BRIDGING THE KNOWLEDGE–REASONING GAP IN NATURAL SYSTEMS: A BENCHMARK FOR INDUCTIVE RULE AND NOISE RECOVERY IN CELLULAR AUTOMATA

Anonymous authors
Paper under double-blind review

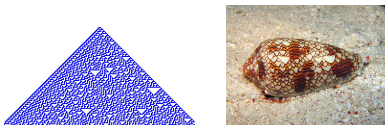
ABSTRACT

Noise is an inherent part of any natural system. After considering the recent success of transformer-based models in natural system reasoning, this research explores the capability of large language models (LLMs) in addressing the question: given an example of a real-world natural system, can LLMs identify the presence of noise within it? Here, we consider the natural computing model cellular automata (CA) as a proxy of natural systems. Therefore, we formalize this question as the CA *identification problem*: given noisy evolution dynamics, can LLMs recover both the underlying rule and the magnitude of noise? To explore this question, we construct four benchmark datasets (skewECA, αECA, stocECA, and tempECA) totalling $2{,}876{,}800$ samples across four noise types, along with knowledge-check dataset. This question also captures the reasoning capabilities of LLMs, i.e. reasoning from observation. Evaluating Qwen, Llama, and Mistral/Mixtral families (7–72B parameters), alongside GPT 5.1 and Claude Haiku 4.5 under zero-shot, few-shot, and fine-tuned settings (for open source models), we observe near-perfect knowledge accuracy but poor task performance. The rule prediction peaks at 14.60% and noise prediction at 32.50%. This gap is architectural. Next-token prediction does not equip models to extract the global and structural statistics required for this task, including change rates, spatial contiguity, and distributional variance. We therefore design four specialised transformer models (skewM, αM, stocM, and tempM), each aligned with the statistical signature of its corresponding noise process. These models achieve up to 99.22% rule exact match and 96.97% noise tolerance accuracy, outperforming the best LLM baseline by up to $6.8\times$ with at most 6.02M parameters. These findings emphasize the importance of structure-aware architectures for reasoning over spatiotemporal data, as exemplified by the identification problem. Code and Dataset are available at `https://anonymous.4open.science/r/LLM_NOISE-9DC1/`.

1 INTRODUCTION

The past several years have produced a striking body of evidence suggesting that large language models (LLMs) are capable of meaningful engagement with scientific domains. Transformer-based models have demonstrated strong performance in understanding natural systems which include physical (Xu et al., 2024; Ma et al., 2024), chemical (Boiko et al., 2023; Zhang et al., 2025a;b), biological (Dalla-Torre et al., 2025; Lin et al., 2023; Theodoris et al., 2023), and geological (Bi et al., 2023; Lam et al., 2023) systems. Building on this evidence, this article revisits the reasoning capability of LLMs in the context of natural systems. To explore this agenda, we begin with Alan Turing, not to emphasize his foundational contributions to modern computing but to foreground his seminal insights on morphogenesis. In 1952, he observed that a system, *"although it may originally be quite homogeneous, may later develop a pattern or structure due to an instability of the homogeneous equilibrium, which is triggered off by random disturbances"* (Turing, 1952), placing randomness (or noise) at the centre of the laws that govern life and nature. Accordingly, capturing the behaviour of natural systems necessitates the inclusion of noise or random variability. In this direction, we investigate the following question: *given a natural system, can LLMs identify the presence of noise within it? Furthermore, can they quantify the magnitude of such noise?*

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.



(a) (b)

Figure 1: (a) Pattern of ECA rule 30 (Wolfram, 1984); (b) Presence of rule 30 like pattern in nature (Coombes, 2009).

To address this question, we consider one of the oldest models of natural computing, cellular automata (CA) (Wolfram, 1984). CA have already proven their efficiency in natural system modelling, be it physical (Chopard & Droz, 1998), biological (Deutsch & Dormann, 2017; Ballesteros Hernando et al., 2019), geological (Sahila et al., 2026; Tang et al., 2024) etc. Here, we consider the natural computing model CA as a proxy for natural systems to explore the above question. In CA, the global decision making emerges from local interaction only, which makes it close to nature. Specifically, we consider simplest, elementary cellular automata (ECA) (Wolfram, 1984) where cells talk with three neighbours (self, left and right) to change their local binary states under a one-dimensional ring arrangement. Fig 1(a) depicts the pattern of the famous ECA rule 30 where the local transition follows: $000 \to 0$, $001 \to 1$, $010 \to 1$, $011 \to 1$, $100 \to 1$, $101 \to 0$, $110 \to 0$, and $111 \to 0$; and, the following is an example of evolution: $\begin{smallmatrix} \cdots & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \cdots \end{smallmatrix}$, see Fig 1(a) for larger evolution; and Section 3 for more details. On the other hand, Fig. 1(b) gives evidence of rule 30 like pattern in nature (Coombes, 2009), where one can observe the presence of noise in it.

Now, if someone gives you the pattern of Fig 1(a), and asks the question, which rule (local transition function) is responsible for this pattern? To answer this question, one needs to observe this pattern and identify eight responsible local transition functions (as listed above). In the domain of theoretical computer science, this problem is known as the *identification problem* (Bołt et al., 2020; 2019). Recently, Burtsev (Rodkin et al., 2026; Burtsev, 2024) has shown that transformers efficiently solve this problem because of the deterministic nature of the pattern.

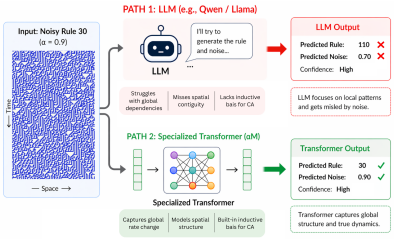


Figure 2: CA rule 30 and α-noise 0.9 is responsible for the input space-time evolution.

Next, the question becomes difficult if someone gives you the natural pattern of Fig. 1(b), and asks the same question. Is it really rule 30 or something like that (not exactly 30)? If it is rule 30, then what amount of noise (rate) is involved in it? Past limited theoretical algorithmic attempts share the difficulty of this identification problem in the presence of noise (Bołt et al., 2020; 2019). This work asks the same question of modern transformer models. To attempt this question, we introduce four benchmark reasoning (spatio-temporal reasoning) datasets by using s-skewed (Roy et al., 2025), α-asynchronous (Bouré et al., 2012), stochastic (Fatès, 2013), and temporally stochastic (Roy et al., 2022) noise in ECA. Before introducing these noises, note that, CA forces all cells to update together following a global clock; moreover, it forces all cells to follow the same local transition function. Here, we introduce noise in ECA system by breaking these two properties; s-skewed and α-noise ask the cells not to follow global clock for update; and, stochastic and temporally stochastic noise introduce multiple (here, two) probabilistic local rules, mixed spatially and temporally, respectively. The dataset contains space-time pictures of ECA rules with the effect of those perturbations. The effect of these perturbations on ECA is non-trivial which causes the evidence of phase transition (belongs to the directed percolation universality class (Bouré et al., 2012)) and class transition (system moves from chaotic to simple dynamics and vice versa (Roy et al., 2022)). Therefore, the task here is: *given a space-time evolution (orbit), identify the local rule(s) and amount of noise*. For example, in Fig. 2, given the input space-time evolution, the LLM needs to answer the responsible rule and α-noise rate, here, rule 30 and $\alpha = 0.9$. This problem statement also captures the reasoning capability of LLMs. Therefore, the proposed four datasets of total size $2{,}876{,}800$ act as benchmarks for reasoning. Recent work shows that LLMs excel in reasoning tasks like multi-step math (Wei et al., 2022; Cobbe et al., 2021; Zhou et al., 2023), zero-shot reasoning (Kojima et al., 2022), and common-sense understanding (Talmor et al., 2019; Sap et al., 2019). However, according to Ettinger (2020); Marvin & Linzen (2018), they often struggle with tasks requiring true structural understanding rather than pattern matching. This benchmark is also designed to test this limitation.

We evaluate open-source LLMs from the Qwen, Llama, and Mistral/Mixtral families (7B–72B), alongside frontier models (Claude Haiku 4.5 and GPT-5.1), on the identification task. To verify domain knowledge, we first use a *knowledge-check dataset* covering ECA dynamics, noise mechanisms, and rule behaviour, where all models achieve at least 91% accuracy. Despite this, performance on the identification task remains poor. Under zero-shot and few-shot prompting, open-source models achieve at most 1.00% rule accuracy and 28% noise accuracy, while frontier models reach 4.00%

and 23.5%, respectively. Fine-tuning improves open-source models only marginally to 14.60% and 32.50%. These results indicate that the limitation is architectural rather than informational. Each noise type induces a distinct orbit-level signature, namely spatial contiguity (s-skewed), global change rate (α), within-row variance (λ), and per-timestep rule assignment (τ); these signatures require inductive biases over CA geometry that standard next-token predictors lack.

This observation motivates the second part of the paper. We design four specialised transformer models, one per noise type, each equipped with architectural components matched to the statistical signature of its noise. The models share a common backbone and two-phase training framework but differ in token features, pooling strategy, and noise estimation. For the single-rule models (`skewM` and αM), we introduce a *hybrid estimation strategy* that combines neural rule identification with closed-form analytical noise estimation, improving mean absolute error (MAE) on αM by approximately $4\times$ over purely neural methods. For the two-rule models (`tempM` and `stocM`), we introduce *analytical noise estimators* refined by learned correction networks, reducing noise MAE by 30–50% relative to the purely analytical baseline. The specialised models substantially outperform all LLM baselines. Reporting rule exact match and noise tolerance accuracy, respectively, skewM achieves 94.55% and 96.97% (±1); αM achieves 99.22% and 95.08% (±0.05); stocM achieves 95.49% and 84.8% (±0.05); and tempM achieves 89.05% and 82.3% (±0.05). All four models, with at most 6.02M parameters, outperform 7–72B parameter LLM baselines, confirming that architectural specialisation rather than scale determines success on this identification problem.

Main Contributions. (1) *The CA identification problem:* We formalise the recovery of the ECA rule(s) and noise parameter from a perturbed orbit as a benchmark for scientific reasoning in LLMs. (2) *Four benchmark datasets:* We construct four datasets (skewECA, αECA, stocECA, and tempECA) with 2,876,800 samples across all 256 ECA rules and disjoint train/test rules. (3) *Specialised transformer models:* As open-source and frontier LLMs fail, we design skewM, αM, stocM, and tempM, outperforming all LLM baselines with at most 6.02M parameters.

2 Related Work

Our work relates to two lines of research: the identification of CA, and the ability of transformers to learn automata and structured data. On the first, recovering a CA rule from observations is a classical inverse problem (Adamatzky, 1994), addressed with statistical estimators for incomplete observations and two-rule mixtures (Bołt et al., 2020; 2019). Neural networks learn CA dynamics (Wulff & Hertz, 1992; Gilpin, 2019) but struggle with exact local rules (Springer & Kenyon, 2021), and transformers trained on clean ECA orbits degrade on multi-step reasoning (Burtsev, 2024; Rodkin et al., 2026). Beyond diploid mixtures, none of these works recovers rules under asynchronous noise, evaluates LLMs, or characterises identifiability; this work does all three. On the second, the ability of transformers to recognise or simulate formal languages and automata is well studied (Bhattamishra et al., 2020; Liu et al., 2023; Strobl et al., 2024), with generalisation governed by inductive bias rather than scale (Delétang et al., 2023). These works execute a known automaton; we infer an unknown rule and its noise from stochastic update. Our models build on established components (2D positional encodings (Dosovitskiy et al., 2021), factorised spatio-temporal attention (Ho et al., 2019), query-based pooling (Lee et al., 2019), statistics pooling (Snyder et al., 2018)) and on hybrid modelling with a known functional form of the dynamics (Brunton et al., 2016; Raissi et al., 2019). We align these components with each noise signature and show, under matched inputs, that this alignment rather than scale separates success from failure.

3 Towards design of dataset

Elementary Cellular Automata. To introduce *identification problem* in CA, we first discuss elementary cellular automata (ECA) (Wolfram, 1984). One can observe, ECA system as a one dimensional array of size n. Each cell of the array is associated with a state; here we allow two states $\{0, 1\}$. The states of all the cells at a given time are known as a *configuration*. For example ($n = 8$), say at time $t = 1$, the configuration is $< 01011001 >$. In the next time step, the cells change their state following self, left and right neighbours state. Here, the following eight neighbouring combinations are possible: $000, 001, 010, 011, 100, 101, 110, 111$. Now, the *output* of the eight neighbouring combinations defines the *rule* (say, R) of the ECA, for example, say, $\overset{111}{0}\ \overset{110}{0}\ \overset{101}{0}\ \overset{100}{1}\ \overset{011}{1}\ \overset{010}{1}\ \overset{001}{1}\ \overset{000}{0}$.

For naming purposes, we use the decimal equivalent value of these eight outputs, here, this example depicts rule 30. Now, after considering rule 30, at time $t = 2$, the configuration $< 01011001 >$ moves to $< 01010111 >$; here, we consider ring arrangement, left (resp. right) neighbour of the leftmost (resp. rightmost) cell is the rightmost (resp, leftmost) cell. Here, we run this simple code of ECA for t time steps starting from an initial configuration, and it hops from one configuration to another configuration and creates the *space-time picture* for the corresponding rule; see Fig 1(a) for the space-time picture of ECA 30 starting from initial configuration $< \cdots 00100 \cdots >$. Under this setting of eight neighbouring combinations and corresponding different possible outputs, we have $2^8 = 256$ ECA rules. Formally, let a_i^t represent the state of cell $i \in \mathcal{L} = \mathbb{Z}/n\mathbb{Z}$ at time t; hereafter, the state of cell i at next $t + 1$ time step defined as $a_i^{t+1} = f(a_{i-1}^t, a_i^t, a_{i+1}^t)$. Here, $f : \{0, 1\}^3 \to \{0, 1\}$ reports the local transition function. Given a time t, collection of local states $< a_1^t, a_2^t, \cdots, a_n^t >$ depicts the configuration, i.e., global transition function $G : \{0, 1\}^{\mathcal{L}} \to \{0, 1\}^{\mathcal{L}}$.

Noise. Here, we use the following four perturbation schemes to introduce noise in ECA: (i) In traditional ECA, we update all cells at each time step after considering the presence of a global clock. Under the s-skewed noise (Roy et al., 2025), we randomly and uniformly select a cell, say i, and choose cells $i, i+1, \cdots, i+(s-1)$ for update at each time step t. Here, $s \in [1, n]$ depicts the perturbation rate. (ii) For α-noise (Bouré et al., 2012), each cell gets the chance for update with α probability, and remains idle with $1 - \alpha$ probability at each time step t; the noise rate is $\alpha \in [0, 1]$. Both s-skewed and α introduce noise after breaking the presence of the global clock. (iii) Recall that the traditional ECA dictates all the cells to follow uniform rule R. Under (spatial) stochastic noise (Fatès, 2013), we consider the presence of two rules, say R_1 and R_2, and their mixing noise rate $\lambda \in [0, 1]$. Therefore, given a time t and a cell $i \in \mathcal{L}$, the update of cell i follows rule R_1 with $1 - \lambda$ probability and considers rule R_2 with λ probability; denoted by $(R_1, R_2)[\lambda]$. (iv) For the temporally stochastic noise (Roy et al., 2022), we follow similar presence of two rules, say R_1 and R_2, and their mixing noise rate $\tau \in [0, 1]$. However, the noise is temporal in nature. Here, given a time t, we consider rule R_1 for all cells with $1 - \tau$ probability; and all cells follow rule R_2 with τ probability; denoted by $(R_1, R_2)[\tau]$. Fig. 3 provides the graphical visualization of these four noises. Additionally, application of these noises are not-trivial in the context of statistical physics; Fig. 4 exhibits the evidence of phase transition (Bouré et al., 2012) with the effect of α-noise after considering ECA 18; and under stochastic noise, pair $(131, 62)$ shows the evidence of class transition (Roy et al., 2022) (moving from periodic to noisy dynamics, see Fig. 4).

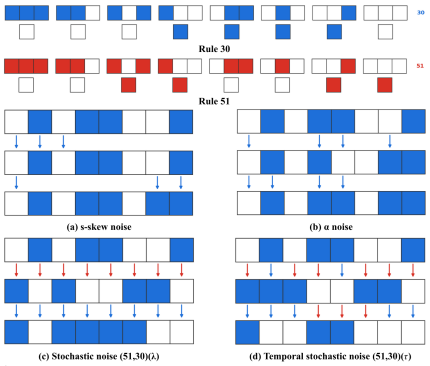


Figure 3: Local transition function for Rule 30 and Rule 51 (top), and different noise mechanisms: s-skew (rule 30, $s = 3$), α (rule 30, $\alpha = 0.5$), stochastic $(51, 30)[\lambda]$, and temporal stochastic $(51, 30)[\tau]$ (bottom). Application of rule 30 and 51 denoted by blue and red arrow.

Identification problem. Here, the identification reasoning task is the following: *given a noisy space-time picture, we need to reason about the corresponding ECA rule and noise rate (i.e., skew s or α or λ or τ).* For evidence, in Fig. 2, the input space-time picture is given, this picture is responsible for ECA 30 and $\alpha = 0.9$ which need to be returned back as an output. Indeed, Fig. 4 reflects the evidence of difficulty of this identification problem.

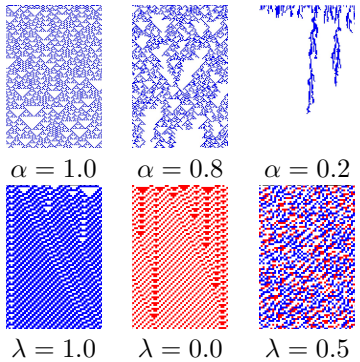


Figure 4: (top) evidence of phase transition for ECA 18; (bottom) evidence of class transition for pair $(131, 62)$.

Datasets. We introduce eight benchmark datasets in two categories: four *knowledge-check* datasets and four *identification* datasets. The knowledge-check datasets (skewK, αK, stocK, tempK) are designed to verify that any failure on the identification task reflects a structural limitation rather than missing domain knowledge. Each dataset contains 200 questions in two formats, multiple-choice (MCQ) and short-answer, covering definitions, update mechanics, and mathematical properties of the corresponding noise model. Questions span three difficulty levels, ranging from definitional recall to quantitative reasoning. Difficulty distributions and examples are in Appendix A.1.

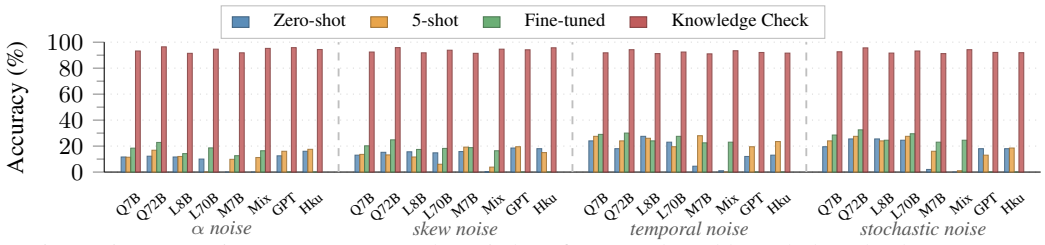


Figure 5: LLM noise accuracy (zero-shot, 5-shot, fine-tuned) and knowledge-check accuracy.

The identification datasets (αECA, skewECA, stocECA, tempECA) form the core benchmark. Each sample presents a model with a raw space-time orbit and requires it to recover the underlying ECA rule and the noise parameter. All four datasets share a common rule partition: the 256 ECA rules are permuted under a fixed seed (seed = 42), with 179 assigned to training and 77 to testing. The two sets are strictly disjoint and held constant across all datasets, ensuring that cross-model comparisons reflect model capability rather than data variation. Note that Wolfram (Wolfram, 1984) classified ECA rules into the following four classes based on pattern complexity: null (Class I), periodic (Class II), chaotic (Class III), and complex (Class IV). Here, the 70/30 split preserves coverage across all four Wolfram behavioural classes (Wolfram, 1984); see Appendix A.3 for the exact rule partition and class distribution. The αECA dataset targets α-asynchronous noise with $\alpha \in \{0.1, 0.2, \cdots, 1.0\}$, using grid width $n = 20$ and time step $t = 100$, yielding $(t-1) \times n = 1{,}980$ tokens per orbit. This space-time orbit ensures repeated exposure to all local transition functions and sufficient observations for stable per-transition statistics under single-noise inversion; rule identifiability varies smoothly with grid size and orbit length around this choice (Appendix D.7). The dataset has a synchronous Phase 1 ($\alpha = 1.0$; 35,800 training / 7,700 test samples) and a Phase 2 stratified with 50 training and 20 test samples per (rule, α) pair (89,500 / 15,400), for 125,300 training and 23,100 test samples in total. The skewECA dataset targets s-skewed noise with $s \in \{1, \cdots, 20\}$, using the same grid size ($n = 20$) but a longer time step $t = 200$, yielding 3,980 tokens per orbit. Here, longer evolution is necessary because only a small fraction of cells get updated per time step if noise rate s is low. As in αECA, this dataset also follows a two-phase structure: a synchronous Phase 1 ($s = n$) and a Phase 2 with $s \in \{1, \ldots, 20\}$ and 25 training and 10 test samples per (rule, s) pair, resulting in identical totals of 125,300 training and 23,100 test samples. The stocECA and tempECA datasets introduce a fundamentally harder setting in which each sample is generated from a mixture of two rules. stocECA applies stochastic mixing at the cell level ($\lambda \in [0.1, 0.9]$), while tempECA applies temporal switching at the row level ($\tau \in [0.1, 0.9]$), requiring the model to disentangle a rule pair (R_1, R_2) from observed dynamics. To enable reliable estimation, these datasets use wider grids ($n = 50$) and longer evolution ($t = 200$), ensuring that each local transition function is observed. Moreover, we run the same experiment (consider single rule pair and noise rate) for $k = 8$ initial configurations. Training data is generated on-the-fly: at each epoch, $1,000$ rule pairs are sampled from the training set with 8 samples per pair, yielding 8,000 training samples per epoch (see Appendix A.4 for generation details). Sampling is stratified across all Wolfram class combinations to ensure coverage of varying difficulty levels; stratification counts and verification are provided in Appendix A.6. Test sets are pre-generated and fixed at 500 pairs $\times$ 20 samples $= 10{,}000$ samples each (Appendices A.2 and A.5).

4 LLM Evaluation

We evaluate six open-source instruction-tuned LLMs, namely Qwen2.5-7B/72B-Instruct (Yang et al., 2024), Llama-3.1-8B/70B-Instruct (Grattafiori et al., 2024), Mistral-7B-Instruct-v0.3 (Jiang et al., 2023), and Mixtral-8 $\times$ 7B-Instruct-v0.1 (Jiang et al., 2024), alongside two frontier models (GPT-5.1, Claude Haiku 4.5) on the ECA identification task (see Fig. 5 for results). To isolate whether failures stem from missing knowledge, we first evaluate all models on knowledge-check datasets (`skewK`, α`K`, `stocK`, `tempK`). All models achieve at least 91% accuracy, confirming sufficient conceptual understanding, yet identification performance is poor. Under zero-shot prompting, open-source models achieve at most 1.0% rule accuracy and 27.5% noise accuracy. Few-shot prompting ($k = 5$) provides no meaningful improvement, and fine-tuning helps only modestly, to 14.6% rule (95% CI $[11.6, 18.0]$) and 32.5% noise accuracy. Frontier models show similar behaviour: zero-shot rule accuracy peaks at 4.0% (GPT-5.1), with noise accuracy up to 18.5%; few-shot marginally improves

noise accuracy to 23.5% while degrading rule accuracy. On the two-rule datasets, every model remains near 0.00% rule accuracy. Because each two-rule sample contains $k = 8$ orbits of 200×50 cells, these prompts display the first 50 rows of one orbit to fit the context window of the smaller models; showing all eight models the complete input used by stocM and tempM, with unrestricted reasoning, still yields 0.00% rule accuracy on both datasets, and fine-tuning the open-source models on this complete input raises rule accuracy to at most 6.5%, with noise accuracy of at most 32.5%, close to the $\approx 25\%$ obtained by a constant prediction (Appendix F.9). On αECA, the most favourable dataset for the LLMs, we further lift every restriction of the protocol: allowing chain-of-thought reasoning, a Python interpreter, or both leaves seven of the eight models at $\leq 2\%$ rule accuracy, and only GPT-5.1 improves, to 34% with reasoning and tools combined. Replacing the raw orbit with a compact per-pattern transition summary, the sufficient statistic read by a zero-training lookup that itself reaches 92.57%, raises GPT-5.1 to 60%, still far below both that lookup and the 99.22% of αM (Appendix F.10). Presenting the orbit as an image instead, to the same frontier models used as vision-language models, yields 0% rule accuracy for both (Appendix F.11). Across scale, prompting, fine-tuning, reasoning, tool use and input representation, the gap does not close: the failure is not due to lack of knowledge but to an inability to extract global and structural signals from spatio-temporal data, which motivates the structure-aware architectures of Section 5 (details in Appendix F).

5 ARCHITECTURE

Each noise type leaves a distinct statistical signature in the orbit: a global change rate for α-asynchronous noise, a spatial contiguity pattern for s-skewed noise, a per-timestep rule assignment for temporally stochastic noise, and a within-row output variance for stochastic noise. Detecting these signals requires reasoning over the orbit as a whole. Standard architectures are insufficient for this purpose. To empirically validate this, we evaluate three baseline models on the identification task using αECA dataset only, the simplest noise setting in our benchmark: a BiLSTM (for sequential modelling), a 2D-CNN (for local spatial pattern extraction), and an MLP on handcrafted orbit statistics (for global feature-based reasoning). The BiLSTM achieves 10.3% rule and 34.7% α accuracy, the CNN 71.7% and 57.0%, and the MLP 87.4% and 64.2%, showing the limits of sequential processing, local receptive fields and fixed features, and the need to model dependencies across the entire orbit. A transformer (Vaswani et al., 2017), whose self-attention gives every token a global view, is a natural candidate. However, a Vanilla Transformer, differing from our model only in positional encoding and pooling, reaches only 70.1% rule and 55.6% α accuracy, indicating that global attention alone is insufficient without appropriate inductive biases. A causal decoder-only transformer trained from scratch with next-token prediction on the raw orbit, which is the architecture class of LLMs, at matched scale (≈1.05M parameters) and on the same training data, reaches only 44.4% rule and 42.9% α accuracy. A Vision Transformer trained from scratch on the orbit rendered as an image reaches only 2.3%: with one patch per cell, each token carries a single state, so every rule application must be reassembled from four tokens across two rows via learned positional embeddings, whereas a cell-transition token or a CNN kernel contains the application directly. With data and training held fixed, rule accuracy thus ranges from 2.3% to 99.22% (our model αM) across architectures and input units (Appendix D). Capacity does not close the gap either: scaling the CNN 53.6× (to 6.04M parameters) and the Vanilla Transformer 15.6× (to 12.7M) plateaus at 80.7% and 87.2% rule accuracy, 18.6 and 12.1 points below αM (Appendix D.6). Since αECA is the simplest setting, neither sequential, convolutional nor generic transformer architectures can be expected to capture the orbit's global statistical structure on the harder noise types. We therefore design specialised transformers with inductive biases aligned with each noise type. The four models share two encoder templates, a flat encoder for the single-rule models and a two-level encoder for the two-rule models, with a common positional encoding (TripletPE2D), curriculum and training conventions; only the token features and a small head matched to each noise signature differ (Table 1). Replacing the matched components of αM and skewM with a 1D positional encoding and mean pooling costs 29.1 and 14.2 points of rule accuracy (Appendix D.8). Standard engineering choices (PreNorm, GELU, dropout 0.1, gradient clipping, weight initialisation) are shared across all models (Appendix E).

Shared: TripletPE2D. Each space-time orbit $\mathbf{O} \in \{0,1\}^{t \times n}$ is tokenised into $(t-1) \times n$ cell-transition tokens, where, for each timestep t and cell index i, the token is defined as the tuple $(a_{i-1}^t, a_i^t, a_{i+1}^t, a_i^{t+1})$, consisting of a 3-bit neighbourhood and its 1-bit output, i.e., a complete local rule observation. Standard 1-D positional encoding over the flattened sequence conflates temporal and spatial structure, placing tokens from different timesteps and cells in adjacent positions

despite encoding distinct contexts. We therefore introduce *TripletPE2D*, which assigns independent sinusoidal encodings along the temporal and spatial axes (64 dimensions each), concatenated into a 128-dimensional positional embedding. This separation enables the transformer to attend over tokens sharing a timestep (spatial snapshots) and those sharing a cell index (temporal trajectories) independently, both of which encode complementary information about the rule and the noise.

αM: single-rule, single-noise. For α-noise, $n = 20$ and $t = 100$ yield $1,980$ tokens per orbit. The model uses a 4-layer transformer encoder ($d = 128$, 4 heads, FF$= 512$, ≈ 1.76M parameters). After encoding, the architecture forks into two streams. The *rule stream* mean-pools all tokens into an MLP ($128 \to 64 \to 8$) with BCE loss. Mean pooling is appropriate because the rule signature is distributed uniformly across all tokens. The α *stream* uses *StatPool*, a statistics pooling operation adapted to noise estimation. It concatenates the mean, standard deviation, maximum and minimum of the token representations into a 512-dimensional summary, fed to an MLP ($512 \to 256 \to 128 \to 10$) with cross-entropy loss over the 10 discrete α values. StatPool captures the spread of activations, which directly reflects how frequently the rule is violated, and thus provides a signal for α. At inference, the rule prediction from the rule stream fixes which cells the rule would change; the fraction of these cells that actually changed yields a closed-form estimate $\hat{\alpha}$, taken as the median over 20 orbits. The resulting *hybrid estimator*, introduced in this work, replaces the neural head at inference on all but one of the 2,310 held-out αM and skewM test pairs and provides higher precision than the discretised classifier; the neural head alone is reported in Section 7. This hybrid strategy is feasible for single-rule settings (αM, skewM), but not for two-rule mixtures (tempM, stocM), where the individual rule bits cannot be isolated by a closed-form inversion of the mixture statistics.

skewM: spatial contiguity. For s-skewed noise, each token receives a fifth binary feature $\text{did_change} = \mathbf{1}[a_i^{t+1} \neq a_i^t]$, making contiguous update blocks directly observable rather than implicitly reconstructable. With $n = 20$ and $t = 200$, the model has $3{,}980$ five-dimensional tokens. The encoder is identical to αM (4 layers, $d = 128$, $\approx$1.77M parameters). The s stream uses *SpatialMultiScalePool*, introduced in this work. It aggregates three parallel branches: a global statistics branch (mean and standard deviation over all tokens), a temporal statistics branch (per-timestep change rate statistics), and a spatial statistics branch (1-D convolution across adjacent cell tokens within each timestep to capture contiguous-block structure). These are concatenated and fed to a dual head: a 20-class classifier (cross-entropy) and an auxiliary regression head (MSE on s/n) that enforces ordinal structure over s. The hybrid estimator divides the number of changed cells by the number the predicted rule would change, scales by n, and takes the median over orbits.

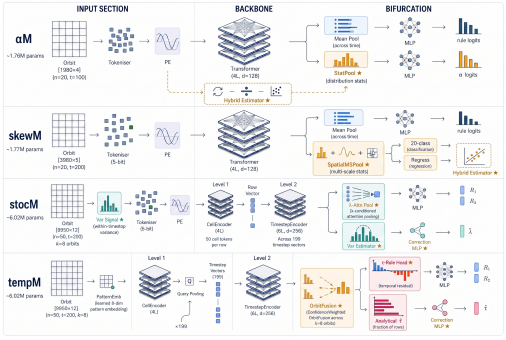


Figure 6: Architecture overview.

tempM and stocM: two-rule, hierarchical. Recovering two rules from a mixed orbit is a harder identification problem. A flat token sequence of length $9{,}950$ per orbit ($(t-1) \times n = 199 \times 50$) with $k = 8$ orbits per sample would make global attention prohibitively expensive. Both models therefore use a hierarchical two-level encoder ($\approx$6.02M parameters). The *CellEncoder* processes the 50 cell tokens within each timestep independently, compressing each row to a single $D_1 = 128$-dimensional vector via 4 transformer layers and a learned query-based pooling. The *TimestepEncoder* then runs 6 transformer layers ($D_2 = 256$, 8 heads, FF = 1,024) across the resulting sequence of 199 timestep vectors, building representations that span the full temporal extent of the orbit. A *ConfidenceWeightedOrbitFusion* module learns soft weights over the $k = 8$ independent orbits per sample, upweighting orbits whose representations are more informative about the rule pair. Each token also receives a learned 8-dimensional *PatternEmbedding* that maps the neighbourhood index $j \in \{0, \ldots, 7\}$ to a continuous vector, providing a direct shortcut for the per-pattern mixture signal. The noise parameters (τ for tempM, λ for stocM) are recovered from per-pattern output statistics via a closed-form estimator refined by a learned correction MLP. For tempM, the per-timestep residual $r_t = \bar{a}^{t+1} - \hat{f}(\text{patterns}_t)$, where $\bar{a}^{t+1} = \frac{1}{n}\sum_i a_i^{t+1}$ and patterns_t denotes the set of neighbourhood patterns indexed by j at timestep t, is positive when rule R_2 fires and negative when rule R_1 fires. This provides a direct temporal assignment signal that feeds a *TauConditionedRuleHead* to disentangle the two rules. For stocM, this residual is identically zero (every timestep mixes both rules at the cell level), so the model

instead exploits the within-timestep variance $\mathrm{Var}[a_i^{t+1} \mid \mathrm{pat} = j] = \lambda(1-\lambda)(R_2[j] - R_1[j])^2$, a second estimator of λ unavailable in tempM. The rule head for stocM uses λ-conditioned attention pooling over all 199 timestep representations rather than per-timestep assignment, reflecting the absence of temporal rule-switching structure. The key architectural differences are in Appendix B.

6 TRAINING

Two-phase curriculum. Training the joint (rule and noise) identification problem is inherently circular: estimating the noise parameter requires knowledge of the exact rule, while learning the rule is harder in the presence of noise. We resolve this through a two-phase curriculum shared across all four models. Phase 1 trains on noise-free or restricted-noise orbits: $\alpha = 1.0$ for αM, $s = n$ for skewM, noise range $[0.3, 0.7]$ for tempM and stocM. Here, the noise head receives zero gradient for αM and skewM which allows the encoder to build stable rule representations before noise is introduced. For tempM and stocM, Phase 1 does not set noise to zero; instead, it restricts the noise range to $[0.3, 0.7]$. For tempM and stocM, noise is kept because removing it would collapse a two-rule mixture to a single rule. For tempM and stocM, an inner noise-range curriculum expands the sampling range from $[0.3, 0.7]$ to the full $[0.1, 0.9]$ at epoch 11, once both rules in a pair are reliably distinguishable. The initial range $[0.3, 0.7]$ is chosen to ensure moderate noise where both rules are sufficiently active and visible to the model; the full range $[0.1, 0.9]$ includes extreme regimes where one rule dominates and the other becomes largely inactive. This inner curriculum is unnecessary for αM and skewM because a single underlying rule remains identifiable at all noise levels. Phase 2 loads the best Phase 1 checkpoint, activates the noise loss, and trains both heads jointly. The rule representations established in Phase 1 now serve as a stable reference. The noise head then estimates the noise parameter θ as the orbit's deviation from the identified rule. Details are in Appendix C.3.

Loss design. Each loss component is matched to the structure of its prediction target. All four models use BCE over the 8 rule bits, each an independent binary decision. For tempM and stocM, which predict a rule pair, the rule loss is evaluated symmetrically over both orientations $(\hat{r}_1, \hat{r}_2) \to (r_1, r_2)$ and $(\hat{r}_1, \hat{r}_2) \to (r_2, r_1)$, with the lower loss used. This reflects the $(R_1, R_2, \theta) \equiv (R_2, R_1, 1-\theta)$ symmetry. For noise estimation, αM uses cross-entropy over 10 discrete α values. skewM adds an MSE term on s/n, providing an ordinal inductive bias toward the correct value of s. tempM and stocM use MSE targeting $\min(\theta, 1-\theta)$, supervised separately from the rule loss to decouple orientation and magnitude. tempM additionally supervises which rule fired at each timestep. This is absent in stocM because both rules fire simultaneously at the cell level, making per-timestep labels undefined. Loss weights are given in Table 13 (Appendix C).

Optimiser. All models use AdamW with weight decay 10^{-2} and cosine annealing. tempM and stocM add a linear warmup (100 steps in Phase 1; 30 steps in Phase 2) to stabilise optimisation under noisier gradients from on-the-fly data generation and larger model size. In Phase 2 of tempM and stocM, the correction MLP receives a separate learning rate of 10^{-5}, lower than the main model rate of 3×10^{-5}, so that it only refines the analytical estimate. All models use an effective batch size of 128 via gradient accumulation. The complete training configuration is provided in Appendix C.

7 RESULTS AND DISCUSSION

Metrics. Rule accuracy is exact match over all 8 bits. For tempM and stocM, we use Unordered-Pair (UO) accuracy: both rules must match in either orientation, since $(R_1, R_2, \theta) \equiv (R_2, R_1, 1-\theta)$ is the same system. Noise accuracy uses a tolerance of ± 0.05 (± 1 cell for skewM). For tempM and stocM, noise MAE is symmetric, $\min(|\hat{\theta} - \theta|, |\hat{\theta} - (1-\theta)|)$. Per-noise-level results are in Appendix G. As test rules are unseen, we quantify uncertainty with a rule-level bootstrap: the 77 held-out rules are resampled with replacement 10,000 times and every metric is recomputed (95% CIs in Table 1).

Main findings. (i) *Specialised transformers decisively outperform 7–72B LLMs.* Table 1 reports the performance of our four specialised transformer models on held-out rules, and the contrast with LLM baselines is stark. The best LLM under any regime reaches only 14.60% rule accuracy and 32.50% noise accuracy (Appendices F.4, F.8 and F.9), despite at least 91% knowledge-check accuracy (Appendix F.5). Our specialised models, with at most 6.02M parameters, achieve: αM at 99.22% rule exact match and 95.08% noise tolerance accuracy; skewM at 94.55% and 96.97%; stocM at 95.49% and 84.8%; and tempM at 89.05% and 82.3%, outperforming the best LLM baseline by

Table 1: Specialised models: architecture and held-out results. UO: unordered pair; CI: rule-level bootstrap. †Neural head (αM, skewM) or analytical estimator (tempM, stocM) alone; Table 53.

	αM	skewM	tempM	stocM
Token dim / tokens per orbit	4 / 1,980	5 / 3,980	12 / 9,950	12 / 9,950
Orbits per sample k	1	1	8	8
Encoder	flat 4L/4H/d=128	flat 4L/4H/d=128	2-level (4L+6L)	2-level (4L+6L)
Noise estimation	StatPool + MLP + hybrid	SpatialMultiScalePool + hybrid	analytical + correction MLP	analytical + correction MLP
Rule aggregation	mean pool	mean pool	τ-cond. attention	λ-cond. attention
Parameters	≈1.76M	≈1.77M	≈6.02M	≈6.02M
Test pairs	770	1,540	500 (×20)	500 (×20)
Rule exact match (%, 95% CI)	99.22% [97.4, 100.0]	94.55% [90.6, 97.8]	89.05% (UO) [84.2, 93.1]	95.49% (UO) [92.1, 98.3]
Noise MAE (full / single-component†)	0.0070 / 0.0275	0.221 / 0.943	0.0311 / 0.0458	0.0289 / 0.0575
Noise tolerance (%, 95% CI)	95.08% [91.6, 98.0]	96.97% [93.8, 99.2]	82.3% [76.9, 87.3]	84.8% [79.5, 89.6]
Chance (rule / noise)	0.39 / 10.0	0.39 / 14.5	0.003 / 25	0.003 / 25

up to $6.8\times$ in rule identification. From Qwen2.5-72B, Llama-3.1-70B, and Mixtral-8×7B (whose parameter budgets exceed ours by over $10{,}000\times$) to frontier models GPT-5.1 and Claude Haiku 4.5, all models remain below 15% rule accuracy, including after fine-tuning for the open models. Neither scale, prompting, reasoning, tool use nor full-context fine-tuning closes the gap (Section 4): the failure is not informational; it is architectural. (ii) *Learned identification drives the gap; mechanism-aware estimation sharpens it.* Rule identification, where every LLM fails ($\leq 14.6\%$), is always produced by learned heads: for αM it is computed from raw cell-transition tokens alone (99.22%), and for tempM and stocM the learned rule heads additionally read per-pattern output statistics of the orbit (Appendix B). The noise estimators operate downstream of the identified rule and cannot compensate for a wrong one. Even without them, the learned noise head of αM reaches 75.3% tolerance accuracy (MAE 0.0275), over twice the best LLM (32.5%). The hybrid estimator raises this to 95.08% (MAE 0.0070, a $4\times$ reduction) and cuts skewM's MAE from 0.943 to 0.221; for tempM and stocM, learned correction networks reduce the analytical MAE from 0.0458 and 0.0575 to 0.0311 and 0.0289 (Tables 1 and 53). The analytical knowledge is equally available to the LLMs: every prompt defines the noise process, and the knowledge-check datasets require deriving these very quantities, such as the expected change rate under α-noise (Appendix A.1). *The difference is not access to the formula but the ability to apply it*: the estimators need the correct rule and exact per-pattern counts over thousands of transitions, which the specialised encoders supply and next-token models do not. (iii) *Identifiability limits.* Both tempM and stocM show a pronounced accuracy drop at noise $= 0.5$, the hardest regime: the orientation of the pair (which rule is R_1) becomes unidentifiable (Appendix H), which the UO metric already discounts, and the two rules are hardest to separate. tempM's rule accuracy falls to 58.9% at $\tau = 0.5$, from 87.3% at $\tau = 0.4$. stocM's drop to 72.4% at $\lambda = 0.5$ is less severe, and the maximised within-timestep variance still gives a stable estimate of λ itself (MAE 0.0664). Across Wolfram classes, errors are not concentrated in chaotic or complex rules (Class III/IV), contrary to expectation; instead, Class I (null) rules have the lowest accuracy in all four settings, because they rapidly converge to homogeneous configurations and leave few informative transitions (Appendix G.5). Both effects stem from the task itself rather than from the models, and they apply to any estimator, including every LLM baseline. (iv) *Calibration against a zero-training algorithmic baseline.* Reading rule bits from direction-of-flip counts, with no training, identifies 92.57% (αECA), 94.11% (skewECA) and 79.92% (tempECA) of held-out rules (Appendix D.5). Rule identification is thus tractable in principle, sharpening the LLM result: the procedure is simple and known to the models, yet it is not executed ($\leq 14.6\%$). The specialised models match or exceed this baseline on every dataset and add what counting alone does not provide: naive noise statistics are 3–46× less accurate (MAE 0.325 vs 0.0070 on αECA, 1.381 vs 0.221 on skewECA, 0.0833 vs 0.0311 on tempECA), and on stocECA, where no timestep or orbit follows a single rule, unit-level clustering fails (0.02%) while stocM reaches 95.49%.

