REASONING IN LIFE: CAN LANGUAGE AGENTS MASTER EVOLVING WORLDS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning is the core capability for large language models (LLMs) to solve complex tasks such as planning and control. Such reasoning tasks can be categorized into three forms: deduction, which predicts the future of a system from its rule and current state; induction, which recovers the hidden rule from observed transitions; and abduction, which reconstructs the antecedent that best explains an observed outcome. Existing benchmarks, however, either consider a single form of reasoning or lack the formal grounding across the three forms, leaving no principled way to measure the divergence of language agents over different reasoning forms. To narrow the gap, we introduce LIFEWORLD, a benchmark built on life-like cellular automata with 2^{18} rules generalizing the famous Conway's Game of Life (GoL), where the evolving step can be characterized by the rule P, a configuration s_t, and its k-step successor s_{t+k}. We curate the dataset for three forms of reasoning: i) deduction ($s_t + P \rightarrow s_{t+k}$) is forward simulation with a unique answer, computable in polynomial time yet Turing complete in expressive power; ii) induction ($s_t + s_{t+k} \rightarrow P$) is rule identification from observed transitions, corresponding to program induction from input-output examples; and iii) abduction ($P + s_{t+k} \rightarrow s_t$) is predecessor reconstruction under irreversible dynamics, which is NP-complete at bounded horizon. We further propose LIFEAGENT, a recursive reasoning harness that operates through two forms of recursion: i) space-time recursion, which breaks a query into sub-grids and sub-horizons, and ii) logical recursion, which solves induction and abduction by a propose-then-verify loop in which candidates are checked through deduction. Extensive experiments across frontier LLMs demonstrate the effectiveness of both recursions in LIFEAGENT and provide a comprehensive analysis of performance across the three reasoning forms, characterizing the behaviors of frontier LLMs.

1 INTRODUCTION

Reasoning is the capability that allows large language models (LLMs) and the agents built on them to move beyond pattern completion toward planning, control, and scientific discovery. Following (Peirce, 1934), reasoning takes three forms. *Deduction* derives what must follow from a rule and a case, *induction* generalizes a rule from observed cases, and *abduction* conjectures the case that best explains an observation under a known rule. The three forms make different demands on a reasoner: deduction requires faithful execution, induction requires search over hypotheses that the data never fully determine, and abduction requires inverting a process whose inverse may be ambiguous or empty. Recent debates about the limits of LLMs are increasingly framed along these lines. As argued in (Zahavy, 2026), current systems have largely mastered induction and are making rapid progress on deduction, yet remain unable to make the abductive leap that produces new explanatory hypotheses, and strong deductive performance does not carry over to abduction (Salimi et al., 2026).

Existing benchmarks usually isolate specific reasoning forms. Rule and function induction is considered by InductionBench (Hua et al., 2025), CellARC (Ložičař, 2025), and the ARC-AGI series (Chollet, 2019; Chollet et al., 2025; ARC Prize Foundation, 2026); deduction by execution prediction (Gu et al., 2024) and synthetic logic corpora (Morishita et al., 2024); and abduction by commonsense explanation tasks (Bhagavatula et al., 2020). Benchmarks that cover several forms (He et al., 2025; Sheng et al., 2025; Sun & Saparov, 2025) pose them over natural-language premises or hand-built ontologies, where each form is instantiated on different content, the correctness of an ab-

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

Table 1: Comparison of LIFEWORLD with existing reasoning benchmarks. *Ded./Ind./Abd.*: whether deduction, induction, and abduction are posed as evaluated tasks. *Shared inst.*: whether different forms are evaluated on the same underlying object, so that performance gaps can be attributed to the direction of inference. *Exact verif.*: whether correctness is decided by a program rather than by reference overlap or model judgment. *Formal ground.*: whether task difficulty is characterized by a complexity-theoretic result. *Scal. diff.*: whether instances can be generated without limit under explicit difficulty parameters. ✓: yes; ○: partial; ✗: no; –: not applicable (single form).

		Reasoning form			Evaluation property			
Benchmark	Substrate	Ded.	Ind.	Abd.	Shared inst.	Exact verif.	Formal ground.	Scal. diff.
ARC-AGI-1/2 (Chollet et al., 2025)	2D grids	✗	✓	✗	–	✓	✗	✗
CellARC (Ložičař, 2025)	1D CA	✗	✓	✗	–	✓	✗	✓
InductionBench (Hua et al., 2025)	Strings	✗	✓	✗	–	✓	✓	✓
CRUXEval (Gu et al., 2024)	Code	✓	✗	✓	✓	✓	✗	✗
ART (Bhagavatula et al., 2020)	Narratives	✗	✗	✓	–	○	✗	✗
IDEA (He et al., 2025)	Puzzles	○	✓	○	○	✓	✗	✗
InAbHyD (Sun & Saparov, 2025)	FOL	✗	✓	✓	✗	✓	✗	✓
LIFEWORLD	2D LLCA	✓	✓	✓	✓	✓	✓	✓

ductive answer often rests on reference matching or model-based judgment, and difficulty is not tied to a formal notion of complexity. In ARC-style tasks the transformation is hand-designed and never written down, so only the inductive direction can be posed with a checkable answer. Consequently, when a model does worse on abduction than on deduction, one cannot tell whether the gap reflects the form of reasoning or incidental differences in domain, format, and grading. What is missing is a single system in which the three forms are instantiated over the same objects, every answer can be verified exactly, and the difficulty of each form is characterized by known complexity results.

We introduce LIFEWORLD, a benchmark built on life-like cellular automata (LLCA), the 2^{18} outer-totalistic rules on the Moore neighborhood that generalize Conway's Game of Life (GoL) (Conway, 1970; Gardner, 1970). An LLCA world couples three objects: a rule $P = (B, S)$, a configuration s_t, and its k-step successor s_{t+k}. Fixing any two and solving for the third recovers Peirce's trichotomy exactly (Figure 1). Deduction ($P + s_t \to s_{t+k}$) is forward simulation, induction ($s_t + s_{t+k} \to P$) is rule identification from observed transitions, and abduction ($P + s_{t+k} \to s_t$) is predecessor reconstruction. Because the three tasks are derived from the same transition, differences in performance can be attributed to the direction of inference rather than to the content. The substrate also grounds each direction formally. Deduction has a unique answer computable in $\mathcal{O}(kn^2)$ time on an $n \times n$ grid, yet GoL is Turing universal (Berlekamp et al., 2004; Rendell, 2002), so in general no shortcut replaces step-by-step simulation (Wolfram & Gad-el Hak, 2003). Induction can be read off in a single pass when $k = 1$ but becomes a search over the rule space once intermediate states are hidden ($k \geq 2$). Abduction inverts an irreversible, many-to-one map: a configuration may have many predecessors or none at all (a Garden of Eden), and deciding whether a predecessor exists is NP-complete (Sutner, 1995). Since LIFEWORLD operates on finite grids, all three problems are decidable, and difficulty is measured by complexity rather than computability; undecidability arises only for global questions on the infinite plane (Kari, 1994). Finally, a simulator checks any candidate answer exactly, and the grid size, horizon, and rule give independent control over difficulty, with the rule family ranging from ordered to chaotic regimes (Langton, 1990). Table 1 compares LIFEWORLD against existing benchmarks. Prior benchmarks either target a single form of reasoning or several forms without a shared instance, an exact verifier, or a formal characterization of difficulty.

Furthermore, we propose LIFEAGENT, a recursive reasoning harness whose design follows from two structural properties of the task. The first is *space-time recursion*. A k-step evolution is a composition of single steps, and each step is local, so a deduction query factors exactly into sub-horizons and overlapping spatial windows whose answers can be pasted together without error. The second is *logical recursion*. Since any two of (P, s_t, s_{t+k}) determine the third, a candidate answer to an induction or abduction query can be checked by re-deriving s_{t+k}, which turns both modes into a propose-then-verify loop in which every check is itself a deduction. The two recursions compose: the check produced by logical recursion is exactly the forward run that space–time recursion decomposes, so gains on deduction carry over to verification in the two harder modes. LIFEAGENT also separates what a checker returns from what it may conclude. Only the exact simulator can accept or reject a candidate, whereas a check performed by the model returns the set of disagreeing cells as a counterexample for the next proposal, and the answer is labeled as verified, conjecture, or fail.

We evaluate three frontier LLMs (DeepSeek-V4.1-Flash, Gemini-3.8-Flash and GPT-5.6-Luna) on LIFEWORLD and find that abduction is by far the hardest form, reaching only 0.18 on hard queries against 0.42 for deduction and 0.46 for induction. Both recursions of LIFEAGENT yield substantial gains: space–time recursion improves deduction in all nine settings, by 0.12 to 0.30, and logical recursion nearly doubles induction accuracy (0.31 to 0.58), well beyond what resampling achieves. The two recursions also compose: tiling the forward run inside the check lets a checker built only from the model come within 0.02 of the exact simulator. In summary, our contributions are:

- We introduce LIFEWORLD, a benchmark that instantiates deduction, induction, and abduction over the same life-like transitions, with exact verification, complexity-theoretic grounding for each form, and independent control over grid size, horizon, and rule. LIFEWORLD is, to our knowledge, the first benchmark that satisfies all of these properties at once: the three forms are derived from the same transition, every answer is checked by simulation, including the non-unique answers of induction and abduction, and the difficulty is tied to a known complexity result.
- We propose LIFEAGENT, a recursive reasoning harness that combines space–time recursion for exact decomposition of forward simulation with logical recursion for propose-then-verify inference, and that labels each answer by how it was obtained.
- We conduct an extensive evaluation of frontier LLMs, characterizing how their behavior diverges across the three forms of reasoning and quantifying the contribution of each recursion.

2 RELATED WORK

Cellular Automata and Learning Systems. Conway's GoL (Conway, 1970; Gardner, 1970) and its life-like generalizations are among the most studied discrete dynamical systems. GoL supports universal computation (Berlekamp et al., 2004; Rendell, 2002), rich behavior concentrates near the boundary between order and chaos (Langton, 1990; 1997), and cellular automata have been proposed as models of physical computation itself (Fredkin, 1990; Wolfram & Gad-el Hak, 2003). Complexity results for cellular automata, including the NP-completeness of the predecessor problem for finite configurations (Sutner, 1995) and the undecidability of reversibility and surjectivity in two dimensions (Kari, 1994), supply the formal backbone of LIFEWORLD. In machine learning, cellular automata serve both as a learning target and as a source of training data. Small networks struggle to learn GoL despite the existence of compact exact solutions (Springer & Kenyon, 2021), transformers can be trained to emulate it (Berkovich & Buehler, 2025), and pre-training on automaton trajectories transfers to language modeling and downstream reasoning (Zhang et al., 2025b; Lee et al., 2026a). Closest to our setting, CellARC (Ľžičař, 2025) evaluates rule induction on one-dimensional automata from a few support pairs. These efforts either train models on automaton data or evaluate a single direction of inference, whereas LIFEWORLD poses all three directions over two-dimensional life-like rules and evaluates frontier agents at inference time.

Deduction, Induction, and Abduction in LLMs. Deductive ability is typically measured through multi-step logical inference, rule application, and program execution (Morishita et al., 2024; Liu et al., 2026a; Gu et al., 2024). Inductive ability has been studied through function induction from examples (Hua et al., 2025), hypothesis search and refinement (Wang et al., 2024; Qiu et al., 2024), and the ARC-AGI series (Chollet, 2019; Chollet et al., 2025; ARC Prize Foundation, 2026); a related line asks whether models can infer the laws of an environment from observation or interaction (Baek et al., 2026; Liu et al., 2026b; Menaged et al., 2026). Abduction has received less attention, from commonsense explanation (Bhagavatula et al., 2020) to the recent taxonomy of (Salimi et al., 2026), which separates hypothesis generation from hypothesis selection, and (Zahavy, 2026) argues that abduction is the form current LLMs lack most. A few works consider the three forms jointly, either as stages of an agent's rule-learning loop (He et al., 2025) or as parallel evaluation suites (Sheng et al., 2025; Sun & Saparov, 2025), and (Agarwal et al., 2026) combine induction with deduction to synthesize formally verified systems. LIFEWORLD differs in three respects: the three forms share one underlying transition, every answer is checked exactly by simulation, and the difficulty of each form is characterized by known complexity results rather than by empirical calibration alone.

Test-time Scaling, Verification, and Agent Harnesses. Allocating more computation at inference time improves LLM reasoning, whether through repeated sampling (Brown et al., 2024), compute-optimal search (Snell et al., 2024), or recurrent computation (Lee et al., 2026b). Much of this gain depends on verification: learned verifiers and process reward models rank candidates (Cobbe et al.,

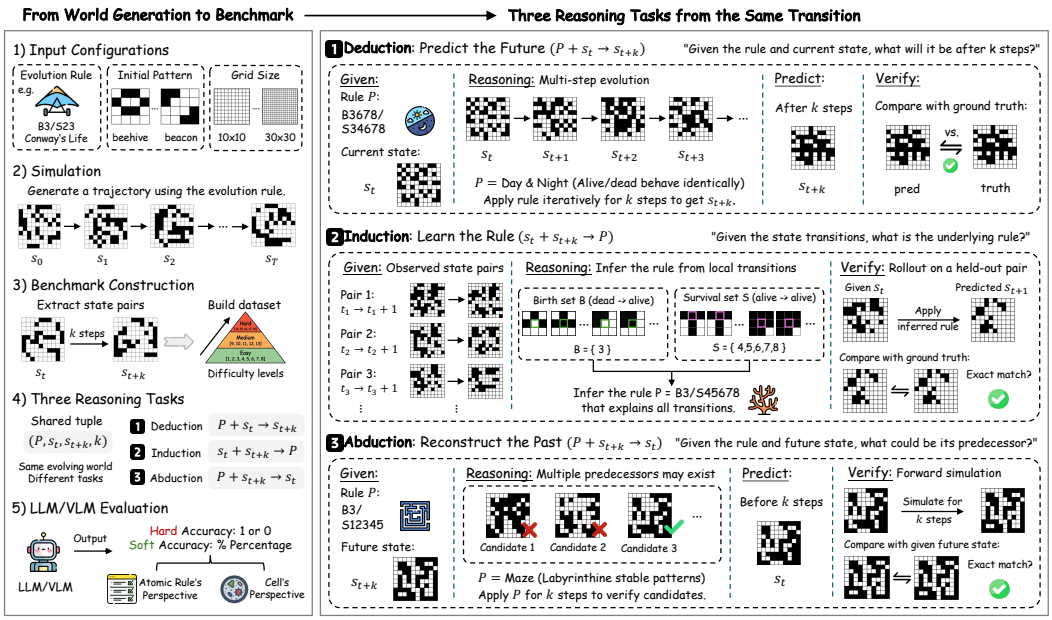

Figure 1: Overview of the LIFEWORLD benchmark.

2021; Lightman et al., 2024), and scaling verification alongside sampling is as important as sampling itself (Zhao et al., 2025). Recursive language models let a model decompose its input and call itself on the pieces (Zhang et al., 2025a). In parallel, the scaffolding around a model, or harness, has become an object of optimization in its own right, through reflective prompt evolution (Agrawal et al., 2026), end-to-end harness search (Lee et al., 2026c), and harnesses that improve themselves offline or online (Zhang et al., 2026; Karten et al., 2026), while SkillsBench evaluates reusable agent skills across tasks (Li et al., 2026). LIFEAGENT is complementary to these approaches. Rather than learning the harness, it derives its recursions from the structure of the task, so that the spatial decomposition is exact and the verifier in the loop is a deduction whose soundness is tracked explicitly.

