolidation**—distilling findings into reusable forms; and (4) **Knowl-**

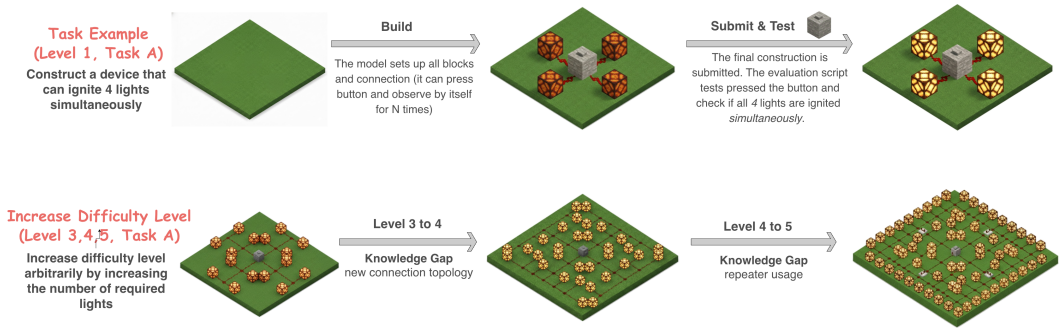

Figure 2: **SCICRAFTERTask Design Illustration. Top (Task Procedure):** The model is tasked with constructing a functional device within a constrained vacant area based on provided instructions. During construction, the agent can interact with the device (e.g., by pressing a button) and observe its behavior to iterate on the design. Finally, the device is evaluated by an automated script that verifies if the output lighting patterns match the target specifications. **Bottom (Task Difficulty):** Task complexity is parametrically scalable by adjusting the required number of lights (N). For some temporal tasks, difficulty is further increased by requiring specific sequential patterns defined by a parameter array $[N_1, N_2, \ldots, N_M]$. See details in the Appendix A.

edge Application—reasoning, planning, and executing precise engineering. We devise "oracle" interventions—such as targeted hints and a "scientist" sub-agent—whose marginal contributions serve as proxies for each capacity gap (Sec. 5.1). For instance, simply providing oracle hints on *what* to discover—without revealing the answer—more than doubles success rates for all models, exposing knowledge gap identification as a primary bottleneck.

Our analysis reveals three key findings. First, while knowledge application constitutes the largest bottleneck for mid-to-low tier models (e.g., GLM-4.7, Qwen3-32B), knowledge gap identification emerges as the primary bottleneck for frontier models (e.g., Gemini-3-Pro). Second, our "scientist" sub-agent and knowledge consolidation methods yield substantial gains (44% to 71% relative improvement), indicating that current LLMs have significant untapped potential in experimental discovery and knowledge consolidation. These findings illuminate the capacity gaps of current LLMs in the discovery-to-application loop and suggest concrete directions for future research.

In summary, our contributions are three-fold:

1. SCICRAFTER, a set of parametrically scalable discovery-to-application tasks in Minecraft.
2. A four-component capacity-gap decomposition evaluated across eight representative LLMs.
3. A "scientist" sub-agent and a knowledge consolidation method yielding 44–71% relative gains.

2 RELATED WORKS

Language model agents. Recent work has turned LLMs into *tool-using* agents by interleaving natural-language reasoning with explicit actions or API calls, including prompting-time agent loops and training-time tool-use objectives Yao et al. (2022); Schick et al. (2023); Karpas et al. (2022); Qin et al. (2023). A parallel line studies *long-horizon planning* and hierarchical control with LLM-generated decompositions or search over intermediate thoughts/plans Yao et al. (2023); Ahn et al. (2022); Huang et al. (2022). For *memory, reflection, and self-improvement*, agents maintain external state and iteratively critique/refine their own outputs, enabling multi-session competence beyond a single context window Shinn et al. (2023); Madaan et al. (2023); Packer et al. (2023); Park et al. (2023). Finally, multi-agent and role-specialization patterns (e.g., scientist/engineer sub-agents that converse and delegate) provide a practical abstraction for complex workflows Wu et al. (2023); Li et al. (2023); Hong et al. (2023), and agent evaluations increasingly target interactive, multi-step settings and real software tasks Liu et al. (2023); Jimenez et al. (2024).

Automated scientific discovery and experimentation. Beyond assisting scientists, recent systems attempt end-to-end automation of the research loop—idea generation, implementation, experimentation, and paper writing—as exemplified by fully autonomous "AI scientist" pipelines Lu et al. (2024);

Song et al. (2025); Jansen et al. (2024). In chemistry, LLM-driven autonomous research agents integrate tool use, planning, and execution for closed-loop discovery Boiko et al. (2023). Complementary work targets discovery in algorithmic domains by coupling LLMs with search or evolutionary procedures to yield novel results Romera-Paredes et al. (2024). These efforts connect naturally to *autonomous experimentation* and *active experimental design* in self-driving labs, where Bayesian optimization and active learning select informative experiments under cost and noise constraints Häse et al. (2019; 2018); Shahriari et al. (2016); Settles (2009). For scientific settings where *causal structure* matters, interventional causal discovery provides principled objectives and guarantees, ranging from classical frameworks to modern scalable optimization-based structure learning Pearl (2009); Spirtes et al. (2000); Hauser & Bühlmann (2012); Peters et al. (2016); Zheng et al. (2018).

Minecraft as testbed. Minecraft has emerged as a rich, controllable sandbox for embodied intelligence, supporting both reinforcement learning and interactive agent evaluation. Foundational platforms and datasets enable reproducible experimentation at scale Johnson et al. (2016); Guss et al. (2019), while newer frameworks broaden task diversity and incorporate internet-scale knowledge for open-ended goals Fan et al. (2022); Wang et al. (2023). Recent approaches learn general behavioral priors from unlabeled human gameplay video Baker et al. (2022) and adapt them to instruction-following with text-conditioned behavior generation Lifshitz et al. (2023); dialogue-grounded building assistants further support interactive construction tasks Gray et al. (2019); Long et al. (2024). Benchmarking suites for open-ended game agents aim to systematically measure generalization across heterogeneous Minecraft tasks Zheng et al. (2025). While Minecraft contains complex circuit-like mechanics (redstone), explicit benchmarks for redstone *reasoning* remain limited; existing work more often treats redstone as a stylistic/structural building component rather than as a dedicated logical reasoning target Huang et al. (2023).

3 TASK CONSTRUCTION

High Level Task Schema We design SCICRAFTERwith two principles in mind: *systematic curriculum* in knowledge gaps and *automated scalability*. Tasks must pose a structured sequence of knowledge gaps that require genuine discovery, and task complexity must scale automatically without manual redesign of each difficulty level.

To satisfy these desiderata, we adopt a simple yet expressive task schema: ignite N lamps in specified patterns (e.g., simultaneously, or following a delay sequence $[t_1, t_2, \ldots, t_n]$) within a fixed area. This design ensures that evaluation remains invariant across difficulty levels, while the knowledge gap to meet the requirements consistently grow, necessitating the discovery of new environmental mechanics. For instance, an agent may successfully synchronize four lamps with basic wiring, but scaling to eight requires discovering the "nested hub" pattern; scaling further from 32 to 64 lamps demands understanding signal degradation and how repeaters can circumvent it (see Figure 2).

Task Families and Knowledge Gaps We design five task families that probe distinct spatial and temporal constraints: **(A) Simultaneous Ignition**—activate N lamps at the same tick; **(B) T-Junction Routing**—connect lamps using a trunk-and-branch layout; **(C) Sequential Activation**—activate lamps with specified inter-stage delays $[t_1, t_2, \ldots, t_n]$; **(D) Distance-Equalized Ignition**—simultaneously activate lamps placed at heterogeneous distances; and **(E) Pulse Extension**—maintain activation for a specified duration τ. Details are provided in Appendix A.

Crucially, as the task requirement parameters (e.g N) scale up, models must overcome certain specific knowledge gaps in order to successfully solve higher levels. We identify three core knowledge dimensions: (1) *local wiring grammar*—adjacency-based connectivity rules and powering mechanics that make seemingly correct layouts non-functional (e.g., omitting a supporting block beneath a button yields a plausible but dead circuit); (2) *attenuation-aware topology*—signal strength decays with distance and vanishes after 15 blocks, forcing hub designs or explicit regeneration for larger footprints; and (3) *repeater semantics*—repeaters regenerate signals but impose directional flow and quantized latency, making timing skew a first-class constraint. Each task family surfaces these dimensions differently (see Figure 2); per-family knowledge gap mapping is detailed in Appendix A.

Formal Definition Let $\mathcal{S}$ denote the Minecraft world-state space with per-tick transition $s_{t+1} \sim \tau(\cdot \mid s_t)$. Each task in our suite $\mathcal{M}$ is a tuple

$$m = (I,\ s_0,\ u,\ \varphi) \in \mathcal{M} \tag{1}$$

where I is the task instruction, $s_0 \in \mathcal{S}$ is the initial world state, $u : \mathcal{S} \to \mathcal{S}$ is the stimulus that triggers evaluation (e.g., pressing a button), and φ is a temporal checker that determines whether the built artifact satisfies the desired function. Given (I, s_0), the agent follows its build policy π to produce an action sequence and a final world state $(a_{0:T-1},\ s^\star) \sim \pi(\cdot \mid I, s_0)$, where T is the agent-determined termination step. The evaluator then applies the stimulus and records a test trace $\mathbf{s} = (s_0^{\text{test}}, s_1^{\text{test}}, \ldots)$ via $s_0^{\text{test}} = u(s^\star)$ and $s_{t+1}^{\text{test}} \sim \tau(\cdot \mid s_t^{\text{test}})$, declaring success iff $\varphi(\mathbf{s}) = 1$. All constraints (e.g., spatial distance, allowed components) and tolerance thresholds (e.g., "simultaneous within ≤ 1 tick") are encoded inside φ. In practice, I is a YAML task description given to the agent, and u, φ are executable build-test scripts (Appendix A).

Environment-Agent Interface We standardize the agent-environment interface using the Model Context Protocol (MCP)[2]; details and specific tool calls are in Appendix D.

4 *Scientist* SUB-AGENT AND KNOWLEDGE CONSOLIDATION

As mentioned in the introduction, we provided a "scientist" sub-agent method to both facilitate the evaluation and as a general method contribution. The sub-agent serves as a way for the main agent to discover new knowledge through experimentation. Every time the main agent believes it encounters a knowledge gap to fill, it just needs to prompt this sub-agent the question to investigate into —for instance, *"How long does a stone button remain pressed after activation?"* And then the scientist sub-agent will not rely on prior assumptions, but instead conduct systematic control experiments to find out. When it finishes, it updates the knowledge book shared with the main agent. The main agent can then reattempt the construction task using the refined knowledge, and prompt new questions to the sub-agent.

4.1 *Scientist* SUB-AGENT DESIGN

The major design for the scientist agent is providing it with a template for conducting experiments and a template for extracting knowledge as the system prompt. Along with them, it is also provided with the current knowledge book, the history of past experiment reports, and a basic wiki of Minecraft from online.

This scientific experiment template design is drawn upon AI Scientists line of works (Lu et al., 2025; Jansen et al., 2024; Elahi et al., 2024). The essence is to formulate hypothesis, design experiments, analyze results and iterate. It follows eight parts: *Research Question* (identify the mechanic under investigation), *Hypothesis* (state a testable prediction), *Experiment Design* (specify independent variables, measurements, and controls), *Experiment Steps* (define preparation and ordered procedures), *Experiment Record* (document trial observations and whether they align with the hypothesis), *Experiment Results* (summarize outcomes concisely), *Analysis & Summary* (interpret patterns and assess implications and uncertainties), and *Next Steps* (propose follow-up experiments to refine, validate, or extend the discovered law).

Note that this workflow is provided as a system prompt rather than enforced as a rigid execution script; the LLM retains flexibility in how each step is carried out. Each experiment is repeated three times for reliability. The eight-part structure also serves as the template for the written experiment report archived after each iteration. Based on this report, the complete experiment history, and existing knowledge book entries, the scientist agent updates the knowledge book.