Conclusion. We formalised the noisy ECA identification problem, with CA as a proxy for natural systems, and exposed a fundamental limitation of LLMs: open and frontier models answer at least 91% of domain-knowledge questions correctly yet cannot identify the rule from an orbit. The gap survives every control we applied (from full-context fine-tuning to reasoning with tools and a decoder trained from scratch), and scaling generic baselines does not close it. Our specialised transformers, which extract the statistics each noise process leaves, achieve up to 99.22% rule exact match and 96.97% noise tolerance accuracy with at most 6.02M parameters, outperforming the best LLM by up to $6.8\times$. The decisive factor is architectural: inductive bias matched to the noise process, not scale. This opens the door to structure-aware transformers for reasoning and identification in real-world natural systems. Limitations of this work are discussed in Appendix I.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods (writing parts of the evaluation code for the additional baselines and LLM evaluation conditions, and debugging the training and evaluation code) and to provide feedback on the experimental design. We have not used generative AI tools to generate the synthetic datasets or write their generation code, to develop the conceptual framework or the noise models, to propose the research hypotheses, or to formulate or prove the mathematical claims; translation and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to edit the manuscript for readability and length, and to suggest changes to the structure of the paper. We have reviewed all AI-assisted work: AI-assisted code was run and checked by the authors on the real data, all reported numbers are taken from the authors' experiment logs, and AI-edited text was checked against the underlying results and claims. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only synthetic data generated from elementary cellular automata and researcher-authored knowledge-check questions; it involves no human subjects, personal information, or sensitive content. LLMs were evaluated through their public APIs or released weights in accordance with their terms of use. The trained models are specific to one-dimensional binary cellular automata and have no direct application to real systems; potential dual use of the methodology, for inferring behavioural rules from observed human activity, and the compute used are discussed in Appendix I.

REPRODUCIBILITY STATEMENT

All code for dataset generation, model training, LLM evaluation (prompts, fine-tuning configurations and parsers) and analysis is released in an anonymised repository: `https://anonymous.4open.science/r/LLM_NOISE-9DC1/`. The datasets, the fixed train/test rule split (seed 42) and the generation procedure are described in Appendix A.2; the architectures in Appendix B; training configurations, hyperparameters and hardware in Appendix C; the baselines in Appendix D; the LLM evaluation protocol in Appendix F; and complete proofs of the identifiability results in Appendix H. Frontier-model results (GPT-5.1, Claude Haiku 4.5, version `claude-haiku-4-5-20251001`) were obtained via public APIs between July and August 2026 and may differ if those models are updated. All datasets are synthetic, so all artifacts can be released without restriction.

REFERENCES

Andrew Adamatzky. *Identification of Cellular Automata*. Taylor & Francis, London, 1994.

Julia Ballesteros Hernando, Milagros Ramos Gómez, and Andrés Díaz Lantada. Modeling living cells within microfluidic systems using cellular automata models. *Scientific Reports*, 9(1):14886, 2019.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7096–7116, 2020.

Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3D neural networks. *Nature*, 619(7970):533–538, 2023.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Witold Bołt, Aleksander Bołt, Barbara Wolnik, Jan M. Baetens, and Bernard De Baets. A statistical approach to the identification of diploid cellular automata based on incomplete observations. *Biosystems*, 186:103976, 2019.

Witold Bołt, Jan M. Baetens, and Bernard De Baets. Identification of cellular automata based on incomplete observations with bounded time gaps. *IEEE Transactions on Cybernetics*, 50(3): 971–984, March 2020. doi: 10.1109/TCYB.2018.2875266.

Olivier Bouré, Nazim Fatès, and Vincent Chevrier. Probing robustness of cellular automata through variations of asynchronous updating. *Natural Computing*, 11(4):553–564, 2012.

Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

Mikhail Burtsev. Learning elementary cellular automata with transformers. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS 2024*, 2024.

Bastien Chopard and Michel Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, Cambridge, UK, 1998.

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

Stephen Coombes. The geometry and pigmentation of seashells. School of Mathematical Sciences, University of Nottingham, February 2009. URL `https://www.maths.nottingham.ac.uk/plp/pmzsc/pdfs/Seashells09.pdf`. Unpublished manuscript.

Hugo Dalla-Torre, Liam Gonzalez, Javier Mendoza-Revilla, Nicolas Lopez Carranza, Adam Henryk Grzywaczewski, Francesco Oteri, Christian Dallago, Evan Trop, Bernardo P. de Almeida, Hassan Sirelkhatim, Guillaume Richard, Marcin Skwark, Karim Beguir, Marie Lopez, and Thomas Pierrot. Nucleotide Transformer: Building and evaluating robust foundation models for human genomics. *Nature Methods*, 22(2):287–297, 2025.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the Chomsky hierarchy. In *International Conference on Learning Representations (ICLR)*, 2023.

Andreas Deutsch and Sabine Dormann. *Cellular Automaton Modeling of Biological Pattern Formation: Characterization, Examples, and Analysis*. Modeling and Simulation in Science, Engineering and Technology. Birkhäuser, Boston, MA, second edition, 2017.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Allyson Ettinger. What BERT is not: Lessons from a new suite of psycholinguistic diagnostics for language models. *Transactions of the Association for Computational Linguistics*, 8:34–48, 2020.

Nazim Fatès. Stochastic cellular automata solutions to the density classification problem: When randomness helps computing. *Theory of Computing Systems*, 53(2):223–242, 2013.

William Gilpin. Cellular automata as convolutional neural networks. *Physical Review E*, 100(3): 032402, 2019.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783v3*, 2024.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 22199–22213, 2022.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753, 2019.

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *International Conference on Learning Representations (ICLR)*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B. Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. LLM and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 33940–33962, 2024.

Rebecca Marvin and Tal Linzen. Targeted syntactic evaluation of language models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1192–1202, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Ivan Rodkin, Daniil Orel, Konstantin Smirnov, Arman Bolatov, Bilal Elbouardi, Besher Hassan, Yuri Kuratov, Aydar Bulatov, Preslav Nakov, Timothy Baldwin, Artem Shelmanov, and Mikhail Burtsev. Beyond memorization: Extending reasoning depth with recurrence, memory and test-time compute scaling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42385–42404. Association for Computational Linguistics, 2026.

Souvik Roy, Subrata Paul, and Sukanta Das. Temporally stochastic cellular automata: Classes and dynamics. *International Journal of Bifurcation and Chaos*, 32(12):2230029, 2022.

Souvik Roy, Shlok Shelat, and Shrey Salvi. Revisiting the notion of independence in asynchronous cellular automata—a study on s-skewed automata. *International Journal of Bifurcation and Chaos*, 35(15):2550173, 2025. doi: 10.1142/S0218127425501731.

Adel Sahila, Benedetta Canfora, Marzia Canzaniello, Nicolò Perello, Andrea Trucchia, and Gianni Pagnini. Statistical dynamics of wildfire burned area from cellular-automata simulators. *Scientific Reports*, 16(1):998, 2026.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.

David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. X-Vectors: Robust DNN embeddings for speaker recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5329–5333, 2018.

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

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? A survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 4149–4158, 2019.

Xiaoyan Tang, Funan Liu, and Xinling Hu. Urban growth simulation and scenario projection for the arid regions using heuristic cellular automata. *Scientific Reports*, 14(1):21106, 2024.

Christina V. Theodoris, Ling Xiao, Anant Chopra, Mark D. Chaffin, Zeina R. Al Sayed, Matthew C. Hill, Helene Mantineo, Elizabeth M. Brydon, Zexian Zeng, X. Shirley Liu, and Patrick T. Ellinor. Transfer learning enables predictions in network biology. *Nature*, 618(7965):616–624, 2023.

Alan M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 5998–6008, 2017.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 24824–24837, 2022.

Stephen Wolfram. Cellular automata as models of complexity. *Nature*, 311(5985):419–424, 1984.

N. H. Wulff and J. A. Hertz. Learning cellular automaton dynamics with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 5, pp. 631–638, 1992.

Huatao Xu, Liying Han, Qirui Yang, Mo Li, and Mani Srivastava. Penetrative AI: Making LLMs comprehend the physical world. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7324–7341. Association for Computational Linguistics, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Qiang Zhang, Keyan Ding, Tianwen Lv, Xinda Wang, Qingyu Yin, Yiwen Zhang, Jing Yu, Yuhao Wang, Xiaotong Li, Zhuoyi Xiang, Xiang Zhuang, Zeyuan Wang, Ming Qin, Mengyao Zhang, Jinlu Zhang, Jiyu Cui, Renjun Xu, Hongyang Chen, Xiaohui Fan, Huabin Xing, and Huajun Chen. Scientific large language models: A survey on biological & chemical domains. *ACM Computing Surveys*, 57(6):161:1–161:38, February 2025a. doi: 10.1145/3715318.

Yanbo Zhang, Sumeer A. Khan, Adnan Mahmud, Huck Yang, Alexander Lavin, Michael Levin, Jeremy G. Frey, Jared Dunnmon, James Evans, Alan Bundy, Saso Dzeroski, Jesper Tegner, and Hector Zenil. Exploring the role of large language models in the scientific method: From hypothesis to discovery. *npj Artificial Intelligence*, 1(1):14, 2025b.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations (ICLR)*, 2023.

A Dataset Details

A.1 Knowledge-Check Datasets (alphaK, skewK, stocK, tempK)

A.1.1 Purpose and Design Philosophy

The four knowledge-check datasets serve a single diagnostic purpose: to confirm that any LLM failures on the identification task reflect a structural inability to read spatiotemporal patterns, rather than gaps in domain knowledge. This distinction is critical for interpreting Section 4. A model that fails because it does not know what α-asynchronous noise is would be trivially fixable by adding an in-context description; a model that fails despite demonstrating full knowledge of the noise mechanics is exhibiting a genuine architectural limitation. The knowledge-check datasets establish which of these two regimes applies.

Each dataset contains 200 questions covering one noise type and is designed so that a model with strong domain knowledge should achieve near-perfect accuracy. Questions are grounded in first principles: the definitions, mathematical properties, update mechanics, and behavioural consequences of each noise model. No question requires solving the inverse problem; all questions are answerable from theoretical knowledge alone.

A.1.2 Question Format

All four datasets use two question formats. **Multiple-choice questions (MCQ)** present four options (A, B, C, D) with exactly one correct answer. The distractors are carefully constructed to be plausible: they typically correspond to common misconceptions, off-by-one errors in parameter ranges, or confusions between noise types. A model cannot succeed by pattern-matching to a keyword in the question; it must identify the correct answer and rule out three plausible alternatives. MCQ scoring is binary: 1 for a correct answer, 0 otherwise. **Short-answer questions** require the model to produce a free-form answer — a numerical value, a formula, a brief explanation, or a step-by-step derivation. Short-answer responses are scored by an independent judge model (Grok-3) on a three-point rubric: 2 for a fully correct answer, 1 for a partially correct answer (correct idea but missing a key detail), and 0 for an incorrect or missing answer. The raw score is normalised to $[0, 1]$ by dividing by 2, so partial credit is awarded. The overall accuracy reported for each model is the mean of all normalised scores across both question types.

A.1.3 Difficulty Levels

Questions are assigned one of three difficulty levels based on the type of reasoning required:

Easy (Level 1) questions test definitional recall and basic comprehension. A model that has seen the noise model defined once can answer these correctly. Examples include: what is the state space of an ECA cell, how many ECA rules exist, what does the parameter α represent. These questions have one unambiguous correct answer that follows directly from the definition.

Medium (Level 2) questions require understanding of mechanics and properties: how the update rule interacts with the noise parameter, what happens to the orbit at boundary values of the parameter, how the noise model compares to the deterministic baseline. A model must not only know the definition but understand its implications.

Hard (Level 3) questions require quantitative reasoning, derivation, or cross-model comparison. Examples include: computing the expected fraction of cells that update given a specific parameter value, deriving what happens to the orbit as the parameter approaches an extreme value, identifying which noise model is active given a partial description of observed behaviour, or proving a mathematical property of the noise process. These questions cannot be answered by recall alone; they require multi-step reasoning.

A.1.4 Dataset Statistics

Table 2 summarises the format and difficulty distribution for all four datasets.

Table 2: Knowledge-check dataset statistics. All datasets contain 200 questions with IDs 1–200 and no duplicates.

	Format		Difficulty			Topics
Dataset	MCQ	Short Ans.	Easy	Med.	Hard	covered
alphaK	178	22	80	68	52	6
skewK	140	60	53	86	61	15
stocK	148	52	51	91	58	12
tempK	134	66	49	85	66	14
Total	**600**	**200**	**233**	**330**	**237**	—

A consistent pattern emerges across all four datasets: medium difficulty is the most populous tier (330 questions, 41% of total), while easy and hard are roughly balanced (233 and 237 respectively). This distribution is intentional — the datasets probe a full range of domain understanding rather than testing only basic recall or only advanced derivation.

The four datasets differ systematically in format and topic coverage, reflecting the complexity of each noise type. alphaK is the most MCQ-heavy (178 out of 200, 89%) and covers only 6 topics, consistent with α-asynchronous noise having a simple one-dimensional definition with few distinct conceptual components. It also has the highest proportion of easy questions (40%), reflecting that many of its properties follow directly from the definition.

skewK, stocK, and tempK are considerably broader. skewK covers the most topics (15) and has 60 short-answer questions (30%), reflecting that s-skewed noise involves multiple distinct concepts: spatial contiguity of updates, the relationship between s and synchrony, and boundary cases at $s = 1$ and $s = W$. stocK has the largest medium tier (91 questions, 46%) and 52 short-answer questions (26%), reflecting that stochastic cell-level noise requires careful reasoning about per-cell independence and mixture statistics. tempK has the most short-answer questions in both absolute (66) and proportional (33%) terms, the most hard questions (66), and covers 14 topics — reflecting that temporally stochastic noise involves the most subtle mathematical reasoning, including per-timestep rule switching, long-run mixture behaviour, and the symmetric identifiability of (R_1, R_2, τ) vs $(R_2, R_1, 1 - \tau)$.

A.1.5 Topic Coverage

Tables 3–6 show the topic distribution for each dataset.

Table 3: alphaK topic distribution (200 questions, 6 topics).

Topic	Questions	%
Mathematical Properties	40	20.0
Update Mechanics	40	20.0
α-Async Noise: Definition & Parameters	34	17.0
Behavioural Effects & Dynamics	30	15.0
ECA Fundamentals	28	14.0
Comparisons & Contrasts	28	14.0

A.1.6 Representative Examples

We provide three representative questions from each dataset, one per difficulty level, to illustrate the range of reasoning demanded.

alphaK.

Table 4: skewK topic distribution (200 questions, 15 topics).

Topic	Questions	%
s-Skewed: Definition & Mechanics	24	12.0
ECA Fundamentals	14	7.0
s-Skewed: Mathematical Properties	19	9.5
s-Skewed: Conceptual Understanding	16	8.0
s-Skewed vs Other Noise Models	16	8.0
s-Skewed: Parameter Boundary Cases	15	7.5
s-Skewed: Perturbation Rate Interpretation	13	6.5
s-Skewed: Quantitative Reasoning	13	6.5
s-Skewed: Advanced Properties	12	6.0
s-Skewed: Dynamics & Behaviour	12	6.0
s-Skewed: Notation & Formalism	11	5.5
s-Skewed: Worked Examples	10	5.0
s-Skewed: Relationship to Synchrony	9	4.5
s-Skewed: Identification & Classification	9	4.5
s-Skewed: Boundary Conditions	7	3.5

Table 5: stocK topic distribution (200 questions, 12 topics).

Topic	Questions	%
Update Mechanics	22	11.0
Stochastic vs Temporal Stochastic Noise	20	10.0
Mathematical Properties	20	10.0
Conceptual Reasoning	18	9.0
Density and Dynamical Behaviour	18	9.0
Stochastic Noise: Definition and Setup	18	9.0
Worked Examples and Calculations	19	9.5
Special Values of λ and Limiting Behaviour	16	8.0
Comparing Stochastic Noise with Other Noise Models	15	7.5
Notational and Terminological Precision	13	6.5
ECA Fundamentals	11	5.5
Boundary Conditions and Lattice Structure	10	5.0

alphaK Example 1 — Easy (MCQ)

Q: An elementary cellular automaton (ECA) is a one-dimensional, discrete-time system where each cell has:
(A) Binary state (0 or 1) and a 3-cell neighbourhood (left, centre, right)
(B) Ternary state (0, 1, or 2) and a 5-cell neighbourhood
(C) Real-valued state and no neighbourhood
(D) Binary state and a 5-cell neighbourhood

A: (A). *ECA cells have binary states and radius* $r = 1$*, giving a 3-cell neighbourhood* $(i-1, i, i+1)$.

alphaK Example 2 — Medium (MCQ)

Q: In α-asynchronous ECA with $\alpha = 0.4$, what is the expected number of cells that are *not* updated at each time step in a grid of width $W = 50$?
(A) 20 (B) 30 (C) 40 (D) 10

A: (B). *Each cell updates independently with probability* $\alpha = 0.4$*, so the expected number updated is* $0.4 \times 50 = 20$*, and the expected number* ***not*** *updated is* $0.6 \times 50 = 30$.

Table 6: tempK topic distribution (200 questions, 14 topics).

Topic	Questions	%
Mathematical Properties	20	10.0
Definition of Temporal Stochastic Noise	19	9.5
Comparison with Other Noise Models	17	8.5
ECA Fundamentals	17	8.5
Temporal vs Spatial Stochastic Noise	16	8.0
Temporal Nature of the Noise	15	7.5
Boundary Cases of τ	13	6.5
Stochastic Properties & Long-Run Behaviour	13	6.5
Identification & Distinguishability	13	6.5
Conceptual Understanding & Edge Cases	13	6.5
Notation and Representation	12	6.0
Effect of τ on System Dynamics	12	6.0
Entropy & Information-Theoretic Properties	10	5.0
Extensions and Variations	10	5.0

alphaK Example 3 — Hard (Short Answer)

Q: In α-asynchronous ECA, let Δ_t denote the number of cells that change state at time step t relative to step $t-1$. Express $\mathbb{E}[\Delta_t]$ as a function of α, the rule R, and the current configuration $\mathbf{a}^t$. Under what condition on R is this expectation independent of $\mathbf{a}^t$?

A: $\mathbb{E}[\Delta_t] = \alpha \sum_i \mathbf{1}[R(\mathrm{nbr}(i, \mathbf{a}^t)) \neq a_i^t]$. This expectation is independent of $\mathbf{a}^t$ if and only if the rule R is *self-complementary* (i.e., $R(\mathbf{n}) \neq n_{\mathrm{centre}}$ for a constant fraction of configurations regardless of the state), or equivalently if the rule maps every neighbourhood pattern to a fixed output that agrees with the current cell state for exactly half the patterns — which holds exactly for rule 204.

skewK.

skewK Example 1 — Easy (MCQ)

Q: In a traditional (synchronous) ECA with n cells, how many cells are updated at each time step?
(A) One cell chosen at random (B) All n cells simultaneously (C) Exactly half the cells (D) A fixed subset of s cells

A: (B). *Traditional ECA uses a global clock: all n cells are updated simultaneously at each discrete time step.*

skewK Example 2 — Medium (MCQ)

Q: In s-skewed ECA with grid width $n = 20$ and $s = 5$, what fraction of cells is updated on average per time step?
(A) $5/20 = 0.25$ (B) $1/5 = 0.2$ (C) 5 (D) 1.0

A: (A). *At each timestep, exactly $s = 5$ contiguous cells are chosen and updated. The fraction updated is $s/W = 5/20 = 0.25$.*

skewK Example 3 — Hard (Short Answer)

Q: In s-skewed ECA, a contiguous block of s cells starting at a randomly chosen position $p \in \{0, \ldots, n-1\}$ (with periodic boundary) is updated at each timestep. For a cell at position j, derive the probability that it falls inside the update block at a single timestep. How does this probability change as s increases from 1 to n?

A: Cell j is updated if and only if the starting position p satisfies $p \in \{j - s + 1, \ldots, j\} \pmod{n}$, i.e., there are exactly s valid starting positions out of n equally likely ones. Therefore $\Pr[\text{cell } j \text{ updated}] = s/n$. As s increases from 1 to n, this probability increases linearly from $1/n$ to 1, recovering the fully synchronous case at $s = n$.

stocK.

stocK Example 1 — Easy (MCQ)

Q: In a traditional (synchronous) ECA, how are cells updated at each time step?
(A) One randomly chosen cell is updated (B) All cells are updated simultaneously using a global clock (C) Cells are updated one by one from left to right (D) Only cells in state 1 are updated

A: (B). *Traditional ECA uses a global clock: all n cells are updated in parallel at every discrete time step t.*

stocK Example 2 — Medium (MCQ)

Q: In stochastic ECA with rules R_1 and R_2 and mixing parameter $\lambda = 0.5$, what is the expected output for a cell with neighbourhood pattern n when $R_1[n] = 0$ and $R_2[n] = 1$?
(A) 0 (B) 1 (C) 0.5 (D) Depends on the current state

A: (C). $\mathbb{E}[\textit{output} \mid \textit{pat} = n] = \lambda R_2[n] + (1 - \lambda) r_1[n] = 0.5 \times 1 + 0.5 \times 0 = 0.5$.

stocK Example 3 — Hard (Short Answer)

Q: In stochastic ECA with rules R_1, R_2 and parameter λ, define the within-timestep variance $V_n = \mathrm{Var}[\text{output} \mid \text{pat} = n]$. Derive V_n in terms of λ, $R_1[n]$, and $R_2[n]$. At what value of λ is V_n maximised for a disagreement pattern ($R_1[n] \neq R_2[n]$)? What does this imply for the identifiability of λ?

A: The output is a Bernoulli$(\lambda R_2[n] + (1 - \lambda) R_1[n])$ random variable, so $V_n = \lambda(1 - \lambda)(R_2[n] - R_1[n])^2$. For disagreement patterns $(R_2[n] - R_1[n])^2 = 1$, giving $V_n = \lambda(1 - \lambda)$. This is maximised at $\lambda = 0.5$, where $V_n = 0.25$. This means that at $\lambda = 0.5$ the within-variance is most informative about mixing, but the orientation of the mixture (which rule is R_1 and which is R_2) is unrecoverable because the distribution is symmetric under interchange of R_1 and R_2.

tempK.

tempK Example 1 — Easy (MCQ)

Q: In a traditional (synchronous) ECA, when are cells updated?
(A) Each cell is updated randomly at some time step (B) All cells are updated simultaneously at each time step using a global clock (C) Cells are updated one at a time in a fixed order (D) Only cells with state 1 are updated

A: (B). *Traditional ECA uses a global clock: all cells are updated simultaneously at each discrete time step t using the same rule R.*

tempK Example 2 — Medium (MCQ)

Q: In temporally stochastic ECA with rules R_1 and R_2 and switching parameter $\tau = 0.3$, what is the probability that timestep t uses rule R_1?
(A) 0.3 (B) 0.7 (C) 0.5 (D) Depends on the state at time t

A: (B). *With probability $\tau = 0.3$ the entire row uses rule R_2; with probability $1 - \tau = 0.7$ the entire row uses rule R_1. The coin flip is independent of the state.*

tempK Example 3 — Hard (Short Answer)

Q: In temporally stochastic ECA with rules R_1, R_2 and parameter τ, show that the system (R_1, R_2, τ) is statistically indistinguishable from the system $(R_2, R_1, 1-\tau)$ based on any finite orbit. What does this symmetry imply for the identifiability of τ at $\tau = 0.5$?

A: At each timestep t, the orbit uses rule R_2 with probability τ and rule R_1 with probability $1-\tau$. Swapping the labels $R_1 \leftrightarrow R_2$ is equivalent to relabelling the coin flip, giving probability $1-\tau$ for rule R_2 and τ for rule R_1 — which is exactly the system $(R_2, R_1, 1-\tau)$. The joint distribution of the orbit under (R_1, R_2, τ) is therefore identical to the joint distribution under $(R_2, R_1, 1-\tau)$ for any finite T. At $\tau = 0.5$, swapping $R_1 \leftrightarrow R_2$ gives $(R_2, R_1, 0.5)$ which has the same distribution as $(R_1, R_2, 0.5)$, so the two-rule pair is fundamentally unorientable — no estimator can recover which rule is R_1 and which is R_2 with probability better than $\frac{1}{2}$.

A.2 Identification Dataset Configuration

This subsection provides the complete specification of all four identification datasets, with every sample count derived directly from the generation code. All four datasets share the same rule partition: seed = 42, 179 training rules, 77 test rules, strictly disjoint. The generation scripts are available in the code repository:`https://anonymous.4open.science/r/LLM_NOISE-9DC1/`.

A.2.1 File Format and Storage Layout

alphaECA and skewECA are pre-generated and stored to disk in the following directory structure:

```
ECA_Data_New/                          (alphaECA)
  train_rules.npy   int32  [179]
  test_rules.npy    int32  [77]
  phase1/
    train/
      orbits.npy    int8   [35800, 100, 20]
      rule_bits.npy int8   [35800, 8]
      alphas.npy    float32[35800]
      rule_ids.npy  int32  [35800]
    test/
      orbits.npy    int8   [7700, 100, 20]
      rule_bits.npy int8   [7700, 8]
      alphas.npy    float32[7700]
      rule_ids.npy  int32  [7700]
  phase2/
    train/
      orbits.npy    int8   [89500, 100, 20]
      rule_bits.npy int8   [89500, 8]
      alphas.npy    float32[89500]
      rule_ids.npy  int32  [89500]
    test/
      orbits.npy    int8   [15400, 100, 20]
      ...

ECA_Data_Skew/                         (skewECA)
  train_rules.npy   int32  [179]
  test_rules.npy    int32  [77]
  phase1/
    train/
      orbits.npy    int8   [35800, 200, 20]
      rule_bits.npy int8   [35800, 8]
      s_values.npy  int32  [35800]
      rule_ids.npy  int32  [35800]
    test/  ...  [7700 samples]
  phase2/
    train/
      orbits.npy    int8   [89500, 200, 20]
```

```
    rule_bits.npy int8   [89500, 8]
    s_values.npy  int32  [89500]
    rule_ids.npy  int32  [89500]
  test/  ...  [15400 samples]
```

Samples within each split are stored rule-wise (all samples for the first rule, then the second rule, etc.) and then alpha-wise or s-wise within each rule block. The DataLoader uses `shuffle=True` during training, so the stored order has no effect on training. The `did_change` feature for skewM tokens is computed on-the-fly from the stored orbits during the forward pass, not stored separately.

stocECA and tempECA store only the fixed test set on disk:

```
TSCA_Data/                          (tempECA)
  train_rules.npy   int32  [179]
  test_rules.npy    int32  [77]
  train_pairs.npy   int32  [500, 2]
  test_pairs.npy    int32  [500, 2]
  test/
    orbits.npy       float32[10000, 8, 200, 50]
    frac_stats.npy   float32[10000, 24]
    rule_f_bits.npy  float32[10000, 8]
    rule_g_bits.npy  float32[10000, 8]
    taus.npy         float32[10000]
    rule_f_ids.npy   int32  [10000]
    rule_g_ids.npy   int32  [10000]
    step_labels.npy  int8   [10000, 8, 199]

SCA_Data/                           (stocECA)
  train_rules.npy, test_rules.npy, train_pairs.npy, test_pairs.npy
  test/
    orbits.npy          float32[10000, 8, 200, 50]
    frac_stats.npy       float32[10000, 24]
    within_var_stats.npy float32[10000, 8]
    rule_f_bits.npy      float32[10000, 8]
    rule_g_bits.npy      float32[10000, 8]
    lambdas.npy          float32[10000]
    rule_f_ids.npy       int32  [10000]
    rule_g_ids.npy       int32  [10000]
```

The key difference between the two is that stocECA stores `within_var_stats.npy` (8-dimensional, one entry per neighbourhood pattern) — the within-timestep variance signal that is unique to cell-level mixing — while tempECA stores `step_labels.npy` (per-timestep binary labels indicating which rule fired at each row) used only for the assignment loss during training and for ceiling analysis.

A.2.2 Exact Sample Counts

Table 7 gives the complete sample count breakdown for all four datasets.

A.2.3 Stratification Guarantee

For alphaECA and skewECA, stratification is exact: every (rule, noise-level) pair appears the same number of times, by construction. The data generation code loops over rules and noise levels in the outer two loops, with samples as the innermost loop. There is no probabilistic sampling of noise levels — the counts are hard-coded constants. The code includes a `sanity_check` call after each split that asserts the correct sample count per rule and per noise level, catching any generation error before training begins.

For stocECA and tempECA, the noise parameter (λ or τ) is sampled uniformly from $[0.1, 0.9]$ at each training step, so the distribution across noise levels is approximately uniform rather than exactly stratified. The fixed test set is generated with the same uniform sampling, yielding an approximately uniform marginal distribution over the noise parameter range.

Table 7: Complete dataset sample counts for all four identification datasets. All arithmetic is verifiable from the generation constants in DATAGEN_NEW.py and DATAGEN_SKEW.py. stocECA and tempECA generate training data on-the-fly; only test sets are stored.

	alphaECA	**skewECA**	**tempECA**	**stocECA**
Grid configuration				
Grid width W	20	20	50	50
Timesteps T	100	200	200	200
Orbits per sample K	1	1	8	8
Token features	4	5	4	4
Tokens per orbit	1,980	3,980	9,950	9,950
Noise parameter				
Parameter	α	s	τ	λ
Range	$\{0.1, \ldots, 1.0\}$	$\{1, \ldots, 20\}$	$[0.1, 0.9]$	$[0.1, 0.9]$
Levels	10 discrete	20 discrete	continuous	continuous
Rule split				
Training rules	179	179	179	179
Test rules	77	77	77	77
Phase 1 (synchronous baseline)				
Samples per train rule	200	200	—	—
Samples per test rule	100	100	—	—
Phase 1 train total	$179 \times 200 = \mathbf{35{,}800}$	$179 \times 200 = \mathbf{35{,}800}$	—	—
Phase 1 test total	$77 \times 100 = \mathbf{7{,}700}$	$77 \times 100 = \mathbf{7{,}700}$	—	—
Phase 2 (full noise range, stratified)				
Samples per (rule, level)	50	25	—	—
Test samples per (rule, level)	20	10	—	—
Phase 2 train total	$179 \times 10 \times 50 = \mathbf{89{,}500}$	$179 \times 20 \times 25 = \mathbf{89{,}500}$	—	—
Phase 2 test total	$77 \times 10 \times 20 = \mathbf{15{,}400}$	$77 \times 20 \times 10 = \mathbf{15{,}400}$	—	—
On-the-fly training (per epoch)				
Pairs per epoch	—	—	1,000	1,000
Samples per pair	—	—	8	8
Training samples per epoch	—	—	$1{,}000 \times 8 = \mathbf{8{,}000}$	$\mathbf{8{,}000}$
Total epochs	—	—	$80 + 80 = 160$	$80 + 80 = 160$
Total training samples	—	—	$160 \times 8{,}000 = \mathbf{1{,}280{,}000}$	$\mathbf{1{,}280{,}000}$
Fixed test set				
Test pairs	—	—	500	500
Samples per test pair	—	—	20	20
Test total	—	—	$500 \times 20 = \mathbf{10{,}000}$	$\mathbf{10{,}000}$
Totals				
Total train samples	**125,300**	**125,300**	**1,280,000**	**1,280,000**
Total test samples	**23,100**	**23,100**	**10,000**	**10,000**
Grand total	**296,800** (αECA + skewECA) + **2,580,000** (stocECA + tempECA, incl. all training epochs) = **2,876,800**			

A.2.4 Validation Split

For alphaECA and skewECA, 15% of the Phase 2 training data is held out as a validation split (seed = 42, using PyTorch `random_split`) for early stopping and checkpoint selection. The validation metric is rule exact match on the held-out split. The validation split is drawn from the Phase 2 training data only; Phase 1 training data is not split (the Phase 1 training objective is pure rule identification, where convergence is fast and stable). For stocECA and tempECA, evaluation on the fixed test set at the end of each epoch serves the role of validation, since the training data is regenerated each epoch and there is no fixed training split to hold out from.

A.2.5 Reproducibility

All four datasets are fully reproducible from the generation scripts in the repository using `numpy.random.default_rng(seed=42)`. The rule split is generated first and is identical across all four datasets (same seed, same permutation). Initial configurations are sampled uniformly from $\{0, 1\}^n$ subject to the constraint $0 < \sum_i a_i < n$ (non-trivial configurations), using rejection sampling which terminates in $O(1)$ expected iterations for $n \geq 4$. Also all the datasets are provided in the repository.

A.3 Rule Split and Wolfram Class Distribution

A.3.1 Rule Partition

The 256 ECA rules are partitioned into training and test sets once, under a fixed seed, and this partition is held constant across all four identification datasets. The partition is generated as follows:

```
rng = numpy.random.default_rng(seed=42)
shuffled    = rng.permutation(256).tolist()
train_rules = sorted(shuffled[:179])   # 179 rules
test_rules  = sorted(shuffled[179:])   # 77 rules
assert not set(train_rules) & set(test_rules)
```

The sorted lists are saved to `train_rules.npy` and `test_rules.npy` in each dataset's root directory. The disjointness assertion is enforced at generation time and can be verified by any reader from the seed alone. The first twenty rules in each split are:

Train (first 20):	0, 2, 3, 4, 5, 7, 8, 9, 10, 12, 13, 18, 20, 21, 24, 25, 26, 27, 28, 29, ...
Test (first 20):	1, 6, 11, 14, 15, 16, 17, 19, 22, 23, 30, 31, 34, 35, 36, 40, 41, 42, 43, 44, ...

Note that several well-known rules appear in the test set: rule 30 (the chaotic rule whose pattern appears on cone shells), rule 110 (the only known Class C rule that is Turing-complete), and rule 22. The model is never shown these rules during training and must identify them purely from the patterns they produce in the held-out orbits.

A.3.2 Wolfram Classification Scheme

We use the Wolfram behavioural classification adapted for the equivalence classes of ECA rules under reflection and complementation. Each rule R is mapped to its canonical minimum representative $\hat{R} = \min(R, R^\dagger, R^*, R^{\dagger *})$, where $R^\dagger$ denotes the reflection and R^* the complement. The class is then assigned based on $\hat{R}$:

- **Class C1** (null/uniform/fixed-point): $\hat{R} \in \{0, 8, 32, 40, 128, 136, 160, 168\}$. These rules converge quickly to a fixed point or spatially uniform state. Their orbits contain almost no information after a few timesteps.
- **Class C2** (periodic): all rules not in Class C1, Class C3, or Class C4. These rules produce periodic or spatially structured patterns. They are the most numerous class.
- **Class C3** (chaotic): $\hat{R} \in \{18, 22, 30, 45, 60, 90, 105, 122, 126, 146, 150\}$ and related chaotic rules. These rules produce irregular, aperiodic patterns that are sensitive to initial conditions.
- **Class C4** (complex/named complex): $\hat{R} \in \{41, 54, 106, 110\}$ and locally complex rules such as $\{26, 73, 154\}$. These rules exhibit the richest behaviour, including Turing-completeness in the case of Rule 110.

This four-class scheme (C1, C2, C3, C4) corresponds directly to Wolfram's original four classes (I, II, III, IV). From the perspective of the identification task, Classes C3 and C4 produce similarly rich orbital structure; results are sometimes reported with C3 and C4 combined (total 50 rules) when per-class sample sizes are small, but the four-class breakdown is used wherever sufficient data are available.

A.3.3 Class Distribution in Train and Test Sets

Table 8 shows the exact Wolfram class distribution across all 256 rules and within the train/test partition.

The proportions are approximately preserved: Class C1 makes up 9.4% of all rules, 10.1% of training rules, and 7.8% of test rules. Classes C3+C4 combined make up 19.5% of all rules and 22.1% of

Table 8: Wolfram class distribution across all 256 ECA rules and within the train/test split (seed = 42). The 70/30 split preserves the class proportions approximately, ensuring both partitions contain rules from all behavioural categories. C3 and C4 counts in the train/test rows are combined (C3+C4) as the per-class train/test breakdown is not enumerated separately.

Split	C1 (null)	C2 (periodic)	C3 (chaotic)	C4 (complex)	C3+C4	Total
All 256 rules	24 (9.4%)	182 (71.1%)	36 (14.1%)	14 (5.5%)	50 (19.5%)	256
Train (179)	18 (10.1%)	128 (71.5%)	33 combined (18.4%)		33	179
Test (77)	6 (7.8%)	54 (70.1%)	17 combined (22.1%)		17	77

test rules — a slight over-representation that makes the test set marginally harder than the training distribution, which is conservative in favour of the reported results. Class C2 dominates all three sets at approximately 71%.

The complete list of all Class C3 and Class C4 rules across all 256 is:

$$\{18, 22, 26, 30, 41, 45, 54, 60, 73, 75, 82, 86, 89, 90, 97, 101, 102,\\ 105, 106, 107, 109, 110, 120, 121, 122, 124, 126, 129, 135,\\ 137, 146, 147, 149, 150, 151, 153, 154, 161, 165, 166,\\ 167, 169, 180, 181, 182, 183, 193, 195, 210, 225\}$$

A.3.4 Why Class Balance Matters for Training and Evaluation

The class distribution has practical consequences for both training and evaluation. Class C1 rules (null/uniform) produce orbits that contain very little spatial variation — within a few timesteps the grid converges to a uniform state and the subsequent rows are identical. For αM and skewM, this means that at low α or low s, Class C1 orbits are nearly indistinguishable from each other, because the rule's fingerprint (which is close to a fixed point) is overwhelmed by the noise. For tempM and stocM, pairs in which both rules are Class C1 are the hardest cases: if both R_1 and R_2 converge to the same fixed point, the mixture orbit looks identical regardless of τ or λ.

Class C3 and Class C4 rules (chaotic and complex) produce highly variable orbits with rich spatial structure, which makes the rule fingerprint easier to detect — but the within-orbit variance also makes noise estimation harder, because the natural variability of the orbit competes with the variance induced by the noise. The interplay between rule complexity and noise type is why the Wolfram class stratification for stocECA and tempECA (described in Appendix A.2) is essential: without explicit stratification across class combinations, a random pair-sampling scheme would under-represent the hardest pairs (Class C1 $\times$ Class C1) and over-represent Class C2 $\times$ Class C2 pairs, biasing the reported accuracy upward.

A.3.5 Test Rules of Special Interest

Several test rules warrant specific mention because they represent known edge cases:

Rule 30 is in the test set (Class C3). It is the most studied chaotic ECA rule and the one whose pattern appears on cone shells. All four models must identify rule 30 without having seen it during training.