3 LIFEWORLD: A BENCHMARK OF REASONING IN EVOLVING WORLDS

Life-Like Cellular Automata. A *life-like cellular automaton* (LLCA) is a grid of cells, each *alive* (1) or *dead* (0), that all update synchronously at each step $t \to t+1$. A cell's next state depends only on its own state $s(c,t)$ and on its live-neighbor count $n(c,t)$ among its eight Moore neighbors, not on where those neighbors lie. A rule P is therefore a *birth set* $B \subseteq \{0,\dots,8\}$ and a *survival set* $S \subseteq \{0,\dots,8\}$, written B/S:

$$s(c,t+1) = 1 \iff \big(s(c,t) = 0 \wedge n(c,t) \in B\big) \vee \big(s(c,t) = 1 \wedge n(c,t) \in S\big).$$

Conway's Game of Life (Gardner, 1970) is the rule B3/S23: a dead cell is born on exactly three live neighbors and a live cell survives on two or three. This short specification is already Turing complete (Berlekamp et al., 2004). Since B and S are arbitrary subsets, the family holds $2^9 \times 2^9 = 2^{18} = 262{,}144$ rules, whose behavior ranges from explosive growth to labyrinths and still lifes. LIFEWORLD draws its worlds from this family.

Three Modes of Reasoning over LLCA. Any life-like automaton couples three objects: a rule $P = (B,S)$, a configuration s_t, and a later configuration s_{t+k} obtained by applying P for k steps. Fixing any two of the three and solving for the third yields exactly the three modes of inference identified by Peirce (Peirce, 1934): *deduction*, *induction*, and *abduction*. LIFEWORLD is organized around this trichotomy, which cleanly separates forward simulation, rule learning, and inverse reasoning. Throughout, we take the grid to be $L \times L$ (hence L^2 cells) and let s_t denote a full configuration; complexities below refer to the direct brute-force algorithm and are summarized in Table 2.

- **Deduction:** $s_t + P \to s_{t+k}$**.** *Given* the rule P and the current state s_t, *predict* the future state s_{t+k}. This is the forward direction of the automaton: apply P mechanically for k steps. Because the dynamics are deterministic, the conclusion is *necessarily* true given the premises—the hallmark of deduction—and involves no search or uncertainty. Structurally it is a classical syllogism: rule

P governs every transition (major premise), the present state is s_t (minor premise), and therefore $s_{t+k} = P^k(s_t)$ (conclusion). Each step touches every cell once, so the cost is $\mathcal{O}(k\,L^2)$.

- **Induction:** $s_t + s_{t+k} \to P$**.** *Given* one or more observation pairs $(s_t^{(i)}, s_{t+k}^{(i)})$, *infer* the underlying rule P. This is the system-identification, or rule-learning, problem, and it is the paradigm of machine learning: fit a model from input–output pairs. Unlike deduction, its conclusion is only *probable*. Two rules $P \neq P'$ may agree on every observed pair yet diverge elsewhere, and no finite sample can uniquely certify a rule over the entire configuration space, an instance of Hume's problem of induction. The inferred rule sharpens with the number and diversity of observations, but certainty is never attained. The cost depends sharply on the horizon k. When $k = 1$, both endpoints of each transition are observed, so every cell contributes a direct constraint, its neighbor count and current state fix a single membership fact about B or S, and the rule is read off in one $\mathcal{O}(L^2)$ pass, with any unobserved counts left undetermined. When $k \geq 2$, the direct approach enumerates the 2^{18} rules and simulates for k steps, at cost $\mathcal{O}(2^{18}kL^2)$.
- **Abduction:** $P + s_{t+k} \to s_t$**.** *Given* the rule P and an observed later state s_{t+k}, *reconstruct* the most plausible antecedent s_t. This is inference to the best explanation (Peirce, 1934), and it is the hardest of the three directions for a structural reason: life-like automata are *irreversible*. The forward map is many-to-one, so a given s_{t+k} may admit many pre-images or none at all, the latter being the *Gardens of Eden* that possess no predecessor whatsoever. Recovering a pre-image is the PREDECESSOR problem, which is NP-complete (Sutner, 1995). Because the answer is not unique, one must appeal to auxiliary criteria (e.g., sparsity, minimum description length, or low entropy)to choose among admissible antecedents, and the result is a *conjecture* rather than a certainty. A brute-force search enumerates all 2^{L^2} candidate predecessors and simulates each forward process with the runtime complexity $\mathcal{O}(2^{L^2}kL^2)$.

Table 2: The three reasoning modes over life-like automata, with the complexity of the direct brute-force algorithm on an $L \times L$ grid. More efficient algorithms exist in special cases; abduction corresponds to the NP-complete PREDECESSOR problem (Sutner, 1995).

Mode	Direction	Solution	Complexity
Deduction	$s_t + P \to s_{t+k}$	unique, necessary	$\mathcal{O}(k\,L^2)$
Induction	$s_t + s_{t+k} \to P$	non-unique, probable	$\begin{cases}\mathcal{O}(L^2) & k = 1\\ \mathcal{O}(2^{18}\,k\,L^2) & k \geq 2\end{cases}$
Abduction	$P + s_{t+k} \to s_t$	non-unique, conjectural	$\mathcal{O}(2^{L^2}\,k\,L^2)$

Constructing LIFEWORLD. Each world is an $L \times L$ grid with periodic boundary, so that every cell has exactly eight neighbors, and $L \in \{10, 20, 30\}$. We use the twelve rules of Table 4. They include Conway's Life, a rule under which nothing dies (B3/S012345678), one under which no cell survives a step (B2/S), and one symmetric under exchanging live and dead cells (B3678/S34678). Each rule is started from 17 structured patterns, placed at the center of an otherwise dead grid, and from 151 random grids with live-cell density between 0.05 and 0.3. Each start is simulated for 100 steps, which gives 2,016 trajectories per grid size. A question is a pair (s_t, s_{t+k}) cut from one trajectory, with horizon $k \in \{1, \ldots, 5\}$. Together with the trajectory's rule P, one pair gives one query in each mode of reasoning, so the three modes are asked about the same pairs. The true s_t is a pre-image of s_{t+k}, so every abduction query has an answer and no Garden of Eden appears. Pairs whose s_t is all dead are dropped. Pairs that stand still ($s_t = s_{t+k}$) or die out (s_{t+k} all dead) are kept, but each kind is capped at five of the 100 pairs in a cell. A rule is 18 bits, one for each (state, count) class: the bit for (dead, n) is $[n \in B]$ and the bit for (alive, n) is $[n \in S]$. A pair *exercises* a bit if some cell of $s_t, \ldots, s_{t+k-1}$ on the true trajectory falls in that class. A bit that is never exercised has no effect on the pair, since every setting of it carries s_t onto the same s_{t+k}. The number of exercised bits sets the difficulty: *easy* (1-8), *medium* (9-13) and *hard* (14-18). It tracks how varied the grid is: easy pairs average 5% live cells and hard pairs 28%. Each mode is scored by the strictest criterion the pair can decide. A deduction is exact if it matches s_{t+k} on every cell. An induction is exact if it matches P on every bit the pair exercises. The other bits are not graded, since no reasoning from the pair could determine them. An abduction is exact if it is *any* grid that P carries onto s_{t+k} in exactly k steps, checked by simulation; it need not be the s_t the pair was cut from. Appendix C gives the details of the full procedure and statistics.

4 LIFEAGENT: A RECURSIVE REASONING AGENT FOR LIFEWORLD

LifeAgent solves a query with a short loop (Figure 2). It proposes a candidate answer using the primitive for the queried mode (*induce*, *abduce* or *deduce*), optionally checks that candidate, revises it in the light of what the check returned, and commits an answer. A schedule specifies how many rounds are available and what each round does, so single-shot prompting and a propose–verify loop are two configurations of one general procedure. Two properties of the task shape the design, and each of them gives rise to a form of recursion: (i) the object under study is recursive, since a k-step evolution is a composition of single steps and each step is local; and (ii) the inference relation is functional in more than one direction, since any two of (s_t, P, s_{t+k}) determine the third, so a candidate for one of them can be re-derived from the other two and compared with what the query already provides, named space-time recursion and logical recursion. The full procedure is in Algorithm 1 in Appendix D.

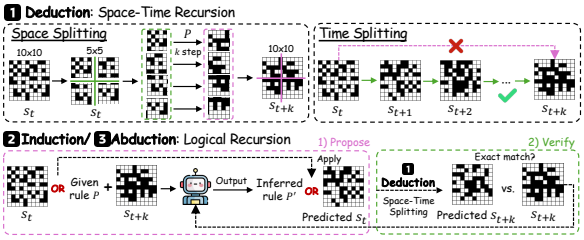

Figure 2: Overview of the LIFEAGENT.

Space-Time Recursion. A model asked for s_{t+k} in a single response has to perform L^2k cell updates without writing anything down. We avoid this by unrolling the automaton's own recursion, in time, in space, or in both. In time we use $\Phi_P^k = \Phi_P^{k-h} \circ \Phi_P^h$: the answer is built in k/h stages of h steps, and stage j is a query about the grid produced by stage $j-1$. The model applies the rule and the agent composes the stages. Intermediate stages are never committed, since s_{t+j} is not the grid the query asks for. In space we use locality. Let $W^{\oplus h}$ denote the dilation of a window W by Chebyshev radius h; repeated application of the neighbourhood map gives

$$\Phi_P^h(s)\big|_W \;=\; \Phi_P^h(\tilde{s})\big|_W \qquad \text{for any } \tilde{s} \text{ agreeing with } s \text{ on } W^{\oplus h}, \tag{1}$$

so a $b\times b$ window of the output depends on the input only through the $(b+2h)\times(b+2h)$ patch around it. The agent can therefore tile the grid, ask about each window separately, and paste the answers together. It plans the tiling and assembles the result, but never applies the rule itself. Because Eq. (1) is an identity, the procedure is exact: if every call is correct, so is the assembled grid. A layout (b, h) determines the work in one call, $w(b, h) = (b + 2h)^2 h$ cell updates, and the number of calls, $n = \lceil L/b \rceil^2 \, k/h$; overlapping halos increase the input by a factor of roughly $(1 + 2h/b)^2$. Eq. (1) describes the forward direction, so only deduction can decompose its own answer in this way. An induced rule is a global object and does not restrict to a window. For abduction the windows are coupled: two adjacent pre-images both constrain the cells near their shared boundary, so windows solved independently may disagree there and pasting them is no longer exact.

Logical Recursion. Suppose the query asks for one of the three objects and the agent has produced a candidate $\hat{c}$. Since any two objects determine the third, $\hat{c}$ and the given objects determine an object the query has already supplied; re-deriving that object and comparing it with what was given is a check, and the re-derivation is another instance of the same relation,

$$(s_t, s_{t+k}) \vdash \hat{P} \;\longrightarrow\; \Phi_{\hat{P}}^k(s_t) \stackrel{?}{=} s_{t+k}, \qquad (P, s_{t+k}) \vdash \hat{s}_t \;\longrightarrow\; \Phi_P^k(\hat{s}_t) \stackrel{?}{=} s_{t+k}. \tag{2}$$

Answering one object therefore reduces to answering another, and since a forward run is the only derivation the given objects support, every check is a deduction. The sub-problem is not easier than the original one: a forward run covers the whole grid and costs accordingly. What we gain is that the check runs in a different direction, where a checker that is allowed to decide may be available. We therefore distinguish what a checker returns from what it is allowed to conclude. Candidates are kept in a set $\mathcal{S} = \mathcal{S}^{\perp} \uplus \mathcal{S}^{+} \uplus \mathcal{S}^{-}$, ordered first by verification status and then by multiplicity, and only the exact simulator, which is correct by construction, may change a status. A check performed by the model is not sound, so rather than a verdict it returns the cells on which the re-derived grid differs from the observed one. A yes or no from an unreliable checker is difficult to use, whereas a set of disagreeing cells is a counterexample that the next proposal can work from, and we present it in the same format as a refutation from the simulator. Whether the simulator is available and whether it may decide are separate settings; a check performed by the model needs neither, which is why it is the fallback when no sound checker exists. When the agent commits, it also reports how the answer was obtained: *verified* if a sound checker accepted it, *conjecture* if nothing sound examined it, and *fail* if the simulator refuted every candidate. The depth of this recursion is the number of rounds R.

Table 3: Single-call exact accuracy in every cell (100 queries each). deepseek: `deepseek-flash`; gemini: `gemini-3.8-flash`; gpt: `gpt-5.6-luna`. Bold marks the best model in each column of each block. Cut-off rates and token counts are in Table 10.

			$L = 10$					$L = 20$					$L = 30$				
	difficulty	model	$k = 1$	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Deduction	easy	deepseek	**1.00**	**0.98**	**1.00**	**0.99**	0.99	**0.98**	**0.96**	0.96	**0.99**	**0.98**	0.93	0.84	0.93	0.89	0.91
		gemini	0.96	**0.98**	0.98	0.95	**1.00**	0.95	0.92	**0.99**	0.97	0.97	**0.95**	**0.99**	**0.96**	**0.99**	**0.98**
		gpt	0.71	0.63	0.72	0.55	0.74	0.53	0.43	0.57	0.55	0.61	0.37	0.37	0.49	0.44	0.44
	medium	deepseek	**0.99**	**0.97**	**0.97**	**0.96**	**0.95**	**0.80**	**0.70**	**0.68**	**0.68**	**0.61**	0.46	0.46	0.48	0.46	0.37
		gemini	0.88	0.82	0.79	0.79	0.79	0.69	0.67	0.62	0.60	0.59	**0.62**	**0.55**	**0.65**	**0.62**	**0.54**
		gpt	0.47	0.26	0.16	0.22	0.17	0.19	0.14	0.18	0.08	0.09	0.10	0.11	0.14	0.07	0.07
	hard	deepseek	**0.98**	**0.98**	**0.95**	**0.76**	**0.77**	**0.92**	**0.51**	**0.34**	**0.41**	**0.32**	0.28	0.18	0.16	0.08	0.11
		gemini	0.78	0.77	0.61	0.53	0.53	0.53	0.46	0.32	0.32	0.25	**0.40**	**0.22**	**0.21**	**0.14**	**0.21**
		gpt	0.23	0.12	0.04	0.01	0.02	0.02	0.03	0.04	0.01	0.03	0.00	0.00	0.01	0.03	0.04
Induction	easy	deepseek	**1.00**	**0.75**	**0.81**	0.66	0.87	**1.00**	0.79	0.79	0.72	0.72	0.84	0.72	0.73	0.70	0.75
		gemini	**1.00**	0.73	0.68	**0.77**	**0.91**	**1.00**	**0.82**	**0.86**	**0.83**	**0.84**	**1.00**	**0.93**	**0.86**	**0.90**	**0.91**
		gpt	0.91	0.44	0.49	0.37	0.64	0.75	0.33	0.51	0.46	0.56	0.76	0.42	0.49	0.49	0.60
	medium	deepseek	**0.99**	**0.54**	0.37	**0.36**	0.35	0.75	0.38	0.31	0.22	0.26	0.66	0.34	0.33	0.26	0.21
		gemini	0.98	0.50	**0.43**	0.34	**0.45**	**0.99**	**0.61**	**0.44**	**0.44**	**0.39**	**0.96**	**0.62**	**0.57**	**0.53**	**0.59**
		gpt	0.73	0.15	0.14	0.18	0.23	0.50	0.13	0.23	0.16	0.23	0.49	0.15	0.27	0.15	0.19
	hard	deepseek	**0.99**	**0.42**	0.23	**0.19**	0.14	0.93	0.30	0.22	0.13	0.16	0.71	0.30	0.18	0.09	0.14
		gemini	0.96	**0.42**	**0.25**	0.18	**0.29**	**0.99**	**0.48**	**0.37**	**0.31**	**0.31**	**0.95**	**0.46**	**0.35**	**0.32**	**0.32**
		gpt	0.48	0.07	0.04	0.10	0.15	0.41	0.10	0.15	0.17	0.18	0.30	0.12	0.17	0.12	0.14
Abduction	easy	deepseek	**0.80**	0.73	0.82	0.72	0.90	0.72	0.69	0.75	0.77	0.77	0.63	0.67	0.74	0.75	0.78
		gemini	0.79	**0.83**	**0.92**	**0.78**	**0.95**	**0.88**	**0.85**	**0.93**	**0.88**	**0.97**	**0.84**	**0.94**	**0.93**	**0.92**	**0.97**
		gpt	0.42	0.39	0.56	0.63	0.72	0.48	0.36	0.58	0.67	0.59	0.27	0.36	0.46	0.61	0.54
	medium	deepseek	0.39	0.42	0.34	0.50	0.48	0.22	0.30	0.25	0.32	0.39	0.17	0.17	0.29	0.33	0.32
		gemini	**0.42**	**0.44**	**0.48**	**0.53**	**0.63**	**0.37**	**0.46**	**0.39**	**0.50**	**0.60**	**0.37**	**0.42**	**0.50**	**0.52**	**0.58**
		gpt	0.15	0.23	0.24	0.31	0.31	0.08	0.18	0.24	0.31	0.30	0.08	0.12	0.19	0.25	0.22
	hard	deepseek	**0.19**	0.21	0.10	**0.28**	0.18	**0.08**	0.06	0.13	0.14	0.18	0.05	0.05	0.13	0.12	0.12
		gemini	0.17	**0.24**	**0.18**	**0.28**	**0.27**	**0.08**	**0.09**	**0.20**	**0.15**	**0.24**	**0.08**	**0.07**	**0.19**	**0.18**	**0.23**
		gpt	0.00	0.08	0.09	0.14	0.12	0.04	0.06	0.13	0.10	0.17	0.00	0.05	0.08	0.11	0.09