4.2 KNOWLEDGE CONSOLIDATION STRUCTURE

After the experiments, the scientist agent consolidate its findings into the knowledge book. We find this structure is critical for later performance (see Table 2). The final design uses four fields: *Claim (law)* (the discovered dynamics statement), *Evidence Proof* (experimental evidence supporting the

[2] `https://modelcontextprotocol.io`

Model	Baseline	Know. Iden. Gap ($\delta_{id}, r_{\delta_{id}}$)	w/ Hint	Discovery Gap ($\delta_{ds}, r_{\delta_{ds}}$)	w/ Hint + Scientist	Residual(App Gap) ($\delta_{app}, r_{\delta_{app}}$)
gemini-3-pro	$25.0_{\pm 12.5}$	Δ33.0 (1.32×)	$58.0_{\pm 6.0}$	Δ11.0 (0.44×)	$69.0_{\pm 3.5}$	Δ31.0 (1.24×)
gpt-5.2	$24.0_{\pm 12.0}$	Δ32.5 (1.35×)	$56.5_{\pm 5.8}$	Δ11.0 (0.46×)	$67.5_{\pm 3.2}$	Δ32.5 (1.35×)
claude-opus-4.5	$22.5_{\pm 11.5}$	Δ31.5 (1.40×)	$54.0_{\pm 5.5}$	Δ11.0 (0.49×)	$65.0_{\pm 3.0}$	Δ35.0 (1.56×)
glm-4.7	$15.5_{\pm 10.5}$	Δ23.0 (1.48×)	$38.5_{\pm 4.5}$	Δ9.5 (0.61×)	$48.0_{\pm 3.2}$	Δ52.0 (3.35×)
grok-4	$14.0_{\pm 9.5}$	Δ22.0 (1.57×)	$36.0_{\pm 4.2}$	Δ10.0 (0.71×)	$46.0_{\pm 3.5}$	Δ54.0 (3.86×)
qwen3-235b	$13.0_{\pm 9.0}$	Δ21.0 (1.62×)	$34.0_{\pm 4.0}$	Δ9.0 (0.69×)	$43.0_{\pm 3.2}$	Δ57.0 (4.38×)
qwen2.5-72b	$9.0_{\pm 5.5}$	Δ11.0 (1.22×)	$20.0_{\pm 3.0}$	Δ5.0 (0.56×)	$25.0_{\pm 2.5}$	Δ75.0 (8.33×)
qwen3-32b	$7.0_{\pm 4.0}$	Δ8.0 (1.14×)	$15.0_{\pm 2.5}$	Δ4.0 (0.57×)	$19.0_{\pm 2.2}$	Δ81.0 (11.57×)

Table 1: **Model Performance and Capacity Gap Decomposition.** The calculated capacity gap gains ($\delta_{id}, \delta_{sd}, \delta_{app}$) and their relative ratios (r_δ) are highlighted in gray. Ratios in parentheses (e.g., $1.32\times$) denote the gap relative to the baseline performance. Delta (δ) values (e.g Δ 33.0) represent average success rates (%) for 8 runs with standard deviations. Ratio (r_δ) values (e.g 1.32x) indicates the ratio between the delta value and the baseline performance.

Knowledge Consolidation Structure	w/Hint + Scientist
Self-determined Summary	60.0 ± 5.5
`Finding-Explanation-Example`	65.5 ± 4.2
`Claim-Proof-Constraints-Example`	69.0 ± 3.5

Table 2: **Knowledge Consolidation Ablation.** Success rates (%) of Gemini 3 Pro w/ Hint + Scientist (8-run average). Unstructured summarization yields only 60%, barely above hints alone (Table 1). The structured `Claim-Proof-Constraints-Example` format performs best.

claim), *Constraints* (conditions where the law applies or fails), and *Example* (a practical application instance). For example, a typical claim is "Redstone signal strength decreases by 1 for every block of distance traveled." The prompt templates and full example entries are in Appendix F; the ablation on consolidation structure is in Table 2.

Formulation We formulate the *scientist agent* as a hierarchical mapping operating across two distinct temporal scales: the environment step t and the discovery iteration k. For a given query Q_k, the agent conducts an experiment consisting of a trajectory $\mathcal{H}_T$ of T environment steps. After the trajectory is complete, the agent immediately generates an experiment report ER_k and updates the knowledge book KB_k. The process for each iteration k is defined as:

$$\mathrm{F}_{\text{Scientist}} : (KB_{k-1}, Q_k, W, ER_{1:k-1}, \mathcal{H}_T) \longrightarrow (\{A_t\}_{t=1}^{T}, ER_k, KB_k) \tag{2}$$

where KB_k is the knowledge book updated incrementally based on the latest findings. Each ER_k is an individual experiment report generated after a trajectory of experiment actions $\mathcal{H}_T = \{(A_t, O_t)\}_{t=1}^{T}$. This ensures that the scientist's evolving understanding is reflected in the knowledge book in real-time as each discovery iteration concludes.

5 Experiments

5.1 Experiment Design Method

As introduced in Section 1, we decompose the loop into four capacities and use oracle interventions to quantify each gap via marginal contribution. The *scientist* sub-agent and knowledge consolidation methods are detailed in Sections 4.1 and 4.2. Notably, the sub-agent requires a consolidation mechanism to relay discoveries to the primary agent, precluding fully isolated evaluation. Since consolidation filters information rather than generating it, we pair the sub-agent with our optimized consolidation method when measuring the discovery gap, providing a more accurate estimate of each model's intrinsic discovery potential.

We formalize the performance of a model M as the success probability $P(S = 1 \mid M, \mathcal{A})$, where $S \in \{0, 1\}$ is a binary random variable indicating task completion and $\mathcal{A}$ is a set of contextual assistances. We define four capacity gaps. The **Knowledge Identification Gap** (δ_{id}) measures the gain achieved by providing oracle identification guidance over the baseline. The **Knowledge Discovery Gap** (δ_{ds}) captures the further gain from introducing the scientific sub-agent that specializes at conducting scientific control experiments. Since it must use one consolidation method or another, and the consolidation method is not adding any new information, we pair it with our optimized consolidation method (h_{kc}^{opt}) so that the measurement reflects intrinsic discovery capacity. The **Consolidation Optimization Gap** (δ_{kc}) is the performance difference between default and optimized consolidation templates. The **Application Gap** (δ_{app}) is the residual gap under the most optimized discovery-consolidation pipeline, representing the fundamental execution bottleneck. This application capacity—the ability to reason and plan with acquired knowledge—underlies every stage of the loop, from identification to discovery to consolidation, and can be regarded as a foundational capability. Formally:

$$\delta_{id} = P(S = 1 \mid M, \{h_{id}\}) - P(S = 1 \mid M, \emptyset) \tag{3}$$

$$\delta_{ds} = P(S = 1 \mid M, \{h_{id}, h_{ds}, h_{kc}^{opt}\}) - P(S = 1 \mid M, \{h_{id}\}) \tag{4}$$

$$\delta_{kc} = P(S = 1 \mid M, \{h_{id}, h_{ds}, h_{kc}^{opt}\}) - P(S = 1 \mid M, \{h_{id}, h_{ds}, h_{kc}^{base}\}) \tag{5}$$

$$\delta_{app} = 1 - P(S = 1 \mid M, \{h_{id}, h_{ds}, h_{kc}^{opt}\}) \tag{6}$$

Following this sequential attribution, the total capacity space (100%) is partitioned as:

$$1 = P(S = 1 \mid M, \emptyset) + \delta_{id} + \delta_{ds} + \delta_{kc} + \delta_{app} \tag{7}$$

We further report the *relative gap ratio* $r_\delta = \delta \,/\, P(S = 1 \mid M, \emptyset)$ for each gap δ to assess its significance relative to baseline performance. For instance, a 4% absolute performance gap translates to a $0.16\times$ relative impact for Gemini-3 Pro, whereas it represents a much more substantial $0.57\times$ impact for Qwen-3-32B.

5.2 Experiment Setup

We evaluate a diverse suite of LLMs—GPT-5.2, Claude-Opus-4.5, Gemini-3-Pro, Grok-4, GLM-4.7, Qwen3-235B-MoE, Qwen2.5-72B-Instruct, and Qwen3-32B—using Claude Code (Anthropic, 2025), a state-of-the-art code agent framework. Each model is given a budget of 50 verification trials per task and evaluated in a curriculum setting (L1→L5) with experience carried forward. The definition of one verification trial is one time that the agent presses the button and receive environmental feedback. Success rate is averaged over eight runs. Full details are provided in Appendix B.

6 Results and Discussion

6.1 Baseline Performance

The comprehensive results of our baseline evaluation across frontier Large Language Models are detailed in Table 1. The highest success rate was achieved by Gemini-3 Pro at 25.0%, with GPT-5.2 (24.0%) and Claude-Opus-4.5 (22.5%) exhibiting similar performance. A secondary tier emerged among GLM-4.7 (15.5%), Grok-4 (14.0%), and Qwen-3 235B (13.0%), while Qwen2.5 72B and Qwen 3 32B formed a third tier. Notably, these models perform comparably despite parameter counts ranging from 72B to an estimated 1.7 trillion for Grok-4, suggesting that increased model size does not inherently resolve fundamental bottlenecks in the discovery-to-application cycle.

6.2 Locating the Capacity Gap

To pinpoint the cognitive deficiencies of current models, we conducted systematic ablations to isolate individual components of the discovery-to-application loop.

Knowledge Gap Identification. We first provided oracle hints (e.g., "signal decay ratio") to guide discovery without revealing solutions. Under these conditions, all models more than doubled their performance. Top-tier models achieved absolute increases of 31.5–33%, reaching 54–58% (gain

ratios of 1.32–1.40×). Mid-tier models benefited even more in relative terms: GLM-4.7, Grok-4, and Qwen-3 235B saw increases of 21–23% (1.48–1.62×). Bottom-tier models gained 8–11% (1.14–1.22×).

Experimental Discovery. Introducing a "scientist" sub-agent with optimized knowledge consolidation yielded further improvements. Top-tier models each gained an additional 11% (0.44–0.49×), reaching 65–69%. Mid-tier models gained 9–10% (0.61–0.71×), reaching 43–48%. Bottom-tier models saw increases of 4–5% (0.56–0.57×), reaching 19–25%.

Knowledge Consolidation. The format for consolidating discovered knowledge proves critical (Table 2). Using Gemini 3 Pro with hints and scientist, unstructured summarization achieved only 60%—merely 2% above hints alone. We tested two structured formats: "Finding-Explanation-Example" improved performance to 65.5%, while "Claim-Proof-Constraints-Example"—recording findings as law-like claims with supporting evidence, constraints, and examples—achieved 69% (see Appendix F for details).

Knowledge Application. The remaining performance gap reflects knowledge application capacity. Top-tier models show gaps of 31–35% (1.24–1.40×), while other models exhibit substantially larger gaps ranging from 52% for GLM-4.7 (3.35×) to 81% for Qwen 3 32B (11.57×).

Additional Results. Experiments in Appendix C show that curriculum learning (L1→L5) outperforms independent evaluation at each level. Performance naturally degrades from L1 to L5; models begin failing all tasks at Level 4 in baseline conditions. Even with full support (hints + scientist), no model succeeds at Level 5, leaving this open for future progress.

6.3 Discussion

Our results reveal that the capacity gap in knowledge gap identification is nearly three times larger than in experimental discovery. While initially surprising, this aligns with Einstein's observation that formulating a problem is often more essential than solving it Einstein & Infeld (1938)—highlighting problem formulation as an important research direction for language models.

The magnitude of the experimental discovery gap is also notable. The "scientist" agent follows a generalized research procedure that should, in principle, already reside within LLM prior knowledge. Moreover, discovery through experimentation represents fundamental reasoning—an area where frontier models claim particular strength. Yet simply requiring models to follow a formal experimental structure yields 0.44–0.71× gains across all tiers, suggesting that autonomous discovery capabilities remain underdeveloped and warrant significant attention as AI is increasingly expected to accelerate scientific breakthroughs.

The stark differences among consolidation formats reveal that LLMs perform poorly at determining how knowledge should be stored. We hypothesize that "Claim-Proof-Constraints-Example" outperforms "Finding-Explanation-Example" because it more explicitly delineates conditions under which claims hold, enabling the model to better understand when and how to apply them. This may offer insights for memory evolution research (Hu et al., 2025; Xu et al., 2025).

Finally, top models like Gemini 3 Pro and GPT 5.2 exhibit application gaps smaller than or equal to their identification gaps, suggesting that we may soon need to shift focus from improving AI for general problem-solving toward problem formulation.

6.4 Limitations and Future Work

First, while our Minecraft-based environment isolates core discovery-to-application challenges, it remains synthetic and may not capture all complexities of real-world scientific discovery. Second, we plan to release tools for procedurally modifying environment mechanics, enabling researchers to generate new environment variants arbitrarily in the future. Third, our current text-based interaction will be extended with multimodal feedback (screenshots, video) to support vision-language model evaluation.