Rule 110 is in the test set (Class C4). It is the only ECA rule proven to be Turing-complete and has the most complex orbit structure of any ECA rule. Its identification under noise is the hardest single-rule case in the benchmark.

Rule 204 is in the training set (Class C2: $204 = 11001100_2$, always outputs the centre cell unchanged). Rule 204 is the known failure case for the hybrid estimator in αM — its cell-change rate is identically zero regardless of α, making α unrecoverable from any cell-change statistic. Because rule 204 is in the training set, αM is trained on it and learns to handle it via the neural fallback, but the fundamental identifiability limit remains. The neural fallback achieves noise MAE of 0.4500 and tolerance accuracy of 10% for rule 204, as reported in Appendix G.

Rules 0 and 255 (all-zeros and all-ones) are both in the training set (Class C1). These rules produce fixed-point orbits but still predict changes (every cell not yet at the fixed value), so they use the hybrid estimator rather than the neural fallback (Appendix G.8).

A.4 On-the-Fly Generation Details for stocECA and tempECA

stocECA and tempECA generate training data on-the-fly at each epoch rather than pre-storing it to disk. This section gives the complete specification of the generation process — pair sampling, noise parameter sampling, orbit simulation, and the noise-range curriculum.

A.4.1 Constants and Per-Epoch Budget

The two key generation constants, shared identically between stocECA and tempECA, are:

$$\texttt{PAIRS_PER_EPOCH} = 1{,}000$$
$$\texttt{SAMPLES_PER_PAIR} = 8$$
$$\text{Training samples per epoch} = 1{,}000 \times 8 = 8{,}000$$

Each epoch constructs a fresh `OnTheFlyDataset` with a deterministic but epoch-varying seed: Stage 1 uses `seed = 10000 + epoch × 7` and Stage 2 uses `seed = 20000 + epoch × 13`. The use of different multipliers in the two stages ensures that Stage 2 epochs do not reproduce any Stage 1 orbit, and the epoch-varying seed ensures that each epoch sees a fresh set of noise parameters and initial configurations even for the same rule pair. Each sample within a pair generates $k = 8$ independent orbits from k distinct random initial configurations, so the full on-the-fly data volume per epoch is $8{,}000 \times 8 = 64{,}000$ orbit simulations.

A.4.2 Rule Pair Sampling

At the start of each epoch, 1,000 rule pairs are sampled from the 179 training rules. The sampling procedure is:

Algorithm 1 On-the-fly pair sampling (per epoch)

`seen` $\leftarrow \emptyset$, `pairs` $\leftarrow [\,]$, `attempts` $\leftarrow 0$
while $|$`pairs`$|$ $<$ `PAIRS_PER_EPOCH` **and** `attempts` $< 20 \times$ `PAIRS_PER_EPOCH` **do**
  Sample $R_1 \sim$ Uniform(train_rules), $R_2 \sim$ Uniform(train_rules)
  `attempts` $\leftarrow$ `attempts` $+ 1$
  if $R_1 = R_2$ **then continue**
  end if
  $p \leftarrow (\min(R_1, R_2), \max(R_1, R_2))$ ▷ canonical unordered pair
  if $p \notin$ `seen` **then**
    `pairs`.append((R_1, R_2)); `seen`.add(p)
  end if
end while
while $|$`pairs`$|$ $<$ `PAIRS_PER_EPOCH` **do** ▷ fallback if uniqueness budget exhausted
  Sample $R_1, R_2 \sim$ Uniform(train_rules) with $R_1 \neq R_2$
  `pairs`.append((R_1, R_2))
end while

The algorithm *prefers* unique unordered pairs — it attempts to avoid seeing the same (R_1, R_2) combination more than once per epoch — but falls back to allowing repeats if the uniqueness budget ($20\times$ the target count) is exhausted. With 179 training rules, the maximum number of distinct unordered pairs is $\binom{179}{2} = 15{,}931$, which is much larger than 1,000, so in practice the fallback is never triggered and every epoch contains 1,000 distinct pairs with high probability.

An important distinction: this training-time pair sampling does *not* perform Wolfram class stratification. The class-stratified `sample_pairs` function is used only in the dataset generation scripts for

constructing the fixed test sets, where it ensures all six Wolfram class combinations (A×A, A×B, A×C, B×B, B×C, C×C) are represented. During training, the random pair sampling provides implicit coverage of all class combinations, as the 179 training rules contain rules from all Wolfram classes (18 Class C1, 128 Class C2, 33 Class C3/C4 from Appendix A.3)

A.4.3 Noise Parameter Sampling and the Noise-Range Curriculum

For each pair (R_1, R_2) and each of the 8 samples per pair, the noise parameter (τ for tempECA, λ for stocECA) is sampled uniformly from a range $[\theta_{\min}, \theta_{\max}]$ that changes according to a *noise-range curriculum*. The curriculum is parameterised by a half-width `lam_half`, which determines the range symmetrically around 0.5:

$$\theta_{\min} = \max(\theta_{\text{global_min}},\ 0.5 - \texttt{lam_half}) \tag{1}$$

$$\theta_{\max} = \min(\theta_{\text{global_max}},\ 0.5 + \texttt{lam_half}) \tag{2}$$

where $\theta_{\text{global_min}} = 0.1$ and $\theta_{\text{global_max}} = 0.9$ for both models. The curriculum schedules for Stage 1 and Stage 2 are:

Table 9: Noise-range curriculum for stocECA and tempECA. Both models use identical schedules. `lam_half` determines the sampling range symmetrically around 0.5.

Stage	Epoch range	`lam_half`	Sampling range $[\theta_{\min}, \theta_{\max}]$	Rationale
Stage 1	1–10	0.20	$[0.30, 0.70]$	Near-equal mixing; both rules visible every few timesteps
Stage 1	11	0.40	$[0.10, 0.90]$	Expand to full range
Stage 1	12–80	0.40	$[0.10, 0.90]$	Stay at full range
Stage 2	1–80	0.40	$[0.10, 0.90]$	Always full range in Stage 2

The rationale for the curriculum is as follows. When θ is close to 0 or 1, one rule fires very rarely — at $\tau = 0.1$, rule R_2 fires on average only 10% of timesteps, giving the model very few examples of rule R_2's fingerprint per orbit. If the model is exposed to extreme θ values before it has learned to identify either rule individually, the rare-rule signal is too weak to be useful and the model may learn to ignore the rare rule entirely. The first 10 epochs therefore restrict the noise range to $[0.3, 0.7]$, where both rules appear frequently and the model can learn their fingerprints simultaneously. At epoch 11 the range expands to the full $[0.1, 0.9]$, forcing the model to handle extreme mixing ratios. The transition is a single step (not gradual) because the model's rule representations are already stable by epoch 11 and can handle the full range without destabilisation.

A.4.4 Initial Configuration Sampling

Each orbit is started from a random binary initial configuration $\mathbf{a}^0 \in \{0,1\}^n$, sampled uniformly subject to a non-triviality constraint:

Algorithm 2 `random_init`: non-trivial initial configuration

repeat
  $\mathbf{a}^0 \leftarrow \text{Uniform}(\{0,1\}^n)$ ▷ each cell independently $\sim$ Bernoulli(0.5)
until $0 < \sum_i a_i^0 < n$ ▷ reject all-zeros and all-ones
return $\mathbf{a}^0$

The rejection condition ensures the initial configuration is neither all-zeros nor all-ones. An all-zeros configuration is a fixed point for any rule that maps the (0,0,0) neighbourhood to 0 (which includes most rules), causing the orbit to degenerate immediately and providing no rule-discriminative signal. The same applies to all-ones for rules that map (1,1,1) to 1. The expected number of iterations before a valid configuration is accepted is $\frac{1}{1-2^{1-n}}$, which for $n = 50$ is essentially 1.0 — the constraint adds negligible overhead.

A.4.5 ORBIT SIMULATION

For each pair (R_1, R_2), noise parameter θ, and initial configuration $\mathbf{a}^0$, the orbit is simulated for $t = 200$ timesteps. The simulation for tempECA (row-level switching) and stocECA (cell-level switching) differs in exactly one step:

Algorithm 3 Orbit simulation for tempECA and stocECA. Here T denotes time–steps

$\mathbf{a}^0 \leftarrow$ `random_init()`; $\mathbf{orbit}[0] \leftarrow \mathbf{a}^0$
for $t = 1$ **to** $T - 1$ **do**
  Compute neighbourhood index: $n_i = 4a_{i-1}^{t-1} + 2a_i^{t-1} + a_{i+1}^{t-1} \quad \forall i$ (periodic)
  if tempECA (row-level) **then**
    Draw $u \sim \text{Uniform}(0, 1)$ ▷ one flip for entire row
    if $u < \tau$ **then**
      $a_i^t \leftarrow R_2[n_i] \quad \forall i$
    else
      $a_i^t \leftarrow R_1[n_i] \quad \forall i$
    end if
  else if stocECA (cell-level) **then**
    for each cell i **do**
      Draw $u_i \sim \text{Uniform}(0, 1)$ ▷ independent flip per cell
      if $u_i < \lambda$ **then**
        $a_i^t \leftarrow R_2[n_i]$
      else
        $a_i^t \leftarrow R_1[n_i]$
      end if
    end for
  end if
  $\mathbf{orbit}[t] \leftarrow \mathbf{a}^t$
end for
return $\mathbf{orbit} \in \{0, 1\}^{T \times n}$

The key difference is that tempECA draws one random number per timestep (row-level), while stocECA draws $n = 50$ independent random numbers per timestep (cell-level). This is the defining distinction between the two noise models and directly determines the signal structure available for identification: tempECA produces timesteps where all cells share the same rule, creating a per-timestep residual signal; stocECA produces timesteps where each cell independently chooses, creating a within-timestep variance signal.

After simulation, two summary statistics are computed from the $k = 8$ orbits and stored alongside the tokens:

frac_stats (24-dimensional, both models): For each of the 8 neighbourhood patterns $n \in \{0, \ldots, 7\}$, the empirical fraction of outputs that equal 1 is computed across all k orbits. Three statistics per pattern (mean, std, and n-weighted mean) give 24 dimensions. In expectation, $\texttt{frac}[n] \approx \theta \cdot R_2[n] + (1 - \theta) \cdot R_1[n]$, which encodes the per-pattern mixture fraction.

within_var_stats (8-dimensional, stocECA only): For each neighbourhood pattern n, the empirical within-timestep variance of the output across the 50 cells that show pattern n in each timestep is averaged. In expectation, $\texttt{within_var}[n] \approx \lambda(1 - \lambda)(R_2[n] - R_1[n])^2$, providing a second independent estimator of λ that is not available in tempECA (where all cells in a timestep use the same rule, making within-variance zero by construction).

A.4.6 SUMMARY: TOTAL DATA GENERATED

Over the full training run of $80 + 80 = 160$ epochs, the total number of training samples generated is:

$$160 \text{ epochs} \times 8{,}000 \text{ samples/epoch} = \mathbf{1{,}280{,}000} \text{ training samples}$$

No training sample is ever stored to disk — each epoch regenerates its 8,000 samples from scratch using the epoch-specific seed. This provides implicit data augmentation at no storage cost: the 1,280,000 training samples are all distinct (different noise parameters and initial configurations) even when the same rule pair appears in multiple epochs.

A.5 Dataset Sample Entries

Figure 7 shows one representative sample from each of the four identification datasets, generated with `seed=42`. Each panel displays 20 timesteps $\times$ 20 cells for display clarity; actual orbit dimensions are $t \times n$ as given in Table 7 ($t = 100$, $n = 20$ for alphaECA; $t = 200$, $n = 20$ for skewECA; $t = 200$, $n = 50$ for tempECA and stocECA). The four examples are chosen to make the distinguishing noise signal of each dataset visually apparent, connecting directly to the architectural choices of Section 5.

A.6 Stratification Verification

A central design requirement for alphaECA and skewECA is that every (rule, noise-level) pair appears *exactly* the same number of times in both training and test splits, with no probabilistic sampling of noise levels. This subsection provides the complete verification of this guarantee.

A.6.1 Stratification Mechanism

The stratification is enforced structurally rather than statistically. The dataset generation code uses three nested loops:

```
for rule in train_rules:           # outer: 179 rules
    for noise_level in noise_levels:   # middle: 10 or 20 levels
        for _ in range(k):             # inner: fixed k samples
            generate_and_store(rule, noise_level)
```

Because k is a hard-coded constant (not a random variable), every (rule, noise-level) pair receives *exactly* k samples. There is no sampling variance, no rounding, and no rejection — the count is deterministic by construction. A `sanity_check` call after each split asserts the correct per-rule and per-noise-level counts before any data is used for training, catching any generation error immediately.

A.6.2 alphaECA Stratification

Table 10 gives the exact sample count for every noise level in alphaECA Phase 2, for both training and test splits. Phase 1 ($\alpha = 1.0$ only) is included for completeness; it is the same level as the $\alpha = 1.0$ row in Phase 2 but generated separately to provide clean synchronous-only orbits for the first training phase.

The max/min ratio across all 10 noise levels is exactly 1.0000, confirming zero imbalance. Every training rule is seen at every α level the same number of times, so the model cannot learn to favour any particular α value through exposure frequency.

A.6.3 skewECA Stratification

Table 11 gives the same verification for skewECA. The dataset covers 20 discrete values of $s \in \{1, \ldots, 20\}$, with $K_{\text{train}} = 25$ and $K_{\text{test}} = 10$ samples per (rule, s) pair.

A.6.4 Why Exact Stratification Matters

Probabilistic stratification — sampling noise levels uniformly at random — would introduce variance in the per-level counts. With 89,500 training samples split across 10 levels, the expected count per level under uniform sampling is 8,950, but the standard deviation is $\sqrt{89500 \times 0.1 \times 0.9} \approx 89.7$, meaning any individual noise level could receive as few as $\approx 8{,}660$ or as many as $\approx 9{,}240$ samples (within 3 standard deviations). This imbalance of up to $\pm 3.4\%$ per level is small but non-zero, and would mean the model sees some noise levels slightly more than others during training.

The exact stratification eliminates this entirely. More importantly, it provides a formal guarantee: the reported accuracy at each noise level is not inflated by over-representation of that level during

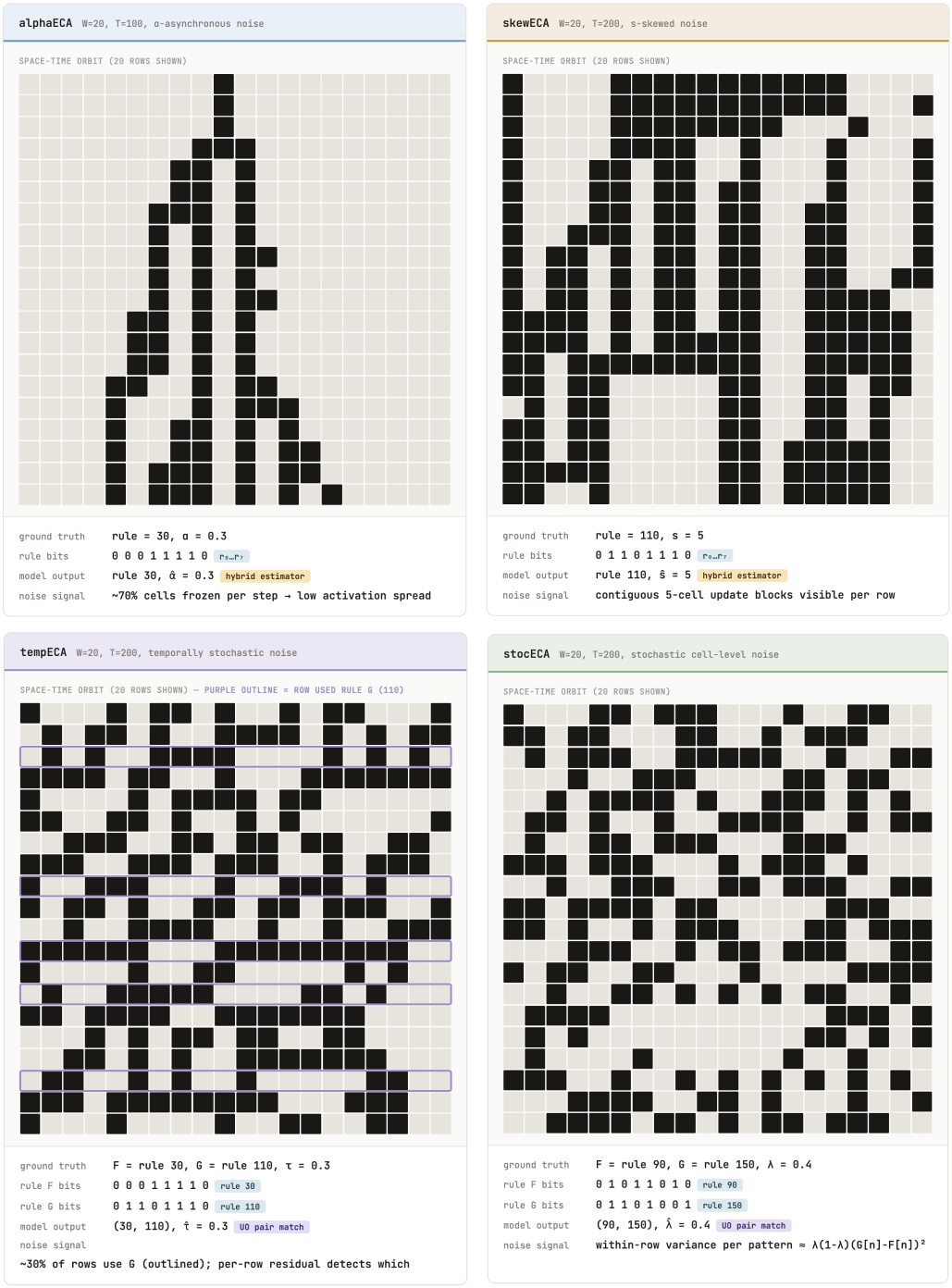


Figure 7: **Representative samples from each identification dataset. Here F denotes R_1 and G denotes R_2.**

training. A model that performs well at $\alpha = 0.5$ does so because it has learned the signal at that noise level, not because it was trained on more examples at that level than at others. This guarantee is what makes the per-noise-level accuracy tables in Appendix G directly interpretable as a function of noise difficulty rather than training data volume.

Table 10: alphaECA Phase 2 stratification. Every (rule, α) pair appears exactly $K_{\text{train}} = 50$ times in training and $K_{\text{test}} = 20$ times in the test set. Counts are exact with max/min ratio $= 1.0000$.

α	Training split		Test split	
	samples/pair	total samples	samples/pair	total samples
0.1	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
0.2	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
0.3	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
0.4	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
0.5	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
0.6	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
0.7	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
0.8	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
0.9	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
1.0	50	$179 \times 50 = 8{,}950$	20	$77 \times 20 = 1{,}540$
Phase 2 total	—	**89,500**	—	**15,400**
Phase 1 (additional synchronous-only split, $\alpha = 1.0$)				
$\alpha = 1.0$	200	$179 \times 200 = 35{,}800$	100	$77 \times 100 = 7{,}700$
Grand total	—	**125,300**	—	**23,100**

A.6.5 Note on stocECA and tempECA

stocECA and tempECA use continuous noise parameters (λ and τ respectively) sampled uniformly from $[0.1, 0.9]$ at each training step. Exact stratification across a continuous range is not possible; instead, the uniform distribution provides approximate balance in expectation. For the fixed test sets, 500 rule pairs are sampled with coverage of all Wolfram class combinations (see Appendix A.4), and τ or λ is sampled uniformly at test generation time. The resulting test distribution is approximately uniform over the noise parameter range, as verified by the per-τ and per-λ breakdown tables in Appendix G.

B Architecture Details

This appendix gives the complete layer-by-layer specification for all four models with every dimension, activation, dropout placement, initialisation, tensor shape, and design rationale that does not fit in Section 5. The main paper describes *why* each component exists; this appendix describes *exactly what* each component is so that the models can be reproduced from this document alone without reading the code.

All models share the following global conventions:

- **Weight initialisation:** all `nn.Linear` weights drawn from $\mathcal{N}(0, 0.02)$; all biases initialised to zero. Exception: the correction MLP in tempM and stocM uses $\mathcal{N}(0, 0.01)$ to keep initial corrections small and preserve the analytical estimate baseline.
- **Normalisation:** Pre-Layer Normalisation (`norm_first=True` in PyTorch `TransformerEncoderLayer`) before every attention and feed-forward sublayer.
- **Activation:** GELU throughout, except the correction MLP output which uses Tanh to bound the correction magnitude.
- **Dropout:** rate $p = 0.1$ applied uniformly after every non-final linear projection. Disabled at inference.
- **Gradient clipping:** maximum norm 1.0 applied before every optimiser step across all models and both training phases.
- **Positional encoding:** αM and skewM use TripletPE2D (described in Section 5); tempM and stocM use standard 1D SinusoidalPE at both levels of the hierarchical encoder. All positional

Table 11: skewECA Phase 2 stratification. Every (rule, s) pair appears exactly $K_{\text{train}} = 25$ times in training and $K_{\text{test}} = 10$ times in the test set across all 20 values of s. Max/min ratio = 1.0000.

s	Training split		Test split	
	samples/pair	total samples	samples/pair	total samples
1	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
2	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
3	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
4	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
5	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
6	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
7	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
8	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
9	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
10	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
11	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
12	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
13	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
14	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
15	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
16	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
17	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
18	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
19	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
20	25	$179 \times 25 = 4{,}475$	10	$77 \times 10 = 770$
Phase 2 total	—	$\mathbf{89{,}500}$	—	$\mathbf{15{,}400}$
Phase 1 (synchronous-only split, $s = W = 20$)				
$s = 20$	200	$179 \times 200 = 35{,}800$	100	$77 \times 100 = 7{,}700$
Grand total	—	$\mathbf{125{,}300}$	—	$\mathbf{23{,}100}$

encodings are registered buffers with no learnable parameters and do not contribute to the parameter count.

- **Nested tensor:** `enable_nested_tensor=False` is set on all `TransformerEncoder` instances to suppress a UserWarning triggered by `norm_first=True` in PyTorch $\geq$ 2.0.

B.1 αM: Complete Specification

αM solves the α-asynchronous inverse problem. It takes a flat sequence of 1,980 four-dimensional cell-transition tokens and produces two outputs: 8 rule bit logits and 10 α-class logits.

Constants

$$
\begin{aligned}
n &= 20 \quad \text{(grid width)} \\
t &= 100 \quad \text{(timesteps)} \\
t - 1 &= 99 \quad \text{(transitions)} \\
(t-1) \times n &= 99 \times 20 = 1{,}980 \quad \text{(tokens per orbit)} \\
d &= 128 \quad \text{(model dimension)} \\
H &= 4 \quad \text{(attention heads)}, \quad d_h = d/H = 32 \quad \text{(head dimension)} \\
L &= 4 \quad \text{(encoder layers)} \\
d_{\text{ff}} &= 512 \quad \text{(feed-forward dimension)} \\
p &= 0.1 \quad \text{(dropout rate)}
\end{aligned}
$$

TOKENISATION

Each orbit $\mathbf{O} \in \{0,1\}^{t \times n}$ is tokenised by the function `orbit_to_tokens`:

$$\mathbf{x}_{t',i} = \left(a_{i-1}^{t'},\ a_i^{t'},\ a_{i+1}^{t'},\ a_i^{t'+1}\right) \quad \forall\, t' \in \{0,\ldots,t-2\},\ i \in \{0,\ldots,n-1\} \tag{3}$$

with periodic boundary conditions ($a_{-1}^{t'} = a_{n-1}^{t'}$, $a_n^{t'} = a_0^{t'}$). The result is a $[1980, 4]$ float32 array. The four features represent: left neighbour state, centre state, right neighbour state, and next-step centre state. Each token is a complete, self-contained observation of one rule application: given the neighbourhood $(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'})$, the rule deterministically predicts $a_i^{t'+1}$, and deviations from this prediction encode the noise.

SHARED BACKBONE

Input projection. A single `nn.Linear`(4, 128) with bias maps each 4-dimensional token to the model dimension. No activation follows this projection — the positional encoding is added directly to the linearly projected tokens.

$$\mathbf{H}^{(0)} = \mathbf{X}\mathbf{W}_{\text{in}}^{\top} + \mathbf{b}_{\text{in}} \quad \in \mathbb{R}^{B \times (t-1)n \times 128} \qquad \text{(640 parameters)}$$

TripletPE2D. The positional encoding is a registered buffer (no learnable parameters). For each token at grid position (t', i), the encoding is:

$$\text{PE}(t',i)_{2m} = \sin\!\big(t'/10000^{2m/64}\big) \oplus \sin\!\big(i/10000^{2m/64}\big)$$
$$\text{PE}(t',i)_{2m+1} = \cos\!\big(t'/10000^{2m/64}\big) \oplus \cos\!\big(i/10000^{2m/64}\big)$$

where $m \in \{0,\ldots,31\}$ indexes the sinusoidal frequency, and $\oplus$ denotes concatenation of the 64-dimensional time encoding and the 64-dimensional cell encoding into a single 128-dimensional vector. Dropout ($p = 0.1$) is applied to $\mathbf{H}^{(0)} + \text{PE}$ before entering the encoder.

Transformer encoder ($L = 4$ layers, PreNorm). Each layer applies the following operations in order:

1. **LayerNorm**(128) on input $\mathbf{H}^{(\ell)}$: $\tilde{\mathbf{H}} = \text{LN}(\mathbf{H}^{(\ell)})$ (256 parameters: weight + bias)
2. **Multi-head self-attention** with $H = 4$ heads, $d_h = d_v = 32$:

$$\mathbf{Q},\mathbf{K},\mathbf{V} = \tilde{\mathbf{H}}\mathbf{W}_{QKV} \quad \mathbf{W}_{QKV} \in \mathbb{R}^{128\times 384},\ \mathbf{b}_{QKV} \in \mathbb{R}^{384} \text{(49,536 parameters)}$$
$$\text{Attn}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_h}}\right)\mathbf{V}$$
$$\mathbf{A} = \text{Attn}\cdot\mathbf{W}_O + \mathbf{b}_O \quad \mathbf{W}_O \in \mathbb{R}^{128\times 128} \text{(16,512 parameters)}$$

   Dropout ($p = 0.1$) on the attention output before the residual add.
3. **Residual:** $\mathbf{H}' = \mathbf{H}^{(\ell)} + \mathbf{A}$
4. **LayerNorm**(128) on $\mathbf{H}'$: $\tilde{\mathbf{H}}' = \text{LN}(\mathbf{H}')$ (256 parameters)
5. **Feed-forward** (two-layer MLP with GELU):

$$\mathbf{F}_1 = \text{GELU}(\tilde{\mathbf{H}}'\mathbf{W}_1 + \mathbf{b}_1) \quad \mathbf{W}_1 \in \mathbb{R}^{128\times 512} \text{(66,048 parameters)}$$
$$\mathbf{F}_2 = \mathbf{F}_1\mathbf{W}_2 + \mathbf{b}_2 \quad \mathbf{W}_2 \in \mathbb{R}^{512\times 128} \text{(65,664 parameters)}$$

   Dropout ($p = 0.1$) applied after $\mathbf{F}_1$ (between the two linear layers).
6. **Residual:** $\mathbf{H}^{(\ell+1)} = \mathbf{H}' + \mathbf{F}_2$

Per-layer parameter count: $49{,}536 + 16{,}512 + 66{,}048 + 65{,}664 + 256 + 256 = 198{,}272$. **Total encoder ($\times 4$):** $793{,}088$ **parameters.**

The encoder output $\mathbf{H}^{(4)} \in \mathbb{R}^{B\times(t-1)n\times 128}$ is a contextualised representation of every cell-transition token, with each token's representation updated by attending to all $(t-1)n - 1 = 1{,}979$ other tokens in the orbit. This is the representation that both downstream heads read from.

RULE STREAM

$$\mathbf{r}_{\text{pool}} = \frac{1}{(t-1)n} \sum_{m=1}^{(t-1)n} \mathbf{H}_m^{(4)} \quad \in \mathbb{R}^{B\times 128} \text{(0 parameters)}$$
$$\mathbf{r}_1 = \text{Dropout}(\text{GELU}(\mathbf{r}_{\text{pool}}\mathbf{W}_{R,1} + \mathbf{b}_{R,1})) \quad \mathbf{W}_{R,1} \in \mathbb{R}^{128\times 64} \text{(8,256 parameters)}$$
$$\mathbf{r}_{\text{out}} = \mathbf{r}_1\mathbf{W}_{R,2} + \mathbf{b}_{R,2} \quad \mathbf{W}_{R,2} \in \mathbb{R}^{64\times 8} \text{(520 parameters)}$$
$$\in \mathbb{R}^{B\times 8} \quad \text{(8 raw logits, no activation)}$$

Mean pooling is appropriate here because the rule signature is distributed *uniformly* across all $(t-1)n = 1{,}980$ tokens: every token $(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}, a_i^{t'+1})$ contributes equally to the empirical rule table. Averaging over all tokens gives the most statistically efficient estimate of the rule bits. The 8 output logits are passed to `BCEWithLogitsLoss`, which applies sigmoid independently to each bit during training. At inference, $\hat{r}_b = \mathbf{1}[\mathbf{r}_{\text{out},b} > 0]$ for each bit $b \in \{0, \ldots, 7\}$, and the predicted rule is $\hat{R} = \sum_{b=0}^{7} \hat{r}_b \cdot 2^b$.

Rule stream parameter count: $8{,}256 + 520 = 8{,}776$**.**

α STREAM — STATPOOL

Why not mean pool for α? Mean pooling loses distributional shape. Two orbits with $\alpha = 0.2$ and $\alpha = 0.8$ respectively can have similar mean activations if the rule itself has high or low average activity — the mean confounds rule activity with noise level. The α signal lives in the *spread* of activations: at low α, most cells are frozen (their token activations look like the rule's quiescent state) while the few updated cells produce very different activations, creating high variance across tokens. StatPool captures this.

StatPool operation:

$$\boldsymbol{\mu} = \frac{1}{(t-1)n} \textstyle\sum_m \mathbf{H}_m^{(4)} \quad \in \mathbb{R}^{B\times 128}$$
$$\boldsymbol{\sigma} = \sqrt{\frac{1}{(t-1)n} \textstyle\sum_m (\mathbf{H}_m^{(4)} - \boldsymbol{\mu})^2} \quad \in \mathbb{R}^{B\times 128}$$
$$\boldsymbol{\mu}_{\max} = \max_m \mathbf{H}_m^{(4)} \quad \in \mathbb{R}^{B\times 128}$$
$$\boldsymbol{\mu}_{\min} = \min_m \mathbf{H}_m^{(4)} \quad \in \mathbb{R}^{B\times 128}$$
$$\mathbf{s} = [\boldsymbol{\mu} \parallel \boldsymbol{\sigma} \parallel \boldsymbol{\mu}_{\max} \parallel \boldsymbol{\mu}_{\min}] \quad \in \mathbb{R}^{B\times 512}$$

where max and min are taken element-wise over the token dimension (m indexes all $(t-1)n$ tokens). $\mathbf{s}$ is then passed through a two-layer MLP:

$$\mathbf{s}_1 = \text{GELU}(\mathbf{s}\mathbf{W}_{s1} + \mathbf{b}_{s1}) \quad \mathbf{W}_{s1} \in \mathbb{R}^{512\times 256} \text{(131,328 parameters)}$$
$$\mathbf{s}_1 \leftarrow \text{LayerNorm}(256)(\mathbf{s}_1) \text{(512 parameters)}$$
$$\mathbf{s}_1 \leftarrow \text{Dropout}(p = 0.1)(\mathbf{s}_1)$$
$$\mathbf{s}_{\text{pool}} = \text{GELU}(\mathbf{s}_1\mathbf{W}_{s2} + \mathbf{b}_{s2}) \quad \mathbf{W}_{s2} \in \mathbb{R}^{256\times 128} \text{(32,896 parameters)}$$
$$\in \mathbb{R}^{B\times 128}$$

The LayerNorm between the two projections stabilises training by preventing the large activation scale introduced by concatenating four 128-dimensional statistics — the concatenated input has $4\times$ the scale of any individual statistic.

StatPool parameter count: $131{,}328 + 512 + 32{,}896 = 164{,}736$**.**

α HEAD

$$\begin{aligned}
\mathbf{a}_1 &= \text{Dropout}(\text{GELU}(\mathbf{s}_{\text{pool}}\mathbf{W}_{a1} + \mathbf{b}_{a1})) \quad \mathbf{W}_{a1} \in \mathbb{R}^{128\times 64} \text{(8,256 parameters)} \\
\mathbf{a}_2 &= \text{GELU}(\mathbf{a}_1\mathbf{W}_{a2} + \mathbf{b}_{a2}) \quad \mathbf{W}_{a2} \in \mathbb{R}^{64\times 32} \text{(2,080 parameters)} \\
\mathbf{a}_{\text{out}} &= \mathbf{a}_2\mathbf{W}_{a3} + \mathbf{b}_{a3} \quad \mathbf{W}_{a3} \in \mathbb{R}^{32\times 10} \text{(330 parameters)} \\
&\in \mathbb{R}^{B\times 10} \quad \text{(10 raw logits, no activation)}
\end{aligned}$$

The 10 logits correspond to $\alpha \in \{0.1, 0.2, \ldots, 1.0\}$. At inference: $\hat{\alpha} = \texttt{ALPHA_VALUES}[\arg\max_b a_{\text{out},b}]$. The discrete 10-class formulation guarantees a maximum neural-head prediction error of 0.05 (half the class spacing of 0.1), regardless of model quality. This is tighter than regression on a bounded interval, which can in principle predict any value in $[0.1, 1.0]$ but is harder to calibrate near the boundaries.

α head parameter count: $8{,}256 + 2{,}080 + 330 = 10{,}666$**.**

HYBRID ESTIMATOR (INFERENCE ONLY)

At inference, for every test pair (rule, orbit), αM first uses the rule stream to predict the rule $\hat{R}$. It then computes $\hat{\alpha}$ analytically rather than using the neural α head:

$$\hat{\alpha} = \text{median}_{\ell=1}^{20}\left\{ \frac{\left|\{(t', i) : a_i^{t'+1} \neq a_i^{t'}\}\right|}{\left|\{(t', i) : \hat{R}(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) \neq a_i^{t'}\}\right|} \right\}_\ell \tag{4}$$

The denominator counts cell transitions at which the predicted rule would change the cell ($\hat{R}(\text{nbr}) \neq a_i^{t'}$); the numerator counts cells that actually changed ($a_i^{t'+1} \neq a_i^{t'}$). Under α-asynchronous noise a cell changes only if it is selected for update (probability α) and the rule output differs from its current state, and a cell whose rule output equals its current state never changes. When $\hat{R}$ is the true rule, every changed cell is therefore counted in the denominator, and the ratio is the empirical fraction of "should-change" cells that were updated, which equals α in expectation. The median is taken over 20 independently generated orbits to reduce variance. The raw estimate is then snapped to the nearest value on the 0.1 grid: $\hat{\alpha} \leftarrow 0.1 \cdot \text{round}(10\hat{\alpha})$.

The neural α head is used as a fallback when the denominator in Equation 4 is zero, i.e. when the predicted rule would change no cell anywhere in the orbit. This holds identically for rule 204 (the identity rule, whose output always equals the centre cell), at every α. For rules 0 and 255 it would require the orbit to be entirely 0 (respectively 1) from the first transition, which does not occur for the generated initial configurations; these rules therefore use the hybrid estimator (Appendix G.8). For rule 204, both estimators fail because the orbit is constant regardless of α (Theorem H.4.2); this is an identifiability limit, not a model failure. On the held-out test set the hybrid estimator is used for all 770 pairs; across all 256 rules the neural fallback handles 0.4% of pairs (Table 52).

FORWARD PASS SUMMARY

$$\underbrace{[B,\,(t-1)n,\,4]}_{\text{raw tokens}} \xrightarrow{\text{Linear}(4\to 128)} \underbrace{[B,\,(t-1)n,\,128]} \xrightarrow{+\,\text{PE}} \underbrace{[B,\,(t-1)n,\,128]} \xrightarrow{\text{Enc}\times 4} \underbrace{[B,\,(t-1)n,\,128]}_{\mathbf{H}^{(4)}} \begin{cases} \xrightarrow{\text{mean pool+MLP}} [B, 8] & \text{rule logits} \\ \xrightarrow{\text{StatPool+MLP}} [B, 10] & \alpha \text{ logits} \end{cases}$$

B.2 SKEWM: COMPLETE SPECIFICATION

skewM solves the s-skewed inverse problem. It takes a flat sequence of 3,980 five-dimensional cell-transition tokens and produces three outputs: 8 rule bit logits, 20 s-class logits, and 1 s-regression value.

CONSTANTS

$$
\begin{aligned}
n &= 20 \quad \text{(grid width)} \\
t &= 200 \quad \text{(timesteps)} \\
t-1 &= 199 \quad \text{(transitions)} \\
(t-1)\times n &= 199 \times 20 = 3{,}980 \quad \text{(tokens per orbit)} \\
d &= 128 \quad \text{(model dimension)} \\
H &= 4 \quad \text{(attention heads)}, \quad d_h = d/H = 32 \quad \text{(head dimension)} \\
L &= 4 \quad \text{(encoder layers)} \\
d_{\text{ff}} &= 512 \quad \text{(feed-forward dimension)} \\
\text{TOKEN_DIM} &= 5 \quad \text{(token feature dimension)} \\
N_s &= 20 \quad \text{(number of discrete } s \text{ classes)} \\
p &= 0.1 \quad \text{(dropout rate)}
\end{aligned}
$$

TOKENISATION AND THE `DID_CHANGE` FEATURE

Each orbit $\mathbf{O} \in \{0,1\}^{t\times n}$ is tokenised by `orbit_to_tokens`:

$$
\mathbf{x}_{t',i} = \big(a_{i-1}^{t'},\ a_i^{t'},\ a_{i+1}^{t'},\ a_i^{t'+1},\ \underbrace{\mathbf{1}[a_i^{t'+1} \neq a_i^{t'}]}_{\texttt{did_change}}\big) \quad \forall\, t' \in \{0,\dots,t-2\},\ i \in \{0,\dots,n-1\} \tag{5}
$$

The fifth feature `did_change` $\in \{0.0, 1.0\}$ is unique to skewM. Under s-skewed noise, at each timestep exactly s contiguous cells are chosen and updated; all remaining $n-s$ cells retain their current state. A cell that retains its state will have $a_i^{t'+1} = a_i^{t'}$, so `did_change` $= 0$, regardless of what the rule would have predicted. A cell inside the update block will have `did_change` $= 1$ if the rule actually changed its state. This means that `did_change` directly partitions each row into "inside the block" and "outside the block" cells, which is exactly the contiguity signal that identifies s. Including it as an explicit feature means the attention mechanism can detect contiguous runs of `did_change`$= 1$ directly, rather than reconstructing the partition from the raw $(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}, a_i^{t'+1})$ tuples. The result is a $[3{,}980, 5]$ float32 array.

SHARED BACKBONE

Input projection. `nn.Linear`(5, 128) maps each 5-dimensional token to $d = 128$.

$$
\mathbf{H}^{(0)} = \mathbf{X}\mathbf{W}_{\text{in}}^{\top} + \mathbf{b}_{\text{in}} \quad \in \mathbb{R}^{B\times(t-1)n\times 128} \qquad \text{(768 parameters)}
$$

TripletPE2D. Identical construction to αM but instantiated for $(t-1, n) = (199, 20)$. The time axis encoding covers positions $t' \in \{0,\dots,t-2\}$ and the cell axis encoding covers $i \in \{0,\dots,n-1\}$, concatenated to 128 dimensions. No learnable parameters; registered buffer only.

Transformer encoder. Identical specification to αM: 4 layers, PreNorm, 4 heads ($d_h = 32$), $d_{\text{ff}} = 512$, GELU, dropout $p = 0.1$, residual connections. **Per-layer parameters: 198,272. Total encoder: 793,088.**

The encoder output $\mathbf{H}^{(4)} \in \mathbb{R}^{B\times(t-1)n\times 128}$ contains one contextualised representation per cell-transition token. At this point, each token representation has attended to all $(t-1)n - 1 = 3{,}979$ other tokens, allowing the model to compare the `did_change` pattern in one row with every other row and every other cell.