Composing Two Recursions. Every check is a deduction, and deductions decompose by equation 1, so the check produced by the logical recursion is exactly the object that the space–time recursion factors. In Algorithm 1 the same procedure RUN appears in both roles, once to produce an answer and once to check one, because w depends only on the layout and not on the mode that issued the call. This keeps two questions apart. Whether the answer to a query decomposes is a property of the mode, and holds for deduction alone; whether a particular forward run decomposes follows from Eq. (1), and holds wherever the run occurs. The practical consequence is that a verifier is no better than the deduction it performs, and that this deduction can be improved by a geometric argument about the object being verified, including in the two modes whose own answers do not decompose.

5 EXPERIMENTAL EVALUATIONS

Research Questions. The experiments address three research questions. **RQ1:** How well do current models perform deduction, induction and abduction in a single call, and how does accuracy depend on the grid size, the horizon and the difficulty (Section 5.1)? **RQ2:** Does unrolling the forward run in space and time improve deduction, and what decides whether it helps (Section 5.2)? **RQ3:** Does the propose–verify loop improve induction and abduction, and how does a checker built from the model depend on the deduction it performs (Section 5.3)?

Setup. Each cell $(L, k, \text{difficulty})$ of the evaluation split, with $L \in \{10, 20, 30\}$ and $k \in \{1, \ldots, 5\}$, holds 100 queries drawn round-robin over rules. We report exact accuracy (Appendix C.5): a deduction must match s_{t+k} on every cell, an induction must match the true rule on every exercised bit, and an abduction may be any pre-image of s_{t+k}. A reply that is cut off or cannot be parsed scores zero. We use `deepseek-flash`, `gemini-3.8-flash` and `gpt-5.6-luna` at default reasoning settings, with text-only prompts, no worked examples and no token budget. Sections 5.2 and 5.3 use `gemini-3.8-flash` only on hard queries due to the limited budget.

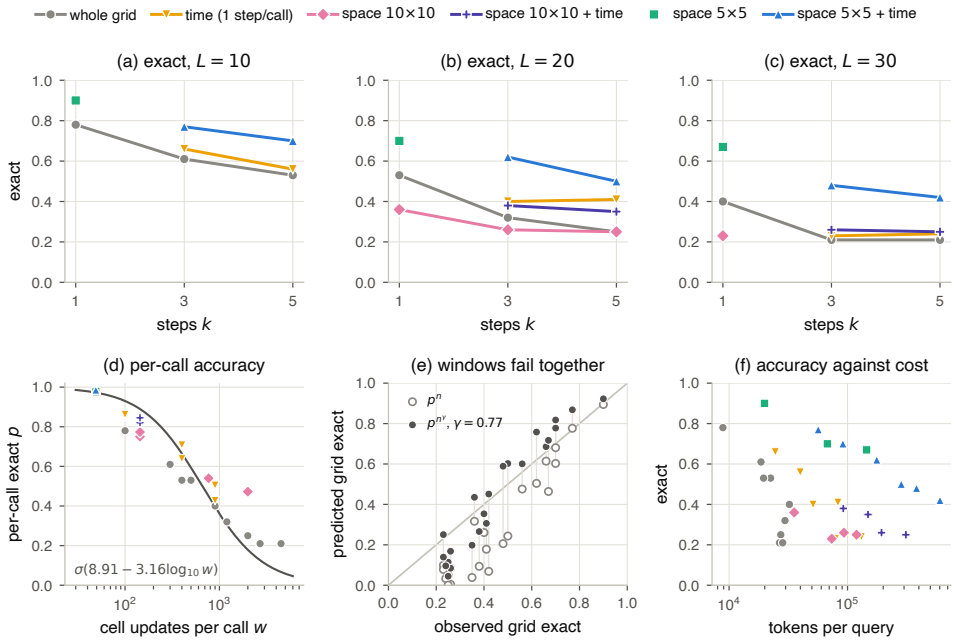

Figure 3: Unrolling deduction (`gemini-3.8-flash`, hard). (a)–(c) Exact accuracy of every layout in every cell; a lone marker is a layout run only at $k = 1$. (d) Accuracy of one call against the cell updates it carries, pooled over layouts and cells, with the logistic fit. (e) Grid accuracy predicted from the per-call accuracy p and the number of calls n, assuming independent (hollow) or clustered (filled) failures. (f) Exact accuracy against tokens per query, one point per layout and cell.

5.1 Single-Call Baselines

Each Mode Has Its Own Shape. Table 3 reports all 40,500 single-call queries. Difficulty is the largest factor: at $L = 30$, $k = 5$, `gemini-3.8-flash` scores 0.98 on easy and 0.21 on hard deduction, and 0.97 and 0.23 on abduction. Grid size and horizon act differently in each mode. Deduction falls along both axes, since one response must carry L^2k cell updates. On hard queries, for example, `gemini-3.8-flash` drops from 0.78 at $L = 10$, $k = 1$ to 0.21 at $L = 30$, $k = 5$. Induction is nearly solved at $k = 1$, where every exercised bit can be read off the pair (≥ 0.95 in all nine cells for `gemini-3.8-flash`). It then collapses to 0.42–0.48 on hard queries at $k = 2$. Per-bit errors show that the difficulty is inverting a composition (Table 12). Abduction instead rises with k, since a longer horizon admits more pre-images (e.g., 0.08 to 0.23 on hard queries at $L = 30$).

Models Fail Differently. `gemini-3.8-flash` is best in 99 of the 135 cells (6 of them ties) and is rarely cut off. `deepseek-flash` leads on hard deduction at $L = 10$ (0.95–0.98 for $k \leq 3$), but it reasons longer and hits its output limit. It is cut off on 5%, 39% and 70% of hard deduction queries at $L = 10$, 20 and 30, and on over half of hard induction queries (Table 10). `gpt-5.6-luna` reasons briefly and is the weakest model in 127 cells, tied in 4 more. `gemini-3.8-flash` and `gpt-5.6-luna` also lean towards Conway's rule. On the induction queries for which B3/S23 is wrong, they still answer it 29% and 16% of the time, against 2% for `deepseek-flash` (Table 11).

5.2 Space-Time Recursion: Unrolling Deduction

Fine Windows Win Everywhere, and Coarse Windows Can Lose. Besides the whole grid, we unroll the forward run in time only ($h = 1$), in space only ($b \in \{10, 5\}$) and in both, giving 23 unrolled conditions over $L \in \{10, 20, 30\}$ and $k \in \{1, 3, 5\}$ (Figure 3; tests in Table 13). 5×5 windows, unrolled in time when $k > 1$, beat the whole grid in all nine cells by +0.12 to +0.30 (all $p < 0.05$), reaching 0.42 against 0.21 at $L = 30$, $k = 5$. Splitting is not beneficial in itself: at $k = 1$, 10×10 windows lose 0.17 at both $L = 20$ and $L = 30$.

The Work per Call Decides the Sign. A tiled answer is also n observations of a call carrying w updates, and the probability $p(w)$ that such a call is exact follows $\sigma(8.91 - 3.16 \log_{10} w)$ (Figure 3d). A 5×5 one-step window ($w = 49$) is exact 97–99% of the time, a 10×10 one ($w = 144$) only 75–85%. Tiling therefore helps only when it raises p enough to pay for extra calls that must all be right. Those calls fail together (Figure 3e): at $L = 30$, $k = 5$, independence predicts 0.07 from $p = 0.985$ and $n = 180$, yet we observe 0.42. Modelling grid accuracy as p^{n^γ}, where $\gamma = 1$ means independent failures, a single $\gamma = 0.77$ fits the 23 unrolled conditions with mean absolute error 0.094, against 0.167 for $\gamma = 1$. A window should keep w below about 50–80 updates for this model, and the gain costs 2.2–21× the tokens of the whole grid (Figure 3f).

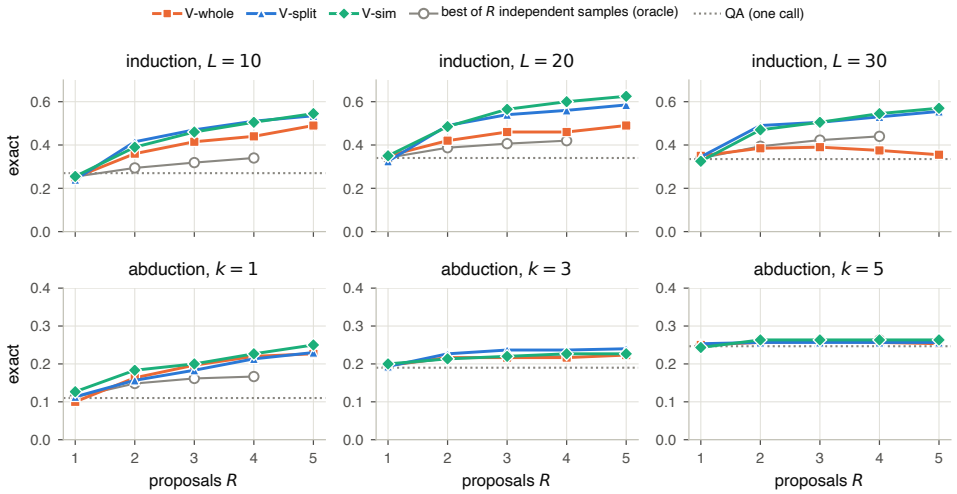

Figure 4: Exact accuracy against proposals R (`gemini-3.8-flash`, hard). Gray: one call (dotted) and the oracle best of R independent calls (hollow). Per-cell results are in Tables 14 and 15.

5.3 Logical Recursion: Propose–Verify

We run the loop with up to $R = 5$ proposals on induction ($k \in \{3, 5\}$) and abduction ($k \in \{1, 3, 5\}$), for $L \in \{10, 20, 30\}$. Each proposal is checked, and the next one sees the disagreement set D. The checker is the exact simulator (V-sim), or a forward run by the model on the whole grid (V-whole) or in 5×5 + time windows (V-split). Under a model checker, a clean run stops the loop; otherwise the candidate with the fewest disagreements is committed. The first proposals share the single-call prompt, so they also give the best of R independent calls as picked by an oracle.

Feedback Improves Induction Proposals but Not Abduction Proposals. On induction, accuracy rises with every proposal under V-sim and V-split, from 0.31 to 0.58 and from 0.30 to 0.56 on average (Figure 4; all gains $p < 0.01$). The best of four independent calls reaches only 0.40, even with an oracle picking it. After four proposals, a right rule is among the candidates for 0.54–0.55 of queries under every checker (Table 16), but V-whole fails to commit it and stays at 0.35–0.39 at $L = 30$. On abduction the loop clearly helps only at $k = 1$, and the proposer, not the checker, is at fault. The checkers accept one wrong pre-image in over 9,000 (Table 17), but only 7% of proposals are valid. After a refutation, the next proposal agrees with s_{t+k} on even fewer cells (Figure 8).

A Checker Is Only as Good as the Deduction It Performs. A right induction candidate can differ from the true rule only on bits the pair never exercises, so checking it asks for the same grid as the deduction query of that item, which Section 5.2 ran independently. In every cell, the share of right candidates a checker accepts matches the deduction accuracy of its layout ($r = 0.97$, mean absolute difference 0.04; Figure 5a). Since all three checkers accept wrong rules at the same rate (0.02), a model checker fails only by rejecting right answers, and its loss grows as its deduction worsens (Figure 5b). V-whole loses 0.24 of queries at $L = 30$, $k = 5$; V-split loses at most 0.05. V-split beats V-whole by $+0.16$ to $+0.21$ ($p < 0.003$) at $L = 20$, $k = 3$ and at $L = 30$, where it comes within 0.02 of V-sim. Costs are reported in Appendix E.4.

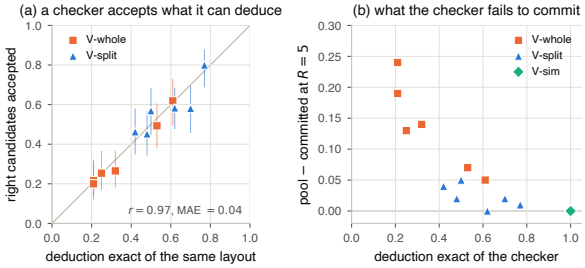

Figure 5: Induction checkers, one point per cell. (a) Right candidates accepted vs. deduction accuracy of the same layout (Section 5.2). (b) Queries with a right rule in pool but wrong committed answer ($R = 5$).

6 Conclusion

We introduced LifeWorld, a benchmark that poses deduction, induction, and abduction over the same life-like transitions, with exact verification and complexity-theoretic grounding for each form. Frontier LLMs show a distinct profile in each mode, and abduction remains by far the hardest. We further proposed LifeAgent, whose space-time recursion decomposes deduction exactly and whose logical recursion turns induction and abduction into propose-then-verify loops.

Limitations. Due to the limited space, we discuss the limitations in Appendix B.

AI USE STATEMENT

We used generative AI tools to assist with research brainstorming, literature search, manuscript editing, and drafting. All AI-assisted content was reviewed and verified by the authors, who take full responsibility for the final content.