7 CONCLUSION

We introduce a task evaluating language models in discovery-to-application scenarios, demonstrating that even the best current models fall substantially short. By employing hints, a "scientist" sub-agent, and structured knowledge consolidation, we isolate performance across four dimensions: knowledge gap identification, experimental discovery, knowledge consolidation, and knowledge application. While most models struggle with knowledge application, state-of-the-art models are increasingly bottlenecked by knowledge gap identification, with experimental discovery and knowledge consolidation also remaining inadequate.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. doi: 10.48550/arXiv.2204.01691.

Anthropic. Introducing claude 4 and claude code. https://www.anthropic.com/news/claude-4, 2025.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. In *Thirty-sixth Conference on Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2206.11795.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabriel Gomes. Autonomous chemical research with large language models. *Nature*, 2023.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083*, 2025. URL https://arxiv.org/abs/2508.00083.

Albert Einstein and Leopold Infeld. *The Evolution of Physics*. Cambridge University Press, 1938.

Muhammad Qasim Elahi, Lai Wei, Murat Kocaoglu, and Mahsa Ghasemi. Adaptive online experimental design for causal discovery. In *Forty-first International Conference on Machine Learning*, pp. 12385–12408. PMLR, 2024. URL https://arxiv.org/abs/2405.11548. Spotlight.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://arxiv.org/abs/2206.08853.

Noah D Goodman, Joshua B Tenenbaum, Jacob Feldman, and Thomas L Griffiths. A rational analysis of rule-based concept learning. *Cognitive Science*, 32(1):108–154, 2008. doi: 10.1080/03640210701802071. URL https://onlinelibrary.wiley.com/doi/abs/10.1080/03640210701802071.

Google Cloud. *Gemini Code Assist Overview*, 2025. URL https://developers.google.com/gemini-code-assist/docs/overview. Official Product Documentation.

Alison Gopnik and Henry M Wellman. Reconstructing constructivism: Causal models, bayesian learning mechanisms, and the theory theory. *Psychological Bulletin*, 138(6):1085, 2012. doi: 10.1037/a0028044. URL https://psycnet.apa.org/doi/10.1037/a0028044.

Jonathan Gray, Kavya Srinet, Yacine Jernite, Haonan Yu, Zhuoyuan Chen, Demi Guo, Siddharth Goyal, C. Lawrence Zitnick, and Arthur Szlam. Craftassist: A framework for dialogue-enabled interactive agents. *arXiv preprint arXiv:1907.08584*, 2019. doi: 10.48550/arXiv.1907.08584.

Tarun Gupta, Wenbo Gong, Chao Ma, Nick Pawlowski, Agrin Hilmkil, Meyer Scetbon, Marc Rigter, Ade Famoti, Ashley Juan Llorens, Jianfeng Gao, Stefan Bauer, Danica Kragic, Bernhard Schölkopf, and Cheng Zhang. The essential role of causality in foundation world models for embodied ai, 2024. URL https://arxiv.org/abs/2402.06665.

William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019. doi: 10.48550/arXiv.1907.13440.

Felix Häse, Loïc M. Roch, and Alán Aspuru-Guzik. Phoenics: A bayesian optimizer for chemistry. *ACS Central Science*, 4(9):1134–1145, 2018. doi: 10.1021/acscentsci.8b00307.

Felix Häse, Loïc M. Roch, and Alán Aspuru-Guzik. Next-generation experimentation with self-driving laboratories. *Trends in Chemistry*, 1(3):282–291, 2019. doi: 10.1016/j.trechm.2019.02.007.

Alain Hauser and Peter Bühlmann. Characterization and greedy learning of interventional markov equivalence classes of directed acyclic graphs. *Journal of Machine Learning Research*, 13:2409–2464, 2012.

Siyu Hong et al. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. doi: 10.48550/arXiv.2308.00352.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.

Shuo Huang, Chengpeng Hu, Julian Togelius, and Jialin Liu. Generating redstone style cities in minecraft. *arXiv preprint arXiv:2307.09777*, 2023. doi: 10.48550/arXiv.2307.09777.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022. doi: 10.48550/arXiv.2201.07207.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. In *Thirty-eighth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.06769. Spotlight.

Carlos E. Jimenez et al. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024. doi: 10.48550/arXiv.2405.15793.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, et al. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022. doi: 10.48550/arXiv.2205.00445.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017. doi: 10.1017/S0140525X16001837.

Guohao Li et al. Camel: Communicative agents for "mind" exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*, 2023. doi: 10.48550/arXiv.2303.17760.

Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *arXiv preprint arXiv:2306.00937*, 2023. doi: 10.48550/arXiv.2306.00937.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023. doi: 10.48550/arXiv.2308.03688.

Marlize Lombard and Anders Högberg. Four-field co-evolutionary model for human cognition: Variation in the middle stone age/middle palaeolithic. *Journal of Archaeological Method and Theory*, 28(1):142–177, 2021. doi: 10.1007/s10816-020-09502-6. URL https://doi.org/10.1007/s10816-020-09502-6.

Qian Long, Zhi Li, Ran (Steven) Gong, Ying Nian Wu, Demetri Terzopoulos, and Xiaofeng Gao. Teamcraft: A benchmark for multi-modal multi-agent systems in minecraft. *arXiv preprint arXiv:2412.05255*, 2024. URL https://arxiv.org/abs/2412.05255.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. doi: 10.48550/arXiv.2408.06292.

Chris Lu, Cong Lu, Jack Hessel, Jixuan Chen, Qihang Lin, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search, 2025.

Aman Madaan et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. doi: 10.48550/arXiv.2303.17651.

Lambros Malafouris. How does thinking relate to tool making? on tools making minds. *Adaptive Behavior*, 29(2):107–121, 2021. doi: 10.1177/1059712320950539. URL https://doi.org/10.1177/1059712320950539.

Allen Newell and Herbert A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, 1976.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. doi: 10.48550/arXiv.2310.08560.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023. doi: 10.48550/arXiv.2304.03442.

Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2nd edition, 2009. ISBN 978-0521895606. URL https://doi.org/10.1017/CBO9780511803161.

Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the American Statistical Association*, 111(513):160–172, 2016. doi: 10.1080/01621459.2015.1075556.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. doi: 10.48550/arXiv.2307.16789.

Bernardino Romera-Paredes et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. doi: 10.48550/arXiv.2302.04761.

Burr Settles. Active learning literature survey. Technical Report 1648, University of Wisconsin–Madison, 2009.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2016. doi: 10.1109/JPROC.2015.2494218.

Noah Shinn et al. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. doi: 10.48550/arXiv.2303.11366.

Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, 3rd edition, 1996. ISBN 0262691914. URL https://mitpress.mit.edu/9780262691914/the-sciences-of-the-artificial-3rd-edition/.

Zhangde Song et al. Evaluating large language models in scientific discovery, 2025. URL https://arxiv.org/abs/2512.15567.

Aditya Bharat Soni, Boxuan Li, Xingyao Wang, Valerie Chen, and Graham Neubig. Coding agents with multimodal browsing are generalist problem solvers. *arXiv preprint arXiv:2506.03011*, 2025. URL https://arxiv.org/abs/2506.03011.

Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. MIT Press, 2000.

Dietrich Stout. Stone toolmaking and the evolution of human culture and cognition. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 366(1567):1050–1059, 2011. doi: 10.1098/rstb.2010.0369. URL https://doi.org/10.1098/rstb.2010.0369.

Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011. doi: 10.1126/science.1192788. URL https://www.science.org/doi/10.1126/science.1192788.

Krist Vaesen. The cognitive bases of human tool use. *Behavioral and Brain Sciences*, 35(4): 203–218, 2012. doi: 10.1017/S0140525X11001452. URL https://doi.org/10.1017/S0140525X11001452.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.16291.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.01030.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. doi: 10.48550/arXiv.2308.08155.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. doi: 10.48550/arXiv.2210.03629.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023. doi: 10.48550/arXiv.2305.10601.

Xinyue Zheng, Haowei Lin, Kaichen He, Zihao Wang, Zilong Zheng, and Yitao Liang. Mcu: An evaluation framework for open-ended game agents. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2310.08367.

Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. Dags with NO TEARS: Continuous optimization for structure learning. In *Advances in Neural Information Processing Systems*, 2018. doi: 10.48550/arXiv.1803.01422.

Level	Family A (Simultaneous)	Family B (Branch Reach)	Family C (Sequential)	Family D (Equal Delay)	Family E (Pulse)
L1	N=4, skew $\leq$1 tick, radius=10	N=4, max reach=8, skew $\leq$1 tick, T-junction	N=4, delays=[1,2,1], tol=$\pm$1 tick	N=4, distance buckets={4,8,12,16}, skew $\leq$1 tick	N=4, τ=4 ticks
L2	N=8, skew $\leq$1 tick, radius=10	N=8, max reach=12, skew $\leq$1 tick, T-junction	N=8, delays=[1,2]$\times$3+[1], tol=$\pm$1 tick	N=8, distance buckets={4,8,12,16}, skew $\leq$1 tick	N=8, τ=6 ticks
L3	N=16, skew $\leq$1 tick, radius=10	N=16, max reach=15, skew $\leq$1 tick, T-junction	N=16, delays=[1,2]$\times$7+[1], tol=$\pm$1 tick	N=16, distance buckets={4,8,12,16}, skew $\leq$1 tick	N=16, τ=8 ticks
L4	N=32, skew $\leq$1 tick, radius=10	N=32, max reach=18, skew $\leq$1 tick, *repeaters required*	N=32, delays=[1,2]$\times$15+[1], tol=$\pm$1 tick	N=32, distance buckets={4,8,12,16}, skew $\leq$1 tick	N=32, τ=10 ticks
L5	N=64, skew $\leq$1 tick, radius=10	N=64, max reach=20, skew $\leq$1 tick, *repeaters required*	N=64, delays=[1,2]$\times$31+[1], tol=$\pm$1 tick	N=64, distance buckets={4,8,12,16}, skew $\leq$1 tick	N=64, τ=12 ticks

Table 3: Task families and level parameters. All families scale the number of output lamps as $N \in \{4, 8, 16, 32, 64\}$ for L1–L5, while introducing family-specific spatial/temporal constraints.

A Task Specifications

This appendix specifies the task contracts and YAML task files used in SciCrafter. Tasks are organized into five families (A–E), each instantiated at five difficulty levels (L1–L5). Difficulty does not increase as a purely quantitative "more blocks" scaling: higher levels cross discrete redstone *mechanism thresholds* (e.g., attenuation and repeater delay semantics), which forces qualitatively different engineering patterns (e.g., hierarchical distribution, delay compensation).

A.1 Common Setup and Contracts

World. All tasks run in a flat, empty creative-mode "redstone lab" with a bounded build region: a radius-10 cube centered at an *anchor* position (default $(0, 4, 0)$). The evaluator resets this region before each new attempt.

Component palette. To focus evaluation on spatiotemporal reasoning rather than component breadth, we restrict the palette to "redstone fundamentals": *stone button, redstone wire, redstone repeater, redstone torch, redstone lamp*, plus inert support blocks (stone, glass) and `air` for removal.

I/O and tick-level logging. Each task provides a single input stimulus (button press) and uses N lamps as outputs. The evaluator records the on/off state of each lamp at every game tick (20Hz) and checks a temporal contract φ with a ± 1 tick tolerance (50ms) unless stated otherwise.

Metrics. Primary evaluation is functional correctness (Acc: whether φ passes). We additionally track interaction and stability signals, including the number of environment verification trials (button presses/observations), Tokens/Attempt, and Engineer rebuild Trials$_{Eng}$.

Difficulty levels. Across task families, difficulty levels scale the number of output lamps as $N \in \{4, 8, 16, 32, 64\}$ for L1–L5 (Table 3), while each family introduces additional spatial/temporal constraints (e.g., topology restrictions, delay patterns, distance heterogeneity, pulse duration).

A.2 FAMILY A: SIMULTANEOUS N-LAMP CONTROL

Goal. Build a circuit that activates N lamps (nearly) simultaneously when the button is pressed.

Why it is challenging. For small N, a symmetric star/branch pattern can often succeed. As N grows within a fixed footprint, the agent must (i) distribute power to many endpoints without unintended dust auto-connections, and (ii) respect signal attenuation (signal strength decays and vanishes after 15), which eventually makes repeaters mandatory. Repeaters, however, are not "free": they are directional and introduce quantized delay, so maintaining simultaneity requires joint reasoning about distance *and* delay.

Level schedule. We use the shared N schedule in Table 3.