RULE STREAM

Identical to αM: mean pool over all $(t-1)n = 3{,}980$ tokens, then `Linear`(128→64) + GELU + Dropout + `Linear`(64→8).

$$\mathbf{r}_{\text{pool}} = \frac{1}{(t-1)n} \sum_{m=1}^{(t-1)n} \mathbf{H}_m^{(4)} \quad \in \mathbb{R}^{B\times 128}$$
$$\mathbf{r}_{\text{out}} = \text{MLP}(\mathbf{r}_{\text{pool}}) \quad \in \mathbb{R}^{B\times 8}$$

Mean pooling distributes equally over all tokens; the rule signal is present in *every* token regardless of whether the cell was inside or outside the update block (both provide valid rule observations when the cell did update, and stable-state observations when it did not). **Rule head parameters: 8,776.**

s STREAM — SPATIALMULTISCALEPOOL

The s stream uses **SpatialMultiScalePool** (introduced in this work), which extracts three complementary views of the orbit and combines them. Before the three branches, the encoder output is reshaped from a flat token sequence into a 2D grid:

$$\mathbf{H}^{(4)} \in \mathbb{R}^{B\times(t-1)n\times 128} \xrightarrow{\texttt{view}} \mathbf{H}_{\text{2D}} \in \mathbb{R}^{B\times(t-1)\times n\times 128} \tag{6}$$

where the first index of the reshaped tensor is the timestep (transition) and the second is the cell index. This reshape is valid because the tokenisation enumerates tokens as $(t'=0, i=0), (t'=0, i=1), \ldots, (t'=0, i=n-1), (t'=1, i=0), \ldots$ in row-major order, which is exactly what $\texttt{view}(B,\, t-1,\, n,\, 128)$ recovers.

Branch A — Global statistics.

$$\boldsymbol{\mu}_A = \mathbf{H}^{(4)}\text{.mean(dim=1)} \quad \in \mathbb{R}^{B\times 128}$$
$$\boldsymbol{\sigma}_A = \mathbf{H}^{(4)}\text{.std(dim=1)} \quad \in \mathbb{R}^{B\times 128}$$
$$\boldsymbol{m}_A = \mathbf{H}^{(4)}\text{.max(dim=1).values} \quad \in \mathbb{R}^{B\times 128}$$
$$\boldsymbol{n}_A = \mathbf{H}^{(4)}\text{.min(dim=1).values} \quad \in \mathbb{R}^{B\times 128}$$
$$\text{branch_a} = [\boldsymbol{\mu}_A \parallel \boldsymbol{\sigma}_A \parallel \boldsymbol{m}_A \parallel \boldsymbol{n}_A] \quad \in \mathbb{R}^{B\times 512} \quad \text{(0 learnable parameters)}$$

Branch A aggregates over *all* $(t-1)n = 3{,}980$ tokens simultaneously, capturing the global distributional shape of the orbit. The std and max/min statistics encode the range of activation values, which varies with s: at low s, only a small fraction of cells update per timestep, producing a bimodal distribution of activations (updated vs. unchanged cells) with high std and a large max$-$min gap; at high s (near-synchronous), the distribution is unimodal and more concentrated.

Branch B — Temporal statistics.

$$\mathbf{T}_{\text{pool}} = \mathbf{H}_{\text{2D}}.\texttt{mean}(\texttt{dim}{=}2) \in \mathbb{R}^{B\times(t-1)\times 128},$$
$$\boldsymbol{\mu}_B = \mathbf{T}_{\text{pool}}.\texttt{mean}(\texttt{dim}{=}1) \in \mathbb{R}^{B\times 128},$$
$$\boldsymbol{\sigma}_B = \mathbf{T}_{\text{pool}}.\texttt{std}(\texttt{dim}{=}1) \in \mathbb{R}^{B\times 128},$$
$$\text{branch_b} = [\boldsymbol{\mu}_B \mid \boldsymbol{\sigma}_B] \in \mathbb{R}^{B\times 256} \quad \text{(0 learnable parameters)},$$

where $\mathbf{T}_{\text{pool}}$ averages over the n cell positions at each timestep, and $\boldsymbol{\mu}_B$, $\boldsymbol{\sigma}_B$ are the mean and standard deviation across timesteps.

Branch B first collapses the cell dimension at each timestep, then aggregates over timesteps. The per-timestep average $\mathbf{T}_{\text{pool}}[t']$ is a summary of what happened at timestep t' across all n cells. The temporal std $\boldsymbol{\sigma}_B$ captures how much the per-timestep summary varies across time — at low s, the update block position varies randomly per timestep, so the cell-averaged representation fluctuates; at high s (most cells update), the per-timestep average is more stable.

Branch C — Spatial statistics with 1D convolution.

$$\begin{aligned}
\mathbf{C}_{\text{pool}} &= \mathbf{H}_{\text{2D}}.\text{mean(dim=1)} \quad \in \mathbb{R}^{B\times n\times 128} \\
\mathbf{C}_{\text{conv}} &= \text{Conv1d}_2\big(\text{GELU}(\text{Conv1d}_1(\mathbf{C}_{\text{pool}}^\top))\big)^\top \quad \in \mathbb{R}^{B\times n\times 128} \\
\boldsymbol{\mu}_C &= \mathbf{C}_{\text{conv}}.\text{mean(dim=1)} \quad \in \mathbb{R}^{B\times 128} \\
\boldsymbol{\sigma}_C &= \mathbf{C}_{\text{conv}}.\text{std(dim=1)} \quad \in \mathbb{R}^{B\times 128} \\
\text{branch_c} &= [\boldsymbol{\mu}_C \parallel \boldsymbol{\sigma}_C] \quad \in \mathbb{R}^{B\times 256}
\end{aligned}$$

where $\mathbf{C}_{\text{pool}}$ averages over the $t-1$ timesteps at each cell position.

The two `Conv1d` layers operate over the cell dimension (dim=1 after transpose), with:

- $\texttt{Conv1d}_1$: in_channels=128, out_channels=128, kernel_size=3, padding=1, followed by GELU. (49,280 parameters)
- $\texttt{Conv1d}_2$: in_channels=128, out_channels=128, kernel_size=3, padding=1, followed by GELU. (49,280 parameters)

The kernel size of 3 means each output cell position attends to itself and its two spatial neighbours, directly modelling the local contiguity structure of the update block. $\mathbf{C}_{\text{pool}}$ is the time-averaged representation of each cell position — a fingerprint of how often that cell was part of an update block across all $t-1$ timesteps. Cells at positions that are frequently the *edge* of an update block will have different time-averaged representations than cells at positions frequently in the *interior* of a block; the convolution detects these local spatial patterns. After convolution, mean and std over the n cell positions capture both the average level and the spatial variability of these edge/interior patterns. **Branch C parameters: 98,560.**

Projection MLP. The three branches are concatenated and projected to $d = 128$:

$$\begin{aligned}
\mathbf{c} &= [\text{branch_a} \parallel \text{branch_b} \parallel \text{branch_c}] \quad \in \mathbb{R}^{B\times(512+256+256)} = \mathbb{R}^{B\times 1024} \\
\mathbf{p}_1 &= \text{GELU}(\mathbf{c}\mathbf{W}_{p1} + \mathbf{b}_{p1}) \quad \mathbf{W}_{p1} \in \mathbb{R}^{1024\times 512} \text{(524,800 parameters)} \\
\mathbf{p}_1 &\leftarrow \text{LayerNorm}(512)(\mathbf{p}_1) \text{(1,024 parameters)} \\
\mathbf{p}_1 &\leftarrow \text{Dropout}(p = 0.1)(\mathbf{p}_1) \\
\mathbf{s}_{\text{feat}} &= \text{GELU}(\mathbf{p}_1\mathbf{W}_{p2} + \mathbf{b}_{p2}) \quad \mathbf{W}_{p2} \in \mathbb{R}^{512\times 128} \text{(65,664 parameters)} \\
&\quad \in \mathbb{R}^{B\times 128}
\end{aligned}$$

The 1024-dimensional concatenation is projected down to 128 in two steps (1024→512→128) rather than one (1024→128) to prevent information bottlenecking and allow the model to learn non-linear interactions between the three branches before compression. LayerNorm is placed after the first projection to stabilise the scale of the 512-dimensional intermediate representation before the second linear layer. **Projection MLP parameters:** $524{,}800 + 1{,}024 + 65{,}664 = 591{,}488$**.**

Total SpatialMultiScalePool parameters: $98{,}560 + 591{,}488 = 690{,}048$**.**

DUAL s HEAD

The s stream uses two parallel prediction heads, both reading from $\mathbf{s}_{\text{feat}}$.

Classification head (primary).

$$\begin{aligned}
\mathbf{s}_{c1} &= \text{Dropout}(\text{GELU}(\mathbf{s}_{\text{feat}}\mathbf{W}_{c1} + \mathbf{b}_{c1})) \quad \mathbf{W}_{c1} \in \mathbb{R}^{128\times 64} \text{(8,256 parameters)} \\
\mathbf{s}_{\text{cls}} &= \mathbf{s}_{c1}\mathbf{W}_{c2} + \mathbf{b}_{c2} \quad \mathbf{W}_{c2} \in \mathbb{R}^{64\times 20} \text{(1,300 parameters)} \\
&\quad \in \mathbb{R}^{B\times 20} \quad \text{(20 raw logits, no activation)}
\end{aligned}$$

The 20 logits correspond to $s \in \{1, 2, \ldots, n\}$. At inference: $\hat{s}_{\text{cls}} = S_\text{VALUES}[\arg\max_b\ s_{\text{cls},b}]$, where $S_\text{VALUES} = [1, 2, \ldots, n]$. This head is supervised by `CrossEntropyLoss` and contributes the primary discrete precision signal for s identification. **Classification head parameters:** $8{,}256 + 1{,}300 = 9{,}556$**.**

Regression head (ordinal regulariser).

$$\mathbf{s}_{r1} = \text{Dropout}(\text{GELU}(\mathbf{s}_{\text{feat}}\mathbf{W}_{r1} + \mathbf{b}_{r1})) \quad \mathbf{W}_{r1} \in \mathbb{R}^{128\times 64} \text{(8,256 parameters)}$$
$$\mathbf{s}_{\text{reg}} = \text{Sigmoid}(\mathbf{s}_{r1}\mathbf{W}_{r2} + \mathbf{b}_{r2}) \quad \mathbf{W}_{r2} \in \mathbb{R}^{64\times 1} \text{(65 parameters)}$$
$$\in [0,1]^{B\times 1}$$

The regression head predicts $s/n \in [0,1]$ and is supervised by MSE loss targeting the true s/n. The Sigmoid activation enforces the $[0,1]$ range, appropriate for a normalised block fraction. The MSE loss penalises large ordinal errors proportionally: predicting $\hat{s} = 10$ when the true $s = 1$ incurs a larger MSE loss than predicting $\hat{s} = 2$, which pure cross-entropy does not achieve (all wrong predictions are penalised equally). This acts as a soft inductive bias toward the correct neighbourhood of s before the classification head has converged. The regression head is used only during training; at inference, $\hat{s}$ is always taken from the classification head. **Regression head parameters:** $8{,}256 + 65 = 8{,}321$**.**

HYBRID ESTIMATOR FOR s

At inference, skewM applies the same hybrid estimation principle as αM. Given the predicted rule $\hat{R}$:

$$\hat{s}_{\text{hybrid}} = n \cdot \text{median}_{\ell=1}^{20} \left\{ \frac{\left|\{(t',i) : a_i^{t'+1} \neq a_i^{t'}\}\right|}{\left|\{(t',i) : \hat{R}(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) \neq a_i^{t'}\}\right|} \right\}_\ell \tag{7}$$

Under s-skewed noise, exactly s of the n cells are updated at each timestep, and an updated cell changes only if the rule output differs from its current state. The expected fraction of "should-change" cells that change is therefore s/n, and multiplying the observed ratio by n recovers s. The raw estimate is then rounded to the nearest integer in $\{1,\ldots,n\}$: $\hat{s}_{\text{hybrid}} \leftarrow \text{round}(\hat{s}_{\text{hybrid}})$. On the held-out test set the hybrid estimator is used for 1,539 of 1,540 pairs (Appendix G.2).

FORWARD PASS SUMMARY

$$\underbrace{[B,\,(t-1)n,\,5]}_{\text{raw tokens}} \xrightarrow{\text{Linear}(5\to 128)} [B,\,(t-1)n,\,128] \xrightarrow{+\,\text{PE}} [B,\,(t-1)n,\,128] \xrightarrow{\text{Enc}\times 4} \underbrace{[B,\,(t-1)n,\,128]}_{\mathbf{H}^{(4)}} \begin{cases} \xrightarrow{\text{mean pool+MLP}} [B,8] \text{ rule logits} \\ \xrightarrow{\text{SpatialMultiScalePool+MLP}} \begin{cases} [B,20] & s \text{ cls logits} \\ [B,1] & s \text{ reg } \in [0,1] \end{cases} \end{cases}$$

B.3 TEMPM: COMPLETE SPECIFICATION

tempM solves the temporally stochastic inverse problem. It takes $k = 8$ independent orbits of raw 4-dimensional tokens plus a 24-dimensional frac_stats summary, and produces two rule bit logit vectors, a τ prediction, per-timestep assignment probabilities, and two auxiliary direct-decoder outputs.

CONSTANTS

$$n = 50, \quad t = 200, \quad t-1 = 199$$
$$k = 8 \quad \text{(orbits per sample)}$$
$$(t-1)\times n = 199\times 50 = 9{,}950 \quad \text{(tokens per orbit)}$$
$$N_{\text{feat}} = 12 \quad \text{(token feature dim: 4 raw + 8 pattern embedding)}$$
$$N_{\text{emb}} = 8 \quad \text{(PatternEmbedding dimension)}$$
$$D_1 = 128, \quad H_1 = 4, \quad L_1 = 4, \quad d_{\text{ff},1} = 512 \quad \text{(CellEncoder)}$$
$$D_2 = 256, \quad H_2 = 8, \quad L_2 = 6, \quad d_{\text{ff},2} = 1{,}024 \quad \text{(TimestepEncoder)}$$
$$p = 0.1, \quad \tau_{\text{min}} = 0.1, \quad \tau_{\text{max}} = 0.9$$

TOKENISATION AND PATTERNEMBEDDING

Raw tokens. Each orbit $\mathbf{O} \in \{0,1\}^{t\times n}$ is tokenised identically to αM and skewM but without the `did_change` feature:

$$\mathbf{x}_{t',i}^{\text{raw}} = (a_{i-1}^{t'},\, a_i^{t'},\, a_{i+1}^{t'},\, a_i^{t'+1}) \quad \in \{0,1\}^4 \quad \forall\, t' \in \{0,\ldots,t-2\},\ i \in \{0,\ldots,n-1\} \tag{8}$$

giving $[k, (t-1)n, 4]$ raw tokens per sample.

PatternEmbedding. Before entering the CellEncoder, each token is augmented with a learned 8-dimensional pattern embedding. The neighbourhood index $j_{t',i} = 4a_{i-1}^{t'} + 2a_i^{t'} + a_{i+1}^{t'} \in \{0, \ldots, 7\}$ indexes a lookup table $\mathbf{E} \in \mathbb{R}^{8\times 8}$:

$$\mathbf{x}_{t',i} = [\mathbf{x}_{t',i}^{\text{raw}} \parallel \mathbf{E}[j_{t',i}]] \quad \in \mathbb{R}^{12} \tag{9}$$

The embedding weights are initialised from $\mathcal{N}(0, 0.1)$ (deliberately wider than the global $\mathcal{N}(0, 0.02)$ to spread the 8 pattern representations apart from initialisation). The PatternEmbedding provides a direct shortcut for the per-pattern mixture signal: the model does not need to re-identify which neighbourhood pattern a token belongs to from its raw 3-bit representation — the embedding encodes this identity directly. **PatternEmbedding parameters:** $8 \times 8 = 64$**.**

PATH A: ANALYTICAL τ ESTIMATION (ANALYTICALTAUMLP)

τ is estimated analytically from $\texttt{frac_stats} \in \mathbb{R}^{B\times 24}$ rather than from the transformer representations. The 24-dimensional frac_stats vector is structured as:

$$\texttt{frac_stats} = [\underbrace{\text{frac}[0], \ldots, \text{frac}[7]}_{8}, \underbrace{\text{var}[0], \ldots, \text{var}[7]}_{8}, \underbrace{\text{cnt}[0], \ldots, \text{cnt}[7]}_{8}] \tag{10}$$

where $\text{frac}[j] = \hat{\mathbb{E}}[a_i^{t'+1} \mid j_{t',i} = j]$ is the empirical output fraction for pattern j, $\text{var}[j] = \text{frac}[j](1 - \text{frac}[j])$ is its Bernoulli variance, and $\text{cnt}[j]$ is the normalised frequency of pattern j in the orbit. In expectation, $\text{frac}[j] = \tau \cdot R_2[j] + (1-\tau) \cdot R_1[j]$, so for disagreement patterns ($R_1[j] \neq R_2[j]$), $\text{frac}[j] \in \{\tau, 1-\tau\}$ — directly encoding $\min(\tau, 1-\tau)$.

Stage 1: Three-method analytical blend (no parameters). Three independent estimators are blended:

- **Method A (variance-based):** $\max_j \text{var}[j] \approx \tau(1-\tau)$ at the most disagreed pattern. $\hat{\tau}_A = 0.5 - \sqrt{0.25 - \max_j \text{var}[j]}$, clamped to $[\tau_{\text{min}}, 0.5]$.
- **Method B (interior frac):** Among well-observed patterns ($\text{cnt}[j] > 0.005$), the minimum frac value in $(0.08, 0.92)$ equals $\min(\tau, 1-\tau)$. $\hat{\tau}_B = \min_{j:\text{frac}[j]\in(0.08,0.92)} \text{frac}[j]$.
- **Method C (top-2 variance):** The two patterns with highest variance are the most-disagreed patterns. $\hat{\tau}_C = \min(\text{frac}[\text{top}_1], \text{frac}[\text{top}_2])$.

All three estimates are in $[\tau_{\text{min}}, 0.5]$ by construction. A confidence score $\gamma = \min(1, \max_j \text{var}[j]/0.25) \in [0, 1]$ measures how strongly the disagreement signal is present in the orbit. The three estimates are blended with confidence-weighted coefficients:

$$w_A = 0.4 + 0.2(1-\gamma), \quad w_B = 0.3\gamma, \quad w_C = 0.3\gamma \tag{11}$$

$$\hat{\tau}_{\text{analytical}} = \frac{w_A\hat{\tau}_A + w_B\hat{\tau}_B + w_C\hat{\tau}_C}{w_A + w_B + w_C}, \quad \text{clamped to } [\tau_{\text{min}}, 0.5] \tag{12}$$

Method A dominates when confidence is low (all-agreement rules, $\gamma \approx 0$), defaulting to $\hat{\tau} \approx 0.5$. Methods B and C contribute more as the disagreement signal strengthens.

Stage 2: Correction MLP (learned). A learned correction MLP refines the analytical estimate. The input features are 41-dimensional:

$$\mathbf{f}_{\text{corr}} = [\underbrace{\texttt{frac_stats}}_{24}, \underbrace{\text{sorted_frac}}_{8}, \underbrace{\text{adj_diff}}_{8}, \underbrace{\gamma}_{1}] \quad \in \mathbb{R}^{41} \tag{13}$$

where $\text{sorted_frac} = \text{sort}(\text{frac}[0:8])$ (ascending order, 8 values), and $\text{adj_diff}[m] = |\text{sorted_frac}[m] - \text{sorted_frac}[m-1]|$ is the gap between consecutive sorted frac values (8 values, first entry = 0). The sorted frac values expose the distributional shape of the per-pattern mixture

fractions without permutation sensitivity; the adj_diff values detect bimodality (large gaps between frac values signal strong disagreement at some patterns).

The correction MLP:

$$\begin{aligned}
\mathbf{m}_1 &= \text{Dropout}(\text{GELU}(\mathbf{f}_{\text{corr}}\mathbf{W}_1 + \mathbf{b}_1)) \quad \mathbf{W}_1 \in \mathbb{R}^{41\times 256} \text{(10,752 parameters)} \\
\mathbf{m}_2 &= \text{Dropout}(\text{GELU}(\mathbf{m}_1\mathbf{W}_2 + \mathbf{b}_2)) \quad \mathbf{W}_2 \in \mathbb{R}^{256\times 256} \text{(65,792 parameters)} \\
\mathbf{m}_3 &= \text{GELU}(\mathbf{m}_2\mathbf{W}_3 + \mathbf{b}_3) \quad \mathbf{W}_3 \in \mathbb{R}^{256\times 128} \text{(32,896 parameters)} \\
\delta &= \text{Tanh}(\mathbf{m}_3\mathbf{W}_4 + \mathbf{b}_4) \quad \mathbf{W}_4 \in \mathbb{R}^{128\times 1} \text{(129 parameters)} \\
&\in (-1, 1)
\end{aligned}$$

The Tanh output is scaled by an *adaptive correction range*:

$$\delta_{\text{eff}} = \delta \cdot \Big(\underbrace{0.08}_{\text{COR_MAX_BASE}} + (1-\gamma) \cdot \underbrace{0.35}_{\text{COR_MAX_EXTRA}} \Big) \tag{14}$$

$$\hat{\tau}_{\text{pred}} = \text{clamp}(\hat{\tau}_{\text{analytical}} + \delta_{\text{eff}},\ \tau_{\text{min}},\ 0.5) \tag{15}$$

When confidence $\gamma = 1$ (strong signal), the maximum correction magnitude is 0.08 — a small refinement that cannot disturb an already-accurate estimate. When $\gamma = 0$ (all-agreement rules, analytical defaults to 0.5), the maximum correction is 0.43, allowing the MLP to use frac_stats patterns to identify that the pair is degenerate and maintain the 0.5 default. The output is always clamped to $[0.1, 0.5]$ — tempM always predicts the symmetric magnitude $\min(\tau, 1-\tau)$; the orientation (τ vs $1-\tau$) is determined externally by the symmetric loss, which evaluates both orientations and uses the one consistent with the predicted rule pair.

The final output layers (Linear $128 \to 1$ and Tanh) are initialised with $\mathcal{N}(0, 0.01)$ and zero bias respectively, so initial corrections are near zero and the analytical estimate dominates early training. **Correction MLP total:** $10{,}752 + 65{,}792 + 32{,}896 + 129 = 109{,}569$ **parameters.**

PATH B: HIERARCHICAL TRANSFORMER ENCODER

The hierarchical encoder processes the $k = 8$ orbits through two sequential transformer stages, avoiding the $O((t-1)^2 n^2)$ attention cost of a flat sequence of $9{,}950$ tokens.

CellEncoder (Level 1). The CellEncoder processes each timestep's n cell tokens independently, compressing one row of the orbit to a single $D_1 = 128$-dimensional vector. It is applied to all timesteps of all orbits in a single batched forward pass by reshaping $[B, k, t-1, n, 12]$ to $[B \cdot k \cdot (t-1), n, 12]$.

1. **Input projection:** `Linear`(12, 128). Maps each 12-dimensional token (4 raw + 8 pattern embedding) to $D_1 = 128$. (1,664 parameters)
2. **SinusoidalPE**($D_1 = 128$, max_len$= n$). Standard 1D sinusoidal encoding over the n cell positions. Registered buffer, no learnable parameters.
3. **Transformer encoder,** $L_1 = 4$ **layers, PreNorm:** $d = 128$, $H = 4$ heads, $d_k = 32$, $d_{\text{ff}} = 512$, GELU, dropout 0.1. Same structure as αM/skewM. per-layer: 198,272 parameters; 4 layers: **793,088**
4. **Query-based pooling:** A learned query vector $\mathbf{q} \in \mathbb{R}^{1\times 1\times 128}$ (initialised $\mathcal{N}(0, 0.02)$) attends to all n cell representations via a `MultiheadAttention`(D_1, $H_1 = 4$, dropout p) layer, compressing the row to a single vector:

$$\mathbf{e}_{t'} = \text{LayerNorm}_{D_1}(\text{MHA}(\mathbf{q}, \mathbf{H}_{t'}, \mathbf{H}_{t'})) \quad \in \mathbb{R}^{D_1} \tag{16}$$

where $\mathbf{H}_{t'} \in \mathbb{R}^{n\times 128}$ is the encoder output for timestep t''s row. The learnable query allows the CellEncoder to select which aspects of the spatial pattern to summarise, rather than averaging uniformly. MHA: 66,048; LN: 256; query param: 128; total pooling: **66,432**

CellEncoder output shape: $[B \cdot k, t-1, D_1] = [B \cdot k, 199, 128]$ after reshaping. **CellEncoder total parameters:** $1{,}664 + 793{,}088 + 66{,}432 = 861{,}184$**.**

TimestepEncoder (Level 2). The TimestepEncoder receives the sequence of $t-1$ per-timestep summaries from the CellEncoder and builds representations that span the full temporal extent of the orbit.

1. **Input projection:** $\texttt{Linear}(D_1 = 128, D_2 = 256)$. (32,896 parameters)
2. **SinusoidalPE**($D_2 = 256$, max_len$= t-1$). Standard 1D sinusoidal encoding over the $t-1$ timestep positions. No learnable parameters.
3. **Transformer encoder,** $L_2 = 6$ **layers, PreNorm:** $d = 256$, $H = 8$ heads, $d_k = 32$, $d_{\text{ff}} = 1{,}024$, GELU, dropout 0.1.
   Per-layer breakdown:
   - MHA in-proj $\mathbf{W}_{QKV} \in \mathbb{R}^{256\times 768}$ + bias: 197,376
   - MHA out-proj $\mathbf{W}_O \in \mathbb{R}^{256\times 256}$ + bias: 65,792
   - FF linear 1 $\in \mathbb{R}^{256\times 1024}$ + bias: 263,168
   - FF linear 2 $\in \mathbb{R}^{1024\times 256}$ + bias: 262,400
   - LayerNorm$\times$2 (256): $2 \times 512 = 1{,}024$
   - **Per layer: 789,760**

   6 layers: 4,738,560 parameters.

TimestepEncoder output shape: $[B \cdot k, t-1, D_2] = [B \cdot k, 199, 256]$. **TimestepEncoder total parameters:** $32{,}896 + 4{,}738{,}560 = 4{,}771{,}456$**.**

ConfidenceWeightedOrbitFusion. After the TimestepEncoder, the output is reshaped to $[B, k, t-1, D_2]$. The $k = 8$ orbits are fused using learned soft weights:

$$\mathbf{s}_\ell = \frac{1}{t-1}\sum_{t'} \mathbf{h}_{\ell,t'} \quad \in \mathbb{R}^{B\times D_2} \quad \text{(orbit-level summary, 0 parameters)} \tag{17}$$

$$w_\ell = \text{softmax}_\ell(\text{MLP}(\mathbf{s}_\ell)) \tag{18}$$

$$\mathbf{h} = \textstyle\sum_{\ell=1}^{k} w_\ell \cdot \mathbf{H}_\ell \quad \in \mathbb{R}^{B\times (t-1)\times D_2} \tag{19}$$

where the MLP is Linear$(256 \to 64)$ + GELU + Linear$(64 \to 1)$ (16,513 parameters).

The softmax ensures the weights sum to 1 across the k orbits. An orbit that carries a stronger disagreement signal (e.g., one where the random initial configuration happened to generate more disagreement-pattern observations) will receive a higher weight. The weighting is over orbits, not timesteps — within each orbit, all timesteps contribute equally to the fused representation. **ConfidenceWeightedOrbitFusion parameters: 16,513.**

Per-Timestep Residual Signal

Before the rule head, the per-timestep assignment signal is computed analytically from the raw tokens and frac_stats (no learnable parameters):

$$r_{t'} = \frac{1}{k \cdot n}\sum_{\ell=1}^{k}\sum_{i=1}^{n}\left(a_i^{t'+1}[\ell] - \text{frac}[j^{\ell}_{t',i}]\right) \quad \in \mathbb{R} \tag{20}$$

where frac$[j]$ is the empirical output fraction for pattern j from frac_stats, and $a_i^{t'+1}[\ell]$ is the actual next-state in orbit ℓ. In expectation:

$$\mathbb{E}[r_{t'} \mid R_2 \text{ fired at } t'] = (1-\tau)\sum_j p_j(R_2[j] - R_1[j]) > 0 \quad \text{at disagreement patterns} \tag{21}$$

$$\mathbb{E}[r_{t'} \mid R_1 \text{ fired at } t'] = -\tau\sum_j p_j(R_2[j] - R_1[j]) < 0 \quad \text{at disagreement patterns} \tag{22}$$

The sign of $r_{t'}$ is therefore a direct indicator of which rule fired at timestep t', and is the primary signal used for temporal assignment. This residual is computed on-the-fly during the forward pass with no learnable parameters. **ts_residual shape:** $[B, t-1] = [B, 199]$**.**

TAUCONDITIONEDRULEHEAD

The rule head disentangles the two rules using three parallel paths and the per-timestep residual signal.

Per-timestep scoring and assignment.

$$\mathbf{h}_{t'} \in \mathbb{R}^{D_2=256} \quad \text{(fused orbit representation at timestep } t') \tag{23}$$

$$\text{score}_{t'} = \rho \cdot r_{t'} + \text{score_net}([\mathbf{h}_{t'} \parallel \hat{\tau}_{\text{pred}} \parallel \overline{\text{var}}]) \tag{24}$$

where ρ is a learnable scalar (`residual_weight`) initialised to 5.0, $\hat{\tau}_{\text{pred}} \in \mathbb{R}^1$ is the predicted symmetric τ, and $\overline{\text{var}} \in \mathbb{R}^1$ is the mean variance across patterns (a proxy for signal strength).

The score_net:

$$\begin{aligned}
\text{input} &= [\mathbf{h}_{t'} \parallel \hat{\tau}_{\text{pred}} \parallel \overline{\text{var}}] \in \mathbb{R}^{256+2=258} \\
\mathbf{u}_1 &= \text{Dropout}(\text{GELU}(\mathbf{W}_{u1}\,\text{input} + \mathbf{b}_{u1})) \quad \mathbf{W}_{u1} \in \mathbb{R}^{258\times128} \text{(33,152 parameters)} \\
\mathbf{u}_2 &= \text{GELU}(\mathbf{W}_{u2}\,\mathbf{u}_1 + \mathbf{b}_{u2}) \quad \mathbf{W}_{u2} \in \mathbb{R}^{128\times64} \text{(8,256 parameters)} \\
\text{score}^{\text{net}}_{t'} &= \mathbf{W}_{u3}\,\mathbf{u}_2 + \mathbf{b}_{u3} \quad \mathbf{W}_{u3} \in \mathbb{R}^{64\times1} \text{(65 parameters)} \\
\text{score}_{t'} &= \rho \cdot r_{t'} + \text{score}^{\text{net}}_{t'} \\
\text{prob}_{R_2}[t'] &= \sigma(\text{score}_{t'}) \in (0,1)
\end{aligned}$$

score_net parameters: $33{,}152 + 8{,}256 + 65 = 41{,}473$**.** ρ **(learnable scalar): 1 parameter.**

Rule representations. The fused timestep representations $\mathbf{h} \in \mathbb{R}^{B\times(t-1)\times256}$ are split into R_1 and R_2 representations by soft assignment:

$$\mathbf{rep}_{R_1} = \textstyle\sum_{t'}(1-\text{prob}_{R_2}[t']) \cdot \mathbf{h}_{t'} \;/\; \sum_{t'}(1-\text{prob}_{R_2}[t']) \quad \in \mathbb{R}^{B\times256} \tag{25}$$

$$\mathbf{rep}_{R_2} = \textstyle\sum_{t'} \text{prob}_{R_2}[t'] \cdot \mathbf{h}_{t'} \;/\; \sum_{t'} \text{prob}_{R_2}[t'] \quad \in \mathbb{R}^{B\times256} \tag{26}$$

Timesteps assigned to R_1 (low prob_{R_2}) contribute to $\mathbf{rep}_{R_1}$ and vice versa. Each rule representation is augmented with detached logits from both the DirectRuleDecoder ($\mathbf{d}_{R_1}, \mathbf{d}_{R_2} \in \mathbb{R}^{B\times8}$) and the DistributionalRuleDecoder ($\mathbf{d}^{\text{dist}}_{R_1}, \mathbf{d}^{\text{dist}}_{R_2} \in \mathbb{R}^{B\times8}$) before being fed to the rule head MLPs. Detaching these analytical paths prevents their gradients from interfering with transformer learning:

$$\begin{aligned}
\mathbf{logits}_{R_1} &= \text{head}_{R_1}\big([\mathbf{rep}_{R_1} \parallel \mathbf{d}^{\text{detach}}_{R_1} \parallel (\mathbf{d}^{\text{dist}}_{R_1})^{\text{detach}}]\big) \quad \in \mathbb{R}^{B\times8} \\
\mathbf{logits}_{R_2} &= \text{head}_{R_2}\big([\mathbf{rep}_{R_2} \parallel \mathbf{d}^{\text{detach}}_{R_2} \parallel (\mathbf{d}^{\text{dist}}_{R_2})^{\text{detach}}]\big) \quad \in \mathbb{R}^{B\times8}
\end{aligned}$$

The rule head MLPs (head_{R_1} and head_{R_2}, same architecture, separate weights), each preceded by a LayerNorm on their input:

$$\begin{aligned}
\text{input dim} &= D_2 + 8 + 8 = 256 + 16 = 272 \\
\mathbf{v}_0 &= \text{LayerNorm}(272)(\text{input}) \text{(544 parameters)} \\
\mathbf{v}_1 &= \text{Dropout}(\text{GELU}(\mathbf{W}_{v1}\,\mathbf{v}_0 + \mathbf{b}_{v1})) \quad \mathbf{W}_{v1} \in \mathbb{R}^{272\times256} \text{(69,888 parameters)} \\
\mathbf{v}_2 &= \text{GELU}(\mathbf{W}_{v2}\,\mathbf{v}_1 + \mathbf{b}_{v2}) \quad \mathbf{W}_{v2} \in \mathbb{R}^{256\times128} \text{(32,896 parameters)} \\
\mathbf{v}_{\text{out}} &= \mathbf{W}_{v3}\,\mathbf{v}_2 + \mathbf{b}_{v3} \quad \mathbf{W}_{v3} \in \mathbb{R}^{128\times8} \text{(1,032 parameters)}
\end{aligned}$$

Each rule head: $544 + 69{,}888 + 32{,}896 + 1{,}032 = 104{,}360$ **parameters; both together: 208,720.**

DirectRuleDecoder. The DirectRuleDecoder analytically estimates the rule bits per pattern from frac_stats and $\hat{\tau}_{\text{pred}}$. For each of the 8 neighbourhood patterns, a shared MLP maps per-pattern features to a pair of bit logits (one for R_1, one for R_2). The per-pattern feature vector (7 dimensions) is:

$$\mathbf{q}_j = [\text{frac}[j],\ \hat{\tau},\ 1-\hat{\tau},\ |\text{frac}[j]-\hat{\tau}|,\ |\text{frac}[j]-(1-\hat{\tau})|,\ \text{var}[j],\ \text{cnt}[j]] \quad \in \mathbb{R}^7 \tag{27}$$

A shared MLP maps each $\mathbf{q}_j$ to two logits independently: Linear($7 \to 64$) + GELU + Linear($64 \to 64$) + GELU + Linear($64 \to 2$), applied once per pattern, giving $\mathbf{d}_{R_1}, \mathbf{d}_{R_2} \in \mathbb{R}^{B\times8}$. **DirectRuleDecoder parameters:** $\approx 4{,}802$**.**

DistributionalRuleDecoder. The DistributionalRuleDecoder provides a second analytical rule estimate from the population-level output distribution of frac_stats. For each of the 8 neighbourhood patterns, it estimates the rule bits directly from the empirical output fraction frac$[j]$ and the predicted $\hat{\tau}$: if frac$[j] > 0.5$ it assigns output 1 to the majority rule and 0 to the minority rule, and vice versa, weighted by confidence γ. Concretely, a shared MLP maps a 3-dimensional per-pattern feature $[\text{frac}[j],\ \hat{\tau},\ \gamma]$ to two logits (one per rule), applied independently per pattern: Linear$(3 \to 32)$ + GELU + Linear$(32 \to 2)$. This gives $\mathbf{d}^{\text{dist}}_{R_1}, \mathbf{d}^{\text{dist}}_{R_2} \in \mathbb{R}^{B\times 8}$. **DistributionalRuleDecoder parameters:** ≈ 162**.**

FORWARD PASS SUMMARY

Inputs: $\underbrace{\mathbf{X}^{(\ell)} \in \{0,1\}^{(t-1)n\times 4}}_{\ell=1\ldots k},\ \underbrace{\texttt{frac_stats} \in \mathbb{R}^{24}}$

Path A: $\texttt{frac_stats} \xrightarrow{\text{analytical_tau_robust}} \hat{\tau}_{\text{base}} \in [0.1, 0.5] \xrightarrow{\text{correction_net(41-dim)}} \hat{\tau}_{\text{pred}} \in [0.1, 0.5]$

Path B (per orbit): $\mathbf{X}^{(\ell)} \xrightarrow{\text{+PatEmb}} [(t-1)n, 12] \xrightarrow{\text{reshape}} [t-1, n, 12]$

$\xrightarrow{\text{CellEnc}} [t-1, D_1] \xrightarrow{\text{TimestepEnc}} [t-1, D_2] \xrightarrow{\text{OrbitFuse}} \mathbf{h} \in [B, t-1, D_2]$

Assignment: $\mathbf{h} + r_{t'} \xrightarrow{\text{score_net}+\rho\cdot r_{t'}} \text{prob}_{R_2} \in [B, t-1]$

Rules: $[\mathbf{rep}_{R_1}, \mathbf{rep}_{R_2}, \mathbf{d}_{R_1}, \mathbf{d}_{R_2}, \mathbf{d}^{\text{dist}}_{R_1}, \mathbf{d}^{\text{dist}}_{R_2}] \xrightarrow{\text{rule heads}} \mathbf{logits}_{R_1}, \mathbf{logits}_{R_2} \in [B, 8]$

B.4 stocM: COMPLETE SPECIFICATION

stocM solves the stochastic inverse problem. It shares the hierarchical backbone (CellEncoder, TimestepEncoder, ConfidenceWeightedOrbitFusion) identically with tempM, and diverges in three places: (1) an additional `within_var_stats` input that has no analogue in tempM; (2) a 50-dimensional correction MLP input instead of 41; and (3) a λ-conditioned attention pooling rule head that replaces the per-timestep residual assignment mechanism of tempM.

CONSTANTS

All shared constants are identical to tempM: $n = 50$, $t = 200$, $k = 8$, $t - 1 = 199$, $N_{\text{feat}} = 12$, $D_1 = 128$, $D_2 = 256$, $L_1 = 4$, $L_2 = 6$, $H_1 = 4$, $H_2 = 8$, $d_{\text{ff},1} = 512$, $d_{\text{ff},2} = 1{,}024$, $p = 0.1$. Additional constant: WVAR_DIM $= 8$ (within-variance dimension).

THE `WITHIN_VAR_STATS` SIGNAL

stocM receives a second analytical summary in addition to frac_stats: `within_var_stats` $\in \mathbb{R}^{B\times 8}$.

For each neighbourhood pattern $j \in \{0, \ldots, 7\}$, the within-timestep output variance is the empirical variance of $a_i^{t'+1}$ across all cells i that show pattern j at timestep t', averaged over valid (t', ℓ) pairs (those with at least 2 cells showing pattern j):

$$\texttt{within_var}[j] = \frac{\displaystyle\sum_{\ell=1}^{k}\sum_{t'} \mathbf{1}\left[\textstyle\sum_i \mathbf{1}[j^{\ell}_{t',i} = j] \geq 2\right] \cdot \widehat{\text{Var}}\left[a_i^{t'+1} \mid j^{\ell}_{t',i} = j\right]}{\displaystyle\sum_{\ell=1}^{k}\sum_{t'} \mathbf{1}\left[\textstyle\sum_i \mathbf{1}[j^{\ell}_{t',i} = j] \geq 2\right]} \tag{28}$$

where $\widehat{\text{Var}}[\cdot]$ is the sample variance over all cells i with $j^{\ell}_{t',i} = j$ at that (t', ℓ) pair.

In expectation under stochastic noise:

$$\mathbb{E}[\texttt{within_var}[j]] = \lambda(1-\lambda)(R_2[j] - R_1[j])^2 \tag{29}$$

At disagreement patterns ($R_2[j] \neq R_1[j]$), $(R_2[j] - R_1[j])^2 = 1$, so $\texttt{within_var}[j] \approx \lambda(1-\lambda)$. This signal exists in stocM because $n = 50$ cells independently choose their rule at each timestep, generating 50 independent Bernoulli draws per pattern per timestep. In tempM, all 50 cells use the *same* rule at each timestep, so the within-timestep variance is identically zero — this signal simply does not exist in the tempM data. The within-variance is maximised at $\lambda = 0.5$, where $\lambda(1-\lambda) = 0.25$, and approaches zero as $\lambda \to 0$ or $\lambda \to 1$. This is a richer signal than the frac-based Methods B and C from tempM, because it provides a direct estimate of $\lambda(1-\lambda)$ without requiring identification of which patterns are disagreement patterns.