ETHICS STATEMENT

This work involves no human subjects, crowdsourcing, or personal data. All benchmark instances are synthetic configurations of life-like cellular automata generated by simulation, and the evaluated models were accessed through public commercial APIs under their providers' terms of use. We do not see any ethical risks arising from this work.

REPRODUCIBILITY STATEMENT

The complete codebase, benchmark datasets, and execution scripts can be accessed at https://anonymous.4open.science/r/LifeWorld-03D5/

REFERENCES

Shubham Agarwal, Alexander Krentsel, Shu Liu, Mert Cemri, Audrey Cheng, Rui Meng, Tomas Pfister, Chun-Liang Li, Sylvia Ratnasamy, Aditya Parameswaran, et al. Inductive deductive synthesis: Enabling AI to generate formally verified systems. *arXiv preprint arXiv:2605.23109*, 2026.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *ICLR*, pp. 8479–8565, 2026.

ARC Prize Foundation. ARC-AGI-3: A new challenge for frontier agentic intelligence. *arXiv preprint arXiv:2603.24621*, 2026.

Doojin Baek, Gyubin Lee, Junyeob Baek, Hosung Lee, and Sungjin Ahn. Learning to theorize the world from observation. *arXiv preprint arXiv:2605.03413*, 2026.

Jaime A Berkovich and Markus J Buehler. LifeGPT: Topology-agnostic generative pretrained transformer model for cellular automata. *npj Artificial Intelligence*, 1(1):23, 2025.

Elwyn R Berlekamp, John H Conway, and Richard K Guy. *Winning Ways for Your Mathematical Plays*. AK Peters/CRC Press, 2004.

Chandra Bhagavatula, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Hannah Rashkin, Doug Downey, Wen tau Yih, and Yejin Choi. Abductive commonsense reasoning. In *ICLR*, 2020.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. ARC-AGI-2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

John Conway. Conway's game of life. *Scientific American*, 223:120–123, 1970.

Edward Fredkin. Digital mechanics. *Physica D*, 45(1-3):254–270, 1990.

Martin Gardner. Mathematical games. *Scientific American*, 222(6):132–140, 1970.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

Kaiyu He, Mian Zhang, Shuo Yan, Peilin Wu, and Zhiyu Chen. IDEA: Enhancing the rule learning ability of large language model agent through induction, deduction, and abduction. In *Findings, ACL*, pp. 13563–13597, 2025.

Wenyue Hua, Tyler Wong, Fei Sun, Liangming Pan, Adam Jardine, and William Yang Wang. InductionBench: LLMs fail in the simplest complexity class. In *ACL*, pp. 26526–26546, 2025.

Jarkko Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48(1):149–182, 1994.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026.

Chris G Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: nonlinear phenomena*, 42(1-3):12–37, 1990.

Christopher G Langton. *Artificial Life: An Overview*. MIT press, 1997.

Dan Lee, Seungwook Han, Akarsh Kumar, and Pulkit Agrawal. Training language models via neural cellular automata. *arXiv preprint arXiv:2603.10055*, 2026a.

Sangyun Lee, Sean McLeish, Tom Goldstein, and Giulia Fanti. Do language models need sleep? offline recurrence for improved online inference. *arXiv preprint arXiv:2605.26099*, 2026b.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026c.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2024.

Yang Liu, Jiaqi Li, and Zilong Zheng. RuleReasoner: Reinforced rule-based reasoning via domain-aware dynamic sampling. In *ICLR*, 2026a.

Ziming Liu, Sophia Sanborn, Surya Ganguli, and Andreas Tolias. From Kepler to Newton: Inductive biases guide learned world models in transformers. *arXiv preprint arXiv:2602.06923*, 2026b.

Miroslav Lžičař. CellARC: Measuring intelligence with cellular automata. *arXiv preprint arXiv:2511.07908*, 2025.

Reef Menaged, Gili Lior, Shauli Ravfogel, Roee Aharoni, and Gabriel Stanovsky. Can llm agents infer world models? evidence from agentic automata learning. *arXiv preprint arXiv:2606.16576*, 2026.

Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. Enhancing reasoning capabilities of LLMs via principled synthetic logic corpus. In *NeurIPS*, pp. 73572–73604, 2024.

Charles Sanders Peirce. *Collected Papers of Charles Sanders Peirce*, volume 5. Harvard University Press, 1934.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. In *ICLR*, 2024.

Paul Rendell. Turing universality of the game of life. In *Collision-based Computing*, pp. 513–539. Springer, 2002.

Moein Salimi, Shaygan Adim, Danial Parnian, Nima Alighardashi, Mahdi Jafari Siavoshani, and Mohammad Hossein Rohban. Wiring the 'why': A unified taxonomy and survey of abductive reasoning in LLMs. *arXiv preprint arXiv:2604.08016*, 2026.

Yu Sheng, Wanting Wen, Linjing Li, and Daniel Zeng. Evaluating generalization capability of language models across abductive, deductive and inductive logical reasoning. In *Proceedings of the 31st international conference on computational linguistics*, pp. 4945–4957, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Jacob M Springer and Garrett T Kenyon. It's hard for neural networks to learn the game of life. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021.

Yunxin Sun and Abulhair Saparov. Language models do not follow Occam's razor: A benchmark for inductive and abductive reasoning. *arXiv e-prints*, pp. arXiv–2509, 2025.

Klaus Sutner. On the computational complexity of finite cellular automata. *Journal of Computer and System Sciences*, 50(1):87–97, 1995.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah Goodman. Hypothesis search: Inductive reasoning with language models. In *ICLR*, 2024.

Stephen Wolfram and M Gad-el Hak. A new kind of science. *Appl. Mech. Rev.*, 56(2):B18–B19, 2003.

Tom Zahavy. Position: LLMs can't jump. In *ICML Position Paper Track*, 2026.

Alex L Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025a.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026.

Shiyang Zhang, Aakash Patel, Syed A Rizvi, Nianchen Liu, Sizhuang He, Amin Karbasi, Emanuele Zappala, and David van Dijk. Intelligence at the edge of chaos. In *ICLR*, 2025b.

Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. In *ICML*, 2025.

A Frequent Asked Questions (FAQs)

A.1 The benchmark

Q1. Why life-like cellular automata, and why on a torus? They give three reasoning modes over a single object with an exact answer. A rule P, a state s_t and a later state s_{t+k} are tied together so that any two determine the third. Deduction, induction and abduction are therefore the same relation queried in three directions, and they can be compared on the same pairs. Each answer can be checked exactly by simulation, and difficulty can be varied independently along three axes: the grid size L, the horizon k, and the number of rule bits a pair exercises. The family also spans very different dynamics (Table 4), so a model cannot succeed by knowing one rule well. A finite grid needs a boundary condition, and we use periodic boundaries rather than the infinite lattice: every cell then has eight neighbours and obeys the rule exactly as stated, with no special case at the edge (Appendix C.1). The prompts state the boundary explicitly (Listing 1).

Q2. Could a model have memorised the answers? Most pairs cannot be memorised. At least three quarters of the pairs in every difficulty bin start from a random grid, and 89% of hard pairs do (Table 6). A random grid under one of twelve rules is not a known object. Only one rule, B3/S23, has a large literature, and it supplies about a tenth of the pairs (Table 4). The structured seeds (Table 5) are named after their behaviour under B3/S23 and are most common in the easy bins.

Q3. What does the difficulty level measure, and why is induction graded only on the bits a pair exercises? The difficulty level is the number of the 18 rule bits that the pair exercises along its true trajectory: 1–8 for easy, 9–13 for medium and 14–18 for hard (Appendix C.3). It tracks how varied the grid is, and in practice its density: easy pairs are 5% live and hard pairs 28% (Table 6). The level matters for every model. At $L = 30$, $k = 5$, for example, `gemini-3.8-flash` scores 0.98 on easy and 0.21 on hard deduction, 0.91 and 0.32 on induction, and 0.97 and 0.23 on abduction (Table 3). A bit that the trajectory never exercises has no effect on the pair: every setting of it carries s_t onto the same s_{t+k}, so no reasoning from the pair could determine it. Grading all 18 bits would reward a guess about bits the data leaves open. Induction is therefore graded on exactly the bits that set the pair's difficulty, 14 to 18 of them on a hard pair (Appendix C.5).

Q4. Abduction accepts any pre-image. Can a model score by returning the observation unchanged? It can, on some queries. If s_{t+k} lies on a cycle whose period divides k, then s_{t+k} is its own pre-image, and returning it is exact. This holds for 35% of easy, 23% of medium and 12% of hard pairs, and on hard pairs it rises from 3% at $k = 1$ to 16% at $k = 5$ (Appendix C.4). The rate should be read beside abduction accuracy. On hard queries at $k = 5$, where 16% of pairs are their own pre-image, a single call scores between 0.09 and 0.27 across models and grid sizes (Table 3), not far above this trivial level. We keep these pairs because a static or periodic pre-image is a legitimate answer; rejecting it would make the metric depend on how the pair was generated, not on whether the answer is correct. Every abduction query has at least one answer, since each pair is cut from a simulated trajectory and the true s_t is a pre-image of s_{t+k}. No query asks for the predecessor of a Garden of Eden (Appendix C.3).

A.2 The single-call baseline

Q5. Why does accuracy depend on k so differently in the three modes? Because each mode's answer requires different work, and k changes that work in different ways (Table 3).

- **Deduction** is mechanical but requires L^2k cell updates in one response. On hard queries `gemini-3.8-flash` falls from 0.78 at $L = 10$, $k = 1$ to 0.21 at $L = 30$, $k = 5$, and deduction is the only mode that degrades with L for every model. Section 5.2 measures the same effect call by call: a call that carries w updates is exact with probability $p(w) = \sigma(8.91 - 3.16 \log_{10} w)$, which is 0.99 at $w \approx 23$ but 0.50 at $w = 659$, while a whole $L = 30$, $k = 5$ grid carries $w = 4{,}500$.
- **Induction** collapses between $k = 1$ and $k = 2$ (hard, `gemini-3.8-flash`: 0.95–0.99 at $k = 1$, 0.42–0.48 at $k = 2$) and then declines more slowly. At $k = 1$ every exercised bit can be read directly off the pair: each cell's state and neighbour count are visible in s_t, and its outcome in s_{t+1}. From $k = 2$ on, the intermediate states are hidden, and the rule has to be inverted through its own composition.

- **Abduction** gets *easier* with k (hard, `gemini-3.8-flash`: 0.17 to 0.27 at $L = 10$, 0.08 to 0.23 at $L = 30$). A longer horizon admits more pre-images, any one of which is accepted, and the share of pairs that are their own pre-image rises with k as well (Q4).

Q6. Why does `deepseek-flash` rank below `gemini-3.8-flash`, and does spending more tokens help? `deepseek-flash` runs out of output tokens. It reasons for twice as many tokens per query (42k against 21k) and is cut off at its 65,536-token limit on 32% of queries, including about half of hard induction and hard abduction queries. A reply that is cut off contains no answer and scores zero. On the hard queries it does finish, it is exact on 0.83 of deduction and 0.76 of induction queries, although these conditional rates favour the easier queries that finish (Appendix E.1). More tokens are not a sign of a better answer. Within one model, runs that use more tokens are *less* accurate, because harder cells draw longer reasoning (Figure 6): tokens per query track difficulty, not effort. No run imposed a token budget.

A.3 LIFEAGENT

Q7. Is unrolling deduction just handing the problem to a simulator, and why are only deduction answers decomposed? It is not. The harness never applies the rule. It cuts the grid into patches, sends each patch to the model and pastes the model's answers back, so every cell update is made by the model (Section 4, Appendix D.5). What makes the procedure exact is the light-cone identity equation 1: if every call is right, the assembled grid is right. That identity describes the forward direction only. An induced rule is a single global object that does not restrict to a window, and for abduction adjacent windows constrain the cells along their shared boundary, so pre-images solved independently need not agree there. Every check of an induction or abduction candidate, however, is a forward run, and forward runs do decompose. This is how V-split brings the decomposition to the two modes whose answers do not decompose.

Q8. If an exact simulator exists, why use the model as a checker at all? The setting the method targets is one where no sound checker exists. V-sim is the reference: it shows what the loop is worth when every verdict is correct, and there it is both the most accurate and the cheapest arm (Figure 9). The question the experiments answer is how close a checker built only from the model can come to it. On induction, V-split comes within 0.02 at $L = 30$ (Table 14). The comparison is controlled: a refutation is shown to the proposer in the same format whichever checker produced it, with the grid the check produced, the observed grid and the same instruction not to repeat the candidate (Listing 6). The proposer is not told which checker it faces, and the three arms differ only in how accurate the verdict was.

Q9. Why does even the exact simulator accept wrong rules? A rule can carry s_t exactly onto s_{t+k} and still differ from the true rule on a bit the pair exercises only at an intermediate step. Such a rule is consistent with everything the query shows, so a checker that tests consistency with the observation cannot reject it. The simulator accepts 2% of wrong induction candidates for this reason, and these make up 8% of the rules it accepts (Table 17). This is a floor for any checker in this setting. The two model checkers accept wrong rules at the same 2%.

A.4 RESULTS

Q10. Is the gain of the unrolled layouts just the result of spending more tokens? Spending alone does not explain it. More calls can make results worse: at $k = 1$, 10×10 windows use four to nine calls where the whole grid uses one, and they are significantly *worse* (-0.17 at $L = 20$ and at $L = 30$; Table 13). What decides the sign of the effect is the work per call, not the number of calls. 5×5 one-step windows ($w = 49$) are exact 97–99% of the time, while 10×10 one-step windows ($w = 144$) are exact only 75–85% of the time, and only the former gain in every cell. On propose–verify, V-split and V-whole accept wrong rules equally rarely (2%), but V-split accepts 0.57 of right rules against 0.33 for V-whole. That is the accuracy of the deduction each one performs (Figure 5), so the difference is one of verdict quality, not of quantity. Nor is the loop's gain a matter of drawing more samples: the best of four independent calls, even when an oracle picks it, reaches 0.40 on induction, while V-split commits 0.53 after four proposals (Table 16). We have not run a control that gives V-whole the same token budget as V-split in some other form, for example several samples of the whole-grid run.

Q11. Why model grid accuracy as p^{n^γ}, and how should one choose the window size? Failures are not independent. For $L = 30$, $k = 5$ in 5×5+t, $p = 0.985$ and $n = 180$ calls would predict 0.07 under independence, but we observe 0.42, because errors concentrate on the same queries. The exponent $\gamma = 0.77$ in p^{n^γ}, fitted once over the 23 unrolled conditions, predicts grid accuracy with a mean absolute error of 0.094, against 0.167 for $\gamma = 1$ (Section 5.2). To choose a window, keep the work per call where the model is reliable (below about 50–80 updates for `gemini-3.8-flash`) and use γ to judge how many calls a grid can afford. The curve $p(w)$ is a property of the model and has to be measured again for another model. The experiments compare 5×5 and 10×10 windows; other sizes were not run.