I/O contract. Let t_{press} be the tick when the button is activated, and t_i be the first tick when lamp i turns on. The simultaneity contract requires:

$$\forall i, j \in [1, N] : |t_i - t_j| \leq 1 \text{ tick}. \tag{8}$$

Example task file (L1).

```
task_id: A_simultaneous_lights_L1
family: A
level: L1
task_name: Simultaneous Lights (N=4)
task_description: |
  Build a redstone circuit that turns on 4 lamps with (near) zero skew after a single button
     press.
  This level is meant to validate basic wiring semantics: strong powering via a support
     block, dust connectivity, and simple fan-out.
difficulty: beginner

world:
  anchor: [0, 4, 0]
  radius: 10

allowed_blocks:
  - minecraft:stone_button
  - minecraft:redstone_lamp
  - minecraft:redstone_wire
  - minecraft:redstone_torch
  - minecraft:redstone_repeater
  - minecraft:stone
  - minecraft:glass
  - minecraft:air

inputs:
  button:
    type: minecraft:stone_button
    quantity: 1

outputs:
  lamps:
    type: minecraft:redstone_lamp
    quantity: 4

contract:
  type: simultaneity
  tolerance_ticks: 1
  num_outputs: 4

requirements:
  - All 4 lamps must turn on within +-1 tick of each other after the button press
  - Construction must fit within the radius-10 build region
  - Use only the allowed block palette

evaluation_metrics:
  functional_correctness:
    description: Verify all lamps activate
    weight: 1.0
    check_method: verify_all_lamps_activated
  simultaneity:
    description: Max skew across lamps <= 1 tick
    weight: 1.0
    check_method: verify_max_skew_within
    params:
      tolerance_ticks: 1
```

Level	N	Knowledge Gap	Hint
L1	4	Button powering primitives: a button must strongly power a supporting block; dust only conducts via cardinal (N/S/E/W) adjacency; lamps require adjacent powered dust.	Strong-Power-Support-Block
L2	8	Nested fanout topology (hub $\rightarrow$ branch hubs) to reach off-axis/diagonal lamps without diagonal dust connectivity; preserve symmetry to avoid accidental skew.	Nested-Hub-Fanout
L3	16	Signal strength attenuates with wire distance; multi-radius branching (e.g., two rings) must be planned so all lamps receive nonzero power within the dust budget.	Signal-Strength-Decay
L4	32	High-fanout distribution under radius and attenuation constraints: shared backbones/rails and dense symmetric branching; avoid unintended dust auto-connections/shorts in crowded layouts.	Attenuation-Aware-Fanout
L5	64	Repeater semantics for scaling: where to insert repeaters to regenerate signal past the attenuation limit; repeaters are directional and add delay, so boosters must be placed symmetrically to maintain ± 1 tick simultaneity.	Repeater-Signal-Regeneration

Table 4: **Breakdown of knowledge gaps per level for Task A**. This task variant requires activating N lamps simultaneously. **Knowledge Gap** lists the environment knowledge agents must discover to design a working solution. **Hint** is a concise directional keyword indicating the missing area without revealing the specific mechanism.

```
test_cases:
  - name: single_press
    sequence:
      - action: press_button
      - action: check_lamps_on

metadata:
  learning_objectives:
    - strong_power_support_block
    - dust_connectivity
    - simple_fanout
  estimated_difficulty_score: 1.0
```

A.3 Family B: Branch Reach (T-Junction Expansion)

Goal. Reach N off-axis lamps from a central trunk using T-junction branches, while maintaining valid powering (attenuation-aware) and near-simultaneous activation.

Why it is challenging. Unlike Family A, which primarily rewards symmetry, Family B constrains the *wiring topology*: solutions must implement a trunk-and-branch routing pattern. This exposes different failure modes: long trunk lines need repeater placement that respects attenuation, while crowded branches are prone to dust auto-connections that create unintended shortcuts.

Level schedule. We use the shared N schedule in Table 3. The maximum required reach (measured in wire path length) increases from 8 to 20 blocks across levels.

I/O contract. Same simultaneity requirement as Family A (± 1 tick), with an additional topology constraint that the solution must contain at least one explicit T-junction branch (fan-out node) in the redstone dust graph.

Example task file (L1).

```
task_id: B_branch_reach_L1
family: B
level: L1
task_name: Branch Reach (T-junction) (N=4)
task_description: |
  Build a trunk-and-branch wiring layout that reaches 4 lamps that are not collinear with the
     button.
  The circuit must use at least one explicit T-junction and still satisfy the same +/-1 tick
     simultaneity tolerance.
difficulty: beginner
```

```yaml
world:
  anchor: [0, 4, 0]
  radius: 10

allowed_blocks:
  - minecraft:stone_button
  - minecraft:redstone_lamp
  - minecraft:redstone_wire
  - minecraft:redstone_torch
  - minecraft:redstone_repeater
  - minecraft:stone
  - minecraft:glass
  - minecraft:air

inputs:
  button:
    type: minecraft:stone_button
    quantity: 1

outputs:
  lamps:
    type: minecraft:redstone_lamp
    quantity: 4

contract:
  type: branch_reach
  tolerance_ticks: 1
  num_outputs: 4
  max_reach_blocks: 8
  require_t_junction: true

requirements:
  - All 4 lamps must turn on within +/-1 tick of each other after the button press
  - At least one T-junction (fan-out) must be present in the wiring topology
  - The solution must respect redstone attenuation (no dead outputs)
  - Construction must fit within the radius-10 build region

evaluation_metrics:
  functional_correctness:
    description: Verify all lamps activate
    weight: 1.0
    check_method: verify_all_lamps_activated
  simultaneity:
    description: Max skew across lamps <= 1 tick
    weight: 1.0
    check_method: verify_max_skew_within
    params:
      tolerance_ticks: 1

test_cases:
  - name: single_press
    sequence:
      - action: press_button
      - action: check_lamps_on

metadata:
  learning_objectives:
    - t_junction_branching
    - off_axis_routing
    - attenuation_awareness
  estimated_difficulty_score: 1.0
```

A.4 Family C: Sequential Activation (Delay Pattern)

Goal. Activate lamps in a specified temporal order with target inter-stage delays.

Why it is challenging. This family tests *temporal synthesis*: the agent must realize a target delay sequence using repeaters with discrete delay settings (1–4 ticks), while preventing accidental bypasses (e.g., dust auto-connections) that would trigger later stages early.

Level schedule. We use the shared N schedule in Table 3. Each YAML file specifies an inter-stage delay vector $\boldsymbol{\delta}$ of length $N - 1$ (e.g., an alternating pattern), which the agent must realize with quantized repeater delays.

I/O contract. Let δ_i be the specified delay between stage i and stage $i+1$. The contract requires:

$$\forall i \in [1, N-1] : |t_{i+1} - t_i - \delta_i| \leq 1 \text{ tick.} \tag{9}$$

Example task file (L1).

```
task_id: C_sequential_activation_L1
family: C
level: L1
task_name: Sequential Activation (N=4, delays=[1,2,1])
task_description: |
  After a button press, activate 4 lamps in order with inter-stage delays [1, 2, 1] ticks
     (+/-1 tick tolerance).
  This level checks that the agent can realize a basic multi-stage delay line using repeaters.
difficulty: beginner

world:
  anchor: [0, 4, 0]
  radius: 10

allowed_blocks:
  - minecraft:stone_button
  - minecraft:redstone_lamp
  - minecraft:redstone_wire
  - minecraft:redstone_torch
  - minecraft:redstone_repeater
  - minecraft:stone
  - minecraft:glass
  - minecraft:air

inputs:
  button:
    type: minecraft:stone_button
    quantity: 1

outputs:
  lamps:
    type: minecraft:redstone_lamp
    quantity: 4

contract:
  type: sequential
  tolerance_ticks: 1
  delays_ticks: [1, 2, 1]

requirements:
  - Lamps must activate in order with the specified inter-stage delays

evaluation_metrics:
  functional_correctness:
    description: Verify order and timing per delays
    weight: 1.0
    check_method: verify_sequence_with_delays
    params:
      delays_ticks: [1, 2, 1]
      tolerance_ticks: 1

test_cases:
  - name: single_press
    sequence:
      - action: press_button
      - action: check_sequence
        params:
          delays_ticks: [1, 2, 1]
          tolerance_ticks: 1

metadata:
  learning_objectives:
    - repeater_delay_basic
    - ordered_activation
  estimated_difficulty_score: 1.0
```

A.5 Family D: Equal-Delay Distribution

Goal. Deliver a signal to lamps placed at heterogeneous distances, while ensuring all lamps still activate (nearly) simultaneously by compensating shorter paths with additional delay.

Why it is challenging. Family D stresses *delay alignment* under heterogeneous geometry. Some outputs are inherently "slow" because they require longer routes and/or mandatory repeaters for signal regeneration. To synchronize all outputs, the agent must intentionally *slow down the fast paths* by inserting compensation repeaters, turning repeaters into timing buffers rather than just range extenders.

Level schedule. We use the shared N schedule in Table 3. Lamps are partitioned into distance buckets to induce heterogeneous inherent delays; the agent must compensate shorter paths so that all outputs satisfy the same simultaneity tolerance.

I/O contract. Despite different path lengths, all lamps must activate simultaneously within tolerance:

$$\forall i, j \in [1, N] : |t_i - t_j| \leq 1 \text{ tick}. \tag{10}$$

Example task file (L1).

```
task_id: D_equal_delay_distribution_L1
family: D
level: L1
task_name: Equal-Delay Distribution (N=4, distances={4,8,12,16})
task_description: |
  Place and wire a circuit so that 4 lamps at different distances still turn on within +/-1
     tick of each other after a button press.
  The intended solution uses repeaters as compensatory delay elements on the faster (shorter)
     paths.
difficulty: beginner

world:
  anchor: [0, 4, 0]
  radius: 10

allowed_blocks:
  - minecraft:stone_button
  - minecraft:redstone_lamp
  - minecraft:redstone_wire
  - minecraft:redstone_torch
  - minecraft:redstone_repeater
  - minecraft:stone
  - minecraft:glass
  - minecraft:air

inputs:
  button:
    type: minecraft:stone_button
    quantity: 1

outputs:
  lamps:
    type: minecraft:redstone_lamp
    quantity: 4

contract:
  type: equal_delay
  tolerance_ticks: 1
  target_distances: [4, 8, 12, 16]

requirements:
  - All 4 lamps must turn on within +/-1 tick (skew <= 1)
  - The solution must compensate different distances using repeater delays

evaluation_metrics:
  functional_correctness:
    description: Verify all lamps activate
    weight: 1.0
    check_method: verify_all_lamps_activated
  skew_tolerance:
    description: Max skew across lamps <= 1 tick
    weight: 1.0
    check_method: verify_max_skew_within
    params:
      tolerance_ticks: 1

test_cases:
  - name: single_press
    sequence:
      - action: press_button
```

```
      - action: check_lamps_on

metadata:
  learning_objectives:
    - delay_compensation_basic
    - repeater_as_buffer
  estimated_difficulty_score: 1.0
```

A.6 FAMILY E: PULSE SHAPING (FIXED ON-DURATION)

Goal. On each button press, keep all N lamps lit for a target duration τ ticks, then return to off.

Why it is challenging. The input button produces a pulse width determined by the game mechanics. Family E requires shaping this pulse into a target on-duration τ, then distributing the resulting waveform to N outputs without introducing large skew. This typically requires a pulse shaper (e.g., monostable / edge-triggered variants realizable with torches and repeaters), not just copying the input wire.

Level schedule. We use the shared N schedule in Table 3. The target duration increases from $\tau = 4$ to $\tau = 12$ ticks across levels.

I/O contract. Let t_{press} be the button press time. For each lamp i, define t_i^{on} as the first on tick and t_i^{off} as the first off tick after activation. The contract requires:

$$t_i^{\text{on}} \in [t_{\text{press}},\, t_{\text{press}} + 1], \tag{11}$$

$$t_i^{\text{off}} \in [t_{\text{press}} + \tau - 1,\, t_{\text{press}} + \tau + 1]. \tag{12}$$

YAML note. For compactness, this appendix includes YAML examples for Families A–D. Family E follows the same schema and is released with the benchmark harness.