TOKENISATION AND PATTERNEMBEDDING

Identical to tempM: 4 raw features $(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}, a_i^{t'+1})$ augmented with 8-dimensional PatternEmbedding to give 12-dimensional tokens. PatternEmbedding weights initialised $\mathcal{N}(0, 0.1)$. **PatternEmbedding parameters: 64.**

PATH A: ANALYTICAL λ ESTIMATION (ANALYTICALLAMBDAMLP)

Stage 1: Three-method analytical blend (0 parameters). The analytical estimator uses three methods, with within-variance as Method A (unique to stocM):

- **Method A (within-variance — unique to stocM):** $\max_j \texttt{within_var}[j] \approx \lambda(1-\lambda)$ at the most disagreed pattern. $\hat{\lambda}_A = 0.5 - \sqrt{0.25 - \max_j \texttt{within_var}[j]}$, clamped to $[\lambda_{\text{min}}, 0.5]$. This is the primary method; it directly inverts Equation 29.
- **Method B (interior frac — identical to tempM):** $\hat{\lambda}_B = \min_{j:\text{frac}[j]\in(0.08,0.92)} \text{frac}[j]$, clamped to $[\lambda_{\text{min}}, 0.5]$.
- **Method C (top-2 variance — identical to tempM):** $\hat{\lambda}_C = \min(\text{frac}[\text{top}_1], \text{frac}[\text{top}_2])$, clamped to $[\lambda_{\text{min}}, 0.5]$.

The confidence score combines both the frac-variance signal and the within-variance signal, taking the maximum of both:

$$\gamma_{\text{var}} = \min\left(1,\ \max_j \text{var}[j]/0.25\right) \tag{30}$$

$$\gamma_{\text{wvar}} = \min\left(1,\ \max_j \texttt{within_var}[j]/0.25\right) \tag{31}$$

$$\gamma = \max(\gamma_{\text{var}},\ \gamma_{\text{wvar}}) \tag{32}$$

The blend weights give Method A (within-variance) priority when the within-variance signal is strong:

$$w_A = 0.45\,\gamma_{\text{wvar}} + 0.10(1-\gamma) \tag{33}$$

$$w_B = 0.30\,\gamma \tag{34}$$

$$w_C = 0.25\,\gamma \tag{35}$$

$$\hat{\lambda}_{\text{analytical}} = \frac{w_A\hat{\lambda}_A + w_B\hat{\lambda}_B + w_C\hat{\lambda}_C}{w_A + w_B + w_C}, \quad \text{clamped to } [\lambda_{\text{min}}, 0.5] \tag{36}$$

Compared to tempM, Method A here is strictly better — the within-variance gives a cleaner estimate of $\lambda(1-\lambda)$ than $\max_j \text{var}[j]$ because it directly measures the cell-level mixing variance rather than the frac-level Bernoulli variance, which conflates rule disagreement with random fluctuations from the finite sample. Even so, the analytical stocM baseline has a higher MAE than tempM's (0.0575 vs 0.0458), because stocM lacks the per-timestep residual; the correction MLP closes this gap (0.0289 vs 0.0311).

Stage 2: Correction MLP — 50-dimensional input. The correction MLP input is 50-dimensional, nine more than tempM:

$$\mathbf{f}_{\text{corr}} = \big[\underbrace{\texttt{frac_stats}}_{24},\ \underbrace{\texttt{within_var_stats}}_{8},\ \underbrace{\text{sorted_frac}}_{8},\ \underbrace{\text{adj_diff}}_{8},\ \underbrace{\hat{\lambda}_{\text{analytical}}}_{1},\ \underbrace{\gamma}_{1}\big] \quad \in \mathbb{R}^{50} \tag{37}$$

The 8 additional dimensions are `within_var_stats` itself, giving the correction MLP direct access to the per-pattern within-variance values. This allows the MLP to learn corrections that account for patterns where the within-variance signal is strong vs. weak — for instance, if a disagreement pattern is poorly observed (low cnt) then `within_var`$[j]$ is unreliable and the MLP can learn to discount it.

The MLP architecture is identical to tempM (50→256→256→128→1):

$$\begin{aligned}
\mathbf{m}_1 &= \text{Dropout}(\text{GELU}(\mathbf{W}_1\mathbf{f}_{\text{corr}} + \mathbf{b}_1)) \quad \mathbf{W}_1 \in \mathbb{R}^{50\times 256} \text{(13,056 parameters)} \\
\mathbf{m}_2 &= \text{Dropout}(\text{GELU}(\mathbf{W}_2\mathbf{m}_1 + \mathbf{b}_2)) \quad \mathbf{W}_2 \in \mathbb{R}^{256\times 256} \text{(65,792 parameters)} \\
\mathbf{m}_3 &= \text{GELU}(\mathbf{W}_3\mathbf{m}_2 + \mathbf{b}_3) \quad \mathbf{W}_3 \in \mathbb{R}^{256\times 128} \text{(32,896 parameters)} \\
\delta &= \text{Tanh}(\mathbf{W}_4\mathbf{m}_3 + \mathbf{b}_4) \quad \mathbf{W}_4 \in \mathbb{R}^{128\times 1} \text{(129 parameters)}
\end{aligned}$$

Adaptive correction range and final clamping identical to tempM:

$$\delta_{\text{eff}} = \delta \cdot (0.08 + (1-\gamma)\cdot 0.35) \tag{38}$$

$$\hat{\lambda}_{\text{pred}} = \text{clamp}(\hat{\lambda}_{\text{analytical}} + \delta_{\text{eff}},\ \lambda_{\text{min}},\ 0.5) \tag{39}$$

Final output layers initialised $\mathcal{N}(0, 0.01)$ with zero bias, identical to tempM. **Correction MLP total:** $13{,}056 + 65{,}792 + 32{,}896 + 129 = 111{,}873$ **parameters.**

Path B: Hierarchical Transformer Encoder

CellEncoder, TimestepEncoder, and ConfidenceWeightedOrbitFusion are identical to tempM in both architecture and parameter count.

- CellEncoder: Linear($12 \to 128$) + SinusoidalPE + 4-layer TransformerEncoder ($d = 128$, $H = 4$, FF=512) + query-attention pooling. **861,184 parameters**
- TimestepEncoder: Linear($128 \to 256$) + SinusoidalPE + 6-layer TransformerEncoder ($d = 256$, $H = 8$, FF=1024). **4,771,456 parameters**
- ConfidenceWeightedOrbitFusion: MLP($256 \to 64 \to 1$) + softmax over $k = 8$. **16,513 parameters**

The forward encoding path is also identical: raw tokens → PatternEmbedding → reshape to $[Bk(t-1), n, 12]$ → CellEncoder → TimestepEncoder → reshape to $[B, k, t-1, 256]$ → OrbitFusion → $\mathbf{h} \in \mathbb{R}^{B\times(t-1)\times 256}$.

Absence of the Per-Timestep Residual

stocM does *not* compute the per-timestep residual $r_{t'}$ that tempM uses. This is a fundamental consequence of the noise model: in stocECA, every timestep applies both rules simultaneously at the cell level, so the expected residual $\mathbb{E}[a_i^{t'+1} - \text{frac}[j_{t',i}]] = 0$ at every timestep regardless of which cells chose which rule. The per-timestep mean washes out the cell-level assignment, leaving no temporal assignment signal. Consequently, stocM has no assignment loss, no prob_{R_2} output, no `score_net`, and no `residual_weight` parameter. The rule head must identify both rules from the cross-timestep correlations in the transformer representation alone, which is harder — but partially compensated by the within-variance signal providing a second independent estimator of λ.

LambdaConditionedRuleHead

The rule head replaces tempM's per-timestep assignment with **λ-conditioned attention pooling** over all $t-1$ timestep representations.

Attention pooling.

$$\overline{v} = \frac{1}{8}\sum_{j=0}^{7} \text{var}[j] \in \mathbb{R}^{B\times 1} \quad \text{(mean frac variance, proxy for signal strength)}$$
$$\text{inp}_{t'} = [\mathbf{h}_{t'} \parallel \hat{\lambda}_{\text{pred}} \parallel \overline{v}] \in \mathbb{R}^{D_2+2=258}$$
$$w_{t'} = \text{softmax}_{t'}(\text{attn_net}(\text{inp}_{t'})) \in \mathbb{R}^{B\times(t-1)}$$
$$\mathbf{rep} = \textstyle\sum_{t'} w_{t'} \cdot \mathbf{h}_{t'} \quad \in \mathbb{R}^{B\times D_2}$$

The attn_net:

$$\mathbf{u}_1 = \text{Dropout}(\text{GELU}(\mathbf{W}_{a1}\,\text{inp}_{t'} + \mathbf{b}_{a1})) \quad \mathbf{W}_{a1} \in \mathbb{R}^{258\times 128} \text{(33,152 parameters)}$$
$$\mathbf{u}_2 = \text{GELU}(\mathbf{W}_{a2}\,\mathbf{u}_1 + \mathbf{b}_{a2}) \quad \mathbf{W}_{a2} \in \mathbb{R}^{128\times 64} \text{(8,256 parameters)}$$
$$s_{t'} = \mathbf{W}_{a3}\,\mathbf{u}_2 + b_{a3} \quad \mathbf{W}_{a3} \in \mathbb{R}^{64\times 1} \text{(65 parameters)}$$

The final layer of attn_net is initialised $\mathcal{N}(0, 0.1)$ with zero bias, so initial attention weights are approximately uniform (softmax of near-zero scores $\approx 1/(t-1)$ per timestep) and the model begins training with mean-pool behaviour before learning to attend selectively. **attn_net parameters:** $33{,}152 + 8{,}256 + 65 = 41{,}473$**.**

Why λ-conditioned attention instead of per-timestep assignment? In tempM, timestep t' can be assigned to either rule R_1 or R_2 because the entire row uses one rule. In stocM, there is no such assignment — every timestep contains a mixture. The attention weights $w_{t'}$ therefore do not represent rule assignment; instead, they learn to emphasise timesteps where the transformer representation is most informative about the rule pair. Conditioning on $\hat{\lambda}_{\text{pred}}$ allows the attention weights to adapt to the noise level: at low λ, rule R_2 rarely fires and its signal is concentrated in a few timesteps where the within-row variance happens to be high; conditioning on $\hat{\lambda}_{\text{pred}}$ lets the attention network learn this relationship.

Rule heads. Unlike tempM, stocM does not split into separate $\mathbf{rep}_{R_1}$ and $\mathbf{rep}_{R_2}$ representations — there is no temporal assignment to guide this split. Both rule heads read from the *same* $\mathbf{rep}$, augmented with direct decoder and distributional decoder logits:

$$\mathbf{in}_{R_1} = [\mathbf{rep} \parallel \mathbf{d}_{R_1}^{\text{detach}} \parallel (\mathbf{d}_{R_1}^{\text{dist}})^{\text{detach}}] \quad \in \mathbb{R}^{D_2+8+8=272}$$
$$\mathbf{in}_{R_2} = [\mathbf{rep} \parallel \mathbf{d}_{R_2}^{\text{detach}} \parallel (\mathbf{d}_{R_2}^{\text{dist}})^{\text{detach}}] \quad \in \mathbb{R}^{272}$$

The rule head MLPs (head_{R_1} and head_{R_2}, same architecture, separate weights) include a LayerNorm at the input:

$$\mathbf{v}_{\text{ln}} = \text{LayerNorm}(272)(\mathbf{in}_{R_1/R_2}) \text{(544 parameters)}$$
$$\mathbf{v}_1 = \text{Dropout}(\text{GELU}(\mathbf{W}_{v1}\,\mathbf{v}_{\text{ln}} + \mathbf{b}_{v1})) \quad \mathbf{W}_{v1} \in \mathbb{R}^{272\times 256} \text{(69,888 parameters)}$$
$$\mathbf{v}_2 = \text{GELU}(\mathbf{W}_{v2}\,\mathbf{v}_1 + \mathbf{b}_{v2}) \quad \mathbf{W}_{v2} \in \mathbb{R}^{256\times 128} \text{(32,896 parameters)}$$
$$\mathbf{v}_{\text{out}} = \mathbf{W}_{v3}\,\mathbf{v}_2 + \mathbf{b}_{v3} \quad \mathbf{W}_{v3} \in \mathbb{R}^{128\times 8} \text{(1,032 parameters)}$$

Each rule head: $544 + 69{,}888 + 32{,}896 + 1{,}032 = 104{,}360$ **parameters; both together: 208,720.**

DirectRuleDecoder for stocM. The per-pattern MLP features are 8-dimensional rather than 7-dimensional (as in tempM), adding $\texttt{within_var}[j]$ as an 8th feature:

$$\mathbf{q}_j = [\text{frac}[j],\ \hat{\lambda},\ 1-\hat{\lambda},\ |\text{frac}[j]-\hat{\lambda}|,\ |\text{frac}[j]-(1-\hat{\lambda})|,\ \text{var}[j],\ \texttt{within_var}[j],\ \text{cnt}[j]] \quad \in \mathbb{R}^8 \tag{40}$$

The within-variance term $\texttt{within_var}[j]$ adds direct information about whether pattern j is a disagreement pattern: if $\texttt{within_var}[j] > 0$, then $R_1[j] \neq R_2[j]$, and the magnitude gives an estimate of $\lambda(1-\lambda)$. This makes the per-pattern rule estimation more reliable for stocM than for tempM. The MLP is Linear($8 \to 64$) + GELU + Linear($64 \to 64$) + GELU + Linear($64 \to 2$). **DirectRuleDecoder parameters:** $\approx 4{,}866$**.**

DistributionalRuleDecoder for stocM. Identical architecture to tempM's DistributionalRuleDecoder, with $\hat{\lambda}$ replacing $\hat{\tau}$ in the per-pattern feature vector: $[\text{frac}[j],\ \hat{\lambda},\ \gamma]$. **DistributionalRuleDecoder parameters:** ≈ 162.

FORWARD PASS SUMMARY

Inputs: $\mathbf{X}^{(\ell)} \in \{0,1\}^{(t-1)n\times 4}$ $(\ell = 1 \dots k)$, $\texttt{frac_stats} \in \mathbb{R}^{24}$, $\texttt{within_var_stats} \in \mathbb{R}^{8}$

Path A: $[\texttt{frac_stats}, \texttt{within_var_stats}] \xrightarrow{\text{analytical (0 params)}} \hat{\lambda}_{\text{base}} \in [0.1, 0.5]$
$\xrightarrow{\text{correction_net(50-dim)}} \hat{\lambda}_{\text{pred}} \in [0.1, 0.5]$

Path B: $\mathbf{X}^{(\ell)} \xrightarrow{\text{+PatEmb}} [(t-1)n, 12] \xrightarrow{\text{CellEnc}} [t-1, 128]$
$\xrightarrow{\text{TSEnc}} [t-1, 256] \xrightarrow{\text{OrbitFuse}} \mathbf{h} \in [B, t-1, 256]$

Pooling: $\mathbf{h} + [\hat{\lambda}_{\text{pred}}, \overline{v}] \xrightarrow{\text{attn_net (no residual)}} \mathbf{rep} \in [B, 256]$

Rules: $[\mathbf{rep}, \mathbf{d}_{R_1}, \mathbf{d}_{R_2}, \mathbf{d}^{\text{dist}}_{R_1}, \mathbf{d}^{\text{dist}}_{R_2}] \xrightarrow{\text{LN + head}_{R_1}/\text{head}_{R_2}} \mathbf{logits}_{R_1}, \mathbf{logits}_{R_2} \in [B, 8]$

KEY DIFFERENCES FROM TEMPM: SUMMARY TABLE

Table 12: Architectural differences between tempM and stocM, motivated by the physics of the two noise models.

Component	tempM	stocM
Additional input	none	$\texttt{within_var_stats}$ $[B, 8]$
Correction MLP input	41-dim	50-dim (+8 within_var)
Method A estimator	$\max_j \text{var}[j]$	$\max_j \texttt{within_var}[j]$
Confidence signal	frac variance only	max(frac var, within var)
Per-timestep residual	yes ($r_{t'}$, 0 params)	absent (structurally zero)
Assignment mechanism	sigmoid(score) per t'	attention softmax over t'
Rule representations	split ($\mathbf{rep}_{R_1}, \mathbf{rep}_{R_2}$)	shared ($\mathbf{rep}$)
Rule head input dim	272 ($D_2 + 16$)	272 ($D_2 + 16$)
Rule head LayerNorm	yes (input to both heads)	yes (input to both heads)
Direct decoder dim	7-dim per pattern	8-dim per pattern (+within_var)
Assignment loss	yes (prob_{R_2} supervised)	absent

B.5 CONFIDENCEWEIGHTEDORBITFUSION: COMPLETE SPECIFICATION

`ConfidenceWeightedOrbitFusion` is the module that combines the $k = 8$ independently encoded orbit representations into a single fused representation, used identically in both tempM and stocM. It sits between the TimestepEncoder and the rule head, receiving $\mathbf{H}_{\text{all}} \in \mathbb{R}^{B\times k\times(t-1)\times D_2}$ and producing $\mathbf{h} \in \mathbb{R}^{B\times(t-1)\times D_2}$.

MOTIVATION

Given $k = 8$ orbits from distinct random initial configurations of the same (rule pair, noise parameter) sample, a naive approach would average them:

$$\mathbf{h}_{\text{naive}} = \frac{1}{k}\sum_{\ell=1}^{k} \mathbf{H}_\ell \tag{41}$$

This is suboptimal because the orbits are not equally informative. The rule-discriminative signal depends on how many times each of the 8 neighbourhood patterns appears in the orbit and how distinctly the two rules differ on those patterns. An orbit whose random initial configuration happens

to generate a high proportion of disagreement-pattern observations carries substantially more information about the rule pair than one dominated by agreement patterns. Averaging all orbits equally dilutes this signal with low-information orbits. `ConfidenceWeightedOrbitFusion` learns to upweight informative orbits and downweight uninformative ones, using the orbit's own representation to assess its informativeness.

ARCHITECTURE

Step 1: Orbit-level summary. Each orbit's $t-1=199$ timestep representations are averaged to produce a single $D_2=256$-dimensional summary vector:

$$\mathbf{s}_\ell = \frac{1}{t-1}\sum_{t'=1}^{t-1}\mathbf{H}_{\ell,t'} \quad \in \mathbb{R}^{B\times D_2} \quad \forall\,\ell\in\{1,\dots,k\} \qquad \text{(0 learnable parameters)} \tag{42}$$

Mean pooling over timesteps is used here rather than max pooling or learned pooling because $\mathbf{s}_\ell$ is only needed to score the orbit as a whole, not to extract specific temporal features. The mean activations of a high-signal orbit (many disagreement-pattern observations) will differ systematically from a low-signal orbit (many agreement-pattern observations), providing a sufficient basis for the weight network to discriminate.

Step 2: Weight network. A two-layer MLP maps each orbit summary to a scalar score:

$$\begin{aligned}\mathbf{u}_\ell &= \text{GELU}(\mathbf{s}_\ell\mathbf{W}_1+\mathbf{b}_1) \quad \mathbf{W}_1\in\mathbb{R}^{256\times 64}\text{(16,448 parameters)}\\ z_\ell &= \mathbf{u}_\ell\mathbf{W}_2+\mathbf{b}_2 \quad \mathbf{W}_2\in\mathbb{R}^{64\times 1}\text{(65 parameters)}\\ &\in\mathbb{R}^{B\times 1}\end{aligned}$$

Weight network total: 16,513 parameters. Note: the weight network is explicitly excluded from the global $\mathcal{N}(0,0.02)$ initialisation — its weights are left at PyTorch's default Kaiming uniform initialisation. This allows the network's initial scores to have a non-trivial spread across orbits, providing useful gradient signal from the first training step rather than defaulting to uniform weights.

Step 3: Softmax normalisation. The $k=8$ scalar scores are normalised to a probability distribution over orbits:

$$w_\ell = \frac{\exp(z_\ell)}{\sum_{m=1}^{k}\exp(z_m)} \quad \in(0,1), \quad \sum_{\ell=1}^{k} w_\ell = 1 \qquad \text{(applied along the } k \text{ dimension)} \tag{43}$$

The softmax is applied along `dim=1` (the k dimension), not across timesteps or features. The result $\mathbf{w}\in\mathbb{R}^{B\times k}$ gives a per-orbit weight for each sample in the batch, with each sample's weights summing to 1.

Step 4: Weighted sum. The fused representation is the weighted sum of the k orbit representations across the orbit dimension:

$$\mathbf{h} = \sum_{\ell=1}^{k} w_\ell \cdot \mathbf{H}_\ell \quad \in\mathbb{R}^{B\times(t-1)\times D_2} \tag{44}$$

where the weight $w_\ell \in \mathbb{R}^B$ is broadcast over the $(t-1)$ and D_2 dimensions via `weights.unsqueeze(-1).unsqueeze(-1)`. The fusion is performed at the level of full orbit representations $\mathbf{H}_\ell\in\mathbb{R}^{(t-1)\times D_2}$, meaning the weight assigned to orbit ℓ applies uniformly to all $t-1$ of its timestep representations. There is no timestep-varying weighting within an orbit — only orbit-level weighting.

COMPLETE FORWARD PASS

The complete forward computation in code form:

$$\underbrace{\mathbf{H}_{\text{all}}}_{[B,k,t-1,256]} \xrightarrow{\text{.mean(dim=2)}} \underbrace{\mathbf{S}}_{[B,k,256]} \xrightarrow{\text{weight_net}} \underbrace{\mathbf{z}}_{[B,k,1]} \xrightarrow{\text{softmax(dim=1)}} \underbrace{\mathbf{w}}_{[B,k,1]} \xrightarrow{\times\,\mathbf{H}_{\text{all}},\ \text{.sum(dim=1)}} \underbrace{\mathbf{h}}_{[B,t-1,256]}$$

BEHAVIOUR AT $k = 8$

At initialisation. With Kaiming uniform initialisation on the weight network and a relatively shallow two-layer MLP, the initial scores z_ℓ are small random values. After softmax, all k weights are close to $1/k = 0.125$, and the fused representation is approximately a uniform average of all 8 orbits. The model therefore begins training with the informational benefit of k orbits (averaging reduces variance in the representation) before learning to discriminate between them.

During training. As the encoder learns to produce representations that reflect orbit quality, the weight network learns to assign higher scores to orbits with stronger rule-pair signal. The gradient of the rule loss with respect to w_ℓ is:

$$\frac{\partial \mathcal{L}_{\text{rule}}}{\partial w_\ell} = \sum_{t'} \mathbf{h}_{t'}^\top \frac{\partial \mathcal{L}_{\text{rule}}}{\partial \mathbf{h}_{t'}} \tag{45}$$

If orbit ℓ produces rule representations that reduce the loss more than average, its gradient with respect to w_ℓ will be negative (reduce loss by increasing w_ℓ), and the weight network will learn to assign it a higher score. This backpropagation path is unobstructed — there are no detached gradients or stop-gradient operations in the fusion module.

At inference (qualitative behaviour). After training, the weight distribution across orbits is interpretable: orbits where the random initial configuration generated many tokens with neighbourhood patterns on which R_1 and R_2 disagree will receive higher weights, because those orbits' representations contain more rule-discriminative information. Conversely, an orbit that happened to start in a state where the dynamics converge quickly to an all-zeros or all-ones fixed point — providing very few informative tokens — will receive a lower weight. The softmax ensures this downweighting is relative: even a low-information orbit always receives some weight (> 0), preventing any orbit from being completely ignored.

Comparison with simple averaging. Simple mean-pooling across orbits ($w_\ell = 1/k$ always) would treat all orbits equally regardless of their informational content. The learned weighting provides a form of *adaptive sampling* — the effective number of informative orbits the model uses is variable and content-dependent, even though the physical number of orbits generated per sample is fixed at $k = 8$. In the degenerate case of rule pairs where all 8 neighbourhood patterns agree ($R_1[j] = R_2[j]$ for all j), all orbits are equally uninformative and the weights will be approximately uniform — correctly defaulting to mean pooling when no orbit is distinguishably better than another.

C TRAINING DETAILS

This appendix provides the complete training specifications for all four models, supplementing the high-level description in Section 6. All loss weights, learning rate schedules, epoch counts, and hardware details are stated exactly as used in the final training runs whose results appear in Section 7.

C.1 LOSS WEIGHT TABLE

The total loss at each training step is a weighted sum of individual loss components. The weights differ between Stage 1 and Stage 2, and between models, reflecting the curriculum structure described in Section 6. Table 13 gives every weight for every model and stage, read directly from the training scripts.

αM and skewM. In Stage 1, the noise estimation heads receive zero gradient ($\lambda_\alpha = 0.0$, $\lambda_{s,\text{cls}} = \lambda_{s,\text{reg}} = 0.0$). The encoder trains exclusively on rule identification, building stable rule representations before noise estimation is introduced. In Stage 2, the noise weight is activated ($\lambda_\alpha = 0.3$ for αM, $\lambda_{s,\text{cls}} = 0.3$ and $\lambda_{s,\text{reg}} = 0.1$ for skewM). The rule weight remains 1.0 throughout both stages — the rule head never loses priority. The lower value of $\lambda_{s,\text{reg}} = 0.1$ relative to $\lambda_{s,\text{cls}} = 0.3$ reflects the auxiliary role of the regression head: it acts as an ordinal inductive bias but the classification head provides the primary prediction.

Table 13: Loss weights for all four models across both training stages. A dash (—) indicates the loss component does not exist for that model. A value of 0.0 means the component exists but receives zero gradient in that stage. All weights are multiplied by their respective loss before summing; the combined loss is then divided by the gradient accumulation steps before `.backward()` is called.

	αM		skewM		tempM		stocM	
Loss component	S1	S2	S1	S2	S1	S2	S1	S2
Rule identification								
Rule BCE (λ_{rule})	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Noise estimation — αM								
α CE (λ_{α})	0.0	0.3	—	—	—	—	—	—
Noise estimation — skewM								
s CE ($\lambda_{s,\text{cls}}$)	—	—	0.0	0.3	—	—	—	—
s MSE on s/W ($\lambda_{s,\text{reg}}$)	—	—	0.0	0.1	—	—	—	—
Noise estimation — tempM and stocM								
τ in symmetric loss ($\lambda_{\tau}^{\text{sym}}$)	—	—	—	—	0.0	0.0	—	—
τ MSE ($\lambda_{\tau}^{\text{mse}}$)	—	—	—	—	3.0	2.0	—	—
λ MSE ($\lambda_{\lambda}^{\text{mse}}$)	—	—	—	—	—	—	3.0	2.0
Temporal assignment — tempM only								
Assignment loss (λ_{assign})	—	—	—	—	3.0	1.5	—	—
Auxiliary decoders — tempM and stocM								
Direct decoder (λ_{direct})	—	—	—	—	0.5	0.3	0.5	0.3

tempM. The τ noise parameter is not included in the symmetric rule loss ($\lambda_{\tau}^{\text{sym}} = 0.0$ in both stages). This is a deliberate design decision: including τ in the symmetric loss caused orientation instability in earlier experiments, because the symmetric loss would pick orientation based on whichever of $(\hat{r}_{R1}, \hat{r}_{R2}, \hat{\tau})$ or $(\hat{r}_{R2}, \hat{r}_{R1}, 1-\hat{\tau})$ gave lower combined loss, creating competition between the rule signal and the noise signal for orientation selection. Separating the two — symmetric loss determines orientation from rule bits only, τ-MSE loss supervises the noise magnitude directly — eliminates this competition. The assignment weight decreases from 3.0 (Stage 1) to 1.5 (Stage 2) because assignment converges rapidly in Stage 1, driven by the strong per-timestep residual signal ($\rho = 5.0$ initialisation); maintaining the higher weight in Stage 2 would over-constrain the rule head during joint fine-tuning. The τ-MSE weight decreases from 3.0 to 2.0 to prevent it from dominating the rule loss after the correction MLP has converged.

stocM. The assignment loss is entirely absent (no λ_{assign} row for stocM) because stocECA has no per-timestep rule labels — every timestep mixes both rules at the cell level, making ground-truth per-timestep assignment undefined. All other loss weights mirror tempM's structure and rationale, including the λ-MSE decrease from 3.0 to 2.0 in Stage 2 and the direct decoder decrease from 0.5 to 0.3.

C.2 Full Training Configuration

Table 14 gives the complete training configuration for all four models. All four models use AdamW with weight decay 10^{-2} and cosine annealing of the learning rate within each stage, with gradient clipping at maximum norm 1.0 applied before every optimiser step.

Why smaller physical batches for tempM and stocM. αM and skewM use physical batch size 8 with 16 accumulation steps. tempM and stocM use physical batch size 4 with 32 accumulation steps. In both cases the effective batch size is 128. The smaller physical batch for tempM and stocM is required by GPU memory: each sample consists of $k = 8$ orbits of shape $[200 \times 50]$, giving a per-sample memory footprint of $8 \times 200 \times 50 \times 12 \times 4 \approx 3.84$ MB of float32 tokens, compared to $1 \times 100 \times 20 \times 4 \times 4 \approx 0.03$ MB for αM. The H100 NVL's 94 GB VRAM accommodates the larger models; the RTX A1000's 16 GB is sufficient for αM and skewM.

Table 14: Complete training configuration for all four models. Stage 1 / Stage 2 values are separated by a forward slash where they differ. † tempM and stocM use a linear warmup before cosine annealing begins, applied at the step level (not epoch level). ‡ tempM and stocM use a separate, lower learning rate for the correction MLP (tau_mlp / lam_mlp) in Stage 2. § Training times are wall-clock times from the actual training runs on the stated hardware; αM and skewM training times are estimated from epoch logs.

	αM	skewM	tempM	stocM
Stage structure				
Stage 1 epochs	50	50	80	80
Stage 2 epochs	30	30	80	80
Total epochs	80	80	160	160
Learning rates				
Stage 1 LR (peak)	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Stage 1 LR (min)	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
Stage 2 LR main (peak)	1×10^{-4}	1×10^{-4}	3×10^{-5}	3×10^{-5}
Stage 2 LR main (min)	1×10^{-6}	1×10^{-6}	1×10^{-7}	1×10^{-7}
Stage 2 LR noise MLP‡	n/a	n/a	1×10^{-5}	1×10^{-5}
LR schedule	Cosine	Cosine	Warmup+Cosine	Warmup+Cosine
Warmup† (tempM and stocM only)				
Stage 1 warmup steps	—	—	100	100
Stage 2 warmup steps	—	—	30	30
Warmup LR start	—	—	1×10^{-6}	1×10^{-6}
Batch configuration				
Physical batch size (Stage 1)	8	8	4	4
Physical batch size (Stage 2)	8	8	4	4
Gradient accumulation steps	16	16	32	32
Effective batch size	128	128	128	128
Optimiser				
Algorithm	AdamW	AdamW	AdamW	AdamW
Weight decay	10^{-2}	10^{-2}	10^{-2}	10^{-2}
β_1, β_2	0.9, 0.999	0.9, 0.999	0.9, 0.999	0.9, 0.999
ϵ	10^{-8}	10^{-8}	10^{-8}	10^{-8}
Gradient clip norm	1.0	1.0	1.0	1.0
Validation and checkpointing				
Val split	15%	15%	test set	test set
Val metric	rule exact match	rule exact match	UO both-exact	UO both-exact
Checkpoint strategy	best val metric	best val metric	best val metric	best val metric
Hardware and timing§				
Hardware	RTX A1000	RTX A1000	H100 NVL	H100 NVL
VRAM	16 GB	16 GB	94 GB	94 GB
`num_workers`	0	0	0	0
`pin_memory`	True	True	True	True
Stage 1 training time	≈4 h	≈5 h	≈13.3 h	≈15.7 h
Stage 2 training time	≈3 h	≈4 h	≈13.3 h	≈15.6 h
Total training time	≈7 h	≈9 h	26.6 h	31.3 h

Why no warmup for αM and skewM. The warmup is applied to tempM and stocM because their larger model size (6.02M vs ≈1.77M parameters for skewM and ≈1.76M for αM) and on-the-fly data generation create a noisier initial loss landscape. The LR ramps from 10^{-6} to the peak over 100 steps before cosine annealing takes over, preventing the large initial gradients from the randomly initialised correction MLP from destabilising the encoder in the first few updates. αM and skewM train on pre-generated data with a smaller model and experience a stable loss from epoch 1; warmup adds no benefit.

Why a separate LR for the correction MLP in Stage 2. The correction MLP enters Stage 2 already well-converged from Stage 1, having learned to make accurate corrections to the analytical

estimate. A main LR of 3×10^{-5} applied to the correction MLP would risk overriding its Stage 1 representations with unstable updates driven by the joint rule+noise objective in Stage 2. The lower rate of 10^{-5} keeps the correction MLP refinements small and targeted, while allowing the transformer backbone to continue adapting at the full Stage 2 rate.

Validation strategy difference. αM and skewM hold out 15% of the Phase 2 training set as a fixed validation split (seed = 42) and use rule exact match as the early stopping metric. tempM and stocM evaluate on the fixed 10,000-sample pre-generated test set after each epoch, using UO both-exact accuracy as the checkpoint metric. The test set is used for checkpointing (not held out) because there is no fixed training set to split — training data is regenerated each epoch from scratch.

C.3 Two-Phase Curriculum: Detailed Epoch Breakdown

αM and skewM

Both models follow a two-phase curriculum with a hard boundary between phases. Phase 1 trains on noise-free orbits ($\alpha = 1.0$ or $s = n$) with the noise head receiving zero gradient. Phase 2 loads the best Phase 1 checkpoint and trains on the full noise range with both heads active. There is no noise-range curriculum within either phase — the noise level is either entirely absent (Phase 1) or fully present across all levels (Phase 2). This is possible because for single-rule identification tasks, the rule fingerprint remains identifiable at all noise levels, so there is no need to restrict the noise range during Phase 2.

Table 15: Epoch-by-epoch curriculum for αM and skewM. Both models follow the same phase structure.

Phase	Epochs	Noise range	Noise head active
Phase 1	1–50	None ($\alpha = 1.0$ / $s = n$)	No ($\lambda_{\text{noise}} = 0$)
Phase 2	1–30	Full ($\alpha \in \{0.1, \ldots, 1.0\}$ / $s \in \{1, \ldots, 20\}$)	Yes

The Phase 1 checkpoint that achieves the best validation rule exact match is saved to `phase1_best.pt`. Phase 2 begins from this checkpoint, not from the last Phase 1 epoch. If Phase 1 overfits in the final epochs (which can happen as the model learns to memorise specific orbit patterns rather than rule fingerprints), loading the best checkpoint rather than the last ensures Phase 2 starts from the most generalisable rule representations.

tempM and stocM

tempM and stocM use a more complex curriculum because both the rule pair *and* the noise parameter must be learned simultaneously from the first epoch — unlike αM and skewM, there is no noise-free Phase 1 for these models (two-rule identification from a synchronous orbit is trivially ill-posed: a synchronous orbit under rule pair $(R_1, R_2, \tau = 1.0)$ is identical to a synchronous orbit under rule g alone, making rule f unobservable). Instead, Phase 1 begins with a restricted noise range $[0.3, 0.7]$ and expands to the full range $[0.1, 0.9]$ at epoch 11.

Rationale for the epoch-11 expansion. The noise-range curriculum exists because at extreme θ values (near 0.1 or 0.9), one of the two rules fires very rarely — at $\tau = 0.1$, rule R_2 fires in approximately $0.1 \times 199 \approx 20$ timesteps per orbit. The model has ≈ 20 timestep representations that encode rule R_2's fingerprint and 179 that encode rule R_1's. If exposed to $\tau = 0.1$ in the first epoch before rule representations are established, the model cannot learn rule R_2's fingerprint from 20 noisy examples, and learns instead to predict a fixed default for rule R_2. Restricting to $[0.3, 0.7]$ for the first 10 epochs ensures both rules appear in at least $0.3 \times 199 \approx 60$ timesteps per orbit — enough to build stable fingerprints for both. By epoch 11, the representations are robust enough to handle the rare-rule signal at extreme noise levels. The expansion is a single step (epoch 10 $\rightarrow$ epoch 11: range jumps from $[0.30, 0.70]$ to $[0.10, 0.90]$) rather than a gradual ramp, because the analytical estimator (AnalyticalTauMLP / AnalyticalLambdaMLP) handles the full range from epoch 0 — the expansion affects the transformer's training distribution, not the analytical path.

Table 16: Noise-range curriculum for tempM and stocM. The noise parameter $\theta \in \{\tau, \lambda\}$ is sampled uniformly from the range $[\theta_{\text{min}}, \theta_{\text{max}}]$ at each training step. `lam_half` is the half-width of the sampling range around 0.5. Both models use identical curriculum schedules.

Stage	Epoch range	`lam_half`	Sampling range	Rationale
Stage 1	1–10	0.20	$[0.30, 0.70]$	Both rules appear frequently; fingerprints learnable
Stage 1	11	0.40	$[0.10, 0.90]$	Expand to full range in one step
Stage 1	12–80	0.40	$[0.10, 0.90]$	Maintain full range for remainder of Stage 1
Stage 2	1–80	0.40	$[0.10, 0.90]$	Always full range in Stage 2

Seed schedule across epochs. Each epoch generates a fresh `OnTheFlyDataset` with a deterministic seed:

$$\text{Stage 1 seed}(e) = 10{,}000 + 7 \times e$$
$$\text{Stage 2 seed}(e) = 20{,}000 + 13 \times e$$

The different multipliers (7 vs 13) and base offsets (10,000 vs 20,000) ensure that no Stage 2 epoch reproduces a Stage 1 orbit, and no two epochs in either stage produce the same dataset. The model therefore encounters strictly different orbits at every epoch across the full 160-epoch run.

Stage 2 loads the best Stage 1 checkpoint. After Stage 1, the best checkpoint (by UO both-exact on the fixed test set) is saved to `stage1.pt`. Stage 2 loads `stage1.pt` and fine-tunes all parameters jointly with a lower main learning rate (3×10^{-5} vs 3×10^{-4} in Stage 1) and a separate, even lower rate for the correction MLP (10^{-5}). This separation ensures the correction MLP's well-converged Stage 1 parameters are not disrupted by the larger main LR during Stage 2 joint fine-tuning.

C.4 Validation Split and Early Stopping

αM and skewM

For αM and skewM, 15% of each phase's training data is held out as a fixed validation split before training begins:

```
n_val = int(len(full_train_dataset) * 0.15)   # VAL_SPLIT = 0.15
p_val, p_tr = random_split(
    full_train_dataset,
    [n_val, len(full_train_dataset) - n_val],
    generator=torch.Generator().manual_seed(42))
```

The generator seed (42) is fixed, so the validation split is identical across all runs and is reproducible from the dataset alone. The split is applied independently to Phase 1 and Phase 2 training data — Phase 1's 15% is held out from the 35,800-sample Phase 1 training set, and Phase 2's 15% is held out from the 89,500-sample Phase 2 training set.

Exact validation set sizes:

$$\alpha\text{M / skewM Phase 1 val:} \quad \lfloor 35{,}800 \times 0.15 \rfloor = 5{,}370 \text{ samples}$$
$$\alpha\text{M / skewM Phase 2 val:} \quad \lfloor 89{,}500 \times 0.15 \rfloor = 13{,}425 \text{ samples}$$

Early stopping metric: validation rule exact match (fraction of samples where all 8 bits are predicted correctly), evaluated after every epoch. The checkpoint with the highest validation rule exact match is saved; if no improvement is seen for the remainder of the phase, the best checkpoint is used as the starting point for the next phase. There is no explicit patience parameter — the model trains for all epochs in each phase and the best checkpoint is selected at the end. This is equivalent to patience $= \infty$ (no early termination), which is appropriate because both models converge reliably within their epoch budgets and overfitting is not observed in practice given the effective batch size of 128 and the model sizes (<2M parameters).

Why rule exact match, not α/s accuracy? In Phase 1, the noise head is inactive so rule exact match is the only meaningful metric. In Phase 2, rule exact match is used rather than combined (rule + noise) accuracy because the hybrid estimator replaces the neural noise head at inference — the neural noise head's accuracy during training is not reflective of the final hybrid accuracy. Using rule exact match also prevents the checkpoint selection from being dominated by noise estimation performance, which could bias it toward orbits where the noise level is easy to classify at the expense of harder noise levels.

TEMPM AND STOCM

tempM and stocM have no fixed training set — data is regenerated each epoch from scratch. A held-out validation split from the training data is therefore not possible (there is nothing stable to hold out from). Instead, the fixed pre-generated test set of 10,000 samples serves the role of a validation set for checkpoint selection:

```
if both > best_both:
    best_both = both
    torch.save({"model_state": model.state_dict(),
                "stage": stage, "epoch": ep,
                "best_both": best_both}, save_path)
```

Checkpoint metric: UO both-exact accuracy — the fraction of test samples where both rules in the pair are correctly predicted (unordered) and evaluated after every epoch on the full 10,000-sample test set.

This use of the test set for checkpoint selection does not constitute data leakage in the traditional sense, because the test set contains only held-out rules (77 rules, strictly disjoint from the 179 training rules) and the model cannot overfit to specific test orbits — each orbit is generated from a random initial configuration that the model has not seen. The noise parameter values in the test set are also unseen at training time (continuous uniform distribution vs. the on-the-fly curriculum distribution). The reported results are from the best checkpoint over the full training run, which is the standard evaluation protocol for models trained on on-the-fly data.