Q12. Why does the loop help induction much more than abduction, and how many proposals are needed? The two modes fail for different reasons. On induction, refutations make the proposer better under every checker: after four proposals a right rule is among the candidates for 0.54–0.55 of queries, against 0.40 for the best of four independent calls (Table 16). The checker then decides how much of this is committed. V-whole loses its gain at $L = 30$, where a whole-grid forward run is exact only 21% of the time, while V-split keeps it (mean exact 0.32 for QA, 0.45 with V-whole, 0.56 with V-split, 0.58 with V-sim). On abduction the checkers are accurate: of more than $9{,}000$ wrong pre-images, they accept one. The proposer is what fails. Only about 7% of proposals are valid, and after a refutation the next one agrees with the observation on fewer cells than the first (Figure 8). The loop helps only at $k = 1$ and at $L = 10$, $k = 3$ (Table 15). Most of the gain arrives by the third proposal: on induction, V-sim rises from 0.31 at $R = 1$ to 0.51 at $R = 3$ and 0.58 at $R = 5$, and V-split from 0.30 to 0.51 to 0.56 (Figure 4). At $R = 1$ every loop equals QA within noise, as it must.

Q13. With 100 queries per cell, are the differences real? Every comparison is paired by query and tested with McNemar's exact test (Section 5), and the appendix tables give paired-bootstrap 95% confidence intervals. A single-cell difference below about 0.10 cannot be resolved at this sample size, so the conclusions rest on patterns across cells. For example, 5×5 windows beat the whole grid in all nine deduction cells, and V-split gains significantly over QA in all six induction cells.

A.5 Scope and cost

Q14. Why are Sections 5.2 and 5.3 run on one model and on hard queries only? Hard queries leave the most room above single-call accuracy, and `gemini-3.8-flash` is the one model without a systematic failure in the baseline (Section 5.1): `deepseek-flash` is cut off on a third of its queries, and `gpt-5.6-luna` scores at most 0.04 on hard deduction beyond $L = 10$, $k \leq 2$. The cost of the tiled checker also limits how far these experiments can go (Q15). We list this as a limitation (Section B); the per-call curve $p(w)$ in particular is expected to differ from model to model.

Q15. What did the experiments cost, and can they be reproduced? They used 3.33B tokens over $476{,}219$ calls, and 91% of output tokens were thinking (Table 18). The unrolled checker is the expensive part: V-split costs 6.7–125× the tokens of QA per query, up to 2.6M tokens per query at induction $L = 30$, $k = 5$, and it accounts for 1.48B of the 2.07B tokens spent on propose–verify. The cost is justified only where the whole-grid run is unreliable, which on induction means $L \geq 20$ (Appendix E.4). The results can be reproduced. The benchmark is regenerated from the construction scripts with fixed seeds (Appendix C). Every prompt is given verbatim (Appendix D.2), and each arm is a fixed schedule with no learned or model-driven control (Appendix D.4). Every call is logged with its prompt, reply, thinking and token usage, and every figure and table in Section 5 is computed from these logs without further API calls (Appendix D.6).

B LIMITATIONS

Text only. Every grid reaches the model as rows of `0/1` digits, and every grid answer is returned as a list of integers (Appendix D.2). In this serialisation the two horizontal neighbours of a cell are adjacent characters, while its vertical neighbours are a full row away. A model that sees the grid as an image may have a different per-call curve $p(w)$, and the window size that pays off would move with it. We have not tried image input, vision-language models or other serialisations.

No self-improvement. Every query is solved from scratch with frozen weights. A refutation is shown to the next proposal of the same query and then discarded; nothing is carried over to the next query, whether a rule that was verified, a mistake that recurred, or a correction of the prior towards B3/S23. Under V-sim the loop produces verified answers, which could serve as training data or as a memory of worked cases, and we have not used them in either way. Nor does the agent learn how to run itself: the layout, the checker and R are set by hand for each arm (Appendix D.4), although the logs would let an agent estimate its own $p(w)$ and γ and choose its layout from them.

Experimental scope. The unrolling and propose-verify experiments (Sections 5.2–5.3) use one model, `gemini-3.8-flash`, on hard queries, with one sample per call and 100 queries per cell. Single-cell differences below about 0.10 therefore cannot be resolved, and conclusions rest on patterns across cells. The per-call curve $p(w)$ and the clustering exponent γ are properties of this model and need not transfer: in the baseline, `deepseek-flash` is cut off on 32% of queries and `gpt-5.6-luna` nearly fails hard deduction. We compared only 5×5 and 10×10 windows. As a checker we ran only the 5×5+t layout, which costs up to $125\times$ the tokens of a single call (2.6M per query at induction $L = 30$, $k = 5$); a cheaper layout may verify well enough, and we have not given V-whole the same token budget in another form, such as several whole-grid runs per check. All queries come from twelve life-like rules on a binary two-dimensional torus, and the models lean towards one of them: `gemini-3.8-flash` answers B3/S23 on 42% of induction queries, whereas only 10% of the queries use that rule. Every checker tests only whether a candidate reproduces the observed pair, so even the exact simulator accepts 2% of wrong rules, and the model checkers could be graded only because such a simulator exists here. Finally, the loop does not repair the abduction proposer. The only feedback we tried is the disagreement set, after which the next pre-image agrees with the observation on fewer cells, and we did not try to decompose the pre-image.

C LIFEWORLD CONSTRUCTION DETAILS

This appendix describes how the benchmark is built, from simulated trajectories to scored questions: the simulator (Appendix C.1), the rules and initial configurations it is run on (Appendix C.2), how questions are cut from trajectories and assigned a difficulty (Appendix C.3), what the resulting split contains (Appendix C.4), and how an answer is scored (Appendix C.5).

C.1 SIMULATOR

Section 3 defines life-like automata on the infinite lattice $\mathbb{Z}^2$. The benchmark uses a finite $L \times L$ grid with periodic boundary, so that every cell has exactly eight neighbours and no rule has to specify what lies beyond the edge. One step computes every cell's live-neighbour count on the torus and then updates all cells simultaneously: a dead cell with count in B becomes alive, a live cell with count in S stays alive, and every other cell is dead at the next step.

Rule bits. A rule is fully described by 18 bits, one for each pair (state, count) with state $\in \{\text{dead}, \text{alive}\}$ and count $\in \{0, \ldots, 8\}$: the bit for (dead, n) is $[n \in B]$ and the bit for (alive, n) is $[n \in S]$. Each cell at each step reads exactly one of these bits. We say that a pair of states (s_t, s_{t+k}) *exercises* a bit if some cell of $s_t, \ldots, s_{t+k-1}$ on the true trajectory is in that (state, count) class. A bit that is never exercised has no effect on the pair: every setting of it carries s_t onto the same s_{t+k}. This count is the difficulty measure of Appendix C.3, and it defines the induction metric of Appendix C.5.

C.2 RULES AND INITIAL CONFIGURATIONS

Rules. The benchmark uses the twelve life-like rules of Table 4. They span growth, decay, stable structures and chaos: a rule under which nothing dies (B3/S012345678), one under which no cell survives a step (B2/S), and a rule, B3678/S34678, that is symmetric under exchanging live and dead cells.

Table 4: The twelve rules of LIFEWORLD, with the number of evaluation pairs drawn from each (of 4,500).

name	rule	behaviour	pairs
Conway's Life	B3/S23	The classic rule; Turing complete, with rich dynamics.	445
HighLife	B36/S23	Close to Life, with natural self-replicators.	417
Day & Night	B3678/S34678	Symmetric under live/dead inversion.	374
Seeds	B2/S	No survival; explosive growth, every live cell dies each step.	413
Life without Death	B3/S012345678	Cells are born but never die; fractal growth.	391
Maze	B3/S12345	Grows labyrinthine, largely stable corridors.	439
Anneal	B4678/S35678	Smooths boundaries into large blobs.	363
Replicator	B1357/S1357	Every pattern replicates itself; highly chaotic.	294
Diamoeba	B35678/S5678	Amoeba-like blobs that roam and expand.	330
Coral	B3/S45678	Slowly growing reef-like structures.	345
34 Life	B34/S34	Birth and survival on three or four neighbours.	354
Gnarl	B1/S1	Any seed grows into fractal, triangular patterns.	335

Initial configurations. Each rule is started from the 17 structured patterns of Table 5 and from random grids. A structured pattern is placed at the centre of an otherwise dead grid. Any of its cells that fall outside the grid are dropped, which truncates the Gosper glider gun (36 cells wide) on every grid size used. A random grid has each cell alive independently with probability ρ, and is drawn once at $\rho = 0.3$ and then 30 times (seeds 1000–1029) at each of $\rho \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$. The patterns are named after their behaviour under B3/S23; under the other rules they are simply structured seeds, which is what they contribute to the benchmark.

Trajectories. Every (rule, initial configuration) pair is simulated for 100 steps on each grid size $L \in \{10, 20, 30\}$. This gives $12 \times (18 + 5 \times 30) = 2{,}016$ trajectories of 101 states per grid size.

Table 5: Structured initial patterns, grouped by their behaviour under Conway's rule.

class	patterns
still life	block, beehive, loaf, boat
oscillator	blinker, toad, beacon (period 2); pulsar (3); pentadecathlon (15)
spaceship	glider, lightweight, middleweight and heavyweight spaceship
gun	Gosper glider gun
methuselah	R-pentomino, acorn, diehard

C.3 From trajectories to questions

Windows. A question is a pair (s_t, s_{t+k}) taken from one trajectory, for every start t with $t + k \leq 100$ and every $k \in \{1, \ldots, 5\}$; $t = 0$ is allowed, so s_t can be the initial configuration itself. A grid size therefore offers $2{,}016\,(101 - k)$ windows at horizon k. The same pair, together with the trajectory's rule P, answers all three modes, so one pair yields one deduction, one induction and one abduction query. Every abduction query therefore has at least one pre-image, namely the true s_t, and no Garden of Eden appears in the benchmark.

Difficulty. The complexity of a window is the number of rule bits it exercises (Appendix C.1), counted over the k transitions $s_t \to \cdots \to s_{t+k}$; it ranges from 1 to 18. Windows are binned as *easy* (1–8 bits), *medium* (9–13) and *hard* (14–18). A hard window leaves at most four bits of the rule unobserved, while an easy one leaves at least ten.

Degenerate windows. Windows are sorted into four kinds.

- *Empty*: s_t is all dead, so s_{t+k} is all dead too, and the question is trivial in every mode. These windows are dropped. They are 46% of all windows at $L = 10$, 33% at $L = 20$ and 29% at $L = 30$, mostly from rules and patterns that die out.
- *Static*: $s_t = s_{t+k}$, not empty.
- *Dying*: s_{t+k} is all dead, s_t is not.
- *Normal*: everything else.

Static and dying windows are kept, since they are legitimate questions, but each kind is capped at 5% of a file so that they do not dominate the low-difficulty bins.

Sampling. Each file holds one cell $(L, k, \text{difficulty})$: $3 \times 5 \times 3 = 45$ files of 100 pairs, 4,500 pairs in all. A file takes up to five static and up to five dying windows at random from its bin, and fills the rest with normal windows. Normal windows are allocated as evenly as possible across rules, and when a rule has too few windows in the bin the shortfall is filled at random from the other rules' remaining windows. Sampling uses seed 42. The number of normal pairs per rule within a file ranges from 1 to 21, because some rules rarely reach some bins; 37 of the 45 files contain all twelve rules, and the rest contain ten or eleven. The experiments read each file round-robin over rules, so that any prefix of a file is as balanced across rules as the file allows. A training split is built in the same way with seed 43 and excludes every pair whose (s_t, s_{t+k}) appears in the evaluation split, so that the two share no question. All experiments in this paper use the evaluation split.

C.4 Statistics of the evaluation split

Table 6 summarises the evaluation split by difficulty.

- **Difficulty tracks density.** Easy pairs are sparse: a mean of 5% live cells, with a quarter of them grown from structured patterns. Hard pairs are 28% live and 89% come from random grids. A pair exercises many rule bits only if its grid holds many different neighbourhoods, and a dense random grid does.
- **Every file has 100 pairs.** Static and dying pairs are 4.8% and 4.4% of the split. They are fewer in the hard bins, where a pair that exercises 14 or more bits rarely stands still or dies out.
- **Returning the observation is often a valid answer to abduction.** If s_{t+k} lies on a cycle whose period divides k (a still life, or an oscillator of suitable period), then s_{t+k} is its own k-step pre-image. Because any pre-image is accepted, answering with the observed grid unchanged is then

correct. This holds for 35% of easy, 23% of medium and 12% of hard pairs. On hard pairs it rises with k: 3%, 10%, 13%, 15% and 16% for $k = 1, \ldots, 5$. It is one reason abduction accuracy rises with k (Section 5.1).

Table 6: The evaluation split by difficulty (15 files, 1,500 pairs each). *Density*: mean fraction of live cells. *Identity valid*: share of pairs for which s_{t+k} is itself a valid abduction answer.

	exercised bits	normal	static	dying	random init	density s_t / s_{t+k}	identity valid
easy	1–8 (mean 6.9)	1,350	75	75	0.75	0.05 / 0.05	0.35
medium	9–13 (mean 11.5)	1,359	75	66	0.79	0.21 / 0.20	0.23
hard	14–18 (mean 15.4)	1,380	65	55	0.89	0.28 / 0.27	0.12

C.5 Scoring

Each mode is scored by the strictest criterion the given pair can decide, and an answer that is missing or cannot be read scores zero.

- **Deduction.** The answer is exact if it equals s_{t+k} on every cell. A grid of the wrong shape is wrong.
- **Induction.** The answer is first put in canonical form: the B and S parts are located, and only the digits 0–8 are kept. It is exact if it agrees with the true rule on every bit the pair exercises, that is, on the (state, count) classes met along the *true* trajectory from s_t to s_{t+k}. The remaining bits are not graded, since no reasoning from this pair could determine them. The number of graded bits is therefore exactly the complexity that sets the pair's difficulty (Table 6). At $k = 1$ every graded bit can be read directly off the pair; at $k \geq 2$ the classes met at intermediate steps are hidden, and a rule that carries s_t onto s_{t+k} through different intermediate states is still wrong.
- **Abduction.** The answer is exact if it is *any* grid that the true rule carries onto s_{t+k} in exactly k steps; it need not be the s_t the pair was cut from. The check runs the simulator of Appendix C.1.

D LIFEAGENT IMPLEMENTATION DETAILS

This appendix gives what is needed to reproduce the runs of Section 5: the prompts, verbatim (Appendix D.2); how a reply is turned into an answer (Appendix D.3); the schedule that drives each arm (Appendix D.4); the details of the tiled forward run (Appendix D.5); and the model interface and logging (Appendix D.6).