A.7 TASK FILE FORMAT (YAML)

Each task is specified in YAML and stored under `appendices/task_specs/`. The released files follow the same high-level structure:

```
# Identity
task_id: <string>
family: <A|B|C|D|E>
level: <L1|L2|L3|L4|L5>
task_name: <string>
task_description: <string>

# Build region and palette
world:
  anchor: [x, y, z]
  radius: 10
allowed_blocks:
  - minecraft:stone_button
  - minecraft:redstone_wire
  - minecraft:redstone_repeater
  - minecraft:redstone_torch
  - minecraft:redstone_lamp
  - minecraft:stone
  - minecraft:glass
  - minecraft:air

# I/O specification
inputs:  { ... }      # one stone button
outputs: { ... }      # N lamps
contract: { ... }     # family-specific parameters (e.g., tolerance, delays, tau)

# Test harness
test_cases:
  - name: <string>
    sequence:
      - {action: press_button}
      - {action: check_<...>, params: {...}}
```

Metric	Definition
Acc	Fraction of tasks whose constructed device satisfies the spatiotemporal contract φ over all evaluator test cases.
Tokens/Attempt	Total LLM tokens consumed per task attempt (including planning and tool calls), averaged over attempts.
Trials_{Eng}	Number of build–revise cycles executed by the Engineer within a task before final submission (lower is better).
Verification Trials	Number of environment interaction trials used for exploration and validation (e.g., button press plus subsequent observations), subject to a fixed budget.

Table 5: Evaluation metrics tracked for SCICRAFTER.

```
# Optional metadata (not used by the checker)
metadata:
  learning_objectives: [ ... ]
```

B EXPERIMENT DETAILS

Code Agent Framework We evaluate all models using a state-of-the-art code agent framework (Dong et al., 2025; Google Cloud, 2025; Anthropic, 2025). Our tasks require sustained interaction and complex reasoning, making the code agent paradigm well-suited as an evaluation vehicle: it is designed to interface with complex environments and construct functional artifacts through executable code (Soni et al., 2025; Wang et al., 2024), providing a faithful representation of current frontier AI capabilities. Specifically, we use Claude Code (Anthropic, 2025), though our method generalizes to other code agents.

Model Selection Our selection spans closed- and open-source models, frontier reasoning systems, and various parameter scales within model families: GPT-5.2, Claude-Opus-4.5, Gemini-3-Pro, Grok-4, GLM-4.7, Qwen3-235B-MoE, Qwen2.5-72B-Instruct, and Qwen3-32B. Models are evaluated by swapping the underlying API while keeping all other components fixed.

Verification Trials We enforce a budget of 50 verification trials per task as a normalized compute metric that sidesteps token-count variability across model architectures. A verification trial consists of the agent triggering a mechanism (e.g., pressing a button) and observing subsequent block-state transitions. Importantly, the environment returns only raw state changes, not success signals from the evaluation script; agents must autonomously interpret these responses to assess functionality. All trials conducted by the scientist sub-agent count toward this global budget, forcing agents to trade off between exploration and final validation.

Evaluation Protocol Models are evaluated in a curriculum setting: agents progress from L1 to L5, carrying forward accumulated experience. The primary metric is success rate (percentage of tasks completed), averaged over eight independent runs. Results for independent (non-curriculum) evaluation are reported in Section C.

Model	Independent.				Curriculum.			
	baseline	w/ hint	w/ scientist sub-agent	w/ hint + scientist	baseline	w/ hint	w/ scientist sub-agent	w/ hint + scientist
gemini-3-pro-preview	$18.0_{\pm 10.5}$	$48.5_{\pm 5.5}$	$26.5_{\pm 11.0}$	$55.0_{\pm 3.2}$	$25.0_{\pm 12.5}$	$58.0_{\pm 6.0}$	$38.5_{\pm 13.0}$	$69.0_{\pm 3.5}$
gpt-5.2	$17.5_{\pm 10.2}$	$46.5_{\pm 5.2}$	$25.0_{\pm 10.5}$	$53.5_{\pm 3.0}$	$24.0_{\pm 12.0}$	$56.5_{\pm 5.8}$	$37.0_{\pm 12.5}$	$67.5_{\pm 3.2}$
claude-opus-4.5	$16.5_{\pm 9.8}$	$45.0_{\pm 5.0}$	$23.5_{\pm 10.0}$	$51.0_{\pm 2.8}$	$22.5_{\pm 11.5}$	$54.0_{\pm 5.5}$	$35.0_{\pm 12.0}$	$65.0_{\pm 3.0}$
glm-4.7	$11.0_{\pm 8.5}$	$32.0_{\pm 4.0}$	$16.5_{\pm 8.0}$	$35.0_{\pm 3.2}$	$15.5_{\pm 10.5}$	$38.5_{\pm 4.5}$	$24.0_{\pm 10.0}$	$48.0_{\pm 3.2}$
grok-4	$10.0_{\pm 8.0}$	$30.0_{\pm 3.8}$	$15.0_{\pm 7.5}$	$33.0_{\pm 3.5}$	$14.0_{\pm 9.5}$	$36.0_{\pm 4.2}$	$22.0_{\pm 9.5}$	$46.0_{\pm 3.5}$
qwen3-235b	$9.5_{\pm 7.5}$	$28.0_{\pm 3.5}$	$14.0_{\pm 7.0}$	$31.0_{\pm 3.5}$	$13.0_{\pm 9.0}$	$34.0_{\pm 4.0}$	$20.0_{\pm 9.0}$	$43.0_{\pm 3.2}$
qwen2.5-72b	$6.5_{\pm 4.0}$	$16.0_{\pm 2.5}$	$10.0_{\pm 5.0}$	$18.0_{\pm 2.2}$	$9.0_{\pm 5.5}$	$20.0_{\pm 3.0}$	$14.0_{\pm 6.5}$	$25.0_{\pm 2.5}$
qwen3-32b	$5.0_{\pm 3.0}$	$12.0_{\pm 2.0}$	$8.0_{\pm 4.0}$	$14.0_{\pm 1.8}$	$7.0_{\pm 4.0}$	$15.0_{\pm 2.5}$	$10.5_{\pm 5.5}$	$19.0_{\pm 2.2}$

Table 6: **Main Performance + Ablations.** Success rates (%) for **Independent and Curriculum settings** - whether the knowledge can be carried across tasks from low to high difficulty levels within the same family. **Hints** guide agents toward the correct inquiry, revealing the capacity gap in "asking the right questions". The **scientist sub-agent** enhances the ability to do systematic control experiments, highlighting gaps in robust knowledge discovery. All of the experiments are given 50 trial budgets.

Level	Independent Setting (Baseline)						Curriculum Setting (Baseline)					
Description	Task A	Task B	Task C	Task D	Task E	Avg	Task A	Task B	Task C	Task D	Task E	Avg
L1 (Primitive)	$52.0_{\pm 18.5}$	$44.0_{\pm 16.0}$	$40.0_{\pm 15.5}$	$48.0_{\pm 17.5}$	$56.0_{\pm 19.5}$	**48.0**$_{\pm 17.5}$	$63.0_{\pm 20.5}$	$57.0_{\pm 19.5}$	$54.0_{\pm 18.5}$	$60.0_{\pm 20.0}$	$66.0_{\pm 21.5}$	**60.0**$_{\pm 20.0}$
L2 (Basic)	$27.0_{\pm 12.5}$	$21.0_{\pm 10.5}$	$18.0_{\pm 9.5}$	$24.0_{\pm 11.5}$	$30.0_{\pm 13.5}$	**24.0**$_{\pm 11.5}$	$38.0_{\pm 15.5}$	$32.0_{\pm 13.5}$	$29.0_{\pm 12.5}$	$35.0_{\pm 14.5}$	$41.0_{\pm 16.5}$	**35.0**$_{\pm 14.5}$
L3 (Intermediate)	$2.0_{\pm 2.5}$	0.0	0.0	0.0	$3.0_{\pm 3.0}$	**1.0**$_{\pm 1.5}$	$10.0_{\pm 6.5}$	$6.0_{\pm 4.5}$	$4.0_{\pm 3.5}$	$8.0_{\pm 5.5}$	$13.5_{\pm 7.5}$	**8.3**$_{\pm 5.5}$
L4 (Advanced)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L5 (Complex)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Task Avg	$20.3_{\pm 11.5}$	$16.3_{\pm 9.5}$	$14.5_{\pm 8.5}$	$18.0_{\pm 10.5}$	$22.3_{\pm 12.0}$	**18.0**$_{\pm 10.5}$	$27.8_{\pm 14.0}$	$23.8_{\pm 12.5}$	$21.8_{\pm 11.5}$	$25.8_{\pm 13.0}$	$30.1_{\pm 15.0}$	**25.0**$_{\pm 12.5}$

Table 7: **Baseline Performance Breakdown.** Success rates of Gemini 3 Pro without any assistance.

C More Experiment Results

We report more experiment results here.

In Table 6, we report results under the Independent setting, where tasks are solved independently rather than in curriculum order (L1 to L5 for each task). While the main paper centers on the Curriculum setting—which better reflects models' upper-bound capacity—we include Independent results to examine how models perform without progressive scaffolding. As the results show, the performance of independent setting are all lower than that in curriculum setting, indicating how appropriate application challenges can help promote the discoveries.

We also reports results with the scientist sub-agent alone (using the optimized knowledge consolidation structure) without hints. The main paper reports the performance difference between models with hints and models with hints plus the scientist sub-agent, ensuring all gaps sum to 100%. Here, we isolate the contribution of the scientist sub-agent alone. The results show that it alone attains slightly bigger improvement than along with hints. For example Gemini 3 Pro improved 13.5% from baseline to w/scientist sub-agent, which is 3.5% larger than the 10% improvement from w/hint to w/hint+scientist. This indicates that a good discovery ability can inspire better knowledge identification.

Tables 7 through 10 provide a detailed breakdown of Gemini 3 Pro's performance by task type and difficulty level. We report results across four conditions—baseline, with hints, with scientist alone, and with hints plus scientist—under both Independent and Curriculum settings. The baseline model fails completely at L4, but with hints or the scientist sub-agent, models begin to tackle L4 challenges. However, none succeed at L5, leaving this as an open challenge for future models. We also observe that Task C is consistently the most difficult, achieving the lowest scores across all conditions. This may be because its sequential requirements pose greater demands on precise spatial-temporal reasoning—a notable weakness of current language models.

Level	Independent Setting (w/ Hint)						Curriculum Setting (w/ Hint)					
Description	**Task A**	**Task B**	**Task C**	**Task D**	**Task E**	**Avg**	**Task A**	**Task B**	**Task C**	**Task D**	**Task E**	**Avg**
L1 (Primitive)	$98.0_{\pm 3.2}$	$94.0_{\pm 4.5}$	$92.0_{\pm 5.2}$	$96.0_{\pm 3.8}$	100.0	**96.0**$_{\pm 3.5}$	$99.0_{\pm 2.2}$	$97.0_{\pm 3.5}$	$96.0_{\pm 4.0}$	$98.0_{\pm 2.8}$	100.0	**98.0**$_{\pm 2.5}$
L2 (Basic)	$82.0_{\pm 8.5}$	$76.0_{\pm 9.2}$	$72.0_{\pm 9.8}$	$78.0_{\pm 8.8}$	$86.0_{\pm 7.5}$	**78.8**$_{\pm 8.5}$	$89.0_{\pm 7.8}$	$83.0_{\pm 8.5}$	$80.0_{\pm 8.8}$	$86.0_{\pm 8.2}$	$92.0_{\pm 6.5}$	**86.0**$_{\pm 7.8}$
L3 (Intermediate)	$45.0_{\pm 10.5}$	$38.0_{\pm 9.8}$	$34.0_{\pm 9.5}$	$42.0_{\pm 10.0}$	$48.0_{\pm 10.8}$	**41.4**$_{\pm 10.2}$	$63.0_{\pm 11.2}$	$57.0_{\pm 10.5}$	$54.0_{\pm 10.0}$	$60.0_{\pm 10.8}$	$66.0_{\pm 11.5}$	**60.0**$_{\pm 10.8}$
L4 (Advanced)	0.0	0.0	0.0	0.0	0.0	0.0	$12.0_{\pm 5.5}$	$8.0_{\pm 4.2}$	$6.0_{\pm 3.2}$	$10.0_{\pm 4.8}$	$16.0_{\pm 6.5}$	**10.4**$_{\pm 4.8}$
L5 (Complex)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Task Avg	$54.8_{\pm 5.5}$	$49.6_{\pm 5.2}$	$47.0_{\pm 4.8}$	$52.2_{\pm 5.4}$	$57.0_{\pm 5.8}$	**48.5**$_{\pm 5.5}$	$65.8_{\pm 6.0}$	$61.3_{\pm 5.8}$	$59.0_{\pm 5.5}$	$63.5_{\pm 5.9}$	$68.5_{\pm 6.2}$	**58.0**$_{\pm 6.0}$

Table 8: **Augmented with Hint.** Breakdown of performance when the agent is provided with discovery target hints.