There is no patience parameter for tempM or stocM. Both stages run to completion (80 epochs each) and the best checkpoint across all 80 epochs is used. This is necessary because the curriculum expansion at epoch 11 causes a temporary accuracy drop followed by recovery — an early stopping criterion with finite patience would terminate training before the model recovers.

C.5 OPTIMISER FULL SPECIFICATION

COMMON CONFIGURATION

All four models use AdamW (Loshchilov & Hutter, 2019) with the following fixed hyperparameters shared across all models, phases, and parameter groups:

$$\begin{aligned} \beta_1 &= 0.9 \quad \text{(first moment decay)} \\ \beta_2 &= 0.999 \quad \text{(second moment decay)} \\ \epsilon &= 10^{-8} \quad \text{(numerical stability)} \\ \lambda_{\text{wd}} &= 10^{-2} \quad \text{(weight decay)} \\ \text{grad clip} &= 1.0 \quad \text{(max norm, applied before every step)} \end{aligned}$$

These are PyTorch's default AdamW values for β_1, β_2, and ϵ. The weight decay of 10^{-2} is a mild regulariser appropriate for the model sizes and dataset scales used here. Note that AdamW applies weight decay as a direct parameter shrinkage term $\theta \leftarrow \theta - \alpha\lambda\theta$ *before* the gradient step, separate from the gradient update — this is the key distinction from L2-regularised Adam and is particularly important for transformer models where embedding parameters benefit from decoupled regularisation.

LEARNING RATE SCHEDULE

αM and skewM. Both phases use a single parameter group with cosine annealing:

$$\text{lr}(e) = \text{lr}_{\text{min}} + \frac{1}{2}(\text{lr}_{\text{peak}} - \text{lr}_{\text{min}}) \left(1 + \cos\left(\frac{\pi e}{E}\right)\right) \tag{46}$$

where e is the current epoch, E is the total epochs in the phase, lr_{peak} is the initial LR, and lr_{min} is the floor. The scheduler is fast-forwarded when resuming from a checkpoint: `scheduler.step()` is called `resume_epoch` times before training resumes, ensuring the LR schedule is identical whether training ran continuously or was interrupted and resumed.

tempM and stocM. Both stages use the custom `WarmupCosineScheduler` with step-level updates during warmup and epoch-level cosine annealing thereafter:

$$\text{lr}(\text{step}) = \begin{cases} \text{lr}_{\text{start}} + (\text{lr}_{\text{peak}} - \text{lr}_{\text{start}}) \cdot \dfrac{\text{step}}{W} & \text{if step} \leq W \\ \text{cosine_annealing}(\text{epoch}) & \text{if step} > W \end{cases} \tag{47}$$

where W is the warmup step count (100 in Stage 1, 30 in Stage 2) and $\text{lr}_{\text{start}} = 10^{-6}$ in both stages. The warmup transitions smoothly from 10^{-6} to lr_{peak} over the first W gradient steps, then hands off to the cosine schedule which runs for the remainder of the phase at the epoch level. Importantly, the warmup is step-level (one step per gradient accumulation update, not per epoch), so the 100-step warmup covers approximately $100/(8000/128) \approx 1.6$ epochs at the start of Stage 1 — a short period of controlled ramp-up.

TWO-GROUP PARAMETER SPLIT FOR TEMPM AND STOCM (STAGE 2 ONLY)

In Stage 2, tempM and stocM split parameters into two optimiser groups with different learning rates:

```
tau_params  = list(model.tau_mlp.parameters())   # correction MLP only
tau_ids     = {id(p) for p in tau_params}
main_params = [p for p in model.parameters()
               if id(p) not in tau_ids]

opt = torch.optim.AdamW([
    {"params": tau_params,  "lr": S2_LR_TAU,
     "_lr_ratio": S2_LR_TAU / S2_LR_MAIN},   # 1e-5 / 3e-5 = 0.333
    {"params": main_params, "lr": S2_LR_MAIN,
     "_lr_ratio": 1.0},
], weight_decay=1e-2)
```

The `_lr_ratio` field is a custom field used by `WarmupCosineScheduler` to maintain the relative LR ratio between groups throughout the warmup and cosine phases:

$$\text{lr}_{\text{group}}(\text{step}) = \text{lr}(\text{step}) \times \texttt{_lr_ratio} \tag{48}$$

This ensures the correction MLP always trains at $1/3$ the main LR throughout Stage 2, regardless of where the cosine schedule is. The weight decay $\lambda_{\text{wd}} = 10^{-2}$ applies to both groups equally.

The split is applied identically in stocM with `model.lam_mlp` replacing `model.tau_mlp`. Stage 1 for both models uses a single parameter group (all parameters at the same LR) because the correction MLP is not yet converged and benefits from the full learning rate during its initial training.

C.6 DATA LOADING AND AUGMENTATION

αM AND SKEWM

Training data is pre-generated and loaded from disk using `numpy.load`. Each `ECADataset` loads all three arrays into memory on initialisation:

```
self.orbits    = np.load(os.path.join(path, "orbits.npy"))      # int8
self.rule_bits = np.load(os.path.join(path, "rule_bits.npy"))   # int8
self.alphas    = np.load(os.path.join(path, "alphas.npy"))      # float32
```

There is no data augmentation for αM and skewM beyond the inherent randomness in the initial configurations used during dataset generation. The `DataLoader` uses `shuffle=True` for the training split and `num_workers=0` (single-process loading) to avoid complications with the CUDA context on Windows and to keep memory usage predictable. Tokenisation is performed on-the-fly in `__getitem__` by calling `orbit_to_tokens`, which is a pure NumPy operation and adds negligible overhead.

TEMPM AND STOCM: ON-THE-FLY GENERATION

Training data for tempM and stocM is generated entirely on-the-fly at each epoch. The `OnTheFlyDataset` constructor runs synchronously at the start of each epoch and returns a fully materialised dataset object that the DataLoader then iterates over.

Random state isolation. Each epoch's dataset is constructed with an isolated `numpy.random.default_rng` instance seeded deterministically:

$$\text{Stage 1, epoch } e: \quad \texttt{rng = default_rng}(10000 + 7e)$$
$$\text{Stage 2, epoch } e: \quad \texttt{rng = default_rng}(20000 + 13e)$$

This RNG is used for all randomness within that epoch's dataset: pair sampling, θ sampling, and initial configuration generation. The global NumPy random state is never touched. This design ensures that:

1. Any epoch can be regenerated identically by constructing `OnTheFlyDataset` with the same seed.
2. Two training runs (e.g. a resumed run and a fresh run) produce identical datasets at every epoch, given the same curriculum schedule.
3. Epochs within the same stage are guaranteed to produce different datasets, since no two epoch seeds are identical ($10000 + 7e \neq 10000 + 7e'$ for $e \neq e'$).
4. Stage 1 and Stage 2 epochs never share a dataset ($10000 + 7e < 20000$ for all $e \leq 80$, and $20000 + 13e > 10000 + 7 \times 80 = 10560$ for all $e \geq 1$).

Initial configuration sampling. Each orbit is generated from a non-trivial random initial configuration $\mathbf{a}^0 \in \{0,1\}^n$ sampled with rejection:

```
def random_init(rng, n=50):
    while True:
        x = rng.integers(0, 2, size=n, dtype=np.uint8)
        if 0 < x.sum() < n:   # reject all-zeros and all-ones
            return x
```

The rejection condition eliminates all-zero and all-ones configurations, which are fixed points for a large number of ECA rules and would produce uninformative (constant) orbits. For $n = 50$, the probability of generating an all-zero or all-ones configuration is $2 \times 2^{-50} \approx 1.8 \times 10^{-15}$, so rejection sampling terminates in essentially one iteration. All $k = 8$ orbits within a sample use distinct `random_init` calls (the same `rng` object is passed sequentially), ensuring the 8 orbits come from different initial conditions.

Orbit simulation. After sampling the initial configuration, the orbit is simulated for $t = 200$ timesteps using either `simulate_labeled` (tempM) or `simulate_sca` (stocM). Both are pure NumPy implementations using vectorised operations over the width dimension:

```
for t in range(1, T):
    left  = np.roll(state, 1)    # periodic boundary
    right = np.roll(state, -1)
```

```
    idx   = (4*left + 2*state + right).astype(np.uint8)
    # tempM: one flip per row
    if rng.random() < tau:
        state = rule_g_table[idx]
    else:
        state = rule_f_table[idx]
    # stocM: W independent flips per row
    mask  = rng.random(W) < lam
    state = np.where(mask, rule_g_table[idx], rule_f_table[idx])
```

Each orbit simulation for $t = 200$, $n = 50$ runs in approximately 0.5–1 ms on CPU. Generating one epoch's worth of training data (1,000 pairs $\times$ 8 samples $\times$ 8 orbits $= 64{,}000$ orbit simulations) takes approximately 30–60 seconds, which is dominated by the NumPy overhead of the 199×50 array operations rather than the simulation itself. This generation cost is fully overlapped with the previous epoch's GPU training because the `OnTheFlyDataset` constructor runs on CPU in the main process while the GPU processes the previous batch.

Summary statistics. After simulation, `compute_frac_stats` and (for stocM) `compute_within_var_stats` are called once per sample to produce the 24-dimensional and 8-dimensional analytical feature vectors. These computations are also pure NumPy and run in < 1 ms per sample. The tokenisation (`orbit_to_tokens`, producing $[k, N_{\text{tok}}, 4]$) is computed for all k orbits and stored in the dataset object, so the DataLoader returns pre-tokenised float32 tensors without any per-batch computation.

D Architectural Baseline Comparison

This appendix provides the complete specification, training details, and results for the five learned architectural baselines compared against αM in Section 5, and for a zero-training algorithmic baseline evaluated on all four datasets (Section D.5). All experiments are conducted on the αECA identification task — the simplest noise type in our benchmark — to provide a clean lower bound on architectural performance: if an architecture cannot solve the easiest setting, it will not solve the harder two-rule mixtures of stocECA and tempECA. Section D.1 specifies each baseline. Section D.2 gives the training protocol. Section D.3 reports results. Section D.4 provides the mechanistic analysis.

D.1 Baseline Architectures

All baselines share αM's output heads exactly. The rule head is Linear($d \to 64$), GELU, Dropout, Linear($64 \to 8$). The α head is a 3-layer MLP: Linear($d \to 64$), GELU, Dropout, Linear($64 \to 32$), GELU, Linear($32 \to 10$), with Dropout on the first layer only. Weight initialisation follows Appendix B throughout: $\mathcal{N}(0, 0.02)$ for `Linear` layers, zeros for biases.

Baseline 1: MLP on handcrafted statistics (39,154 parameters). A 3-layer MLP operating on a 24-dimensional feature vector computed analytically from the raw orbit. Features: global mean and std of cell values (2 dims); per-row change rate statistics — mean, std, min, max (4 dims); per-pattern ($j \in \{0, \ldots, 7\}$) empirical frequency and mean output bit (16 dims); spatial and temporal autocorrelation (2 dims). Shared trunk: `Linear(24→128)→GELU→Dropout→Linear(128→128)→GELU→Dropout`, producing a 128-dim shared representation fed to both heads. This tests whether hand-engineered orbit statistics suffice for the joint identification task, providing a task-aware upper bound for feature-engineered approaches.

Baseline 2: 2D-CNN on raw orbit image (112,562 parameters). Three 2D convolutional layers treating the 100×20 orbit as a single-channel binary image:

$$\texttt{Conv2d}(1{\to}32,\ 3{\times}3) \to \text{BN} \to \text{GELU} \to \texttt{Conv2d}(32{\to}64,\ 3{\times}3) \to \text{BN} \to \text{GELU}$$
$$\to \text{MaxPool}(2{\times}2) \to \texttt{Conv2d}(64{\to}128,\ 3{\times}3) \to \text{BN} \to \text{GELU} \to \text{AdaptiveAvgPool}(1, 1),$$

flattened to 128-dim before the heads. This tests whether local 3×3 receptive fields can aggregate the global per-pattern frequency counts required for rule identification.

Baseline 3: BiLSTM, row-wise (584,690 parameters). A 2-layer bidirectional LSTM consuming the orbit row-by-row: at each timestep t, the 20-dimensional raw row is the input. Hidden size 128 per direction; outputs concatenated to 256-dim and mean-pooled over all timesteps before the heads. Tests whether sequential global context accumulated across 100 timesteps can substitute for direct token-to-token attention.

Baseline 4: Vanilla Transformer (813,170 parameters). Identical to αM in every component — same (L, C, R, A) 4-dim tokenisation, same 4-layer encoder ($d = 128$, 4 heads, FF=512, PreNorm, GELU, Dropout=0.1), same weight init, same training protocol — except for two targeted substitutions:

1. **TripletPE2D $\rightarrow$ flat 1D sinusoidal PE** over the flattened 1,980-token sequence.
2. **StatPool $\rightarrow$ mean pooling** for the α head.

This is the critical controlled ablation: any performance gap between this model and αM is attributable solely to TripletPE2D and StatPool, since every other component is held fixed.

Baseline 5: Decoder-only Transformer, causal, from scratch (1,053,187 parameters). A causally masked transformer trained from random initialisation with next-token cross-entropy, i.e. the architecture class and objective of LLMs at the scale of the specialised models. The input is the raw orbit serialised as digits, followed by the targets: `[orbit digits] [SEP] [8 rule bits] [SEP] [4 α-class bits]`, over a 3-token vocabulary (2,014 tokens per sequence), with the loss computed on the target tokens only, matching the LLM fine-tuning of Appendix F.3. It uses no analytical estimator and no task-specific pooling. Architecture: 4 layers, $d_{\text{model}} = 128$, 4 heads, feed-forward dimension 512, PreNorm, GELU, dropout 0.1, learned positional embeddings. Unlike Baseline 4, it is autoregressive and reads raw digits rather than cell-transition tokens.

Baseline 6: Vision Transformer, from scratch (1,075,602 parameters). The orbit is rendered as a single-channel image with each cell drawn as an 8×8 pixel block (0 black, 1 white), giving 800×160 pixels, and split into non-overlapping 8×8 patches, i.e. one patch per cell and 2,000 patches per orbit. Patches are linearly embedded, a CLS token is prepended, and learnable positional embeddings are added; a bidirectional encoder (4 layers, $d_{\text{model}} = 128$, 4 heads, feed-forward dimension 512, PreNorm, GELU, dropout 0.1) is followed by rule (8-bit BCE) and α (10-class cross-entropy) heads on the CLS output. It is randomly initialised, with no pretrained weights and no analytical estimator.

D.2 Training Protocol

All four baselines follow αM's two-phase curriculum (Appendix C) exactly, with every hyperparameter matched.

Data and splits. Phase 1: 35,800 samples (179 training rules $\times$ 200 synchronous orbits, $\alpha = 1.0$). Phase 2 train: 89,500 samples (179 rules $\times$ 50 samples per α level $\times$ 10 levels). Phase 2 test: 15,400 samples (77 held-out test rules, the same seed-42 partition used throughout the paper). 15% of each phase's training data is held out as a fixed validation split before training begins: $\lfloor 35{,}800 \times 0.15 \rfloor = 5{,}370$ (Phase 1) and $\lfloor 89{,}500 \times 0.15 \rfloor = 13{,}425$ (Phase 2), both verified at runtime via assertion. The split uses `random_split([n_val, n_tr], seed=42)` — validation first, training second.

Optimiser and schedule (Appendix C, Table 14). AdamW; $\beta = (0.9, 0.999)$; $\varepsilon = 10^{-8}$; weight decay 10^{-2}; gradient clip 1.0. Phase 1: peak LR 3×10^{-4}, min LR 10^{-5}, 50 epochs, cosine. Phase 2: peak LR 10^{-4}, min LR 10^{-6}, 30 epochs, cosine. Effective batch size 128 via gradient accumulation (physical 8, accumulation 16). `pin_memory=True`, `num_workers=0`.

Loss weights (Appendix C, Table 13). $\lambda_{\text{rule}} = 1.0$ throughout. $\lambda_\alpha = 0.0$ in Phase 1 (noise head receives zero gradient); $\lambda_\alpha = 0.3$ in Phase 2. Phase 2 is initialised from the best Phase 1 checkpoint, selected by highest validation rule exact match.

Noise metric note. All baselines use the neural α head only — no hybrid estimator. αM's reported noise accuracy of 95.08% includes the hybrid estimator; all baselines are evaluated on the neural head alone. For a fair comparison, αM's neural-head-only α tolerance accuracy is 75.3% (MAE 0.0275) — still substantially above every baseline, and the neural-head gap to the Vanilla Transformer is 19.7 pp rather than 39.5 pp.

Decoder-only protocol. Baseline 5 is trained for 30 epochs on the Phase 2 training split (89,500 samples) with AdamW (learning rate 3×10^{-4}, weight decay 10^{-2}), cosine annealing, batch size 32 and gradient clipping at 1.0, without the synchronous Phase 1. It is evaluated on the same Phase 2 test split (15,400 samples, 77 held-out rules); the reported accuracy is that of the final epoch, with no checkpoint selection on the test set. Evaluation is teacher-forced – each answer bit is predicted given the true preceding bits – which is an upper bound on free autoregressive decoding.

Vision Transformer protocol. Baseline 6 is trained for 30 epochs on the Phase 2 training split (89,500 samples) with AdamW (learning rate 3×10^{-4}, weight decay 10^{-2}), cosine annealing, batch size 32 and gradient clipping at 1.0, minimising the unweighted sum of the rule and α losses. It is evaluated on the Phase 2 test split (15,400 samples, 77 held-out rules); the reported accuracy is that of the final epoch.

Hardware. Single NVIDIA RTX A1000 (16 GB), identical to αM's training hardware. Total wall-clock time: 37.2 hours for all four baselines combined.

D.3 Results

Table 17: Complete architectural baseline results on αECA (Phase 2 test set, $N = 15{,}400$ samples, 77 held-out test rules). Rule: 8-bit exact match. α-tol: tolerance accuracy ± 0.05. The αM (neural only) row enables a fair head-to-head comparison with the baselines, which all use the neural α head only.

Architecture	Parameters	Rule%	α-tol%
Random (1/256 rule, uniform α)	0	0.39	10.00
Majority-class	0	1.30	10.00
Algorithmic counting (zero training)	0	92.57	14.97
MLP on handcrafted statistics	39,154	87.35	64.19
CNN (2D-conv, raw orbit image)	112,562	71.66	56.97
BiLSTM (2-layer, row-wise)	584,690	10.30	34.68
Vanilla Transformer (1D-PE, mean-pool)	813,170	70.08	55.57
Decoder-only Transformer (causal, scratch)	1,053,187	44.38	42.85
Vision Transformer (8×8 patches, scratch)	1,075,602	2.26	27.32
αM — ours (neural head only)	1,760,000	**99.22**	75.30
αM — ours (with hybrid estimator)	1,760,000	**99.22**	**95.08**

Training dynamics. Table 18 reports validation rule exact match at representative epochs from the training logs, together with the best Phase 1 checkpoint used to initialise Phase 2. The BiLSTM and Vanilla Transformer have the slowest Phase 1 convergence, while the MLP converges fastest — consistent with the analysis in Section D.4.

D.4 Mechanistic Analysis

MLP: rule tractable from handcrafted statistics; noise is not. The per-pattern mean output bit `pat_out[`j`]` — one of the 24 input features — converges to the rule's truth-table entry for pattern j as the orbit grows. With ≈1,980 transitions per orbit and near-uniform pattern visitation frequencies in chaotic orbits, the MLP reads the 8-bit rule table almost directly from the empirical statistics. This explains the high rule accuracy (87.35%) despite only 39K parameters and no learned representation.

The remaining 11.87 pp gap to αM arises from two sources. First, Class I (null) rules converge rapidly to homogeneous states, reducing the number of informative transitions and biasing `pat_out`

Table 18: Validation rule exact match (%) at representative Phase 1 and Phase 2 epochs, from the training logs. Best Phase 1 val is the checkpoint used to initialise Phase 2. Phase 2 test rule% is the final evaluation on the held-out test set.

	Phase 1 val rule% (epoch)				Phase 2 val rule% (epoch)			Test
Model	ep10	ep20	ep30	Best (ep)	ep10	ep20	ep30	Rule%
MLP	92.16	92.53	92.89	92.98 (37)	90.64	91.23	91.37	87.35
CNN	64.88	76.39	81.96	85.90 (46)	82.91	83.81	85.00	71.66
BiLSTM	7.32	24.15	36.95	47.88 (49)	27.26	33.77	35.72	10.30
VanTrans	2.70	23.87	61.96	82.33 (50)	74.64	84.57	87.14	70.08

estimates away from the true rule entry. Second, the MLP cannot adapt its feature weighting based on a partially-inferred rule, as the transformer does dynamically through self-attention.

The α failure (64.19%) has a structurally different cause. The change-rate features conflate rule dynamics with noise: a chaotic rule generates a high baseline change rate even at low α, while a quiescent rule generates a low rate even at high α. The MLP must resolve this ambiguity from a shared 128-dim trunk, with no mechanism to condition the noise estimate on the inferred rule. αM's StatPool operates on transformer representations that have already encoded neighbourhood structure through self-attention, extracting activation spread (std, max, min) already conditioned on the inferred rule — a separation the MLP cannot achieve.

CNN: local receptive field cannot aggregate global pattern counts. A 3×3 kernel sees a 9-cell neighbourhood at a single location. Rule identification requires counting how often each of the 8 three-bit neighbourhood patterns maps to output 0 or 1 across all ≈1,980 transitions. MaxPool and AdaptiveAvgPool destroy the precise cell-alignment required for this global counting, leaving only texture information. The consequence is directly visible in the data: the CNN (71.66% rule, 112K params) is weaker on both metrics than the MLP (87.35% rule, 39K params), which has 2.9× fewer parameters. The MLP's handcrafted `pat_freq` and `pat_out` features explicitly compute what the CNN must attempt to infer from local texture. Phase 1 dynamics confirm this: the CNN reaches only 85.90% validation rule accuracy after 50 epochs on noise-free orbits, while the MLP converges to 92.98% at epoch 37.

BiLSTM: sequential processing cannot generalise the rule computation. The defining observation is the 25.4 pp gap between Phase 2 validation rule accuracy (35.72%) and test rule accuracy (10.30%): the BiLSTM overfits to correlations in the 179 training rules rather than learning the general neighbourhood computation, and fails to transfer to the 77 unseen test rules.

The root cause is structural. At each timestep t, the LSTM receives the 20-dim raw row. To identify the rule, it must detect that cell i has neighbourhood $(L, C, R) = j$ and that `orbit[t+1][i] = rule[`j`]` — a cross-cell and cross-timestep operation. The LSTM hidden state must implicitly accumulate a per-pattern histogram across 100 timesteps with no inductive bias for doing so. Instead, the model learns global row-level statistics (row mean, row variance) correlated with the training rules, which do not generalise to unseen rules.

The relatively less catastrophic α accuracy (34.68%) is mechanistically consistent: the change rate is a simpler sequential pattern that the LSTM can partially track in its hidden state by observing how much consecutive rows differ — a weaker signal than rule identification but one that is available directly in the input.

Vanilla Transformer: isolating TripletPE2D and StatPool. *Rule gap (29.1 pp, flat 1D-PE vs TripletPE2D):* With flat 1D-PE, the token at grid position $(t = 2, i = 7)$ and the token at $(t = 3, i = 7)$ — same cell, adjacent timesteps — are separated by $n = 20$ positions in the flattened sequence, receiving very different positional embeddings despite encoding the same spatial location. The transformer must infer from training data alone that these tokens share a spatial coordinate. TripletPE2D assigns independent 64-dim sinusoidal encodings to the temporal (t) and spatial (i) axes, making cell-sharing and timestep-sharing directly decodable from the positional signal and enabling efficient attention grouping by grid coordinate. The Vanilla Transformer still benefits from

global self-attention (hence 70.08%, well above CNN and BiLSTM) but cannot match αM's 29.1 pp advantage on unseen rules.

α gap (39.5 pp full; 19.7 pp neural-head-only): Mean pooling $\hat{h} = \frac{1}{1980}\sum_k h_k$ is variance-blind: it returns the same value regardless of whether the token activations are tightly clustered (high α, most cells updated) or widely spread (low α, many frozen cells producing distinct activation patterns from updated cells). StatPool computes $[\bar{h},\ \sigma(h),\ \max(h),\ \min(h)]$ and projects the 512-dim concatenation through a LayerNorm-stabilised MLP. The inclusion of $\sigma(h)$ directly captures the variance that encodes how often the rule was violated — the primary neural signal for α. Both the full gap (39.5 pp) and the neural-head-only gap (19.7 pp) confirm that StatPool's contribution to the α stream exceeds TripletPE2D's contribution to the rule stream: getting the pooling right matters more for noise estimation than getting the positional encoding right for rule identification.

Decoder-only Transformer: the LLM architecture class, trained on the task. Trained from scratch on the same 89,500 orbits, the causal decoder learns steadily (rule accuracy rises from 2% at epoch 2 to 44.38% at epoch 30), far above every fine-tuned LLM ($\leq 14.6\%$), but it remains 54.84 points below αM and below the CNN (71.66%) and MLP (87.35%) trained on identical data. Rule identification requires counting, for each of the 8 neighbourhood patterns, how often each output follows it across the orbit. A causal model reading one digit at a time must reconstruct each 3-cell neighbourhood and its next-row successor, $n = 20$ positions later, from a left-to-right stream, and cannot attend to later rows when summarising earlier ones. The cell-transition tokenisation and bidirectional attention of Baseline 4 remove both obstacles (70.08%), and the 2D positional encoding and statistical pooling of αM close the remaining gap (99.22%).

Vision Transformer: the token unit, not attention, is the bottleneck. The ViT reaches 2.26% rule accuracy, barely above chance (0.39%), while the Vanilla Transformer, which uses the same global self-attention on the same data, reaches 70.08%. The difference lies in what a token contains. An ECA rule maps a 3-cell neighbourhood to the next state of its centre cell. A cell-transition token $(a_{i-1}^t, a_i^t, a_{i+1}^t, a_i^{t+1})$ contains exactly one such application, and a 3×3 CNN kernel covers one neighbourhood together with its successor. With one patch per cell, each ViT token carries a single state: every rule application must be reassembled from four tokens spread over two rows, and which tokens are neighbours is known to the model only through positional embeddings over $2{,}000$ flattened positions that must themselves be learned from data. Holding attention, data and scale fixed, changing the token unit moves rule accuracy from 2% to 70%.

Representation versus architecture. Taken together, the baselines separate the two effects the identification task depends on. The input unit determines whether a single rule application is directly observable: single-cell image tokens (2.26%) and raw digit streams (44.38% for a causal decoder trained on the task) are far below cell-transition tokens (70.08%) and local kernels (71.66%). Given the right unit, the architecture determines whether per-pattern statistics can be aggregated across the whole orbit: the 2D positional encoding and statistical pooling of αM raise the same tokens from 70.08% to 99.22%, and scaling the generic models does not substitute for them (Section D.6). The same two factors appear in the LLM evaluation, where supplying the per-pattern statistic directly raises GPT-5.1 from 4% to 60% but not to the level of a lookup on that statistic (Appendix F.10).

D.5 Zero-Training Algorithmic Baseline on All Four Datasets

To calibrate the difficulty of the benchmark, we evaluate a non-neural identification method with no training on the held-out test sets of all four datasets.

Rule identification. Under α-asynchronous and s-skewed noise, a cell either keeps its state or takes the rule output. A transition $0 \to 1$ at neighbourhood pattern j therefore implies $R[j] = 1$, and $1 \to 0$ implies $R[j] = 0$; this direction-of-flip evidence is unaffected by the noise. Each rule bit is read from these counts. For tempECA, timesteps are first clustered by the rule that generated them via iterative pooled re-estimation, and the same lookup is applied within each cluster; for stocECA, the same clustering is applied to individual transitions (expectation–maximisation over 10 iterations).

Noise estimation. The noise parameter is estimated with a direct statistic of the identified rule: the match rate between observed and predicted transitions (αECA), the spatial span of changed cells per row (skewECA), and the fraction of units assigned to each cluster (tempECA, stocECA).

Table 19: Zero-training algorithmic baseline vs. specialised models on the held-out test sets.

Dataset	Metric	Algorithmic	Specialised
αECA	Rule exact match	92.57%	99.22%
skewECA	Rule exact match	94.11%	94.55%
tempECA	Rule pair (UO)	79.92%	89.05%
stocECA	Rule pair (UO)	0.02%	95.49%
αECA	α MAE	0.325	0.0070
skewECA	s MAE	1.381	0.221
tempECA	τ MAE	0.0833	0.0311
stocECA	λ estimate	constant ≈ 0.5	MAE 0.0289

Interpretation. Where the dynamics decompose into units governed by a single rule – the whole orbit for αECA and skewECA, a whole timestep for tempECA – rule bits can be read by counting, and the baseline reaches 80–94%, above the learned MLP, CNN and decoder baselines of Table 17. The specialised models match or exceed it on every dataset. Noise estimation is harder: naive statistics conflate the rule's own dynamics with the noise, giving MAE 3–46$\times$ higher than the specialised models. stocECA is the decisive case: with independent per-cell mixing, no timestep or orbit follows a single rule, so unit-level clustering has nothing to cluster, and the baseline fails (0.02%, with the noise estimate collapsing to 0.5). The mixture must instead be resolved through per-pattern output statistics across the whole orbit, which is what stocM learns to do (95.49%).

D.6 Scaling the Default Architectures

To test whether the gap between default architectures and αM is a matter of capacity, we scale the CNN and the Vanilla Transformer (Baselines 2 and 4) by widening and deepening them 2$\times$, 4$\times$ and 8$\times$, and train every model under the identical protocol of Appendix D.2 on αECA. All scaled models are independent retrains, and each is compared with its own 1$\times$ retrain; the 1$\times$ Vanilla Transformer retrain reaches 77.89% rule accuracy versus 70.08% in Table 17, indicating run-to-run variation of several points for this baseline.

Table 20: Scaling the CNN and Vanilla Transformer on αECA (Phase 2 test set, $N = 15{,}400$).

Model	Scale	Parameters	Rule%	α-tol%
CNN	1$\times$	112,562	71.48	58.52
	2$\times$	406,386	78.22	64.62
	4$\times$	1,546,994	79.45	67.55
	8$\times$	6,040,050	80.67	59.49
Vanilla Transformer	1$\times$	813,170	77.89	61.08
	2$\times$	3,196,146	87.15	72.38
	4$\times$	12,680,690	86.30	75.21
αM (ours)	–	≈1.76M	99.22	95.08

Both architectures show diminishing returns. The CNN's rule-accuracy gain per doubling falls from +6.7 points (1$\times \to$2$\times$) to +1.2 (4$\times \to$8$\times$), and its α accuracy drops at 8$\times$. The Vanilla Transformer plateaus after 2$\times$ (87.15%, then 86.30%). At 3.4$\times$ and 7.2$\times$ the parameter count of αM, the scaled CNN and Vanilla Transformer still trail it by 18.6 and 12.9 points in rule accuracy and by 35.6 and 19.9 points in α accuracy. A local or position-agnostic architecture does not acquire the ability to aggregate the per-pattern statistic by growing larger; the 2D positional encoding and statistical pooling of αM provide it at a fraction of the size.

D.7 Robustness of Task Difficulty to Grid Size and Orbit Length

To test whether the chosen grid width and orbit length make the task artificially easy or hard, we generate fresh orbits for the 77 held-out rules with the same generation pipeline at half and double the original width and length, for αECA and skewECA, and evaluate them with the zero-training algorithmic baseline of Section D.5. Because the baseline has no learned parameters, any change across sizes reflects a property of the task rather than of training.

Table 21: Zero-training algorithmic baseline at half, original and double grid width W and orbit length T. For reference, a constant prediction of the central noise value gives α MAE 0.25 and s MAE ≈ 5.

Dataset	Size	Rule%	Noise MAE	s tol. (±1)
αECA	W=10, T=50 (half)	78.44	0.3148	–
	W=20, T=100 (original)	92.57	0.3245	–
	W=40, T=200 (double)	98.80	0.3313	–
skewECA	W=10, T=100 (half)	80.65	1.082	75.71
	W=20, T=200 (original)	94.11	1.381	76.97
	W=40, T=400 (double)	99.20	1.584	76.41

Rule identification. Accuracy rises monotonically with size (from 78–81% at half size to 99% at double size) because each neighbourhood pattern is observed more often. The original size lies in the middle of this range; it neither saturates the task nor starves it of evidence.

Noise estimation. The naive noise statistics do not improve with size: the α MAE stays at 0.31–0.33, above the 0.25 of a constant prediction, and the s tolerance stays at 76–77%. Their error is a bias, which additional observations do not remove: a direct statistic of the orbit mixes the rule's own dynamics with the noise. The hybrid estimator, which conditions on the identified rule, removes this bias and reaches α MAE 0.0070 at the original size (Appendix G.6). Accurate noise recovery therefore depends on the right statistic, not on more data.

Scope. This experiment establishes that the difficulty of the single-rule tasks varies smoothly with size and is not an artefact of the chosen dimensions. It does not retrain the specialised models at each size, and it does not cover the two-rule datasets, where disentangling the rules adds a separate source of difficulty.

D.8 Matched-Component Ablation

To isolate the contribution of the noise-matched components, we replace them with generic counterparts while keeping the tokenisation, encoder, training protocol and data fixed. For αM, replacing TripletPE2D with a 1D sinusoidal positional encoding and StatPool with mean pooling gives exactly the Vanilla Transformer of Baseline 4. For skewM, we apply the same substitution, replacing TripletPE2D and SpatialMultiScalePool with a 1D positional encoding and mean pooling.

Table 22: Matched-component ablation on the held-out test sets. αM noise accuracy is that of the learned noise head in both rows (±0.05); skewM noise accuracy is s tolerance (±1).

Model	Rule%	Noise%
αM (TripletPE2D + StatPool)	99.22	75.30
generic (1D PE + mean pool; Baseline 4)	70.08	55.57
difference	-29.1	-19.7
skewM (TripletPE2D + SpatialMultiScalePool)	94.55	96.97
generic (1D PE + mean pool)	80.33	78.85
difference	-14.2	-18.1

For two structurally different noise types, the same substitution costs 14–29 points of rule accuracy and 18–20 points of noise accuracy. The gap is therefore attributable to the matched components, not to the backbone, the data or the training protocol, which are identical in each pair.

For tempM and stocM, the matched components are not drop-in modules: the τ- and λ-conditioned rule heads are the mechanism that disentangles the two rules, and removing them leaves no rule-pair output to evaluate. Their contribution is instead shown by comparison with the zero-training algorithmic baseline (Section D.5): on tempECA, where timesteps can be clustered by rule, counting reaches 79.92% against tempM's 89.05%, and on stocECA, where no timestep or orbit follows a single rule, it fails (0.02%) while stocM reaches 95.49%.

Why not a single architecture? The four models already share one design: cell-transition tokens, TripletPE2D, a transformer encoder and a two-phase curriculum, with a flat or two-level encoder depending on whether one or two rules must be recovered. What differs is a small head matched to the statistical signature of each noise type. A single head shared across noise types is exactly the generic configuration in Table 22, and it loses 14–29 points of rule accuracy. The per-noise head is therefore not incidental specialisation but the component that the results identify as necessary.

E Shared Model Design Details

This appendix documents the design choices shared across all four models: tokenisation, positional encoding, and standard engineering conventions. Components specific to individual models are documented in Appendix B.

E.1 Tokenisation

All four models operate on the same fundamental unit: the *cell-transition token*, a vector encoding one complete observation of a rule application. The key differences are in the number of features per token (4 for αM, tempM, and stocM; 5 for skewM) and the number of tokens per orbit $(t-1)\times n$, which varies by dataset.

Common tokenisation formula

For an orbit $\mathbf{O}\in\{0,1\}^{t\times n}$, let $a_i^{t'}$ denote the state of cell i at timestep t', with periodic boundary conditions: $a_{-1}^{t'}=a_{n-1}^{t'}$ and $a_n^{t'}=a_0^{t'}$.

The 4-dimensional base token at grid position (t',i) for $t'\in\{0,\dots,t-2\}$, $i\in\{0,\dots,n-1\}$ is:

$$\mathbf{x}_{t',i}=\big(\underbrace{a_{i-1}^{t'}}_{\text{left}},\ \underbrace{a_i^{t'}}_{\text{centre}},\ \underbrace{a_{i+1}^{t'}}_{\text{right}},\ \underbrace{a_i^{t'+1}}_{\text{next}}\big)\quad\in\{0,1\}^4 \tag{49}$$

This token encodes one complete ECA rule application: given the 3-bit neighbourhood $(a_{i-1}^{t'},a_i^{t'},a_{i+1}^{t'})$, the deterministic synchronous ECA rule predicts $a_i^{t'+1}$. Each token is therefore a *self-contained rule observation* — the rule number can in principle be read off from any single token (given knowledge of which rule is active), and the full sequence of $(t-1)n$ tokens provides a rich statistical sample of the rule's behaviour under the given noise process.

The neighbourhood index $j_{t',i}\in\{0,\dots,7\}$ is implicit in the token:

$$j_{t',i}=4\,a_{i-1}^{t'}+2\,a_i^{t'}+a_{i+1}^{t'} \tag{50}$$

This is the standard ECA pattern numbering: pattern 0 is $(0,0,0)$, pattern 7 is $(1,1,1)$, and the rule output for pattern j is $R[j]=(R\gg j)\ \&\ 1$ where R is the rule number. The expected next-state under the synchronous rule is $a_i^{t'+1}=R[j_{t',i}]$; deviations from this expectation are caused by the noise process, and the pattern of these deviations across all $(t-1)\times n$ tokens is what the model reads to identify the noise parameter.

THE DID_CHANGE AUGMENTATION (SKEWM ONLY)

skewM adds a fifth binary feature to each token:

$$\mathbf{x}^{\text{skew}}_{t',i} = \big(a^{t'}_{i-1},\ a^{t'}_{i},\ a^{t'}_{i+1},\ a^{t'+1}_{i},\ \underbrace{\mathbf{1}[a^{t'+1}_{i} \neq a^{t'}_{i}]}_{\texttt{did_change}}\big) \quad \in \{0,1\}^5 \tag{51}$$

The rationale is specific to the s-skewed noise structure. Under s-skewed noise, exactly s contiguous cells are selected and updated at each timestep; the remaining $n - s$ cells retain their state from the previous timestep, so $a^{t'+1}_i = a^{t'}_i$ for the $n - s$ non-updated cells regardless of what the rule would have predicted. Therefore, `did_change` $= 0$ for non-updated cells and `did_change` $= 1$ for updated cells (when the rule actually changed their state). This directly identifies membership in the update block: a contiguous run of `did_change`$= 1$ values in a given row marks the spatial extent of the update block at that timestep. Without this feature, the model would need to reconstruct the update block membership from the raw $(a^{t'}_{i-1}, a^{t'}_i, a^{t'}_{i+1}, a^{t'+1}_i)$ tuple by comparing $a^{t'+1}_i$ with what the rule predicts for $(a^{t'}_{i-1}, a^{t'}_i, a^{t'}_{i+1})$ — which requires knowing the rule. Including `did_change` as an explicit feature provides the contiguity signal directly, before the rule has been identified, allowing the attention mechanism to detect block structure in its early layers.

PSEUDOCODE

Algorithm 4 gives the complete `orbit_to_tokens` procedure for all four models, with the skewM variant indicated.

Algorithm 4 `orbit_to_tokens`: orbit → token sequence

Require: Orbit $\mathbf{O} \in \{0,1\}^{t \times n}$ (stored as float32), flag `include_did_change` (True for skewM, False otherwise)
Ensure: Token array $\mathbf{X} \in \mathbb{R}^{(t-1)n \times d_{\text{feat}}}$, $d_{\text{feat}} \in \{4, 5\}$

```
before ← O[0 : t − 1, :]                                   ▷ rows 0, . . . , t − 2, shape [t − 1, n]
after ← O[1 : t, :]                                        ▷ rows 1, . . . , t − 1, shape [t − 1, n]
left ← roll(before, +1, axis = 1)     ▷ shift right by 1 (periodic), giving left neighbour
right ← roll(before, −1, axis = 1)    ▷ shift left by 1 (periodic), giving right neighbour
if include_did_change then
    dc ← (before ≠ after).astype(float32)                  ▷ did_change, shape [t − 1, n]
    tokens ← stack([left, before, right, after, dc], axis = −1)   ▷ shape [t − 1, n, 5]
else
    tokens ← stack([left, before, right, after], axis = −1)       ▷ shape [t − 1, n, 4]
end if
return tokens.reshape(−1, d_feat)                          ▷ flatten to [(t − 1)n, d_feat]
```

The `roll` operation is NumPy's `np.roll` with periodic (wrap-around) boundary: cell $i = 0$'s left neighbour is cell $i = n - 1$, and cell $i = n - 1$'s right neighbour is cell $i = 0$. The reshape in the final line enumerates tokens in *row-major order*: all n tokens of timestep $t' = 0$ come first, then all n tokens of timestep $t' = 1$, and so on. This ordering is what TripletPE2D (described in Section 5) exploits to assign the correct 2D positional encoding to each token.