D.1 ALGORITHMIC PROCEDURE OF LIFEAGENT

Algorithm 1 LifeAgent: A Recursive Reasoning Agent for LifeWorld

Require: query $q = (\mathit{mode}, E, k)$ on an $L \times L$ torus; rounds R; layout (b, h); checker $V \in \{\Phi, M\}$
Ensure: a committed answer and its regime

procedure RUN(s, P) ▷ *the space–time recursion*
  for $j = 1$ **to** k/h **do**
    for all windows W of a b-tiling of the torus **do**
      $u_W \leftarrow M\big(s|_{W^{\oplus h}},\ P,\ h\big)$ ▷ *one call, load* $w(b, h)$
    end for
    $s \leftarrow$ PASTE$(\{u_W\})$
  end for
  return s; **end procedure**
$\mathcal{S} \leftarrow \emptyset$
if *mode* $=$ DEDUCTION **then**
  return $\big($RUN$(s_t, P),$ conjecture$\big)$ ▷ *the forward run is the answer*
end if
for $r = 1$ **to** R **do**
  $\hat{c} \leftarrow M_{\mathit{mode}}(E, \mathcal{S})$ ▷ *a refinement once* $\mathcal{S} \neq \emptyset$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{\hat{c}\}$ with $\sigma(\hat{c}) \leftarrow \perp$
  $(s_0, P_0) \leftarrow (s_t, \hat{c})$ if *mode* $=$ INDUCTION, else $(\hat{c}, P)$
  if $V = \Phi$ **then**
    $\tilde{s} \leftarrow \Phi^k_{P_0}(s_0)$
  else
    $\tilde{s} \leftarrow$ RUN(s_0, P_0) ▷ *the same recursion, inside the check*
  end if
  $D \leftarrow \{x : \tilde{s}(x) \neq s_{t+k}(x)\}$; attach D to $\hat{c}$ ▷ *a disagreement set, not a verdict*
  if $V = \Phi$ **then** $\sigma(\hat{c}) \leftarrow +$ if $D = \emptyset$, else $-$ ▷ *only a sound checker decides*
  if $D = \emptyset$ **then break**
end for
return COMMIT$(\mathcal{S})$ ▷ *verified, conjecture, or fail*

D.2 PROMPTS

Every call is a single user message: there is no system prompt, no worked example and no instruction text beyond the prompt shown here. Grids are written as rows of space-separated `0/1` digits inside a Markdown code fence, with `1` for a live cell; rules are written in B/S notation (e.g. `B3/S23`). Every prompt asks for one JSON object whose `reasoning` field comes before its `answer` field. Four prompts cover every call the experiments make (Table 7). The three task prompts (Listings 2–4) share the rule background of Listing 1. The window prompt (Listing 5) advances one patch of a grid. The refinement preamble (Listing 6) is placed in front of a task prompt from the second proposal onward.

A check is an ordinary deduction. The forward run that checks a candidate is sent the deduction prompt (or, when tiled, the window prompt) with the candidate filled in: the proposed rule applied to the given s_t on induction, the given rule applied to the proposed s_t on abduction. Nothing in the prompt says that the call is a check or that the rule or grid came from an earlier call, so a checking call and an answering call on the same (s, P, k) are the same request.

Table 7: Every kind of call the experiments make, and the prompt it is sent. (s_0, P_0) is the subject of a check: $(s_t, \hat{c})$ on induction, $(\hat{c}, P)$ on abduction.

call	prompt	used by
first proposal	task prompt of the queried mode, Lst. 2–4	every arm
later proposal	refinement preamble (Lst. 6) + task prompt	V-sim, V-whole, V-split
forward run, whole grid	deduction prompt (Lst. 3) on (s_0, P_0), horizon k	V-whole
one stage, whole grid	deduction prompt on the current grid, horizon 1	*time*
one window	window prompt, Lst. 5	$b\times b$ layouts, V-split

Listing 1: The rule background, inserted wherever a template shows ⟨*rule background*⟩.

```
The rule is two sets over the Moore neighbourhood (8 neighbours),
    periodic (toroidal) boundary, simultaneous update:
- B (Birth):    a dead cell with n live neighbours is born     if n
    in B
- S (Survival): a live cell with n live neighbours survives    if n
    in S
```

Listing 2: The induction task prompt. Slots are shown as ⟨. . . ⟩; a grid slot is filled with the rows of the grid.

````
You are given two states of a life-like cellular automaton (CA) |
    State T and State T+⟨k⟩ | separated by ⟨k⟩ step(s). Infer the
    underlying rule in B.../S... notation.

## CA rule background

⟨rule background⟩

## Input

States are 2D grids where 1 = live, 0 = dead.

State T:
```
⟨State T⟩
```

State T+⟨k⟩:
```
⟨State T+k⟩
```

## Output

Respond with exactly one JSON object and nothing else outside it:

```json
{
  "reasoning": "<your step-by-step reasoning>",
  "answer": "<one valid rule in B.../S... notation>"
}
```
````

Listing 3: The deduction task prompt. It is also the prompt of a whole-grid forward run that checks a candidate, and of one stage of the *time* layout (with horizon 1).

```
You are given a life-like cellular automaton (CA) rule and an initial
     grid (State T). Simulate ⟨k⟩ step(s) and output the resulting
    grid (State T+⟨k⟩).
```

````
## CA rule: ⟨rule⟩

⟨rule background⟩

## State T

```
⟨State T⟩
```

## Output

Respond with exactly one JSON object and nothing else outside it:

```json
{
  "reasoning": "<step-by-step simulation>",
  "answer": <State T+⟨k⟩ as a 2-D list of 0/1 integers, same shape as
    State T>
}
```
````

Listing 4: The abduction task prompt.

````
You are given a life-like cellular automaton (CA) rule and a grid (
    State T+⟨k⟩). Recover a State T that evolves into it in exactly
    ⟨k⟩ step(s).

## CA rule: ⟨rule⟩

⟨rule background⟩

Any State T that forward-simulates onto the given State T+⟨k⟩ is fully
     correct | you do not have to recover the original one.

## State T+⟨k⟩

```
⟨State T+k⟩
```

## Output

Respond with exactly one JSON object and nothing else outside it:

```json
{
  "reasoning": "<how you worked backwards, and the forward check you
    ran>",
  "answer": <State T as a 2-D list of 0/1 integers, same shape as
    State T+⟨k⟩>
}
```
````

Listing 5: The window prompt: one call of a tiled forward run. The patch is the $(b+2h) \times (b+2h)$ region of the torus around a $b \times b$ window, and the call advances it by h steps. In every tiled run reported here $h = 1$ (Appendix D.5), except the space-only layouts, where $h = k$.

```
You are simulating one *window* of a life-like cellular automaton (CA
    ).

The grid below is a ⟨b+2h⟩x⟨b+2h⟩ patch cut out of a larger toroidal
    grid, together with a margin of ⟨h⟩ cell(s) on every side. You are
     asked for the central ⟨b⟩x⟨b⟩ window only | rows ⟨h⟩..⟨h+b-1⟩ and
    columns ⟨h⟩..⟨h+b-1⟩ of the patch, counting from 0.
```

```
## CA rule: ⟨rule⟩

⟨rule background⟩

## Patch of State T (with margin)

```
⟨patch⟩
```

What the margin is for

The margin is exactly wide enough that the central ⟨b⟩x⟨b⟩ window is
    determined by this patch alone: after ⟨h⟩ step(s), no cell outside
     the patch can have influenced it. Cells *in* the margin are a
    different matter | their own neighbours lie outside the patch, so
     whatever you compute for them will drift after the first step.
    Simulate the whole patch if that is easiest, but trust only the
    centre, and treat the outermost ring as unknown rather than as
    dead.

Output

Respond with exactly one JSON object and nothing else outside it:

```json
{
  "reasoning": "<step-by-step simulation>",
  "answer": <the central ⟨b⟩x⟨b⟩ window after ⟨h⟩ step(s), a 2-D list
    of 0/1 integers>
}
```

The answer must have exactly ⟨b⟩ rows of ⟨b⟩ integers | the window,
    not the whole patch.
```

The refinement preamble. When the candidate set $\mathcal{S}$ already holds candidates, the proposal prompt is the task prompt with the preamble of Listing 6 in front of it. The preamble lists every refuted candidate, newest first. Only the newest carries its full counterexample: the grid the check produced, the observed s_{t+k} it should have matched and, on abduction, the candidate pre-image itself, since an abduction candidate is otherwise named only by a hash label (`grid#9980(10x10,live=10)`). Older ones are listed by name with their count of disagreeing cells, which keeps most of the constraint at a fraction of the tokens. The attempt number counts the rounds of the loop, not the proposals, and a check is a round of its own; when every proposal is checked, the second proposal is attempt #3 and the fifth is attempt #9.

A refutation reads the same whichever checker produced it. Under V-sim the grid shown is the simulator's $\Phi^k_{P_0}(s_0)$; under V-whole and V-split it is the grid the model's forward run returned, placed under the same heading, with the same wording and the same "do not repeat" instruction. The proposer is therefore not told which checker it faces, and the three arms differ only in how accurate the refutation was. A second block, *Still on the table*, lists candidates that no check has refuted. With checks run after every proposal it appears only when a check produced no readable grid (Appendix D.3). Listings 7 and 8 show a preamble and a window call from saved runs.

Listing 6: The refinement preamble. Slots marked ⟨*induction*⟩ and ⟨*abduction*⟩ are alternatives: one of the two lines is sent, according to the mode. A candidate that could not be run at all (for example a grid of the wrong shape) is listed with a note instead of a count.

```
## Attempt #⟨round⟩: earlier candidates did not hold up

## Ruled out so far
```

- **⟨*newest refuted candidate*⟩** | rejected (⟨*m*⟩ cell(s) differ from the observed state)

  That State T was: ⟨*abduction only*⟩
```
⟨the candidate pre-image⟩
```

  Applying ⟨*candidate*⟩ to State T for ⟨*k*⟩ step(s) gives: ⟨*induction*⟩
  Evolving it under ⟨*rule*⟩ for ⟨*k*⟩ step(s) gives: ⟨*abduction*⟩
```
⟨the grid the check produced⟩
```
  but the observed State T+⟨*k*⟩ is:
```
⟨State T+k⟩
```

- **⟨*older refuted candidate*⟩** | rejected (⟨*m'*⟩ cell(s) differ from the observed state)

⟨*... one line per older refuted candidate, newest first ...*⟩

Every candidate above is **wrong** | do not repeat any of them. Use the mismatches to constrain what the answer must look like.

Still on the table (unverified) ⟨*only if non-empty*⟩

⟨*candidate*⟩ (proposed ⟨μ⟩x), ...

These have not been disproven. Either confirm one of them with a sharper argument, or propose something better.

⟨*the task prompt of the queried mode, unchanged*⟩

Listing 7: A refinement preamble as sent (V-sim, induction, $L = 10$, $k = 3$; third proposal), copied from a saved run. Both earlier proposals were refuted by the simulator; only the newer one carries its counterexample. The induction task prompt (Listing 2) follows it.

```
## Attempt #5: earlier candidates did not hold up

## Ruled out so far

- **B34/S34** | rejected (51 cell(s) differ from the observed state)

  Applying B34/S34 to State T for 3 step(s) gives:
```
1 1 1 0 0 0 1 1 1 0
0 0 0 0 0 1 1 1 1 0
0 0 1 0 0 0 1 1 0 0
0 0 1 0 1 0 1 1 1 0
1 1 1 1 1 1 1 1 1 1
1 1 0 0 1 0 1 0 1 1
0 0 0 0 1 1 0 1 1 0
1 1 0 0 1 1 1 1 1 0
1 1 0 1 1 0 0 1 1 1
1 0 1 0 0 0 1 0 1 1
```
  but the observed State T+3 is:
```
1 0 0 0 1 0 1 0 1 0
0 0 0 0 1 0 1 1 0 0
1 0 0 0 1 0 1 1 1 1
0 1 0 1 1 0 1 1 1 0
0 0 1 1 1 0 1 1 0 1
0 0 0 1 0 0 1 0 1 0
```
```

```
1 0 1 1 0 0 0 1 1 0
0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 1 1
```

- **B3/S23** | rejected (44 cell(s) differ from the observed state)

Every candidate above is **wrong** | do not repeat any of them. Use the mismatches to constrain what the answer must look like.

Listing 8: A window call as sent (V-split, induction, $L = 10$, $k = 3$), copied from a saved run: stage 1 of the check of the first proposal, for the top-left 5×5 window, whose 7×7 patch is shown.

```
You are simulating one *window* of a life-like cellular automaton (CA
    ).

The grid below is a 7x7 patch cut out of a larger toroidal grid,
    together with a margin of 1 cell(s) on every side. You are asked
    for the central 5x5 window only | rows 1..5 and columns 1..5 of
    the patch, counting from 0.

## CA rule: B3/S23

The rule is two sets over the Moore neighbourhood (8 neighbours),
    periodic (toroidal) boundary, simultaneous update:
- B (Birth):    a dead cell with n live neighbours is born     if n
    in B
- S (Survival): a live cell with n live neighbours survives    if n
    in S

## Patch of State T (with margin)

```
1 1 0 1 1 1 0
1 0 0 0 1 1 0
0 0 0 0 0 1 0
0 0 0 0 0 1 1
0 0 0 0 1 0 1
0 0 0 0 1 0 0
0 0 1 0 1 0 0
```

What the margin is for

The margin is exactly wide enough that the central 5x5 window is
    determined by this patch alone: after 1 step(s), no cell outside
    the patch can have influenced it. Cells *in* the margin are a
    different matter | their own neighbours lie outside the patch, so
     whatever you compute for them will drift after the first step.
    Simulate the whole patch if that is easiest, but trust only the
    centre, and treat the outermost ring as unknown rather than as
    dead.

Output

Respond with exactly one JSON object and nothing else outside it:

```json
{
  "reasoning": "<step-by-step simulation>",
  "answer": <the central 5x5 window after 1 step(s), a 2-D list of
    0/1 integers>
}
```
```

```
The answer must have exactly 5 rows of 5 integers | the window, not
    the whole patch.
```

D.3 From reply to answer

A reply is reduced to an answer in the same way in every arm, and every reply that does not yield an answer is scored as wrong.

- **Cut-off and failed replies.** A reply that stopped at the output limit (`finish_reason=length`) is not parsed, since what it holds is where the model was cut off rather than what it answered. A request that raised an error is retried (Appendix D.6); if every attempt fails, the call is logged as an error and yields no answer. A reply that the provider returned but marked as failed (a content filter, or a generation dropped mid-stream) is logged as an error and not retried.
- **The JSON object.** The parser takes the last brace-balanced object in the reply, removes `//` and `/* */` comments, and decodes it while allowing raw newlines inside strings, which models routinely emit in `reasoning`. If that fails, it decodes the first JSON value from the opening brace and ignores trailing text. If that also fails and the `answer` field opens a list, it reads the rows of integers that follow, which recovers a grid from an object that is broken elsewhere (for instance by a missing comma).
- **Rules.** An induction answer must name exactly one rule in B/S notation. Case and whitespace are ignored, and the rule is put in canonical form (the distinct digits of B and of S, sorted), so `b32/s3` and `B23/S3` are the same candidate. If the `answer` field does not name exactly one rule, the last rule string anywhere in the reply is taken instead, ignoring the bare `B/S`, which in prose names the notation; since the reasoning comes first, this is usually the rule the model settled on.
- **Grids.** A grid answer must be a 2-D list of the query's shape; any other shape is wrong. A window answer may be the $b \times b$ window or the whole $(b+2h) \times (b+2h)$ patch. Models sometimes return the patch they were given, advanced in full, and its interior is exactly the window, so it is cropped rather than discarded.
- **The thinking.** If a reply finished but its visible text yields no answer, the same extraction is applied to the model's thinking, when the provider returns it. The thought summary that `gemini-3.8-flash` places at the head of its reply is stripped before the visible text is parsed.

A check compares the grid it produced with s_{t+k} cell by cell and files the number of disagreeing cells. A check whose grid could not be read, or has the wrong shape, files no count and no verdict. Its candidate stays unrefuted, and it is the case in which the *Still on the table* block of Listing 6 appears.

D.4 The loop and the arms

Each arm is a fixed schedule for the controller: a table of parameters that the controller reads each round, and a priority list that turns the parameters and the state of $\mathcal{S}$ into one action. No LLM is involved in choosing actions. The first applicable rule wins.