Level	Independent Setting (w/ Scientist)						Curriculum Setting (w/ Scientist)					
Description	**Task A**	**Task B**	**Task C**	**Task D**	**Task E**	**Avg**	**Task A**	**Task B**	**Task C**	**Task D**	**Task E**	**Avg**
L1 (Primitive)	$68.0_{\pm 18.5}$	$60.0_{\pm 16.5}$	$58.0_{\pm 16.0}$	$64.0_{\pm 17.5}$	$70.0_{\pm 19.0}$	**64.0**$_{\pm 17.5}$	$79.0_{\pm 20.5}$	$73.0_{\pm 19.5}$	$70.0_{\pm 19.0}$	$76.0_{\pm 20.0}$	$82.0_{\pm 21.5}$	**76.0**$_{\pm 20.0}$
L2 (Basic)	$40.0_{\pm 13.5}$	$32.0_{\pm 11.5}$	$30.0_{\pm 11.0}$	$36.0_{\pm 12.5}$	$42.0_{\pm 14.0}$	**36.0**$_{\pm 12.5}$	$58.0_{\pm 16.5}$	$52.0_{\pm 15.5}$	$49.0_{\pm 15.0}$	$55.0_{\pm 16.0}$	$61.0_{\pm 17.5}$	**55.0**$_{\pm 16.0}$
L3 (Intermediate)	$14.0_{\pm 8.5}$	$8.0_{\pm 6.5}$	$6.0_{\pm 5.5}$	$10.0_{\pm 7.5}$	$18.0_{\pm 9.5}$	**11.2**$_{\pm 7.5}$	$33.0_{\pm 12.5}$	$27.0_{\pm 11.5}$	$24.0_{\pm 10.5}$	$30.0_{\pm 12.0}$	$36.0_{\pm 13.5}$	**30.0**$_{\pm 12.0}$
L4 (Advanced)	0.0	0.0	0.0	0.0	0.0	0.0	$6.0_{\pm 5.2}$	$2.0_{\pm 3.5}$	0.0	$4.0_{\pm 4.5}$	$8.5_{\pm 6.5}$	**4.1**$_{\pm 5.0}$
L5 (Complex)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Task Avg	$29.8_{\pm 11.0}$	$24.6_{\pm 9.5}$	$23.2_{\pm 9.0}$	$27.2_{\pm 10.0}$	$31.6_{\pm 11.5}$	**26.5**$_{\pm 11.0}$	$44.0_{\pm 13.5}$	$38.5_{\pm 12.5}$	$35.8_{\pm 12.0}$	$41.3_{\pm 13.0}$	$46.9_{\pm 14.0}$	**38.5**$_{\pm 13.0}$

Table 9: **Augmented with Scientist Sub-agent.** Breakdown of performance when the agent is augmented with the Scientist sub-agent.

Level	Independent Setting (w/ Hint + Scientist)						Curriculum Setting (w/ Hint + Scientist)					
Description	**Task A**	**Task B**	**Task C**	**Task D**	**Task E**	**Avg**	**Task A**	**Task B**	**Task C**	**Task D**	**Task E**	**Avg**
L1 (Primitive)	$99.0_{\pm 1.2}$	$97.0_{\pm 1.8}$	$96.0_{\pm 2.2}$	$98.0_{\pm 1.5}$	100.0	**98.0**$_{\pm 1.2}$	100.0	100.0	100.0	100.0	100.0	**100.0**
L2 (Basic)	$85.0_{\pm 4.5}$	$79.0_{\pm 5.2}$	$76.0_{\pm 5.8}$	$82.0_{\pm 4.8}$	$88.0_{\pm 3.8}$	**82.0**$_{\pm 4.8}$	$99.0_{\pm 0.8}$	$97.0_{\pm 1.2}$	$96.0_{\pm 1.5}$	$98.0_{\pm 1.2}$	100.0	**98.0**$_{\pm 1.2}$
L3 (Intermediate)	$58.0_{\pm 7.5}$	$52.0_{\pm 7.2}$	$48.0_{\pm 6.8}$	$55.0_{\pm 7.4}$	$62.0_{\pm 7.8}$	**55.0**$_{\pm 7.4}$	$87.0_{\pm 4.5}$	$81.0_{\pm 5.2}$	$78.0_{\pm 5.8}$	$84.0_{\pm 4.8}$	$90.0_{\pm 3.8}$	**84.0**$_{\pm 4.8}$
L4 (Advanced)	$6.0_{\pm 3.5}$	$2.0_{\pm 1.8}$	0.0	$4.0_{\pm 2.5}$	$10.0_{\pm 4.8}$	**4.4**$_{\pm 3.2}$	$33.0_{\pm 7.5}$	$27.0_{\pm 7.2}$	$24.0_{\pm 6.8}$	$30.0_{\pm 7.4}$	$35.0_{\pm 7.8}$	**29.8**$_{\pm 7.4}$
L5 (Complex)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Task Avg	$60.6_{\pm 3.5}$	$56.4_{\pm 3.2}$	$54.0_{\pm 2.8}$	$58.8_{\pm 3.4}$	$63.8_{\pm 3.8}$	**55.0**$_{\pm 3.2}$	$79.7_{\pm 3.8}$	$76.3_{\pm 3.5}$	$74.5_{\pm 3.2}$	$78.1_{\pm 3.6}$	$81.3_{\pm 4.0}$	**69.0**$_{\pm 3.5}$

Table 10: **Full Method (Hint + Scientist).** Breakdown of performance with both augmentations.

D ENV SPECIFICATIONS

Environment Setup In our benchmark, the Minecraft environment is set into *creative* mode where block items are provided to agents so that we can focus on building instead of collecting resources.

To focus evaluation on spatiotemporal-causal reasoning rather than component breadth, this version restricts to common redstone components: *stone button, redstone wire, redstone repeater, redstone torch, redstone lamp* (with inert supports such as stone and glass). More advanced components (e.g., comparators, observers) are excluded and reserved for future extensions.

MCP Calls To facilitate the community to test our tasks, we wrap the interaction interface between the environment and the agent using **Model Context Protocol (MCP)**[3]. MCP has become the standard function call protocol for LLMs so the capability of LLMs would not be affected by the function call format.

Specifically, our observation MCP calls include `get-block-state` to capture desired block state, `get-event-stream` to receive recent event sequence, and `scan-redstone-area` to return nearby redstone components. To configure blocks, the calls include `set-block` to any type, position, or state, thereby capturing all building operations. The `activate-button` action automatically locates and toggles the device's button.

E PROMPT AND EXPERIMENT TEMPLATES

This appendix provides full prompt templates used by the Scientist and Engineer agents, along with the experiment write-up template used for controlled game-mechanics exploration.

E.1 SCIENTIST AGENT PROMPT TEMPLATE (YAML)

```
# Scientist Agent Prompt Template
# Purpose: Discover environment laws through controlled experimentation
# Based on: SciCraft paper Section 4.1 (Scientist Agent)

agent_name: Scientist Agent
agent_role: Environment Law Discovery and Documentation
version: 1.0

# System prompt - describes the agent's role and capabilities
system_prompt: |
  You are the Scientist Agent in the Minecraft framework. Your role is to discover
  fundamental environment laws through controlled experimentation in Minecraft.

  Core Principle: **NEVER GUESS**. All laws must be validated through reproducible experiments
  with minimum support count >= 3.

question: |
  {question_from_engineer}

# Main task prompt with placeholders
experiment_template: |
  {experiment_template}

knowledge_book: |
  {knowledge_book}
```

E.2 ENGINEER AGENT PROMPT TEMPLATE (YAML)

```
# Engineer Agent Prompt Template
# Purpose: Generate construction plans for functional devices in Minecraft
# Based on: SciCraft paper Section 4.2 (Engineer Agent)

agent_name: Engineer Agent
agent_role: Device Construction and Task Execution
version: 1.0

# System prompt - describes the agent's role and capabilities
```

[3] https://modelcontextprotocol.io

```
system_prompt: |
  You are the Engineer Agent in the  Minecraft framework. Your role is to construct
  functional redstone devices in Minecraft that satisfy precise I/O specifications.

  You must NEVER guess about environment laws. When uncertain, delegate to the Scientist
     Agent.

# Main task prompt with placeholders
task_prompt_template: |
  # TASK SPECIFICATION

  {task_specification}

  ---

  # AVAILABLE COMPONENTS

  You may use the following Minecraft blocks:
  {available_blocks}

  ---

  # Knowledge Book

  The following laws have been discovered and validated through experiments:

  {knowledge_book}
```

E.3 GAME MECHANICS EXPLORATION EXPERIMENT TEMPLATE (MARKDOWN)

```
# Game Mechanics Exploration Experiment Template

## Experiment Info
**Experiment #**: ________
**Date**: ________
**Experimenter**: ________

---

## 1. Research Question
What do I want to figure out?

_[e.g., Under what conditions will a device activate? How do two components interact?]_

---

## 2. My Hypothesis
Based on my experience, what do I think will happen?

_[Write down your prediction]_

---

## 3. Experiment Design

### Variables to Test
**What I will change** (Independent Variable): ________

**What I will observe** (Dependent Variable): ________

**What needs to stay constant** (Control Variables):
- ________
- ________
- ________

### Control Setup
- **Control Group**: ________ (baseline test without changes)
- **Experimental Group**: ________ (test with changes)

---

## 4. Experiment Steps

### Preparation
1. ________
2. ________
3. ________
```

```
### Testing Process
**Step 1**: _________
-> Observation: _________

**Step 2**: _________
-> Observation: _________

**Step 3**: _________
-> Observation: _________

(Add more steps as needed)

---

## 5. Experiment Record

### Data Recording Table

| Trial # | Changed Condition | Observed Result | Matches Prediction? | Notes |
|---------|-------------------|-----------------|---------------------|-------|
| 1       |                   |                 | [ ] Yes [ ] No      |       |
| 2       |                   |                 | [ ] Yes [ ] No      |       |
| 3       |                   |                 | [ ] Yes [ ] No      |       |
| 4       |                   |                 | [ ] Yes [ ] No      |       |
| 5       |                   |                 | [ ] Yes [ ] No      |       |

### Detailed Observations
_[Record any interesting phenomena, unexpected situations, details]_


### Screenshots/Diagrams
[Record diagrams of key configurations or phenomena here]

---

## 6. Experiment Results

### What did I discover?
_[Objectively describe the observed phenomena]_

### Was my hypothesis correct?
[ ] Completely correct
[ ] Partially correct
[ ] Completely incorrect

**Explanation**: _________

---

## 7. Analysis & Summary

### Why did this happen?
_[Try to explain the underlying mechanism or pattern]_

### What's the practical application?
_[Real-world use cases]_

### What's still uncertain?
_[List questions or points that need further testing]_

---

## 8. Next Steps
Based on this experiment, I want to test next:

1. _________
2. _________
3. _________

---

## Quick Checklist
- [ ] Research question is clear
- [ ] Only changing one variable at a time
- [ ] Set up control group
- [ ] Recorded all observations
- [ ] Repeated test at least 3 times
- [ ] Documented unexpected situations
- [ ] Summarized patterns or conclusions

---
```

```
**Experiment Notes** (Free recording area):

_[Any additional thoughts, ideas, questions]_
```

F KNOWLEDGE BOOK EVOLUTION

This appendix documents the evolution of the knowledge consolidation format used by the Scientist Agent to distill discoveries into a shared Knowledge Book. We present three formats, each with its generation prompt and an example output.

F.1 SELF-DETERMINED SUMMARY

This baseline asks the model to summarize the experiment findings without enforcing a fixed template.

F.1.1 GENERATION PROMPT

```
Summarize the discovered knowledge from these experiments.
```

F.1.2 EXAMPLE OUTPUT

```
Key discoveries (self-determined summary):

- Redstone dust transmits a signal strength in [0,15]. Starting from 15,
  it decays by 1 per dust block of path length; repeaters refresh the
  signal back to 15.
- Redstone dust connects only to the four cardinal neighbors (N/S/E/W);
  diagonal dust does not connect directly. Many layouts must use
  Manhattan routing.
- Repeaters introduce delay (1-4 redstone ticks) and can be chained; total
  delay is the sum of repeater delays along the path.
- In branching networks, signal strength into a T-junction is shared to
  all outgoing dust. To maximize reach, boost before a split; to control
  branch timing independently, boost after the split.
- For simultaneous activation across unequal path lengths, the longest
  path sets the minimum achievable latency; add compensating delay to
  shorter paths to synchronize edges.
- Solid blocks can be powered in ways that activate adjacent mechanisms
  but do not always propagate to adjacent dust; repeaters/comparators are
  reliable for transmitting power through blocks.
- To generate short, precise pulses from longer inputs (e.g., button
  press), normalize to an edge-triggered pulse and then extend/shaping
  with a small delay network.
```

F.2 FINDING-EXPLANATION-EXAMPLE FORMAT

This format structures each discovery into a reusable finding, an explanation with constraints, and a minimal example.