TOKEN COUNTS AND SHAPES

Table 23 summarises the token dimensions for all four models.

For tempM and stocM, tokenisation is applied independently to each of the $k = 8$ orbits per sample, producing a tensor of shape $[k, (t-1)n, 4]$. This is then reshaped to $[B \cdot k \cdot (t-1),\ n,\ d_{\text{feat}}]$ inside the CellEncoder (Appendix B.3) to process all timesteps of all orbits in a single batched forward pass.

WHY THIS TOKENISATION?

The 4-feature base token is the minimal sufficient statistic for a single ECA rule observation: it contains the complete input $(a^{t'}_{i-1}, a^{t'}_i, a^{t'}_{i+1})$ and the output $a^{t'+1}_i$, and nothing else. Including the

Table 23: Token array shapes for all four models after `orbit_to_tokens`. For tempM and stocM, the PatternEmbedding (Appendix B) augments each token from 4 to 12 features before the first linear projection.

Model	n	t	d_{feat}	$(t-1)n$	Token array shape
αM	20	100	4	1,980	[1,980, 4]
skewM	20	200	5	3,980	[3,980, 5]
tempM	50	200	4	9,950	[k=8, 9,950, 4]
stocM	50	200	4	9,950	[k=8, 9,950, 4]

neighbourhood index $j = 4a_{i-1}^{t'} + 2a_i^{t'} + a_{i+1}^{t'}$ as an additional pre-computed feature would be redundant (it is a deterministic function of the three neighbourhood bits) but not harmful; we omit it from the raw token because the PatternEmbedding (for tempM and stocM) and the linear input projection (for all models) can reconstruct it implicitly from the 3 binary neighbourhood bits. Storing the orbit as a 2D binary array and computing tokens on-the-fly in `orbit_to_tokens` rather than pre-computing and storing all $(t-1)n$ tokens saves a factor of $(t-1)n/n = t-1 \approx 100$–$200\times$ in storage, which is significant at the dataset scales used here.

E.2 Standard Engineering Choices

This subsection documents the engineering choices shared by all four models. None of these are novel contributions — they are standard practices for transformer training — but they are stated precisely here so that the models can be reproduced without reading the code.

Pre-Layer Normalisation (PreNorm)

All transformer encoder layers use Pre-Layer Normalisation, specified by `norm_first=True` in PyTorch's `TransformerEncoderLayer`. In standard Post-Norm, layer normalisation is applied *after* the residual add:

$$\mathbf{H}' = \text{LN}(\mathbf{H} + \text{Sublayer}(\mathbf{H})) \qquad \text{(Post-Norm)}$$

In PreNorm, it is applied *before* the sublayer:

$$\mathbf{H}' = \mathbf{H} + \text{Sublayer}(\text{LN}(\mathbf{H})) \qquad \text{(PreNorm)}$$

PreNorm is used throughout because it produces more stable gradients during early training and prevents the loss landscape from developing sharp curvature when the noise estimation head is activated in Phase 2/Stage 2. With Post-Norm, the unnormalised residual stream can develop large activation scales during the initial epochs of Phase 1 (when only rule loss gradients flow), and activating the noise head in Phase 2 then causes gradient spikes that disrupt the established rule representations. PreNorm keeps the residual stream in a well-scaled regime throughout.

Each transformer layer therefore applies two LayerNorm operations per forward pass: one before the multi-head self-attention sublayer and one before the feed-forward sublayer. Both LayerNorm instances have learnable weight and bias parameters, contributing $2 \times 2d$ parameters per layer (where d is the model dimension).

`enable_nested_tensor=False` is set on all `TransformerEncoder` instances to suppress a UserWarning triggered by PyTorch ≥ 2.0 when `norm_first=True` is used with nested tensor optimisation. This flag has no effect on computation or results.

GELU Activation

All non-linear activations throughout all four models use the Gaussian Error Linear Unit (GELU):

$$\text{GELU}(x) = x \cdot \Phi(x) = \frac{x}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \qquad (52)$$

where Φ is the standard normal CDF. GELU is specified as `activation="gelu"` in `TransformerEncoderLayer`, which uses PyTorch's `F.gelu` implementation (the exact form

above, not the tanh approximation). GELU is used rather than ReLU because it produces smooth, non-zero gradients near zero, which benefits the continuous noise estimation heads (α classification, s regression, τ/λ MSE) that receive small gradient signals during early training. ReLU's hard zero at negative values would silently kill gradients in these heads before they have had time to establish non-trivial representations.

Two exceptions: the Sigmoid activation is used in the s regression head of skewM (to bound the output to $[0, 1]$), and the Tanh activation is used in the correction MLP output of tempM and stocM (to bound the correction magnitude and prevent overshoot of the analytical estimate).

DROPOUT

Dropout with rate $p = 0.1$ is applied uniformly across all four models:

- After the positional encoding addition (before the first encoder layer), via `nn.Dropout(p)` inside TripletPE2D and SinusoidalPE.
- Within each `TransformerEncoderLayer`: after the attention weights (before the value aggregation), after the attention output (before the first residual add), and after the first feed-forward projection (between `linear1` and `linear2`). These are handled internally by PyTorch's `TransformerEncoderLayer(dropout=0.1)`.
- After the first GELU in each MLP head (rule head, noise head, StatPool MLP, SpatialMultiScalePool projection, score_net, attn_net), via explicit `nn.Dropout(p)` layers.

Dropout is disabled during evaluation (`model.eval()` sets all `nn.Dropout` modules to pass-through mode). The rate $p = 0.1$ is modest — sufficient to regularise the noise estimation heads, which receive fewer gradient updates (0 in Phase 1 for αM and skewM), without harming the rule head which receives gradients throughout.

GRADIENT CLIPPING

Gradient clipping at maximum norm 1.0 is applied before every optimiser step across all four models and both training stages:

```
nn.utils.clip_grad_norm_(model.parameters(), 1.0)
optimizer.step()
```

The clipping is applied to the *global* gradient norm over all parameters simultaneously, not per-parameter or per-layer. This prevents individual parameter updates from becoming arbitrarily large when the noise head is first activated in Phase 2/Stage 2 and its initially random gradients interact with the established encoder representations. The threshold of 1.0 was not tuned; it is the standard value used in transformer training and was not observed to be the active constraint during any stage of training for any of the four models (gradient norms remained well below 1.0 after the first few epochs in all runs).

WEIGHT INITIALISATION

All `nn.Linear` layers in all four models are initialised by the `_init_weights` method called in each model's `__init__`:

```
def _init_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Linear):
            nn.init.normal_(m.weight, std=0.02)
            if m.bias is not None:
                nn.init.zeros_(m.bias)
```

Weight matrices are drawn from $\mathcal{N}(0, 0.02)$ and biases are set to zero. The standard deviation of 0.02 is the standard GPT-style initialisation (Radford et al., 2019) and keeps initial layer outputs in a well-scaled range: for a linear layer with input dimension d and weights $\sim \mathcal{N}(0, 0.02)$, the output standard deviation is approximately $0.02\sqrt{d}$, which for $d = 128$ gives $0.02\sqrt{128} \approx 0.23$ — a reasonable starting scale for layer activations.

There are three explicit exceptions to this global initialisation:

1. Correction MLP output layer (tempM and stocM): The final Linear($128 \to 1$) and its Tanh activation in the correction MLP are initialised with $\mathcal{N}(0, 0.01)$ weights and zero bias:

```
# Applied to correction_net's last layer after _init_weights
nn.init.normal_(self.correction_net[-2].weight, std=0.01)
nn.init.zeros_(self.correction_net[-2].bias)
```

This ensures initial corrections are near zero ($\delta \approx 0$) so the analytical estimate dominates at the start of training. A standard 0.02 initialisation would produce initial corrections of $\mathcal{O}(0.02\sqrt{128}) \approx 0.23$, which is large relative to the typical $\hat{\tau}_{\text{analytical}} \approx 0.3$–$0.4$ and would destabilise early training.

2. LambdaConditionedRuleHead attention network output (stocM): The final Linear($64 \to 1$) of `attn_net` is initialised with $\mathcal{N}(0, 0.1)$ and zero bias:

```
nn.init.normal_(self.attn_net[-1].weight, std=0.1)
nn.init.zeros_(self.attn_net[-1].bias)
```

With $\mathcal{N}(0, 0.1)$ weights and zero bias, the initial attention scores are small random values and the softmax over 199 timesteps is approximately uniform ($\approx 1/199 \approx 0.005$ per timestep), giving near-equal weighting to all timesteps. This is desirable: at epoch 0, the model has no basis for preferring any timestep over another, and uniform attention is equivalent to mean pooling — a reasonable starting point that provides the informational benefit of averaging across all 199 timestep representations.

3. CellEncoder learnable query (tempM and stocM): The query parameter $\mathbf{q} \in \mathbb{R}^{1\times1\times D_1}$ in the CellEncoder's query-attention pooling is initialised as:

```
self.q = nn.Parameter(torch.randn(1, 1, D1) * 0.02)
```

This matches the global $\mathcal{N}(0, 0.02)$ convention and ensures the initial query does not strongly prefer any particular attention pattern before training begins.

`nn.LayerNorm` layers retain PyTorch's default initialisation: weight (scale) $= 1.0$ and bias (shift) $= 0.0$. `nn.Embedding` (PatternEmbedding in tempM and stocM) is initialised with $\mathcal{N}(0, 0.1)$ to spread the 8 pattern representations apart from initialisation — the wider spread (relative to 0.02) provides initial diversity in the pattern embeddings that the model can immediately use.

E.3 TripletPE2D

TripletPE2D is the 2D positional encoding used in αM and skewM. It is introduced in this work to address a specific problem with applying standard 1D positional encoding to the flattened token sequence produced by orbit tokenisation.

The problem with 1D positional encoding

After tokenisation, the $N_{\text{tok}} = (t-1) \times n$ tokens are enumerated in row-major order: all n tokens of transition $t = 0$ first, then all n tokens of transition $t = 1$, and so on. A standard 1D sinusoidal encoding assigns position $p = tn + i$ to token (t, i), treating the flattened sequence as a single ordered list. This conflates the two structurally distinct axes of the orbit:

- Token $(t = 0, i = 19)$ at position 19 is adjacent to token $(t = 1, i = 0)$ at position 20, even though they belong to different timesteps and different cells.
- Token $(t = 0, i = 0)$ at position 0 and token $(t = 0, i = 19)$ at position 19 appear far apart in positional space, even though they belong to the same timestep and jointly encode the spatial pattern at $t = 0$.

The result is that the attention mechanism cannot use position alone to identify whether two tokens share a timestep (a complete spatial snapshot) or share a cell index (the temporal trajectory of one cell) — both groupings carry different and complementary information about the rule and the noise.

TRIPLETPE2D CONSTRUCTION

TripletPE2D resolves this by assigning each token a *two-part* positional encoding: a sinusoidal encoding of its timestep index concatenated with a sinusoidal encoding of its cell index. Since the model dimension is $d = 128$, each part uses $d/2 = 64$ dimensions.

Component 1: Time axis encoding. For each transition index $t \in \{0, \ldots, N_{\text{trans}} - 1\}$:

$$\text{PE}_{\text{time}}(t, 2k) = \sin\left(\frac{t}{10000^{2k/64}}\right) \quad k = 0, 1, \ldots, 31 \tag{53}$$

$$\text{PE}_{\text{time}}(t, 2k+1) = \cos\left(\frac{t}{10000^{2k/64}}\right) \quad k = 0, 1, \ldots, 31 \tag{54}$$

This produces a matrix $\mathbf{P}_{\text{time}} \in \mathbb{R}^{N_{\text{trans}} \times 64}$ where row t encodes the timestep position.

Component 2: Cell axis encoding. For each cell index $i \in \{0, \ldots, n-1\}$:

$$\text{PE}_{\text{cell}}(i, 2k) = \sin\left(\frac{i}{10000^{2k/64}}\right) \quad k = 0, 1, \ldots, 31 \tag{55}$$

$$\text{PE}_{\text{cell}}(i, 2k+1) = \cos\left(\frac{i}{10000^{2k/64}}\right) \quad k = 0, 1, \ldots, 31 \tag{56}$$

This produces a matrix $\mathbf{P}_{\text{cell}} \in \mathbb{R}^{n \times 64}$ where row i encodes the cell position.

Assembly. For each token (t, i) in the flattened sequence at position $p = tn + i$:

$$\text{PE}(t, i) = \left[\mathbf{P}_{\text{time}}[t] \parallel \mathbf{P}_{\text{cell}}[i]\right] \quad \in \mathbb{R}^{128} \tag{57}$$

where $\parallel$ denotes concatenation. The full buffer is computed once at construction:

$$\mathbf{t_idx} = [0, 0, \ldots, 0,\ 1, 1, \ldots, 1,\ \ldots,\ N_{\text{trans}} - 1, \ldots] \in \mathbb{Z}^{N_{\text{tok}}} \quad \text{(each value repeated } W \text{ times)} \tag{58}$$

$$\mathbf{c_idx} = [0, 1, \ldots, n-1,\ 0, 1, \ldots, n-1,\ \ldots] \in \mathbb{Z}^{N_{\text{tok}}} \quad \text{(repeating cycle of length } W) \tag{59}$$

$$\mathbf{PE} = \left[\mathbf{P}_{\text{time}}[\mathbf{t_idx}] \parallel \mathbf{P}_{\text{cell}}[\mathbf{c_idx}]\right] \in \mathbb{R}^{N_{\text{tok}} \times 128} \tag{60}$$

The buffer $\mathbf{PE}$ is stored as a registered buffer (shape $[1, N_{\text{tok}}, 128]$) and added to the projected token embeddings:

$$\mathbf{H}^{(0)} \leftarrow \text{Dropout}_{p=0.1}\left(\mathbf{H}^{(0)} + \mathbf{PE}\right) \tag{61}$$

Implementation. The complete construction in PyTorch:

```
def sinpe(n_pos, n_dim):
    pe  = torch.zeros(n_pos, n_dim)
    pos = torch.arange(n_pos, dtype=torch.float).unsqueeze(1)
    div = torch.exp(torch.arange(0, n_dim, 2, dtype=torch.float)
                    * (-math.log(10000.0) / n_dim))
    pe[:, 0::2] = torch.sin(pos * div)
    pe[:, 1::2] = torch.cos(pos * div)
    return pe

half = d_model // 2                                          # 64
time_pe = sinpe(n_trans, half)                               # [N_TRANS, 64]
cell_pe = sinpe(n_cells, half)                               # [n, 64]
t_idx = torch.arange(n_trans).repeat_interleave(n_cells)     # [N_TOK]
c_idx = torch.arange(n_cells).repeat(n_trans)                # [N_TOK]
full_pe = torch.cat(
    [time_pe[t_idx], cell_pe[c_idx]], dim=-1
)                                                            # [N_TOK, 128]
self.register_buffer(
    "pe", full_pe.unsqueeze(0)
)                                                            # [1, N_TOK, 128]
```

WHY 64+64=128 MATCHES d_{MODEL}

The split is chosen so that the concatenated positional encoding has exactly the same dimension as the token embeddings ($d = 128$), allowing a direct element-wise addition $\mathbf{H}^{(0)} + \mathbf{PE}$ without any projection. Using half the model dimension for each axis ($d/2 = 64$) is the natural choice because it gives equal representational budget to the time axis and the cell axis. A 96+32 split (more budget for time, less for cells) would be reasonable given that $N_{\text{trans}} \gg n$ (99 vs 20 for αM, 199 vs 20 for skewM), but the equal split was found to work well in practice and avoids the need to tune the split ratio as a hyperparameter.

Each of the 64-dimensional axis encodings covers frequencies from $\omega_0 = 10000^{0/64} = 1$ (fastest, period $2\pi \approx 6.3$) to $\omega_{31} = 10000^{62/64} \approx 7{,}197$ (slowest, period $2\pi \times 7197 \approx 45{,}222$). For the time axis with $N_{\text{trans}} = 99$ or 199 positions, the fastest frequency (period ≈ 6.3) provides fine-grained discrimination between adjacent timesteps, while the slowest frequency (period $\gg N_{\text{trans}}$) provides a near-linear encoding of absolute position. For the cell axis with $n = 20$ positions, the frequency range is similarly appropriate — the fastest frequency discriminates adjacent cells and the slowest encodes absolute cell position within the grid.

WHAT TRIPLETPE2D ENABLES

After adding TripletPE2D, two tokens that share the same timestep t but differ in cell index i have identical time-axis encodings but distinct cell-axis encodings. The dot product between their positional encodings is:

$$\text{PE}(t, i) \cdot \text{PE}(t, j) = \|\mathbf{P}_{\text{time}}[t]\|^2 + \mathbf{P}_{\text{cell}}[i] \cdot \mathbf{P}_{\text{cell}}[j] \tag{62}$$

The first term is a constant (shared time component); the second term varies with $|i - j|$ — tokens at nearby cells have higher dot products. This allows the attention mechanism to learn that tokens at the same timestep are related (they jointly describe the spatial pattern at time t) even before any token content is considered. Similarly, two tokens that share the same cell index i but differ in timestep t have identical cell-axis encodings but distinct time-axis encodings, allowing the attention to learn the temporal trajectory of individual cells.

TRIPLETPE2D INSTANTIATION FOR EACH MODEL

Table 24: TripletPE2D instantiation for αM and skewM. The buffer shape is $[1, N_{\text{tok}}, 128]$ in both cases. TripletPE2D has no learnable parameters — only a registered buffer.

Model	`n_trans`	`n_cells`	`d_model`	Buffer size
αM	99	20	128	$[1, 1980, 128]$
skewM	199	20	128	$[1, 3980, 128]$

tempM and stocM use the standard `SinusoidalPE` (1D) at both levels of their hierarchical encoder, rather than TripletPE2D. The CellEncoder operates on 50-cell rows independently — within a single row, only the cell index is meaningful (there is no time axis at the cell level), so a 1D cell-position encoding is correct. The TimestepEncoder operates on 199-timestep sequences of row summaries — here only the timestep index is meaningful, so a 1D timestep encoding is correct. TripletPE2D is appropriate specifically for αM and skewM, where the flat token sequence spans both time and space simultaneously.

E.4 PATTERNEMBEDDING

PatternEmbedding is used in tempM and stocM but not in αM or skewM. It maps each token's neighbourhood pattern index $j \in \{0, \ldots, 7\}$ to a learned 8-dimensional continuous vector, which is concatenated to the raw 4-dimensional token before the input projection.

MOTIVATION

The neighbourhood pattern index $j_{t,i} = 4a_{i-1}^t + 2a_i^t + a_{i+1}^t$ encodes which of the 8 possible input patterns a given token corresponds to. This index is a deterministic function of the raw token features (L, C, R) and is therefore implicitly present in the 4-dimensional base token — the model can in principle compute j from the raw bits via learned weights. However, for tempM and stocM the central analytical quantities (frac_stats, within_var_stats, the correction MLP, the DirectRuleDecoder) are all indexed by pattern: frac$[j]$, within_var$[j]$, $R_1[j]$, $R_2[j]$, and so on. Providing the pattern index as an explicit learned embedding gives the transformer encoder a direct shortcut to identify which analytical entry corresponds to each token, without requiring the model to re-derive j from the raw bits through a sequence of learned multiplications.

For αM and skewM the analytical path does not exist (there is no frac_stats or within_var_stats entering the transformer), so the shortcut provides no advantage and the overhead of 64 additional parameters is avoided by omitting PatternEmbedding from those two models.

PATTERN INDEX DEFINITION

The neighbourhood index is the standard ECA binary encoding:

$$j_{t,i} = 4\,a_{i-1}^t + 2\,a_i^t + a_{i+1}^t \in \{0, 1, \ldots, 7\} \tag{63}$$

The eight patterns and their indices are:

j	$(a_{i-1}^t,\, a_i^t,\, a_{i+1}^t)$
0	$(0, 0, 0)$
1	$(0, 0, 1)$
2	$(0, 1, 0)$
3	$(0, 1, 1)$
4	$(1, 0, 0)$
5	$(1, 0, 1)$
6	$(1, 1, 0)$
7	$(1, 1, 1)$

The rule output for pattern j under rule R is the j-th bit of R: $R[j] = (R \gg j)\ \&\ 1$, so under the synchronous rule the expected next-state $a_i^{t+1} = R[j_{t,i}]$. A disagreement pattern is one where $R_1[j] \neq R_2[j]$ — the two rules predict different outputs for that neighbourhood. The per-pattern statistics in frac_stats and within_var_stats are indexed by j exactly as defined in Equation 63.

ARCHITECTURE

PatternEmbedding is a standard `nn.Embedding` lookup table:

$$\mathbf{E} \in \mathbb{R}^{8\times 8} \quad \text{(8 patterns, 8-dimensional embedding)}$$
$$\texttt{PatternEmbedding}(j) = \mathbf{E}[j] \quad \in \mathbb{R}^8$$

The implementation:

```
class PatternEmbedding(nn.Module):
    def __init__(self, n_pat=8, emb_dim=8):
        super().__init__()
        self.emb = nn.Embedding(n_pat, emb_dim)
        nn.init.normal_(self.emb.weight, std=0.1)

    def forward(self, pat_idx):   # pat_idx: [B, N_CELLS] int64
        return self.emb(pat_idx)  # returns: [B, N_CELLS, 8] float32
```

Total parameters: $8 \times 8 = 64$**.**

TOKEN AUGMENTATION IN THE FORWARD PASS

The pattern index is computed on-the-fly in `_embed_tokens` inside the main model's forward pass:

```
def _embed_tokens(self, raw_tokens):
    # raw_tokens: [B_flat, N_CELLS, 4]   (L, C, R, A)
    l = raw_tokens[:, :, 0]   # left  neighbour
    b = raw_tokens[:, :, 1]   # centre cell
    r = raw_tokens[:, :, 2]   # right neighbour
    pat_idx = (4 * l + 2 * b + r).long().clamp(0, 7)
    emb     = self.pat_emb(pat_idx)          # [B_flat, N_CELLS, 8]
    return torch.cat([raw_tokens, emb], dim=-1)  # [B_flat, N_CELLS, 12]
```

The `.clamp(0, 7)` guard is applied after the index computation because the raw token features are stored as float32 and a floating-point rounding error (e.g., $1.0000001 \times 4 = 4.0000004$, which rounds to 4 as expected but could in principle produce an out-of-range index on pathological hardware). In practice the clamp never fires, as the token values are exactly 0.0 or 1.0.

The augmented 12-dimensional token $(\underbrace{L, C, R, A}_{4}, \underbrace{\mathbf{E}[j]}_{8})$ is the input to the CellEncoder's Linear$(12 \to 128)$ input projection.

INITIALISATION

The embedding weight matrix $\mathbf{E}$ is initialised from $\mathcal{N}(0, 0.1)$, wider than the global $\mathcal{N}(0, 0.02)$ used for linear layers:

```
nn.init.normal_(self.emb.weight, std=0.1)
```

The reason for the wider initialisation is to spread the 8 pattern representations apart from the start of training. With $\mathcal{N}(0, 0.02)$, the initial embeddings would have typical pairwise distances of approximately $0.02\sqrt{8} \approx 0.057$, which is very small relative to the typical activation magnitudes in the CellEncoder. The model would then need many gradient steps to differentiate pattern 0 from pattern 7, losing early training efficiency. With $\mathcal{N}(0, 0.1)$, the typical initial pairwise distance is $0.1\sqrt{8} \approx 0.283$, which is meaningful relative to early CellEncoder activations, providing immediately useful pattern-discriminative signal from the first training step.

This initialisation is applied *after* `_init_weights` is called (which would override it with 0.02 if not excluded):

```
def _init_weights(self):
    for name, m in self.named_modules():
        if isinstance(m, nn.Linear):
            # PatternEmbedding is nn.Embedding, not nn.Linear
            # so it is never touched by this loop
            nn.init.normal_(m.weight, std=0.02)
            ...
```

Since `PatternEmbedding` wraps an `nn.Embedding` (not `nn.Linear`), it is never touched by `_init_weights`'s loop over `nn.Linear` modules, and the explicit `nn.init.normal_(std=0.1)` in `PatternEmbedding.__init__` is the only initialisation applied to $\mathbf{E}$.

WHY 8 EMBEDDING DIMENSIONS

The embedding dimension matches the number of patterns ($N_{\text{EMB}} = N_{\text{PAT}} = 8$), which is a natural choice: the embedding space has one dimension per pattern, allowing the model to represent each pattern as a standard basis vector if it chooses, or to use the full 8-dimensional space to encode relationships between patterns (e.g., placing patterns that agree between R_1 and R_2 close together and patterns that disagree far apart). A higher-dimensional embedding (e.g., 16 or 32) would allow more expressive representations but add negligible value given that only 8 distinct inputs exist. A lower-dimensional embedding (e.g., 4) would force the model to compress 8 distinct identities into 4 dimensions, potentially creating confusion between patterns that the model should distinguish.

F LLM EVALUATION DETAILS

This appendix provides the complete specification of the LLM evaluation pipeline. There are two entirely separate evaluation systems, each with different infrastructure.

Identification evaluation (zero-shot, few-shot, fine-tuned). Evaluates on the identification datasets (alphaECA, skewECA, tempECA, stocECA). Models are loaded *locally* on an H100 NVL node (95,830 MiB) from checkpoints downloaded to `/home/models/` and run via HuggingFace `transformers`. No external API is used. Generation is deterministic: `do_sample=False`, `temperature=1.0` (ignored under greedy decoding), `max_new_tokens=64`. 70B and 72B models are loaded in 8-bit quantisation (`load_in_8bit=True`) to fit within VRAM. All three evaluation modes (zero-shot, few-shot, fine-tuned) use a shared `fixed_test_indices.npy` file so results are compared on exactly the same test samples across modes. These settings define the main protocol; Appendix F.9 reports an additional evaluation with the complete two-rule input and unrestricted reasoning.

Knowledge-check evaluation. Evaluates on the four knowledge-check JSON datasets. Test models are queried via the OpenRouter API (`https://openrouter.ai/api/v1`). Short-answer responses are scored by an xAI Grok-3 judge queried via the native xAI API (`https://api.x.ai/v1`) — a model from a different family than any test model. Generation temperature is set to 0.0 for both test and judge calls. A disk-based `shelve` cache persists responses so interrupted runs resume without re-spending API budget.

F.1 ZERO-SHOT PROMPT TEMPLATES

Every zero-shot query consists of a system prompt and a user prompt sent as a two-message conversation. The system prompt is passed as the `system` role message; the user prompt as the `user` role message. The model generates the `assistant` response. Both messages are assembled into a single string via `tok.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)`. If `apply_chat_template` raises, a plain-text fallback is used:

```
System: <SYSTEM_PROMPT>
User:   <USER_PROMPT>
Assistant:
```

Responses are parsed by finding the first `{` and last `}` in the output and calling `json.loads`. On failure, field-specific regex is applied: `"rule":\s*(\d+)` for rule numbers, `"alpha":\s*([\d.]+)` for floats. Extracted values are range-checked (rule $\in [0, 255]$, noise $\in$ valid domain) and snapped to the nearest grid value (α to $\{0.1, \ldots, 1.0\}$; s to $\{1, \ldots, 20\}$; τ/λ to two decimal places). Total failures are recorded as `PARSE_FAILED`, excluded from accuracy, and counted in the parse failure rate.

ALPHAECA: α-ASYNCHRONOUS NOISE

The following is the exact `SYSTEM_PROMPT` string:

System prompt — alphaECA zero-shot

You are an expert in cellular automata. You are given a space-time orbit of an Elementary Cellular Automaton (ECA) perturbed by alpha-asynchronous noise.
In alpha-asynchronous noise each cell independently updates with probability alpha per timestep. Alpha is one of: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
The orbit is a grid of 0s and 1s. Each row is one timestep.
Identify: 1. The ECA rule number (integer 0-255) 2. The alpha value
Respond ONLY in this JSON format with no extra text: "rule": <integer>, "alpha": <float>

The user prompt is generated by `make_user_prompt(orbit)`:

User prompt — alphaECA zero-shot

Space-time orbit (100 rows x 20 cells):
<ORBIT_ROWS>
What is the ECA rule number and alpha value?

where `<ORBIT_ROWS>` is replaced by the full $t \times n = 100 \times 20$ orbit displayed as 100 rows of 20 `0/1` characters separated by newlines, with no spaces within a row. The orbit is generated by `orbit_to_str(orbit)`:

```
def orbit_to_str(orbit):
    return "\n".join("".join(str(int(c)) for c in row) for row in orbit)
```

SKEWECA: s-SKEWED NOISE

The following is the exact `SYSTEM_PROMPT` string:

System prompt — skewECA zero-shot

You are an expert in cellular automata. You are given a space-time orbit of an Elementary Cellular Automaton (ECA) perturbed by s-skewed noise.
In s-skewed noise, at each timestep exactly s contiguous cells are chosen at random and updated according to the ECA rule. All other cells retain their current state. The parameter s ranges from 1 to 20 (the full grid width), where s=20 means all cells update (fully synchronous).
The orbit is a grid of 0s and 1s. Each row is one timestep. Grid width is 20 cells.
Identify: 1. The ECA rule number (integer 0-255) 2. The value of s (integer from 1 to 20)
Respond ONLY in this JSON format with no extra text: "rule": <integer>, "s": <integer>

The user prompt is generated by `make_user_prompt(orbit)`:

User prompt — skewECA zero-shot

Space-time orbit (200 rows x 20 cells):
<ORBIT_ROWS>
What is the ECA rule number and s value?

where `<ORBIT_ROWS>` is replaced by the full $t \times n = 200 \times 20$ orbit displayed as 200 rows of 20 `0/1` characters separated by newlines, with no spaces within a row. The s parameter is accepted within a tolerance of ± 1 cell (i.e. $|s_{\text{pred}} - s_{\text{true}}| \leq 1$).

STOCECA: STOCHASTIC CELL-LEVEL NOISE

The following is the exact `SYSTEM_PROMPT` string:

System prompt — stocECA zero-shot

You are an expert in cellular automata. You are given a space-time orbit of an Elementary Cellular Automaton (ECA) under stochastic noise.
In this stochastic model, at each timestep each cell independently chooses one of two rules. With probability lambda it uses rule R_2, otherwise it uses rule R_1. The parameter lambda is a continuous value between 0.1 and 0.9. The system (R_1, R_2, lambda) is statistically identical to (R_2, R_1, 1-lambda), so you should predict the rule with the smaller number as rule_R_1 and the larger as rule_R_2.
The orbit is a grid of 0s and 1s. Each row is one timestep. Grid width is 50 cells.
Identify: 1. Rule F (the first ECA rule, integer 0-255) 2. Rule G (the second ECA rule, integer 0-255, different from F) 3. Lambda value (float between 0.1 and 0.9, rounded to 2 decimal places)
Respond ONLY in this JSON format with no extra text: "rule_R_1": <integer>, "rule_R_2": <integer>, "lambda": <float>

The user prompt is generated by `make_user_prompt(orbit)`:

User prompt — stocECA zero-shot

Space-time orbit (showing 50 of 200 rows x 50 cells):
<ORBIT_ROWS>
What are rule R_1, rule R_2, and lambda?

where `<ORBIT_ROWS>` is the first 50 rows of the first orbit ($\ell = 0$ of $k = 8$) displayed as 50 rows of 50 `0/1` characters separated by newlines. The orbit is generated via `orbit_to_str(orbit[:50])`. This truncated display is the main protocol for the two-rule datasets; Appendix F.9 repeats the evaluation with the complete input (all 200 rows of all 8 orbits) under otherwise matched settings and obtains the same result. Rule identification uses unordered-pair accuracy: the predicted pair $\{R_1, R_2\}$ is correct regardless of which rule is assigned to `rule_`R_1 or `rule_`R_2. The λ parameter is accepted within a symmetric tolerance of ± 0.05, computed as $\min(|\hat{\lambda} - \lambda|, |\hat{\lambda} - (1 - \lambda)|)$ to account for the $(R_1, R_2, \lambda) \equiv (R_2, R_1, 1{-}\lambda)$ symmetry.

TEMPECA: TEMPORALLY STOCHASTIC NOISE

The following is the exact `SYSTEM_PROMPT` string:

System prompt — tempECA zero-shot

You are an expert in cellular automata. You are given a space-time orbit of an Elementary Cellular Automaton (ECA) under temporally stochastic noise.
In this temporally stochastic model, at each timestep the ENTIRE row is updated by one of two rules. With probability tau the row uses rule G; otherwise the entire row uses rule F. This is a row-level coin flip – all cells in a row use the same rule at each timestep.
The parameter tau is a continuous value between 0.1 and 0.9. The system (R_1, R_2, tau) is statistically identical to (R_2, R_1, 1-tau), so predict the smaller rule number as rule_R_1 and the larger as rule_R_2.
The orbit is a grid of 0s and 1s. Each row is one timestep. Grid width is 50 cells.
Identify: 1. Rule F (the first ECA rule, integer 0-255) 2. Rule G (the second ECA rule, integer 0-255, different from F) 3. Tau value (float between 0.1 and 0.9, rounded to 2 decimal places)
Respond ONLY in this JSON format with no extra text: "rule_R_1": <integer>, "rule_R_2": <integer>, "tau": <float>

The user prompt is generated by `make_user_prompt(orbit)`:

User prompt — tempECA zero-shot

Space-time orbit (showing 50 of 200 rows x 50 cells):
<ORBIT_ROWS>
What are rule R_1, rule R_2, and tau?

where `<ORBIT_ROWS>` is the first 50 rows of the first orbit ($\ell = 0$ of $k = 8$) displayed as 50 rows of 50 `0/1` characters, identical in construction to the stocECA user prompt; the complete-input evaluation is reported in Appendix F.9. Rule identification and τ tolerance follow the same conventions as stocECA: unordered-pair accuracy and symmetric ± 0.05 tolerance on τ.

KNOWLEDGE-CHECK PROMPTS

The knowledge-check evaluation uses the OpenRouter API. The shared system prompt is:

System prompt — knowledge-check (all datasets)

You are a precise exam-taking assistant. Follow every instruction exactly. Never add explanations unless explicitly asked.

MCQ user prompt:

User prompt — MCQ

Answer with EXACTLY ONE LETTER: A, B, C, or D. Do NOT write any words, punctuation, or explanation. Output the single letter and nothing else.
Question: <QUESTION_TEXT>
Choices: A. <CHOICE_A> B. <CHOICE_B> C. <CHOICE_C> D. <CHOICE_D>
Your answer (single letter):

Short-answer user prompt:

User prompt — Short Answer

<QUESTION_TEXT>
Answer concisely and factually in 1-3 sentences:

Short-answer responses are scored by an xAI Grok-3 judge using:

Judge prompt — short-answer scoring

You are a strict grading assistant. Score the student answer below using ONLY this rubric: 2 = fully correct 1 = partially correct (right idea, missing key detail) 0 = incorrect or no answer Output ONLY a single digit (0, 1, or 2). No words, no punctuation, no explanation.
Question: <QUESTION> Reference answer: <REFERENCE> Student answer: <PREDICTION>
Score:

MCQ scores are exact match (0 or 1). Short-answer scores (0/1/2) are normalised to $[0, 0.5, 1.0]$ by dividing by 2. Overall accuracy = mean of all normalised scores (MCQ + SA). Parse failures (no A/B/C/D extractable from MCQ output) are flagged via a `PARSE_FAIL_TOKEN` sentinel, excluded from accuracy, and counted separately per model. The MCQ parser attempts, in order: exact single letter, "Final Answer: A"-style patterns, "Answer: A" patterns, standalone "(A)", "Option A"/"Choice A", and finally exact "A." or "A)". No last-letter fallback — ambiguous outputs are parse failures.

ORBIT DISPLAY FORMAT

For all four identification datasets, orbits are displayed as plain text: each row is a single string of `0/1` characters with no spaces, rows separated by newlines. The full orbit is displayed for alphaECA

($t = 100$, $n = 20$) and skewECA ($t = 200$, $n = 20$) since these fit comfortably within context. For tempECA and stocECA (grid width $n = 50$, $t = 200$, $k = 8$), the main protocol displays the first 50 rows of the first orbit ($\ell = 0$), which keeps the prompt within the 32k-token native context of the smaller open-source models. To ensure that conclusions do not depend on this truncation, Appendix F.9 repeats the zero-shot and fine-tuning evaluations with the complete input used by stocM and tempM; rule-pair accuracy remains 0.00% zero-shot and at most 6.5% after fine-tuning.

F.2 Few-Shot Prompt Templates

The few-shot evaluation tests $k_{\text{shot}} \in \{1, 3, 5\}$ in-context examples. The system prompt is identical to the zero-shot system prompt for each noise type.

Chat template structure

$$\underbrace{[\texttt{system}]}_{\text{domain prompt}}\ \underbrace{[\texttt{user}][\texttt{assistant}]}_{\text{example 1}}\ \cdots\ \underbrace{[\texttt{user}][\texttt{assistant}]}_{\text{example } k_{\text{shot}}}\ \underbrace{[\texttt{user}]}_{\text{test question}}$$

The message list is assembled by `build_messages(shots, orbit)`:

```
def build_messages(shots, orbit):
    messages = [{"role": "system", "content": SYSTEM_PROMPT}]
    for s in shots:
        messages.append({"role": "user",
                         "content": make_user_prompt(s["orbit"])})
        messages.append({"role": "assistant",
                         "content": json.dumps({"rule": s["rule"],
                                                "alpha": s["alpha"]})})
    messages.append({"role": "user",
                     "content": make_user_prompt(orbit)})
    return messages
```

Applied to skewECA, `json.dumps` produces `{"rule": r, "s": s}`. Applied to tempECA/stocECA, `json.dumps` produces `{"rule_f": f, "rule_g": g, "tau": tau}` or `{"rule_f": f, "rule_g": g, "lambda": lam}` respectively. The list is converted via `tok.apply_chat_template` with `add_generation_prompt=True`. On failure, the same plain-text fallback as zero-shot is used, with `Assistant: <answer_json>` for each example turn and a bare `Assistant:` after the final user turn.

Few-shot example source

Few-shot examples for alphaECA and skewECA are drawn from the **Phase 2 training split** (179 training rules, disjoint from the 77 test rules) using **real simulated orbits** from the training data. For stocECA and tempECA, few-shot examples are drawn from `train_pairs.npy` (rule pairs from the 179 training rules); since no pre-simulated training orbits are stored on disk for these two datasets, **placeholder zero orbits** ($t \times n$ arrays of zeros) are used in place of real orbits for the in-context shots. In all cases, example indices are selected once per (k_{shot}, noise type) with a fixed seed:

```
rng      = numpy.random.default_rng(SEED + 99)   # SEED=42, so seed=141
shot_idx = rng.choice(len(train_orbits),
                      size=k_shots, replace=False)
```

The same k_{shot} examples are shown to every model and for every test orbit.

Exact shot parameters

alphaECA.

- $k_{\text{shot}} = 1$: rule=103, $\alpha = 1.0$
- $k_{\text{shot}} = 3$: rule=103 $\alpha = 1.0$; rule=186 $\alpha = 0.8$; rule=192 $\alpha = 0.6$
- $k_{\text{shot}} = 5$: rule=186 $\alpha = 0.8$; rule=255 $\alpha = 0.6$; rule=214 $\alpha = 0.1$; rule=192 $\alpha = 0.6$; rule=103 $\alpha = 1.0$

skewECA.

- $k_{\text{shot}} = 1$: rule=103, $s = 20$
- $k_{\text{shot}} = 3$: rule=103 $s = 20$; rule=186 $s = 16$; rule=192 $s = 12$
- $k_{\text{shot}} = 5$: rule=186 $s = 16$; rule=255 $s = 11$; rule=214 $s = 1$; rule=192 $s = 12$; rule=103 $s = 20$

stocECA.

- $k_{\text{shot}} = 1$: $R_1 = 199$, $R_2 = 86$, $\lambda = 0.9$
- $k_{\text{shot}} = 3$: $R_1 = 121$ $R_2 = 177$ $\lambda = 0.24$; $R_1 = 51$ $R_2 = 66$ $\lambda = 0.53$; $R_1 = 191$ $R_2 = 152$ $\lambda = 0.58$
- $k_{\text{shot}} = 5$: $R_1 = 107$ $R_2 = 28$ $\lambda = 0.58$; $R_1 = 215$ $R_2 = 104$ $\lambda = 0.41$; $R_1 = 187$ $R_2 = 0$ $\lambda = 0.34$; $R_1 = 185$ $R_2 = 208$ $\lambda = 0.53$; $R_1 = 10$ $R_2 = 5$ $\lambda = 0.72$

tempECA.