1. Commit if a candidate has passed the simulator, or if a model check has come back clean.
2. If $\mathcal{S}$ is empty, propose (with the plain task prompt) while proposals remain, and otherwise commit, which fails. An unparsable reply therefore uses up one proposal.
3. Under V-sim, check every candidate the simulator has not yet seen.
4. Under V-whole and V-split, run forward the highest-ranked candidate that has not been run yet.
5. If proposals remain, propose again, with the refinement preamble.
6. Otherwise commit.

Candidates are ranked by status and then by multiplicity μ, the number of times a value was proposed; ties keep the older candidate first. A value proposed again is not a new candidate. It raises μ and uses up a proposal, but it is not checked a second time. Under V-sim, COMMIT returns the highest-ranked candidate that passed (verified), or fail if every candidate was refuted, in which case nothing is committed and the query scores zero. Under the two model checkers no candidate is ever passed or refuted, so the committed answer is always a conjecture: a candidate whose run came back clean, if there is one, and otherwise the candidate whose run disagreed with s_{t+k} on the fewest

cells, with candidates that have no readable run placed last. Table 8 gives the parameters of each arm.

Table 8: Schedule of each arm. R: proposals per query. *Rounds*: the cap on loop rounds, where a proposal, a check and a commit each take one round; with $R = 5$ a query needs at most $2R+1 = 11$ rounds. Every call draws one sample.

arm	checker	R	rounds	forward run
QA	none	1	3	–
deduction layouts (Sec. 5.2)	none	1	3	the answer, laid out as (b, h)
V-sim	exact simulator, decides	5	12	–
V-whole	model, clean run commits	5	12	whole grid, one call
V-split	model, clean run commits	5	12	5×5 windows, $h = 1$

Only V-sim consults the exact simulator during an episode; in every other arm it is used only to grade the committed answer.

D.5 The tiled forward run

Tiling. A layout (b, h) divides each axis of the $L \times L$ torus into $\lceil L/b \rceil$ parts that are as equal as possible (so $b = 10$ at $L = 30$ gives three parts of 10). Each window is sent the $(b + 2h) \times (b + 2h)$ patch around it, cut with wrap-around, and is asked to advance it h steps (Listing 5). With $h = 1$ (5×5+t, 10×10+t, and V-split) the run proceeds in k stages, and stage j is cut from the grid assembled at stage $j - 1$. In the space-only layouts, $h = k$ and there is a single stage. If a window is at least as wide as the grid, or its patch would wrap onto itself ($b + 2h > L$), the layout is not tiled and the grid is answered whole. That is why 10×10 coincides with *whole* at $L = 10$. In the time-only layout every stage is a whole-grid call, and it is sent the deduction prompt with horizon 1 rather than the window prompt, since the grid is the whole torus and no margin exists. Table 9 lists the geometry of every condition that was run.

Assembly. Each window is asked once, and its answer is written into the assembled grid. A window whose reply yields no answer is filled with the corresponding cells of the stage's input ("no change"), and the number of such windows is logged. If no window of a stage yields an answer, the run stops. A deduction answer then has no candidate and is scored as wrong, and a check files no verdict. An intermediate stage is never committed or compared with anything, because s_{t+j} is not the grid the query asks for; only the grid after the last stage is committed as an answer or compared with s_{t+k}.

Table 9: Geometry of every condition run in Section 5.2: calls per query n / cell updates per call w, as the planner laid them out (a dash marks a layout that was not run). A V-split check has the geometry of 5×5+t (at $k = 1$, of 5×5), and a V-whole check that of *whole*, for the same L and k; each check is repeated for each of up to $R = 5$ candidates.

L	k	whole	time	10×10	10×10+t	5×5	5×5+t
10	1	1 / 100	–	–	–	4 / 49	–
10	3	1 / 300	3 / 100	–	–	–	12 / 49
10	5	1 / 500	5 / 100	–	–	–	20 / 49
20	1	1 / 400	–	4 / 144	–	16 / 49	–
20	3	1 / 1,200	3 / 400	4 / 768	12 / 144	–	48 / 49
20	5	1 / 2,000	5 / 400	4 / 2,000	20 / 144	–	80 / 49
30	1	1 / 900	–	9 / 144	–	36 / 49	–
30	3	1 / 2,700	3 / 900	–	27 / 144	–	108 / 49
30	5	1 / 4,500	5 / 900	–	45 / 144	–	180 / 49

D.6 MODEL INTERFACE AND LOGGING

All three models are called through their providers' OpenAI-compatible chat-completions endpoints. We set no sampling parameter (temperature, top-p), reasoning-effort setting or output limit, so every call runs at the provider's defaults; for `deepseek-flash` the default output limit is 65,536 completion tokens. For `gemini-3.8-flash` we request the thought summary (`include_thoughts`) so that the record holds what the model thought; the OpenAI endpoint returns only a count of reasoning tokens for `gpt-5.6-luna`. Each request has a timeout of 1,800 s. A request that raises an error (a timeout, a connection or server error) is retried up to five times with exponential back-off, and each retry is a fresh sample, not a replay. Requests are sent concurrently, 100 to 200 at a time; a query's own calls are issued in the order the loop requires, and calls from different queries and from different windows of the same stage are sent together. No token budget is imposed on any episode.

Every call is logged with its round, its kind (proposal, forward run, window), the full prompt, the visible reply, the thinking where the provider returns it, the finish reason, token usage including reasoning tokens, latency, the number of attempts and any error. Window calls also record the window's position and size. The committed answer, its regime (verified, conjecture or fail) and the trajectory of actions are stored with each episode, so every figure in Section 5, including the $R = 1$ and $R = 3$ prefixes and the per-call accuracy $p(w)$, is computed from these logs without further calls.

E ADDITIONAL EXPERIMENTAL RESULTS

E.1 SINGLE-CALL BASELINE

Table 3 in Section 5.1 gives the exact accuracy of every cell. Table 10 gives, for the same blocks and each grid size, the share of calls cut off at the output limit and the tokens spent per query. Figure 6 relates the two.

Table 10: Single-call baseline: share of calls cut off at the output limit (a dash is none) and mean tokens per query (input, output and thinking), per grid size over the five horizons k (500 queries per entry). Cut-off calls score as wrong. Model names as in Table 3.

			cut off			tokens per query		
	difficulty	model	$L = 10$	20	30	$L = 10$	20	30
deduction	easy	deepseek	<1%	<1%	1%	13k	17k	23k
		gemini	–	–	–	7k	9k	12k
		gpt	–	–	–	3k	5k	7k
	medium	deepseek	1%	15%	28%	25k	41k	48k
		gemini	–	–	<1%	13k	21k	24k
		gpt	<1%	–	–	5k	6k	8k
	hard	deepseek	5%	39%	70%	35k	53k	62k
		gemini	–	–	2%	17k	27k	29k
		gpt	–	–	–	6k	6k	7k
induction	easy	deepseek	11%	14%	18%	26k	31k	38k
		gemini	–	–	–	9k	12k	14k
		gpt	–	<1%	<1%	4k	5k	7k
	medium	deepseek	38%	48%	55%	43k	54k	58k
		gemini	–	–	<1%	13k	19k	23k
		gpt	–	–	–	6k	6k	8k
	hard	deepseek	53%	54%	58%	50k	56k	60k
		gemini	–	–	<1%	13k	18k	24k
		gpt	–	–	<1%	6k	6k	8k
abduction	easy	deepseek	20%	23%	19%	25k	28k	32k
		gemini	1%	2%	3%	14k	16k	18k
		gpt	–	–	–	5k	5k	7k
	medium	deepseek	55%	51%	44%	46k	51k	52k
		gemini	2%	2%	14%	30k	35k	37k
		gpt	<1%	–	–	6k	6k	8k
	hard	deepseek	71%	49%	31%	57k	54k	51k
		gemini	1%	1%	3%	33k	36k	34k
		gpt	1%	1%	–	7k	7k	7k

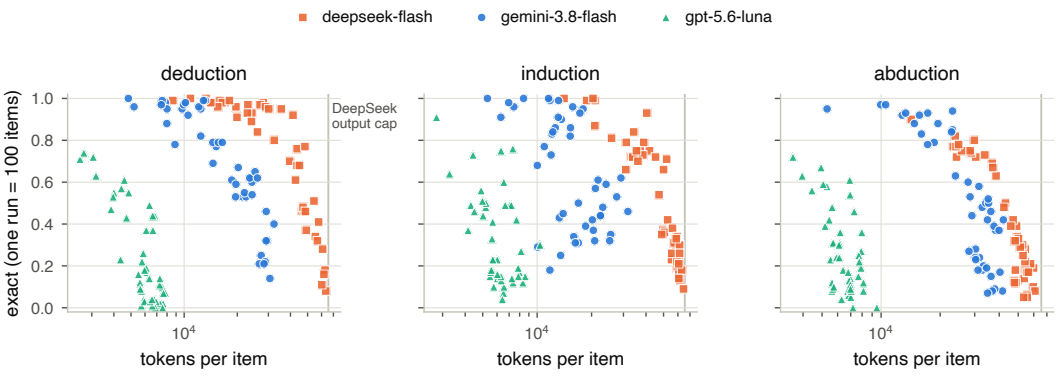

Figure 6: Each point is one run (one model, mode and cell; 100 queries): exact accuracy against mean tokens per query. `deepseek-flash` runs pile up against its output limit (vertical line). Within each model, accuracy falls as tokens rise, because harder cells draw longer reasoning.

Where the Models Fail. The three models fail in different ways.

- `deepseek-flash` reaches its output limit more often as the grid or the difficulty grows (Table 10). A reply cut off at the limit contains no answer. On the hard queries it does finish, it answers 0.83 of deduction and 0.76 of induction queries exactly. These conditional rates favour the easier queries, but they show that the output limit, not the reasoning, bounds its score.
- `gpt-5.6-luna` is cut off on 0.1% of calls, with a median of 3.8k thinking tokens, and 2.5% of its replies do not parse.
- `gemini-3.8-flash` is cut off on 1.2% of calls and shows neither failure, which is why the remaining experiments use it.

Within one model, runs that use more tokens are *less* accurate (Figure 6), because tokens per query track the difficulty of the cell, not the effort.

Table 11: Single-call exact accuracy by rule, over all 45 cells (about 375 queries per entry); rules are sorted by mean accuracy. The last row is the share of induction answers that are B3/S23 on all 18 bits; 10% of queries use that rule. D, G, P: `deepseek-flash`, `gemini-3.8-flash`, `gpt-5.6-luna`.

	deduction			induction			abduction		
rule	D	G	P	D	G	P	D	G	P
B3/S012345678	0.82	0.81	0.28	0.73	0.91	0.54	0.67	0.70	0.46
B3/S23	0.76	0.68	0.31	0.78	0.99	0.62	0.47	0.54	0.37
B3/S12345	0.68	0.64	0.25	0.64	0.79	0.41	0.59	0.61	0.46
B36/S23	0.76	0.69	0.27	0.69	0.69	0.48	0.45	0.53	0.34
B4678/S35678	0.83	0.75	0.37	0.35	0.39	0.30	0.55	0.67	0.48
B3678/S34678	0.74	0.76	0.30	0.45	0.57	0.18	0.41	0.55	0.20
B34/S34	0.73	0.75	0.31	0.42	0.65	0.15	0.33	0.56	0.16
B3/S45678	0.70	0.68	0.23	0.47	0.53	0.26	0.37	0.47	0.22
B35678/S5678	0.72	0.70	0.23	0.36	0.43	0.18	0.31	0.54	0.14
B1357/S1357	0.65	0.72	0.09	0.32	0.56	0.18	0.33	0.50	0.05
B2/S	0.66	0.53	0.14	0.48	0.59	0.28	0.13	0.26	0.09
B1/S1	0.61	0.57	0.18	0.35	0.41	0.24	0.13	0.24	0.12
induction answers that are B3/S23				0.17	0.42	0.23			

Accuracy by Rule, and a Prior towards Conway's Rule. Rules differ widely in difficulty (Table 11). B3/S012345678, under which every live cell survives, is the easiest in every mode. B1/S1 and B2/S are the hardest in induction and abduction. The rules also reveal a prior. `gemini-3.8-flash` answers B3/S23 on 42% of induction queries, and on 35% of the queries whose rule is something else; the other two models do so on 17% and 23% of queries. The prior lifts `gemini-3.8-flash` to 0.99 on induction queries whose rule *is* B3/S23. Its deduction under B3/S23 (0.68) is no better than under other rules, so the models know the name of the rule better than its dynamics.

Table 12: Induction, hard queries, single call: share of exercised rule bits answered wrongly, over replies that contain a rule. *Answered* is the share of queries with a parseable rule. The last five columns split the bits at $k \geq 2$ by how many cell updates along the true trajectory exercise them.

		bits wrong		$k \geq 2$, by updates exercising the bit				
model	answered	$k = 1$	$k \geq 2$	1	2–3	4–10	11–100	>100
`deepseek-flash`	0.45	0.01	0.11	0.09	0.12	0.14	0.08	0.13
`gemini-3.8-flash`	1.00	0.00	0.21	0.26	0.23	0.26	0.19	0.18
`gpt-5.6-luna`	0.97	0.11	0.28	0.32	0.28	0.33	0.27	0.23

Which Rule Bits Induction Gets Wrong. At $k = 1$ each exercised bit can be read directly off the pair: the state and neighbour count of a cell are visible in s_t and its outcome in s_{t+1}. `gemini-3.8-flash` gets 0.3% of these bits wrong and `deepseek-flash` 1% (Table 12).

From $k = 2$ on, the intermediate states are hidden, and 21% and 11% of the bits are wrong. The error rate barely depends on how much evidence the pair holds for a bit: for `gemini-3.8-flash` it is 0.26 for bits exercised once and 0.18 for bits exercised more than a hundred times. The collapse of induction between $k = 1$ and $k = 2$ is therefore a failure to invert the composition of the rule, not a shortage of evidence. `deepseek-flash` returns a rule on only 45% of hard induction queries, so its conditional rates cover the queries it finishes.

E.2 Unrolled deduction

Table 13: Deduction under different layouts of the forward run (`gemini-3.8-flash`, hard, 100 queries per cell). Exact accuracy; bold is the best layout in the row. Marks are McNemar tests against *whole*: $^{*}p < 0.05$, $^{**}p < 0.01$ for a gain, $^{\downarrow}p < 0.05$ for a loss. A dash marks a layout not run in that cell (space-only 5×5 was run only at $k = 1$, where it coincides with 5×5+t). The last two columns give the gain of the finest layout over *whole* with its paired-bootstrap 95% CI, and its token cost relative to *whole*.

L	k	whole	time	10×10	10×10+t	5×5	5×5+t	Δ finest [95% CI]	tokens
10	1	0.78	–	–	–	**0.90***	–	+0.12 [+0.02, +0.23]	2.2×
10	3	0.61	0.66	–	–	–	**0.77***	+0.16 [+0.03, +0.29]	3.0×
10	5	0.53	0.56	–	–	–	**0.70***	+0.17 [+0.02, +0.32]	4.1×
20	1	0.53	–	0.36$^{\downarrow}$	–	**0.70***	–	+0.17 [+0.02, +0.32]	3.5×
20	3	0.32	0.40	0.26	0.38	–	**0.62****	+0.30 [+0.18, +0.42]	6.0×
20	5	0.25	0.41**	0.25	0.35	–	**0.50****	+0.25 [+0.14, +0.36]	10.4×
30	1	0.40	–	0.23$^{\downarrow}$	–	**0.67****	–	+0.27 [+0.15, +0.39]	4.5×
30	3	0.21	0.23	–	0.26	–	**0.48****	+0.27 [+0.17, +0.37]	14.4×
30	5	0.21	0.24	–	0.25	–	**0.42****	+0.21 [+0.12, +0.30]	21.3×

Other Layouts. Unrolling only in time is significant in one cell ($L = 20$, $k = 5$: +0.16); at $L = 30$ a single step still carries 900 updates. At $k \in \{3, 5\}$, 10×10 windows have no effect, because the halo grows with k: a 10×10 window at $k = 5$ reads a 20×20 patch, so the call is no smaller than the whole $L = 20$ grid.