F.2.1 GENERATION PROMPT

```
For each distinct discovery or significant conclusion, generate a knowledge book entry
     strictly following this format:

### Finding
One sentence stating the rule/behavior as a crisp claim.

### Explanation
Explain how/why the finding works and when it applies.
- Include quantitative details, constraints, edge cases, and failure modes.

### Example
Provide one minimal concrete example that demonstrates correct usage (layout, coordinates,
     pseudo-circuit, or scenario).

**Instructions:**
1. **Distill:** Do not just copy text. Extract the core truth from experiences.
```

```
2. **Structure:** Maintain the Finding/Explanation/Example structure for *every* entry.
3. **Coverage:** Ensure all technical details needed for reuse are captured.
4. **Clarity:** Use clear, professional technical language.
```

F.2.2 EXAMPLE OUTPUT

```
### Finding
Diagonal placement allows for compact star topologies.

### Explanation
Redstone dust strictly connects to the four cardinal neighbors (North, South, East, West). It
     does not connect diagonally.
*   **Observation:** Placing dust at `(x, z)` and `(x+1, z+1)` results in two isolated piles
     of dust.
*   **Correction:** Connections must be made via "Manhattan" or "Taxicab" paths (e.g., `(x,
     z) -> (x+1, z) -> (x+1, z+1)`).
*   **Impact:** A visual "Star" with diagonal lines is impossible. Circuits must use
     "Manhattan Star" shapes (cross shapes), which consume more space and can introduce path
     length differences if not strictly symmetrical.

---

### Finding
Signal strength decay limits branch reach; repeaters are required for long-distance
     transmission and must be strategically placed at T-junctions.

### Explanation
Redstone signal strength ($SS$) starts at 15 and decays by 1 per block. $SS_{next} =
     SS_{current} - 1$.
*   **Junction Mechanics:** A signal splitting at a T-junction provides the *same* signal
     strength to all branches. If a wire arrives at a junction with strength 1, it can power
     adjacent dust in 3 directions, but those new dust blocks will have strength 0 (signal
     dies).
*   **Repeater Placement:** To maximize reach after a junction, place a repeater
     *immediately* after the split on each branch, or *immediately* before the split on the
     trunk.
    *   *Before Split:* 1 Repeater powers 3 branches. Efficient, but all branches share the
     delay.
    *   *After Split:* Requires 3 Repeaters. Allows individual delay control per branch.

### Example
**Trunk with Pre-Junction Boost**
Maximizes signal strength entering the junction.
```text
Place components at the following coordinates:
Source: vec3(0, 64, 0)
Wire segment: vec3(0, 64, 1) to vec3(0, 64, 14)
Repeater: vec3(0, 64, 15)
Junction: vec3(0, 64, 16)
Branches: vec3(-1, 64, 16), vec3(1, 64, 16)
Lamps: vec3(-1, 64, 17), vec3(0, 64, 17), vec3(1, 64, 17)
```

Multi-Level T-Branching
For reaching off-axis lamps at varying distances.
```text
Layout:
vec3(10, 5, 10): Trunk Wire
vec3(11, 5, 10): Junction
vec3(11, 5, 9): Branch A Start -> vec3(11, 5, 8): L1
vec3(12, 5, 10): Repeater
vec3(13, 5, 10) to vec3(20, 5, 10): Branch B Wire -> vec3(21, 5, 10): L2
```

Finding
Sequential activation is achieved by chaining repeaters with specific delay settings,
     effectively creating a "Timeline" of events.

Explanation
Redstone Repeaters have configurable delays: 1, 2, 3, or 4 redstone ticks.
*   **Cumulative Delay:** Delays in series add up. $T_{total} = \sum \delta_{repeaters}$.
*   **Parallel Stages:** To trigger events at absolute times $t=0, t=2, t=6$:
    *   Lamp 1: Direct connection ($t=0$).
    *   Lamp 2: Branch -> R(2) -> Lamp ($t=2$).
    *   Lamp 3: Branch -> R(2) -> R(4) -> Lamp ($t=6$).
```

```
*   **Non-Uniform Sequences:** For patterns like [2, 1, 3], you cannot just rely on a single
    chain. You may need independent delay lines branching from the source if the timing
    isn't strictly cumulative or if geometry prevents chaining.

### Example
**Sequential Chain (Cumulative)**
Target: L1(t=0), L2(t=2), L3(t=6)
'''text
Nodes:
n0: vec3(0,0,0) [Source]
n1: vec3(0,0,1) [Wire] -> vec3(1,0,1) [L1]
n2: vec3(0,0,2) [Repeater, delay=2]
n3: vec3(0,0,3) [Wire] -> vec3(1,0,3) [L2]
n4: vec3(0,0,4) [Repeater, delay=4]
n5: vec3(0,0,5) [L3]
'''

**Parallel Delay Lines (Independent)**
Target: L1(t=2), L2(t=4) (Spatially separated)
'''text
vec3(50, 60, 50): Source
vec3(50, 60, 51): Wire Junction
Path 1: vec3(49, 60, 51) [R(2)] -> vec3(48, 60, 51) [L1]
Path 2: vec3(51, 60, 51) [R(4)] -> vec3(52, 60, 51) [L2]
'''

---

### Finding
Delay compensation synchronizes outputs across unequal path lengths by "slowing down the fast
    paths."

### Explanation
When paths differ in length, the longest path (requiring the most repeaters for signal
    strength) sets the minimum system latency ($Latency_{min}$).
*   **The Algorithm:**
    1.  Calculate $Delay_{inherent}$ for every path (ticks from mandatory repeaters needed
    for distance).
    2.  Find $Max(Delay_{inherent})$.
    3.  For every other path $i$, add compensation repeaters: $\delta_{add} = Max(Delay) -
    Delay_i$.
*   **Slack:** Sometimes you intentionally increase the delay of *all* paths to a higher
    common multiple to make the math easier (e.g., synchronize everything to 10 ticks).

### Example
**Equal-Delay Distribution Logic**
*   Path A (20 blocks): Needs 1 Repeater (min 1 tick). Delay=1.
*   Path B (5 blocks): Needs 0 Repeaters. Delay=0.
*   Path C (35 blocks): Needs 2 Repeaters (min 2 ticks). Delay=2.

*Target Delay:* 2 ticks (dictated by Path C).
*Adjustments:*
*   Path A: Add 1 tick (Total 2).
*   Path B: Add 2 ticks (Total 2).
*   Path C: Add 0 ticks (Total 2).

'''text
Network Graph:
Edge(vec3(0,0,0), vec3(10,0,0)) -> Contains R(1) + R(1) -> L3
Edge(vec3(0,0,0), vec3(0,0,10)) -> Contains R(1) + R(1) -> L1
Edge(vec3(0,0,0), vec3(5,0,5))  -> Contains R(2) -> L2
'''

---

### Finding
Monostable circuits and Pulse Limiters are required to generate precise pulse durations
    shorter than the input source.

### Explanation
A stone button generates a long pulse (~10-15 redstone ticks). To control exact duration
    $\tau$:
1.  **Input Normalization:** First, convert the button press into a standardized 1-tick pulse
    using a **Rising Edge Detector**.
2.  **Pulse Shaping:** Extend that 1-tick pulse to exactly $\tau$ ticks.
    *   *Small $\tau$ (1-4):* Use a repeater set to $\tau$ merging with the original signal?
    No, simpler: The 1-tick pulse powers a repeater chain that "holds" the line.
    *   *Medium $\tau$ (4-10):* Use a **Pulse Extender**. A parallel bank of repeaters is
    precise.
    *   *Analog Method:* A Comparator Decay Loop extends pulses but is harder to tune to
    exact ticks.
```

````
### Example
**Precise Pulse Generator (Target $\tau$ = 4 ticks)**

**Step 1: Monostable (Rising Edge)**
Generates a 1-tick pulse regardless of button hold time.
```text
Circuit Topology:
vec3(x,y,z) [Button] connects to:
  1. vec3(x+1,y,z) [Torch] -> vec3(x+2,y,z) [Wire]
  2. vec3(x,y,z+1) [Repeater-2] -> vec3(x+1,y,z+1) [Wire] -> vec3(x+2,y,z+1) [Connect to 1]
  Output at vec3(x+3,y,z) through Inverter.
```

**Step 2: Pulse Extension Bank (The "Timeline" method)**
To output exactly 4 ticks from a 1-tick trigger:
Input splits into 4 parallel lines of delay 1, 2, 3, 4, all merging into Output.
```text
Parallel Array:
Input: vec3(0,0,0)
Branches:
  vec3(1,0,0): R(1)
  vec3(1,0,1): R(2)
  vec3(1,0,2): R(3)
  vec3(1,0,3): R(4)
Merge: vec3(2,0,0..3) connected to Output Line vec3(3,0,0).
```

---

### Finding
Spatial management and component isolation are critical for high-density wiring.

### Explanation
As N increases, wires inevitably cross or run parallel.
*   **Interference:** Two lines of redstone dust placed side-by-side will connect.
*   **Isolation Techniques:**
    1.  **Block Cutting:** Place a solid block between parallel wires.
    2.  **Repeater Tunneling:** Use repeaters to push signal *through* a block, allowing a perpendicular wire to run on top of that block without connecting.
    3.  **Vertical Stacking:** Run one bus line at Y=64 and another at Y=66.
*   **Slabs/Glowstone:** Use transparent blocks to run wire vertically up without cutting the signal.

### Example
**High-Density Bus Routing**
Running 3 parallel signals in a 3-wide space:
```text
Grid Configuration:
Column 0: vec3(0, y, z) -> Signal 1
Column 1: vec3(1, y, z) -> Insulator
Column 2: vec3(2, y, z) -> Signal 2
Column 3: vec3(3, y, z) -> Insulator
Column 4: vec3(4, y, z) -> Signal 3
```

**Crossing Wires (The Bridge)**
Signal A needs to cross Signal B.
```text
Layered Intersection:
Signal A Path: vec3(10, 64, 10) -> vec3(10, 65, 11) [Block Top] -> vec3(10, 64, 12)
Signal B Path: vec3(9, 64, 11) -> vec3(10, 64, 11) [Tunnel] -> vec3(11, 64, 11)
```

---

### Finding
Debug procedure for timing mismatches involves "Tick Counting" and "Edge Observation."

### Explanation
When a circuit fails simultaneity or sequence tests:
1.  **Tick Counting:** Manually trace the path from Source to Lamp, summing the delays of every repeater. Remember: Dust = 0, Torch = 1, Comparator = 1, Repeater = Configured (1-4).
2.  **Edge Observation:** Watch the *activation* (Rising Edge). Sometimes lamps turn ON simultaneously but turn OFF at different times. The contract usually specifies activation time $|t_i - t_j|$.
3.  **Ghost Power:** Ensure blocks aren't being "quasi-powered" or powered indirectly by adjacent strong-powered blocks, which can bypass intended delays.

### Example
````

```
**Debugging Table Construction**
```text
Log Data:
Target: vec3(10,10,10) [L1], Delta=6, Actual=6 (OK)
Target: vec3(20,10,10) [L2], Delta=6, Actual=7 (FAIL) -> Components at vec3(25,10,10) [R1],
     vec3(26,10,10) [R4], vec3(27,10,10) [R2]
```

```
```

F.3 CLAIM-PROOF-CONSTRAINTS-EXAMPLE FORMAT

The final format enforces scientific rigor by recording a law-like claim, evidence-based proof, validity constraints, and a concrete application example.

F.3.1 GENERATION PROMPT

```
For each distinct discovery or significant conclusion, generate an entry in the knowledge
     book strictly following this structure:

### 1. Claim (Law)
State the finding as a universal, law-like statement (similar to a mathematical theorem or
     physical law). It must be concise, precise, and assertive.
*   *Bad:* "Redstone gets weaker as it goes far."
*   *Good:* "The Law of Linear Signal Decay: Redstone signal strength $S$ decreases by
     exactly 1 unit per block of distance $d$, such that $S_{d} = S_{initial} - d$."

### 2. Proof
Provide the rigorous logic that supports this claim. This acts as the "proof."
*   Synthesize specific evidence from the experiments.
*   Connect prior established findings with new data.
*   Outline the deductive steps that lead inevitably to the claim.
*   Maintain a high standard of rigor; mere observation is not proof--explain *why* the
     observation confirms the law.

### 3. Constraints
Identify potential weaknesses or exceptions in the proof.
*   Are there edge cases where the law fails? (e.g., "Does not apply across chunk boundaries
     in specific versions.")
*   Are there assumptions made that require further validation?
*   Note any observed anomalies that contradict the main claim.

### 4. Example
Demonstrate the utility of this finding.
*   Provide a concrete example or usage scenario.
*   Describe a mechanism or design pattern that relies specifically on this law being true.
```

F.3.2 EXAMPLE OUTPUT

```
### 1. The Law of Linear Signal Attenuation

#### Claim
Redstone signal strength $S$ decays linearly by exactly 1 unit for every block of distance
     traversed $d$, governed by the equation $S_{d} = S_{source} - d$. The maximum
     transmission range without amplification is 15 blocks, occurring when $S_{source} = 15$.

#### Proof
*   **Observation:** In all experiments, placing a redstone wire adjacent to a power source
     (Strength 15) results in a strength of 14 on the wire.
*   **Deduction:** The signal terminates (Strength 0) exactly at the 16th block.
*   **Geometry:** Distance is calculated using "Taxicab" or "Manhattan" geometry ($\Delta x +
     \Delta z$), not Euclidean distance. A wire spiraling or changing elevation consumes
     signal strength identical to a straight line of the same block count.

#### Constraints
*   **Constraint:** This law applies strictly to Redstone Dust. Solid blocks powered by
     repeaters or comparators do not degrade the signal *within* the component itself, but
     the dust exiting them restarts the decay.
*   **Edge Case:** Signal strength does not decay when passing through a Comparator in
     comparison mode (it maintains the input strength).

#### Example
*   **Trunk with Pre-Junction Boost:** Place a repeater *immediately* before a trunk line
     splits into multiple branches. This ensures all branches receive maximum signal strength
     (15) and resets the decay counter for the new paths.
```
```

```
    ```text
    B (SS=15)
    |
    [14 blocks of wire]
    |
    R (Repeater, Boosts to SS=15)
    |
    +-------+-------+
    |       |       |
    W(15)   W(15)   W(15) <-- All branches start fresh
    |       |       |
    L1      L2      L3
    ```
*   **Multi-Level T-Branching:** When branching to off-axis lamps at varying distances,
     independent repeaters are required *after* the split to ensure the longer branch
     receives sufficient power without affecting the timing of the shorter branch.
    ```text
    Trunk --+-- Branch A (Short) -- L1
            |
            R (Repeater needed to reach Branch B)
            |
            +-- Branch B (Long) -- L2
    ```
*   **Signal Strength Reader:** A compact circuit using comparators to output a different
     signal strength based on the input signal, allowing for more granular information
     transfer than simple on/off.
    ```text
    Input (SS=0-15) --> Comparator (Subtract Mode) --> Output (SS varies based on input)
    ```

---

2. The Principle of Quasi-Connectivity (Action at a Distance)

Claim
Certain activation components (Pistons, Dispensers, Droppers) behave as if they are two
     blocks high for the purpose of receiving power. A component at (x, y, z) will activate
     if a signal would validly power a theoretical component at $(x, y+1, z)$, even if the
     space at $y+1$ is empty or air.

Proof
*   **Experiment:** A piston placed at y activates when a redstone block is placed at
     $y+2$, provided there is no solid block blocking the "sight" (though air transmits it).
*   **Logic:** The game code checks for power in the block *above* the component to support
     door mechanics, but this check was generalized, resulting in "BUD" (Block Update
     Detector) behavior where the component receives power but doesn't realize it until a
     neighbor updates.

Constraints
*   **Constraint:** This is strictly a **Java Edition** mechanic. It does not function in
     Bedrock Edition.
*   **Anomaly:** The component often requires a "block update" (neighbor changing state) to
     realize it has been quasi-powered or quasi-depowered. This leads to "budding" where a
     piston stays extended even after power is cut.

Example
*   **Quasi-Connectivity Visualization:** Powering the space *above* a piston activates it,
     even with an air gap.
    ```text
    [Redstone Block]  <-- Power Source
          |
        (Air)         <-- "Invisible" connection range
          |
    [Piston/Dropper]  <-- Activates via QC
    ```
*   **Compact Piston Walls:** Powering a wall of pistons from the ceiling to avoid messy
     floor wiring.
    ```text
    [Redstone Dust Line]
    [Solid Block]
    [Piston] [Piston] [Piston] <-- All activate from blocks above
    ```
*   **BUD Switch (Block Update Detector):** A piston or dropper configured to be
     quasi-powered. It only extends/activates when a block *adjacent* to it is updated, even
     if the power source itself hasn't changed. This is used for detecting block changes or
     growth.
    ```text
          (QC Power Source - e.g., Redstone Block 2 blocks above)
          |
    [Piston]  <-- Quasi-powered, but not updated
    [Any Block] <-- When this block updates (e.g., placed/broken), Piston extends
```
```

```
    ```

---

3. The Theorem of Orthogonal Transmission (Manhattan Connectivity)

Claim
Redstone dust connects exclusively to the four cardinal neighbors (North, South, East, West).
     Diagonal transmission is impossible directly; all diagonal signal propagation must
     approximate the hypotenuse via a stepped "Manhattan" path, where $Distance_{steps} =
     |\Delta x| + |\Delta z|$.

Proof
*   **Observation:** Placing dust at (x, z) and $(x+1, z+1)$ results in two isolated
     components.
*   **Logic:** To connect them, dust must be placed at either $(x+1, z)$ or $(x, z+1)$.
*   **Consequence:** A visual "star" topology is topologically impossible. All branching must
     occur at orthogonal T-junctions.

Constraints
*   **Exception:** Vertical transmission via "soft" transparent blocks (slabs, glowstone)
     allows a signal to travel diagonally *upward* in a zigzag pattern without connecting to
     adjacent horizontal wires, effectively creating a diode.

Example
*   **High-Density Bus Routing:** Parallel wires must be separated by blocks (e.g.,
     `Wire-Block-Wire`) to prevent interference.
    ```text
    Top View:
    W1  B   W2  B   W3
    |   |   |   |   |
    W1  B   W2  B   W3
    ```
*   **Crossing Wires (The Bridge):** To cross Signal A over Signal B, A must be routed over a
     block while B passes underneath. This "Bridge" uses the vertical orthogonality to
     separate planar signals.
    ```text
          A
          |
          W
          |
    B -- [Block] -- B  <-- Signal A goes over Signal B
          |
          W (Signal B runs under the block)
          |
          A
    ```
*   **Compact Corner Turns:** Instead of a wide curve, redstone dust can turn 90 degrees
     using a single corner block, adhering to the orthogonal connection rule. This is
     fundamental for compact layouts.
    ```text
    W -- W
         |
         W
    ```

---

4. The Law of Solid Block Conductivity (Hard vs. Soft Power)

Claim
A solid (opaque) block receiving a redstone signal becomes "weakly powered," capable of
     activating adjacent mechanism components (lamps, pistons) but *not* adjacent redstone
     dust. To power adjacent dust, the block must be "strongly powered" by a repeater,
     comparator, or torch.

Proof
*   **Experiment:** Wire pointing into a Stone block activates a lamp touching the Stone, but
     does not activate a wire continuing from the other side of the Stone.
*   **Correction:** Replacing the input wire with a Repeater pointing into the Stone
     activates the output wire.

Constraints
*   **Transparency Exception:** Transparent blocks (Glass, Slabs, Glowstone) *cannot* be
     powered at all. They are insulators. Wire placed on them transmits signal, but the block
     itself does not hold a charge.

Example
*   **Strong vs. Weak Power:** Differentiating how signal propagates through blocks.
    ```text
    (Weak Power - Fails to continue)
```

```
    Wire --> [Block] -/-> Wire (OFF)

    (Strong Power - Successfully transmits)
    Repeater --> [Block] ---> Wire (ON)
    ```
*   **Repeater Tunneling:** Using a repeater to push signal *into* a block allows a
     perpendicular wire to run *on top* of that block without connecting to the signal below.
    ```text
          W (Wire on top)
          |
    R --> [Block] --> R (Signal passes THROUGH block)
          |
          W (Wire on top)
    ```
*   **Redstone Lamp Array:** Activating multiple redstone lamps in a compact space by
     strongly powering a row of solid blocks, each adjacent to a lamp.
    ```text
    R (Repeater) --> [Block] --> [Block] --> [Block]
                      |           |           |
                    [Lamp]      [Lamp]      [Lamp]
    ```

---

5. The Principle of Instantaneous Translocation (Instant Wire)

Claim
The movement of a block by a piston is processed as an instantaneous event ($t=0$ ticks)
     regarding the connectivity of attached components. This allows for "Instant Wire" where
     a signal travels infinite distance in the same game tick.

Proof
*   **Mechanism:** When a piston extends, it pushes a block. The game engine updates the new
     position of the block and its neighbors *within the same tick* logic cycle.
*   **Setup:** A chain of pistons, where Piston A pushing Block A completes a circuit for
     Piston B, will ripple through the entire chain before the next tick is processed.

Constraints
*   **Constraint:** Requires precise sub-tick update order (update suppression or specific
     block placement order) to function reliably in all directions.
*   **Cost:** High noise and potential for client-side visual desync (ghost blocks).

Example
*   **Piston Instant Wire Chain:** A mechanical cascade where moving blocks complete circuits
     instantly.
    ```text
    In -> [Piston] -> [Block A]
                        |
       (Block A moves here to complete circuit)
                        |
                  [Redstone Dust] -> [Piston] -> [Block B] -> Out
    ```
*   **Long-Distance Interrupt:** Transmitting a "Stop" signal to a farm 500 blocks away
     instantly.
*   **T-Flip Flop (Instant Set/Reset):** Using two sticky pistons pushing a redstone block
     back and forth to create a compact T-Flip Flop where a single pulse instantly toggles
     the output state, leveraging instantaneous block updates.
    ```text
    In -> Piston (Sticky) -> [Redstone Block] <- Piston (Sticky) <- In
                                    |
                                  Output
    ```

---

6. The Law of Discrete Quantized Time (Tick Logic)

Claim
Time in Redstone circuits is quantized into discrete "Redstone Ticks" (equivalent to 0.1
     seconds). Events cannot occur "between" ticks. Total latency in a serial circuit is the
     arithmetic sum of all individual component delays: $T_{total} = \sum
     \delta_{components}$.

Proof
*   **Mechanics:** Repeaters have configurable delays of 1, 2, 3, or 4 ticks.
*   **Observation:** A chain of two repeaters set to 2 and 4 results in an exact 6-tick delay.
*   **Logic:** Parallel paths with unequal tick sums arrive at different times. The "slowest"
     path determines the minimum latency for a synchronized system.

Constraints
```
```

````
*   **Sub-Tick Events:** As noted in Law 5, piston events can resolve within a single tick,
     effectively having $\delta=0$.
*   **Pulse Chopping:** If an input pulse is shorter than a repeater's delay setting (e.g.,
     1-tick pulse into 4-tick repeater), the repeater effectively "extends" or stabilizes the
     pulse to its minimum duration in some versions, or ignores it in others.

#### Example
*   **Sequential Activation:** Creating "Waves" of activation (e.g., runway lights) by
     chaining repeaters to trigger lamps at $t, t+2, t+4$.
    ```text
    B --+-- L1 (t=0)
        |
        R(2)
        |
        +-- L2 (t=2)
        |
        R(4)
        |
        L3 (t=6)
    ```
*   **Parallel Delay Lines:** Independent delay lines can be used when spatial separation
     prevents a single chain.
    ```text
          +-- R(2) -- L1
          |
    B -- -+
          |
          +-- R(4) -- L2
    ```
*   **Equal-Delay Distribution Logic:** Synchronizing a "Fast" path with a "Slow" path by
     adding filler repeaters.
    ```text
    Path C: B -- R(1) -- W -- R(1) -- W -- L3 (Native 2 ticks)
    Path A: B -- R(1) -- W -- R(1) -- W -- L1 (1 tick for distance + 1 tick compensation)
    Path B: B -- R(2) -- W -- L2              (2 ticks purely for compensation)
    ```
*   **Pulse Shaping:** Using Monostable circuits (Rising Edge Detectors) to convert
     variable-length button presses into precise 1-tick pulses.
    ```text
          +-- T (Torch) --+
          |               |
    B -- -+-- R(2) -------+-- NOT -- (Output)
    ```
````