Per-Call Accuracy. The fitted curve $p(w)$ reaches 0.95 at $w = 77$, 0.90 at 133 and 0.80 at 240. Work is not its only factor. At equal work, windowed calls do better than whole-grid calls: a 10×10 window at $k = 5$ carries $w = 2{,}000$ and is exact 0.47 of the time, whereas the whole $L = 20$, $k = 5$ grid carries the same w and is exact 0.25 of the time. With the fitted curve in place of the measured p, the clustered prediction of Figure 3e is looser (mean absolute error 0.125, $r = 0.85$), because the curve misses this effect.

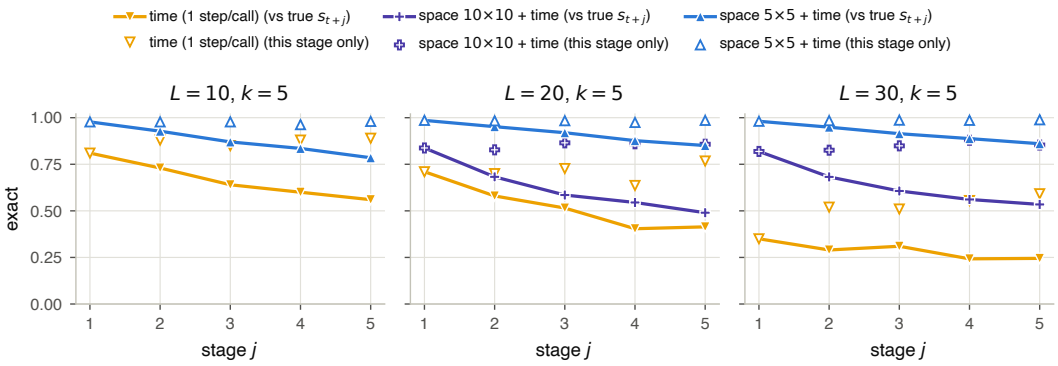

Figure 7: Unrolled runs at $k = 5$, stage by stage. Filled markers: agreement of the assembled grid after stage j with the true s_{t+j}. Hollow markers: the accuracy of stage j alone, i.e. whether its calls advanced their own input correctly by one step.

In Coarse Layouts, Most of the Final Error Is Inherited. Figure 7 follows the unrolled runs stage by stage. In 5×5+t every stage is right about 98% of the time. Agreement with the true intermediate grid decays slowly, from 0.98 to 0.86 over five stages at $L = 30$. In *time*, the per-stage accuracy is lower: 0.64–0.89 at $L \leq 20$ and 0.35–0.59 at $L = 30$. Because each stage builds on the previous one, agreement falls to 0.56, 0.41 and 0.24 by stage 5, so most of the final error was made in earlier stages and carried forward.

E.3 PROPOSE–VERIFY

Tables 14 and 15 give every cell at $R = 5$. Budgets $R < 5$ are read from the same runs by replaying the commit rule on the first R proposals. On the full runs, the replay reproduces the committed answer in all 4,500 episodes.

Table 14: Induction, exact accuracy at $R = 5$ (`gemini-3.8-flash`, hard, 100 queries per cell). Marks are McNemar tests against QA ($^{*}p < 0.05$, $^{**}p < 0.01$). Bold is the best loop in each column. The last row is the paired difference between the two model checkers, tested against each other. *Tokens* is the mean per query over the six cells.

	$L = 10$		$L = 20$		$L = 30$			
	$k = 3$	$k = 5$	$k = 3$	$k = 5$	$k = 3$	$k = 5$	mean	tokens
QA	0.25	0.29	0.37	0.31	0.35	0.32	0.32	18k
V-whole	0.49**	0.49**	0.51*	0.47**	0.40	0.31	0.45	219k
V-split	**0.57****	0.50**	**0.67****	0.50**	0.59**	0.52**	0.56	1.14M
V-sim	0.56**	**0.53****	0.65**	**0.60****	**0.61****	**0.53****	0.58	69k
V-split−V-whole	+0.08	+0.01	+0.16**	+0.03	+0.19**	+0.21**		

Table 15: Abduction, exact accuracy at $R = 5$ (`gemini-3.8-flash`, hard, 100 queries per cell). Conventions as in Table 14.

	$L = 10$			$L = 20$			$L = 30$				
	$k = 1$	3	5	$k = 1$	3	5	$k = 1$	3	5	mean	tokens
QA	0.17	0.18	0.27	0.08	0.20	0.24	0.08	0.19	0.23	0.18	34k
V-whole	0.35**	0.27*	0.29	0.18**	0.20	0.25	0.15*	0.20	0.22	0.23	281k
V-split	0.34**	**0.31****	0.27	**0.22****	0.20	**0.26**	0.13	**0.21**	**0.24**	0.24	878k
V-sim	**0.36****	0.27*	**0.30**	0.21**	**0.21**	0.25	**0.18****	0.20	**0.24**	0.25	187k
V-split−V-whole	−0.01	+0.04	−0.02	+0.04	+0.00	+0.01	−0.02	+0.01	+0.02		

Refinement against Resampling. The first proposal of every loop uses the same prompt as the single call, so each query has four independent samples from the same distribution. The share of queries on which any of them is right is what the best of four independent calls would score if an oracle picked it. On induction this share is 0.33–0.48 per cell (0.40 on average). The first four proposals of a loop contain a right rule 0.47–0.64 of the time (0.54–0.55 on average for every checker), and at $R = 5$ V-sim and V-split commit more than the oracle over independent samples in every cell (Table 16). The pool is nearly the same under all three checkers, even though the disagreement sets of V-whole are often wrong. On abduction, the pool beats resampling by up to 0.11 at $k = 1$. At $k \in \{3, 5\}$ the two agree within 0.02, except at $L = 10$, $k = 3$ (up to 0.06). The last three columns separate generation from selection. V-sim and V-split commit nearly everything in their pool, whereas V-whole drops 0.13–0.24 of queries at $L \geq 20$ on induction.

Grading the Checkers. Table 17 grades every checked candidate against the benchmark. On induction, all three checkers accept about 2% of wrong rules. For V-sim these are rules that carry s_t exactly onto s_{t+k} but differ from the true rule on a bit exercised only at an intermediate step. No checker that tests consistency with the observation can rule them out, so this rate is a floor. As a share of what each checker accepts, wrong rules make up 8% for V-sim, 12% for V-split and 18% for V-whole. The checkers differ mainly in the right rules they reject. Each model checker accepts

Table 16: Where the gain of the loop comes from (`gemini-3.8-flash`, hard, 100 queries per cell). *Independent*: the four first proposals of QA and the three loops, which share one prompt; *one* is the mean accuracy of a single sample, *best of 4* the share of queries on which any of the four is right. *Pool*: share of queries with a right candidate among the first four proposals of the loop. *Committed*: exact accuracy at $R = 5$, with the pool at $R = 5$ minus the committed accuracy in parentheses.

			independent		pool after 4 proposals			committed at $R = 5$ (pool − committed)		
mode	L	k	one	best of 4	V-whole	V-split	V-sim	V-whole	V-split	V-sim
induction	10	3	0.23	0.33	0.47	0.53	0.51	0.49 (0.05)	0.57 (0.01)	0.56 (0.00)
	10	5	0.28	0.35	0.50	0.51	0.50	0.49 (0.07)	0.50 (0.02)	0.53 (0.00)
	20	3	0.35	0.44	0.60	0.62	0.64	0.51 (0.14)	0.67 (0.00)	0.65 (0.00)
	20	5	0.33	0.40	0.56	0.54	0.56	0.47 (0.13)	0.50 (0.05)	0.60 (0.00)
	30	3	0.36	0.48	0.57	0.57	0.57	0.40 (0.19)	0.59 (0.02)	0.61 (0.00)
	30	5	0.32	0.40	0.52	0.53	0.52	0.31 (0.24)	0.52 (0.04)	0.53 (0.00)
abduction	10	1	0.19	0.29	0.35	0.32	0.35	0.35 (0.00)	0.34 (0.00)	0.36 (0.00)
	10	3	0.21	0.25	0.27	0.31	0.27	0.27 (0.02)	0.31 (0.00)	0.27 (0.00)
	10	5	0.28	0.30	0.29	0.29	0.30	0.29 (0.00)	0.27 (0.02)	0.30 (0.00)
	20	1	0.07	0.09	0.17	0.20	0.17	0.18 (0.02)	0.22 (0.00)	0.21 (0.00)
	20	3	0.19	0.21	0.20	0.20	0.21	0.20 (0.00)	0.20 (0.00)	0.21 (0.00)
	20	5	0.25	0.26	0.26	0.26	0.25	0.25 (0.01)	0.26 (0.00)	0.25 (0.00)
	30	1	0.08	0.12	0.15	0.12	0.16	0.15 (0.00)	0.13 (0.00)	0.18 (0.00)
	30	3	0.19	0.20	0.20	0.21	0.20	0.20 (0.00)	0.21 (0.01)	0.20 (0.00)
	30	5	0.22	0.23	0.24	0.24	0.24	0.22 (0.02)	0.24 (0.00)	0.24 (0.00)

Table 17: How the checkers treat right and wrong candidates (`gemini-3.8-flash`, hard), counting each distinct candidate once. For each model checker: the number of right induction candidates it checked, the share it accepted, and the deduction accuracy of the same layout (whole grid, or 5×5+t) from Section 5.2, on the same items (*same*) and over the whole cell (*cell*). The last three columns are the share of wrong candidates accepted. The bottom rows pool all cells; for abduction the check does not reproduce a deduction query of the benchmark.

		V-whole				V-split				wrong accepted		
L	k	right	accepted	ded. same	ded. cell	right	accepted	ded. same	ded. cell	V-whole	V-split	V-sim
10	3	63	0.62	0.57	0.61	65	0.80	0.71	0.77	0.03	0.03	0.04
10	5	71	0.49	0.42	0.53	62	0.58	0.71	0.70	0.01	0.01	0.01
20	3	83	0.27	0.29	0.32	84	0.58	0.57	0.62	0.02	0.02	0.02
20	5	71	0.25	0.23	0.25	65	0.57	0.48	0.50	0.01	0.02	0.02
30	3	74	0.22	0.19	0.21	73	0.45	0.41	0.48	0.01	0.01	0.01
30	5	65	0.20	0.26	0.21	67	0.46	0.48	0.42	0.01	0.02	0.02
all induction		427	0.33	0.32	–	416	0.57	0.56	–	0.02	0.02	0.02
all abduction		219	0.93	–	–	230	0.80	–	–	0.00	0.00	0.00

a right rule about as often as its layout answers the corresponding deduction query (0.33 against 0.32 for V-whole, 0.57 against 0.56 for V-split, pooled over the same items). On abduction, the model checkers accept 0.93 (V-whole) and 0.80 (V-split) of valid pre-images. Of more than 9,000 wrong candidates across the three checkers, one was accepted. Checking is therefore not what limits abduction.

The Abduction Proposer Moves Away after a Refutation. Figure 8 follows the episodes in which all five proposals were refuted, so every point averages the same episodes. The first proposal agrees with s_{t+k} on 0.76–0.78 of the cells after its forward run. The second, made after seeing the first and its disagreement set, agrees on only 0.68–0.71, and later proposals do not recover. Only 7% of abduction proposals are valid pre-images, and 3% after the first. 14% of later proposals repeat an earlier candidate, against 1% on induction. The model does not use the disagreement set to repair a pre-image. It starts again, and it starts from a worse place.

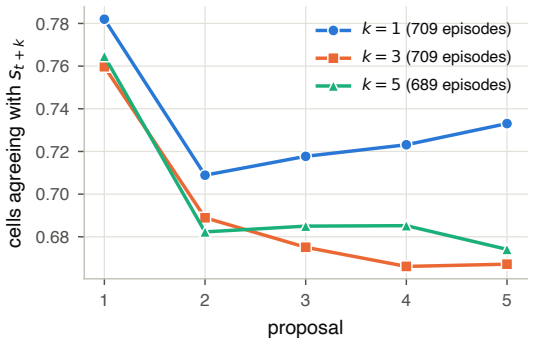

Figure 8: Abduction proposals in episodes that used all five (no checker accepted any of them), pooled over checkers and L: mean share of cells on which the forward run of the proposal agrees with the observed s_{t+k}.

Table 18: Tokens spent, by experiment and arm. *Thinking* is the share of output tokens that were never shown (reasoning). Per-query and per-call figures are totals over all tokens.

arm	queries	calls	tokens	thinking	per query	per call
QA baseline (3 models)	40,500	40,500	925M	96%	23k	23k
Split: time	600	2,390	41M	89%	68k	17k
Split: 10×10	400	2,100	32M	95%	80k	15k
Split: 10×10 + time	400	10,400	75M	91%	186k	7k
Split: 5×5	300	5,600	23M	87%	77k	4k
Split: 5×5 + time	600	44,800	159M	86%	266k	4k
Verify: V-sim	1,500	5,517	210M	96%	140k	38k
Verify: V-whole	1,500	12,045	384M	95%	256k	32k
Verify: V-split	1,500	352,867	1,477M	88%	985k	4k
Total	47,300	476,219	3.33B	91%		

E.4 Cost

All experiments together used 3.33B tokens over 476,219 calls, and 91% of the output tokens were thinking (Table 18). A 5×5+t call averages 4k tokens against 23k for a single QA call, but a query needs up to 180 of them. Relative to the single call, the loop costs the following per query:

- with V-sim, 3.3–6.4× the tokens;
- with V-whole, 5.3–17×;
- with V-split, 6.7–125×, reaching 2.6M tokens per query at induction $L = 30$, $k = 5$.

V-split alone accounts for 1.48B of the 2.07B tokens spent on propose–verify. Because a smaller budget is a prefix of the same run, Figure 9 also shows the trade-off along R. On induction, $R = 3$ keeps most of the gain at about two thirds of the tokens of $R = 5$, under every checker.

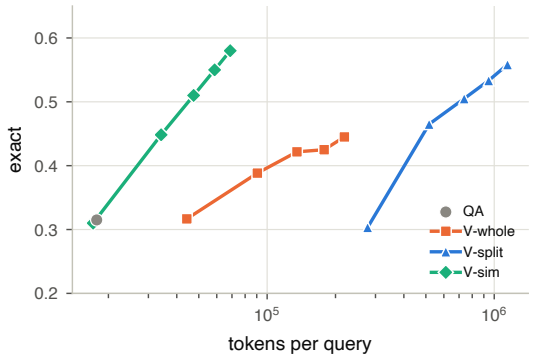

Figure 9: Induction loops: exact accuracy against tokens per query at $R = 1, \ldots, 5$, averaged over the six cells. V-sim is both the most accurate and the cheapest loop, since its check costs no tokens; V-split reaches nearly the same accuracy at far higher cost. The deduction layouts are in Figure 3f.