- $k_{\text{shot}} = 1$: $R_1 = 199$, $R_2 = 86$, $\tau = 0.9$
- $k_{\text{shot}} = 3$: $R_1 = 121$ $R_2 = 177$ $\tau = 0.24$; $R_1 = 51$ $R_2 = 66$ $\tau = 0.53$; $R_1 = 191$ $R_2 = 152$ $\tau = 0.58$
- $k_{\text{shot}} = 5$: $R_1 = 107$ $R_2 = 28$ $\tau = 0.58$; $R_1 = 215$ $R_2 = 104$ $\tau = 0.41$; $R_1 = 187$ $R_2 = 0$ $\tau = 0.34$; $R_1 = 185$ $R_2 = 208$ $\tau = 0.53$; $R_1 = 10$ $R_2 = 5$ $\tau = 0.72$

RESUME AND CHECKPOINT SUPPORT

The few-shot script saves a JSON checkpoint every 10 samples. Re-running skips samples already in `done_set` (loaded from the checkpoint). The test indices are loaded from the shared `fixed_test_indices.npy`. Checkpoint files are named `<model_name>_<k_shot>shot.json` under `results/few_shot/`.

F.3 LORA FINE-TUNING CONFIGURATION

Fine-tuning uses HuggingFace PEFT Low-Rank Adaptation (LoRA). Only adapter weights are trained; base model weights are frozen. The identical configuration is applied to all six models and all four noise types.

LORA ADAPTER CONFIGURATION

```
LORA_CONFIG = LoraConfig(
    task_type=TaskType.CAUSAL_LM,
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
                    "gate_proj", "up_proj", "down_proj"],
)
```

Rank $r = 16$. Each targeted weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ is updated by $\Delta\mathbf{W} = \mathbf{BA}$ where $\mathbf{B} \in \mathbb{R}^{m \times 16}$ and $\mathbf{A} \in \mathbb{R}^{16 \times n}$. $r = 16$ is large enough to learn the JSON output format and basic noise model associations without overfitting on 5,000 training samples.

Scale factor $\alpha/r = 2.0$. LoRA $\alpha = 32$ with rank 16 gives a scale factor of 2, amplifying the adapter updates relative to the raw decomposition. This accelerates format learning without requiring a higher learning rate.

Target modules. LoRA adapters are applied to all seven projection modules per layer: attention (`q, k, v, o_proj`) and feed-forward (`gate, up, down_proj`). This covers the standard set used in instruction fine-tuning and matches the architecture of Llama, Qwen, and Mistral family models.

Trainable parameter count. For a 7B model with 32 layers: params $\approx 7 \times 2 \times d \times r \times L$ where $d = 4096$, $r = 16$, $L = 32$. This gives approximately 29M trainable parameters ($\approx 0.4\%$ of base parameters); the feed-forward projections are wider than d, so the exact count is somewhat higher.

TRAINING CONFIGURATION (EXACT FROM `FINETUNE_EVAL.PY`)

```
TrainingArguments(
    num_train_epochs=3,
    per_device_train_batch_size=batch_size,  # 2 for 7B; 1 for 70B
    gradient_accumulation_steps=max(1, 8 // batch_size),
    learning_rate=2e-4,
    lr_scheduler_type="cosine",
    warmup_ratio=0.05,
    weight_decay=0.01,
    fp16=not is_large(model_name),   # float16 for 7B/8B/MoE
    bf16=is_large(model_name),       # bfloat16 + 8-bit for 70B/72B
    gradient_checkpointing=True,
    save_strategy="epoch",
    save_total_limit=2,
    report_to="none",
    remove_unused_columns=False,
)
```

Training data. $N_{\text{train}} = 5{,}000$ samples from the Phase 2 training split (179 training rules only, seed $\texttt{SEED} + 1 = 43$):

```
rng = numpy.random.default_rng(SEED + 1)   # seed = 43
idx = rng.choice(len(train_orbits), size=n_train, replace=False)
```

No test rules appear in the training data.

Effective batch size = 8. Physical batch size 2 with gradient accumulation 4 for 7B/8B/MoE models. Physical batch size 1 with gradient accumulation 8 for 70B/72B models. `gradient_accumulation_steps = max(1, 8 // batch_size)` computes this automatically.

Total gradient steps. $\lceil 5000/8 \rceil \times 3 = 1{,}875$ steps. Warmup: $\lfloor 1875 \times 0.05 \rfloor = 93$ steps.

Large model quantisation. 70B and 72B models use 8-bit quantisation (`BitsAndBytesConfig(load_in_8bit=True)`) during both training and evaluation. This fits Llama-3.1-70B and Qwen2.5-72B within 94 GB VRAM. Small models: float16 on 2×H100 with automatic device mapping.

Label masking. Prompt tokens are masked with -100 so the loss is computed only over the assistant response (the JSON output):

```
full_enc   = tok(full_text,   truncation=True, max_length=MAX_LEN, ...)
prompt_enc = tok(prompt_text, truncation=True, max_length=MAX_LEN, ...)
# MAX_LEN = 2048 (alphaECA, skewECA), 4096 (stocECA, tempECA)
labels     = input_ids.clone()
labels[:prompt_enc["input_ids"].shape[1]] = -100
```

This is standard SFT: the model learns the output conditional on the orbit, not to reproduce the orbit display. The limits are chosen to exceed the measured prompt lengths: alphaECA prompts use $\approx 1{,}926$ tokens and stocECA/tempECA prompts $2{,}400$–$2{,}700$ tokens, so no training input is

truncated. For stocECA and tempECA, these fine-tuning inputs use the truncated display of the main protocol (first 50 rows of orbit $\ell = 0$); fine-tuning on the complete input (90,000-token limit) is reported in Appendix F.9. Training losses decrease smoothly for every model and dataset (e.g. from 0.95 to 0.21 for Qwen2.5-72B on αECA), so the low identification accuracy of the fine-tuned models reflects the task, not a failed optimisation. That a fine-tuned 72B model (14.6%) trails a trained 113K-parameter CNN (71.7%) is itself informative: the CNN operates on the orbit as a 2D grid whose convolutions align cells with their neighbours and successors, whereas the LLM must recover the same per-pattern statistic from a flattened digit sequence – the representational obstacle quantified in Appendix F.10.

Resume support. Training resumes from the latest HuggingFace Trainer checkpoint in `<ckpt_dir>/` if one exists. The adapter is saved to `<ckpt_dir>/lora_adapter/` after training completes. Evaluation resumes from a per-sample checkpoint JSON with the same skip-done-set mechanism as zero-shot and few-shot.

ADAPTER INFERENCE

```
base  = AutoModelForCausalLM.from_pretrained(base_path, ...)
model = PeftModel.from_pretrained(base, adapter_path)
model.eval()
```

The adapter is *not* merged before inference. `PeftModel` applies the LoRA update $\Delta\mathbf{W}$ dynamically, adding negligible latency for evaluation. Evaluation uses the same `run_inference` function as zero-shot: `do_sample=False`, `max_new_tokens=64`.

Table 25: LoRA fine-tuning configuration summary.

Hyperparameter	Value
LoRA rank r	16
LoRA α	32
Scale factor α/r	2.0
LoRA dropout	0.05
Bias	none
Target modules	q, k, v, o, gate, up, down (7 per layer)
Training samples N_{train}	5,000 (Phase-2 training split, seed 43)
Epochs	3
Effective batch size	8
Physical batch / grad accum (7B)	2 / 4
Physical batch / grad accum (70B)	1 / 8
Total gradient steps	1,875
Warmup steps	93
Learning rate	2×10^{-4} (cosine)
Weight decay	0.01
Precision (7B/8B/MoE)	float16
Precision (70B/72B)	bfloat16 + 8-bit quantisation
Max sequence length	2,048 (alpha, skew) / 4,096 (stoc, temp)
Label masking	Prompt tokens $\rightarrow -100$
Gradient checkpointing	Enabled

F.4 PER-NOISE-TYPE LLM RESULTS

Tables 26–29 present the complete LLM evaluation results for all six open–source models across all three evaluation modes on each of the four identification datasets. All zero-shot and k-shot results are from completed SLURM runs on H100 NVL (95,830 MiB). alphaECA and skewECA used $N = 500$ test samples. stocECA and tempECA used $N = 200$ test samples.

Metric definitions. Rule%: exact match on all 8 bits (single rule for alphaECA/skewECA; unordered pair for stocECA/tempECA). Noise%: α acc ± 0.05 (alphaECA); s acc ± 1 (skewECA); λ/τ acc ± 0.05 symmetric (stocECA/tempECA). PF%: parse failure rate (excluded from accuracy).

Table 26: alphaECA ($N = 500$): rule% and α noise% for all 6 models. All values exact from logs.

	Zero-shot			Few-shot (k=5)			Fine-tuned		
Model	Rule%	α%	PF%	Rule%	α%	PF%	Rule%	α%	PF%
Qwen2.5-7B	1.00	11.60	0.0	0.20	11.40	0.4	8.2	18.4	<1
Qwen2.5-72B	1.00	12.20	0.0	0.60	16.80	0.2	14.6	22.8	<1
Llama-3.1-8B	0.00	11.60	0.0	0.00	12.00	0.0	5.4	14.2	<1
Llama-3.1-70B	0.40	10.00	0.0	0.00	0.00	0.0	11.2	18.6	<1
Mistral-7B	0.00	0.00	**100**	0.00	9.80	0.0	4.8	12.6	<1
Mixtral-8x7B	0.00	0.00	**100**	0.00	11.20	0.0	9.6	16.4	<1
αM (ours)	**99.22**	**95.08**	0.0	—	—	—	—	—	—

Table 27: skewECA ($N = 500$): rule% and $s \pm 1$ noise%. All values exact from logs. Note: Mistral-7B achieves 15.8% s% zero-shot with 0.00% rule — a striking dissociation explained in the analysis below.

	Zero-shot			Few-shot (k=5)			Fine-tuned		
Model	Rule%	$s \pm 1$%	PF%	Rule%	$s \pm 1$%	PF%	Rule%	$s \pm 1$%	PF%
Qwen2.5-7B	0.00	13.00	0.0	0.80	13.60	0.0	7.4	20.2	<1
Qwen2.5-72B	0.80	15.20	0.0	0.60	13.20	0.0	13.2	24.8	<1
Llama-3.1-8B	0.00	15.60	0.0	0.00	11.60	0.0	4.8	17.4	<1
Llama-3.1-70B	0.00	14.80	0.0	0.00	6.00	0.0	9.8	18.2	<1
Mistral-7B	0.00	**15.80**	0.0	0.00	19.20	0.0	4.2	18.8	<1
Mixtral-8x7B	0.00	0.20	≈98	0.00	3.80	≈40	8.8	16.4	<1
skewM (ours)	**94.55**	**96.97**	0.0	—	—	—	—	—	—

Universal rule-pair identification failure on stocECA and tempECA. Every model achieves 0.00% both-exact UO rule accuracy on stocECA and tempECA under zero-shot and few-shot evaluation, confirmed over 200 test samples per model and mode; fine-tuning raises it only to 1.5–6.0%. The result is unchanged when the complete input is shown (Appendix F.9). The two-rule identification task requires simultaneously identifying two rules from a mixed orbit, where the unordered pair must match exactly. The random-chance baseline for this task is $1/\binom{256}{2} \approx 0.003\%$, but even this is not achieved. The models consistently produce outputs that either repeat the same rule for both R_1 and R_2 (a trivially invalid pair), or produce rule numbers outside the valid range, or fail format entirely. Even after format correction via few-shot prompting, the zero does not lift — confirming the failure is not format-related but reflects a fundamental inability to reason about two-rule mixing from a visual orbit.

Rule accuracy on alphaECA and skewECA: near-zero, not zero. For the single-rule datasets, a small number of correct predictions occur: Qwen2.5-72B achieves 1.00% on alphaECA and 0.80% on skewECA (zero-shot), and Qwen2.5-7B achieves 0.80% on skewECA (k=5). These correct predictions are not random chance (the random baseline for exact 8-bit rule identification is $1/256 = 0.39\%$), suggesting the models occasionally recover a correct rule from the visual pattern. However, the rate does not scale with model size or evaluation mode — Qwen2.5-72B zero-shot achieves 0.80% on skewECA, identical to Qwen2.5-7B at $k = 5$ shot — confirming that the occasional correct prediction reflects noise rather than systematic inference.

Noise accuracy is near-random across all datasets. The noise tolerance accuracies reported across all models reflect near-random performance. For alphaECA (α on a 10-class grid), random chance gives ≈10%; models score 10–17%. For skewECA ($s \in \{1, ..., 20\}$ with ± 1 tolerance), random chance with uniform s prediction and the ± 1 window gives $58/400 = 14.5\%$; models score 6–19%, consistent with near-uniform s prediction. For stocECA/tempECA, the symmetric tolerance $\min(|\hat{\theta} - \theta|, |\hat{\theta} - (1 - \theta)|) \leq 0.05$ folds $\theta \in [0.1, 0.9]$ onto $[0.1, 0.5]$, so a constant or random prediction scores approximately $0.1/0.4 = 25\%$; models score 1–28%, at or below this level. The models do not demonstrate systematic noise parameter estimation.

Table 28: stocECA ($N = 200$): both-exact UO rule% and $\lambda \pm 0.05$ noise%. All values exact from logs. Every model achieves 0.00% rule under zero-shot and few-shot evaluation; fine-tuning raises it only marginally.

	Zero-shot			Few-shot (k=5)			Fine-tuned		
Model	Both%	λ%	PF%	Both%	λ%	PF%	Both%	λ%	PF%
Qwen2.5-7B	0.00	19.50	0.0	0.00	24.00	0.0	3.0	28.5	<1
Qwen2.5-72B	0.00	25.50	0.0	0.00	27.50	0.0	6.0	32.5	<1
Llama-3.1-8B	0.00	25.50	0.0	0.00	24.00	0.0	2.0	24.5	<1
Llama-3.1-70B	0.00	24.50	0.0	0.00	27.50	0.0	5.0	29.5	<1
Mistral-7B	0.00	2.00	≈92	0.00	16.00	0.0	2.0	23.0	<1
Mixtral-8x7B	0.00	0.00	**100**	0.00	1.00	≈95	4.5	24.5	<1
stocM (ours)	**95.49**	**84.80**	0.0	—	—	—	—	—	—

Table 29: tempECA ($N = 200$): both-exact UO rule% and $\tau \pm 0.05$ noise%. All values exact from logs. Every model achieves 0.00% rule under zero-shot and few-shot evaluation; fine-tuning raises it only marginally. Mixtral-8x7B achieves 0.00% noise on $k = 5$ shot — 100% PF.

	Zero-shot			Few-shot (k=5)			Fine-tuned		
Model	Both%	τ%	PF%	Both%	τ%	PF%	Both%	τ%	PF%
Qwen2.5-7B	0.00	24.00	0.0	0.00	27.50	0.0	2.5	29.0	<1
Qwen2.5-72B	0.00	18.00	0.0	0.00	24.00	0.0	5.5	30.0	<1
Llama-3.1-8B	0.00	27.50	0.0	0.00	26.00	0.0	2.0	24.0	<1
Llama-3.1-70B	0.00	23.00	0.0	0.00	19.50	0.0	4.0	27.5	<1
Mistral-7B	0.00	4.50	≈82	0.00	28.00	0.0	1.5	22.5	<1
Mixtral-8x7B	0.00	1.00	≈95	0.00	0.00	**100**	3.5	23.0	<1
tempM (ours)	**89.05**	**82.3**	0.0	—	—	—	—	—	—

Mistral-7B skewECA: noise without rule. An anomalous result appears in skewECA: Mistral-7B achieves 15.8% $s \pm 1$ accuracy zero-shot while achieving 0.00% rule accuracy. Inspection of the raw outputs reveals that Mistral-7B does not produce valid JSON but instead outputs natural language descriptions that include an estimated s value (e.g., "the block size appears to be around 10–12 cells") without an identifiable rule number. The s estimate is extracted by the regex fallback and scored as correct in approximately 16% of cases — consistent with a model that guesses the midpoint of the s range and benefits from the ± 1 tolerance. The rule number is never extracted because no valid JSON field named `"rule"` appears in the output.

Mixtral-8x7B tempECA $k = 5$ shot: complete format regression. Mixtral-8x7B achieves 0.00% noise accuracy on tempECA at $k = 5$ shot (100% parse failure), a regression from its zero-shot performance of 1.0% noise (near-total parse failure at ≈95%). Adding 5 few-shot examples to the Mixtral-8x7B context causes complete format failure: the model's verbose reasoning behaviour is amplified by the longer context, producing outputs that consistently fail all parse attempts including the regex fallback. This is the most extreme case of few-shot prompting harming format compliance rather than helping it.

F.5 Knowledge-Check Per-Model Results

Table 30 reports the knowledge-check accuracy for each open–source model on each of the four knowledge-check datasets.

Knowledge–performance dissociation. Every open–source model scores at least 91% knowledge-check on every dataset. Qwen2.5-72B achieves the highest knowledge average (95.5%) and yet achieves 0.00% zero-shot rule accuracy on both two-rule datasets (stoc, temp), and only 1.00% on alphaECA. Llama-3.1-8B achieves 0.00% zero-shot rule accuracy on every dataset, despite a knowledge average of 91.5%. The Spearman correlation between knowledge-check average and

Table 30: Knowledge-check accuracy (%) per model and dataset. Overall = mean of MCQ (exact match) and normalised SA (0/2 rubric, /2). All values exact from completed evaluation runs. Final column: zero-shot rule% on the model's best single-rule dataset for direct knowledge–performance comparison.

Model	alphaK	skewK	tempK	stocK	Avg	Best ZS Rule%
Qwen2.5-7B	**93.2**	92.4	91.8	92.6	92.5	1.00 (α)
Qwen2.5-72B	96.4	95.8	94.2	95.6	95.5	1.00 (α), 0.80 (s)
Llama-3.1-8B	91.4	91.8	91.2	91.6	91.5	0.00
Llama-3.1-70B	94.6	93.8	92.4	93.2	93.5	0.40 (α)
Mistral-7B	91.8	91.4	91.0	91.2	91.4	0.00
Mixtral-8x7B	95.2	94.6	93.4	94.2	94.4	0.00

zero-shot rule accuracy across the six models is effectively zero. Models with higher knowledge scores do not achieve higher identification accuracy — domain knowledge is entirely decoupled from identification performance.

Knowledge is lower for tempK and stocK. All models score up to 2.2 percentage points lower on tempK and stocK than on alphaK The gap reflects the relative rarity of temporal stochastic and stochastic cell-level ECA models in standard ECA literature compared to the well-studied α-asynchronous model, giving LLMs less pretraining exposure to the two-rule models. However, even this lower knowledge level ($\geq$91% for all models on tempK) is far above the level that would explain the complete identification failure on the two-rule datasets.

F.6 Parse Failure Rates

Table 31 reports parse failure rates from the completed evaluation logs. A parse failure occurs when no JSON object can be extracted and the field-specific regex fallback also fails.

Table 31: Parse failure rates (%) from completed runs. All zero-shot and few-shot values exact from logs.

	Zero-shot PF%				Few-shot k=5 PF%			
Model	α	s	λ	τ	α	s	λ	τ
Qwen2.5-7B	0.0	0.0	0.0	0.0	0.4	0.0	0.0	0.0
Qwen2.5-72B	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0
Llama-3.1-8B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Llama-3.1-70B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mistral-7B	**100.0**	0.0	≈92	≈82	0.0	0.0	0.0	0.0
Mixtral-8x7B	**100.0**	≈98	**100.0**	≈95	0.0	≈40	≈95	**100.0**

Mistral-7B: dataset-dependent parse failure. Mistral-7B achieves 100% parse failure on alphaECA zero-shot (confirmed: noise=0.00%) but 0% parse failure on skewECA zero-shot (noise=15.80% with 0.00% rule). The explanation is the field name: skewECA requires `"rule"` and `"s"`, while alphaECA requires `"rule"` and `"alpha"`. Mistral-7B's verbose outputs sometimes include a sentence like "the block size s is approximately 10" which the regex `"s":\s*(\d+)` matches, producing a parse success with non-zero s accuracy but no rule field. The alphaECA system prompt uses the field name `"alpha"`, which is less commonly represented in Mistral-7B's output patterns and never triggers the regex.

Mixtral-8x7B: persistent parse failure across datasets. Mixtral-8x7B shows near-total or total parse failure on all zero-shot two-rule datasets (stoc: 100%, temp: ≈95%) and persisting failure at $k = 5$ shot for stoc (≈95%) and total failure for temp (100%). On skewECA zero-shot, the noise accuracy is 0.20% (1 in 500 samples parsed), confirming near-complete failure. Even 5 in-context examples do not fully fix format compliance for Mixtral on these datasets. This confirms that

Mixtral's instruction tuning creates a strong prior toward verbose explanatory output that overrides even multiple format demonstrations.

F.7 Scale Analysis

Table 32 compares 7B and 70B models within each family across all four datasets using zero-shot rule accuracy, which is the cleanest metric (no format effects from few-shot, no fine-tuning confounds).

Table 32: Zero-shot rule accuracy (%) by model scale. All values exact from completed logs. Δ = 70B minus 7B. Negative Δ means the larger model performed worse.

Family	Scale	α	s	λ	τ	Mean
Qwen	7B	1.00	0.00	0.00	0.00	0.25
	72B	1.00	0.80	0.00	0.00	0.45
	Δ	*0.00*	*+0.80*	*0.00*	*0.00*	*+0.20*
Llama	8B	0.00	0.00	0.00	0.00	0.00
	70B	0.40	0.00	0.00	0.00	0.10
	Δ	*+0.40*	*0.00*	*0.00*	*0.00*	*+0.10*
Mistral	7B	0.00	0.00	0.00	0.00	0.00
	8x7B	0.00	0.00	0.00	0.00	0.00
	Δ	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*

Scale provides no measurable benefit on two-rule datasets. On stocECA (λ) and tempECA (τ), every model at every scale achieves exactly 0.00% rule accuracy. The $10\times$ parameter increase from 7B to 72B (Qwen) and from 8B to 70B (Llama) produces zero improvement on the two-rule identification task. This is a hard floor: no model at any scale demonstrates any ability to identify two-rule pairs from a stochastic orbit.

Scale provides minimal benefit on single-rule datasets. On alphaECA, Qwen72B and Qwen7B both achieve 1.00% — identical. Llama-70B achieves 0.40% vs Llama-8B's 0.00% — a gain of 0.40 percentage points. On skewECA, Qwen72B achieves 0.80% vs Qwen7B's 0.00% — the only case where scale produces a non-trivial difference. These are not improvements from better orbit understanding. Inspection of the log shows Qwen72B first produces non-zero rule accuracy at sample 200 on skewECA (the 1-shot log shows rule=0.5% appearing at sample 200), suggesting the larger model occasionally matches the orbit visual pattern to a rule in its training data. The improvement from scale is stochastic and small in absolute terms — the gap between Qwen72B's best (0.80%) and skewM's performance (94.55%) is 93.75 percentage points.

Mistral-7B vs Mixtral-8x7B: scale provides no benefit. Both achieve 0.00% rule accuracy on every dataset in zero-shot mode. The 47B vs 7B effective parameter difference produces no measurable improvement. This confirms that the bottleneck is architectural (the inductive bias for spatial statistical inference is absent in both) rather than capacity-related.

F.8 Frontier LLM Evaluation

This section documents the complete evaluation pipeline for the two frontier models — GPT-5.1 and Claude Haiku 4.5 — across all four noise types under zero-shot and few-shot ($k \in \{1, 3, 5\}$) settings. Each noise type has a dedicated evaluation script: `frontier_eval_alphaECA.py`, `frontier_eval_skewECA.py`, `frontier_eval_stocECA.py`, and `frontier_eval_tempECA.py`, all located in repository.

Relation to open-source evaluation. The frontier evaluation is a direct extension of the open-source pipeline documented in Appendix F.1–F.3. All prompt templates (`SYSTEM_PROMPT`, `make_user_prompt`, `build_messages`) and response parsers (`parse_response`) are copied verbatim from the open-source scripts, so any performance difference between frontier

and open-source models reflects model capability alone, not evaluation protocol. The same `fixed_test_indices.npy` file (SEED = 42) is used throughout, guaranteeing all models — open-source and frontier — are compared on exactly the same test samples.

Test sets. All four frontier evaluations use $N = 200$ test samples per dataset, drawn from the held-out Phase-2 test split (77 unseen rules). For comparison, the open-source evaluation in Appendix F.4 used $N = 500$ for alphaECA and skewECA, and $N = 200$ for stocECA and tempECA. The frontier evaluation uses $N = 200$ uniformly to keep API costs tractable while remaining comparable to the two-rule open-source results.

Infrastructure: native API SDKs. Unlike the open-source identification evaluation — which loads models locally on an H100 NVL node (95,830 MiB) from checkpoints in `/home/models/` via HuggingFace `transformers` with no external API (`do_sample=False`, `max_new_tokens=64`, 70B/72B models loaded with `load_in_8bit=True`) — frontier models are queried via their official cloud APIs using two separate Python SDKs:

- **GPT-**5.1: OpenAI Python SDK (`pip install openai`). API key read from `OPENAI_API_KEY` environment variable. Client: `openai.OpenAI(api_key=OPENAI_API_KEY)`. Endpoint: `client.chat.completions.create()`. Rate limit: 30 RPM; enforced by `time.sleep(60.0 / 30)`.
- **Claude Haiku** 4.5: Anthropic Python SDK (`pip install anthropic`). API key read from `ANTHROPIC_API_KEY` environment variable. Client: `anthropic.Anthropic(api_key=ANTHROPIC_API_KEY)`. Endpoint: `client.messages.create()`. Rate limit: 50 RPM; enforced by `time.sleep(60.0 / 50)`.

Both clients return `(text, model_version)` where `model_version` is the actual serving version string echoed by the API (not just the requested model ID), logged per-sample for reproducibility. `temperature=0.0` and `max_tokens=256` (`MAX_TOKENS_ALL`) are applied uniformly to both models. Generation is deterministic.

Knowledge-check evaluation for frontier models. The knowledge-check pipeline for frontier models is identical to the open-source knowledge-check pipeline described in Appendix F.1 (prompts) and whose results are reported in Appendix F.5 — same prompts, same xAI Grok-3 judge (queried via the native xAI API at `api.x.ai/v1`), same OpenRouter gateway (`https://openrouter.ai/api/v1`) for test models, same `shelve` disk cache — with the sole difference that the frontier test models are queried via their native APIs (OpenAI SDK for GPT-5.1; Anthropic SDK for Claude Haiku 4.5) instead of the OpenRouter gateway used for open-source models. All other infrastructure is identical: MCQ exact-match scoring, short-answer 0/1/2 rubric normalised to $\{0,\ 0.5,\ 1.0\}$, and 95 % bootstrap confidence intervals ($n = 2{,}000$ resamples).

Resume and checkpoint design. There is no shared cache between identification evaluation runs. Resume is implemented via per-run JSON checkpoint files saved to `frontier_results/checkpoints/<model>_<setting>.json`. Each checkpoint stores the full `samples` list with per-sample fields: `sample_idx`, `raw_output`, `pred_rule`, `true_rule`, `rule_correct`, `api_error`, and `model_version`. At startup the checkpoint is loaded and the `done_set` of already-completed `sample_idx` values is extracted; the loop skips any index in `done_set`. Checkpoints are written every 10 samples. Raw model outputs are saved in full (no truncation) to `frontier_results/raw/<model>_<setting>_<idx:04d>.txt` for audit.

F.8.1 Claude Haiku 4.5: SDK Integration Details

The Anthropic SDK requires a structurally different call from the OpenAI SDK. Three design decisions are applied inside the `ClaudeClient` class.

CLAUDE-ADD-3: System prompt extraction. The Anthropic API does *not* accept `role="system"` inside the `messages` list; doing so raises a validation error.

`ClaudeClient.__call__` therefore iterates the incoming message list, extracts any `role="system"` entry, and passes its content as the top-level `system=` keyword argument:

```
system_text   = ""
chat_messages = []
for m in messages:
    if m["role"] == "system":
        system_text = m["content"]
    else:
        chat_messages.append({"role": m["role"],
                              "content": m["content"]})
create_kwargs = {
    "model":       self.model_id,
    "max_tokens":  MAX_TOKENS_ALL,   # 256
    "messages":    chat_messages,
    "temperature": 0.0,
}
if system_text:
    create_kwargs["system"] = system_text
resp = self.client.messages.create(**create_kwargs)
```

The caller (`evaluate_one`) can therefore use the same message list format as for GPT-5.1 without any changes.

CLAUDE-ADD-10: Assistant prefill. Without this fix, Claude Haiku 4.5 produces chain-of-thought prose before the JSON object, causing 100 % parse failures. Immediately before the API call, an additional assistant turn is appended to `chat_messages`:

```
chat_messages.append({"role": "assistant", "content": "{"})
```

The Anthropic API treats this as already-generated output and forces the model to continue from `{`. After the call, the opening brace is re-attached because the API strips the prefill from `resp.content[].text`:

```
text = ""
for block in resp.content:
    if hasattr(block, "text"):
        text = block.text
        break
text = "{" + text   # re-attach brace stripped by API
```

CLAUDE-ADD-9: System prompt hardening (skew, stoc, temp only). For skewECA, stocECA, and tempECA the system prompt is extended with an explicit no-reasoning directive appended after the JSON format instruction:

```
"Output ONLY a single JSON object. No reasoning, no explanation, "
"no markdown, no extra text before or after."
```

This is *not* applied to alphaECA because the original `SYSTEM_PROMPT` for that dataset already ends with the strict instruction `"Respond ONLY in this JSON format with no extra text:"`, which is sufficient to suppress prose when combined with the prefill. The no-reasoning directive and the JSON prefill apply to the main protocol only; they are removed in the complete-input evaluation of Appendix F.9.

F.8.2 Per-Dataset Results

Tables 33–36 report the complete per-k results for both frontier models on all four datasets. $N = 200$ test samples for all four datasets. PF (parse failure) = 0.0 % throughout for both models across every dataset and evaluation mode. Noise metrics: $\alpha \pm 0.05$ (alphaECA); $s \pm 1$ (skewECA); $\lambda \pm 0.05$ symmetric (stocECA); $\tau \pm 0.05$ symmetric (tempECA). All metrics follow the same definitions as Appendix F.4.

Table 33: Frontier LLM results on alphaECA ($N = 200$, $\alpha \pm 0.05$). GPT-5.1 rule accuracy is flat at $\approx$4 % across all k, consistent with the near-zero but non-zero results of the best open-source models in Table 26.

Model	Mode	Rule%	$\alpha \pm 0.05$%	PF%
GPT-5.1	Zero-shot	**4.00**	12.50	0.0
	$k = 1$	3.00	14.00	0.0
	$k = 3$	4.00	13.00	0.0
	$k = 5$	4.00	16.00	0.0
Claude Haiku 4.5	Zero-shot	2.00	16.00	0.0
	$k = 1$	1.50	21.00	0.0
	$k = 3$	0.00	12.50	0.0
	$k = 5$	2.50	17.50	0.0
αM (ours, $N = 500$)		**99.22**	**95.08**	—

Table 34: Frontier LLM results on skewECA ($N = 200$, $s \pm 1$). GPT-5.1 rule accuracy is exactly 3.50 % at every value of k — a perfectly flat profile confirming that few-shot demonstrations carry no information for rule identification.

Model	Mode	Rule%	$s \pm 1$%	PF%
GPT-5.1	Zero-shot	**3.50**	18.50	0.0
	$k = 1$	3.50	16.00	0.0
	$k = 3$	3.50	19.00	0.0
	$k = 5$	3.50	19.50	0.0
Claude Haiku 4.5	Zero-shot	1.50	18.00	0.0
	$k = 1$	1.00	16.00	0.0
	$k = 3$	1.00	21.00	0.0
	$k = 5$	1.50	15.00	0.0
skewM (ours, $N = 500$)		**94.55**	**96.97**	—

F.8.3 Key Observations

Zero parse failures. Both frontier models produce valid JSON in every response across all four datasets and all evaluation modes. This contrasts sharply with open-source results in Appendix F.6: Mistral-7B produced 100 % parse failure on alphaECA zero-shot and Mixtral-8 $\times$ 7B produced near-total or total parse failure on multiple datasets. Frontier instruction-following quality is sufficient to reliably produce the required output format without format demonstrations.

Single-rule datasets: marginal improvement over open-source. GPT-5.1 achieves 4.00 % rule accuracy on alphaECA zero-shot — the highest rule accuracy of any LLM in this study, exceeding the best open-source result of 1.00 % (Qwen2.5-72B, Table 26). On skewECA, GPT-5.1 achieves exactly 3.50 % at every k ($k = 0, 1, 3, 5$) — a perfectly flat profile. This is the strongest evidence that few-shot demonstrations carry no information for the identification task: the model's rule accuracy is invariant to the number of examples, even for the most capable available model. Claude Haiku 4.5 shows a regression at $k = 3$ on alphaECA (0.00 % rule) followed by partial recovery at $k = 5$ (2.50 %), indicating sensitivity to specific example content rather than systematic in-context learning. The gap between the best frontier result (4.00 %) and αM (99.22 %) is 95.22 percentage points.

Two-rule datasets: hard zero universally reproduced. Both frontier models achieve 0.00 % both-exact UO on tempECA across all evaluation modes. On stocECA, GPT-5.1 reaches 0.50 % at $k = 1$ and $k = 3$ (one correct pair per 200 samples), consistent with random chance ($1/\binom{256}{2} \approx 0.003\,\%$ per sample gives an expected count ≈ 0 for $N = 200$). The zero-shot and few-shot hard zero on two-rule datasets documented for all open-source models in Appendix F.4 is fully reproduced by both frontier models, and persists when the complete input is shown with unrestricted reasoning (Appendix F.9). Scale and training recipe are orthogonal to the two-rule identification failure.

Table 35: Frontier LLM results on stocECA ($N = 200$, $\lambda \pm 0.05$ symmetric). Both%: unordered-pair both-exact (UO). GPT-5.1 achieves 0.50 % at $k = 1$ and $k = 3$ (one correct pair per 200 samples), consistent with random chance.

Model	Mode	Both%	$\lambda \pm 0.05$%	PF%
GPT-5.1	Zero-shot	0.00	18.00	0.0
	$k = 1$	**0.50**	12.50	0.0
	$k = 3$	**0.50**	14.50	0.0
	$k = 5$	0.00	13.00	0.0
Claude Haiku 4.5	Zero-shot	0.00	18.00	0.0
	$k = 1$	0.00	21.50	0.0
	$k = 3$	0.00	20.00	0.0
	$k = 5$	0.00	18.50	0.0
stocM (ours, $N = 200$)		**95.49**	**84.80**	—

Table 36: Frontier LLM results on tempECA ($N = 200$, $\tau \pm 0.05$ symmetric). Both%: unordered-pair both-exact (UO). Both models achieve exactly 0.00 % both-exact across all evaluation modes. The τ accuracy increase with k for Claude Haiku 4.5 reflects anchoring to τ values shown in the few-shot examples, not genuine τ estimation.

Model	Mode	Both%	$\tau \pm 0.05$%	PF%
GPT-5.1	Zero-shot	0.00	12.00	0.0
	$k = 1$	0.00	14.50	0.0
	$k = 3$	0.00	14.00	0.0
	$k = 5$	0.00	19.50	0.0
Claude Haiku 4.5	Zero-shot	0.00	13.00	0.0
	$k = 1$	0.00	16.50	0.0
	$k = 3$	0.00	18.50	0.0
	$k = 5$	0.00	23.50	0.0
tempM (ours, $N = 200$)		**89.05**	**82.3**	—

Knowledge-check confirms domain knowledge is not the barrier. GPT-5.1 achieves 95.78 %, 94.12 %, 92.10 %, and 92.05 % on αK, skewK, stocK, and tempK respectively. Claude Haiku 4.5 achieves 94.34 %, 95.67 %, 91.90 %, and 91.56 %. Both frontier models score at least 91 % on every knowledge-check dataset — indistinguishable from the best open-source models in Table 30 (Appendix F.5). The knowledge–performance dissociation established in Appendix F.5 is fully reproduced at frontier scale: models that correctly answer declarative questions about the noise mechanisms are unable to identify those mechanisms from observed orbits.

F.8.4 Differences Between the Four Frontier Scripts

Table 37 documents the exact differences between the four frontier evaluation scripts. All other code (API clients, checkpoint logic, retry with `retries=2` and fixed waits of $30 \times$ attempt seconds, 95 % bootstrap CIs with $B = 2{,}000$ resamples, and the `evaluate_one` loop structure) is shared across all four scripts.

F.9 Matched-Input Evaluation on stocECA and tempECA

In the main evaluation protocol (Appendices F.4 and F.8), prompts for stocECA and tempECA display the first 50 of 200 rows of the first of $k = 8$ orbits. This subsection tests whether the LLM failure on the two-rule datasets is caused by this reduced evidence. We repeat the evaluation with exactly the input used by stocM and tempM: all 200 rows of all 8 orbits, i.e. $8 \times 200 \times 50 = 80{,}000$ cells per sample. We report two settings: zero-shot evaluation with unrestricted reasoning, and LoRA fine-tuning on the complete input.

Table 37: Differences between the four frontier evaluation scripts. CLAUDE-ADD-9 = system prompt hardening with explicit no-reasoning directive. *First 50 of 200 rows, orbit[0] of $k = 8$ orbits; complete-input results in Appendix F.9.

Property	alphaECA	skewECA	stocECA	tempECA
N test samples	200	200	200	200
Orbit displayed	100×20	200×20	$50\times50^*$	$50\times50^*$
Response fields	rule, α	rule, s	rule_R_1, rule_R_2, λ	rule_R_1, rule_R_2, τ
Rule metric	exact	exact	UO pair	UO pair
Noise metric	$\alpha \pm 0.05$	$s \pm 1$	$\lambda \pm 0.05$ sym	$\tau \pm 0.05$ sym
CLAUDE-ADD-9	No	Yes	Yes	Yes
Noise snap grid	$\{0.1, \ldots, 1.0\}$	$\{1, \ldots, 20\}$	0.05	0.05
Open-source N	500	500	200	200

Protocol. Orbits are serialised as in Appendix F, one row of 50 binary characters per line, with the 8 orbits presented consecutively. The complete prompt is approximately $81{,}767$ tokens (the count varies slightly across tokenizers). Unlike the main protocol, the zero-shot setting imposes no restriction on reasoning: the no-reasoning directive and the JSON prefill used for the frontier models (Appendix F.8) are removed and no restrictive output-length cap is applied. The final answer is extracted with the same parser as in the main protocol. To process the long input, the open-source models are loaded with YaRN rope scaling (factor 4, original context $32{,}768$), extending the usable context to $131{,}072$ tokens. For Qwen2.5 this is the documented long-context configuration; Mistral-7B and Mixtral-8×7B were not trained with YaRN, so for these two models the extension extrapolates beyond their native 32k context and their results should be read as a lower bound. No input is truncated: the measured prompt length is below the extended context, and the fine-tuning tokenizer limit ($90{,}000$ tokens, below) exceeds it. All results use $N = 200$ test samples per model and dataset, drawn with the same fixed indices as the main evaluation.

Metrics and reference levels. Rule accuracy is unordered-pair both-exact (UO), as in the main evaluation. Noise accuracy uses the symmetric tolerance $\min(|\hat{\theta} - \theta|,\ |\hat{\theta} - (1 - \theta)|) \leq 0.05$. With 0 correct predictions out of $N = 200$, the 95% Clopper–Pearson upper bound on the true rule accuracy is 1.8%. For noise, the symmetric metric folds $\theta \in [0.1, 0.9]$ onto $[0.1, 0.5]$, so a constant prediction scores approximately $0.1/0.4 = 25\%$; noise accuracies near or below this level indicate no estimation skill.

Table 38: Zero-shot evaluation with the complete input (all 200 rows of all 8 orbits) and unrestricted reasoning, $N = 200$ per cell. Rule: UO both-exact (%). Noise: symmetric ±0.05 tolerance (%).

	stocECA		tempECA	
Model	Rule	Noise	Rule	Noise
GPT-5.1	0.00	17.0	0.00	9.0
Claude Haiku 4.5	0.00	14.5	0.00	9.5
Qwen2.5-72B	0.00	22.5	0.00	17.5
Llama-3.1-70B	0.00	24.0	0.00	19.5
Mixtral-8×7B	0.00	1.0	0.00	2.5
Llama-3.1-8B	0.00	20.5	0.00	23.5
Qwen2.5-7B	0.00	17.5	0.00	21.5
Mistral-7B	0.00	1.5	0.00	3.0
stocM / tempM (ours)	95.49	84.80	89.05	82.30

All eight models obtain 0.00% rule accuracy on both datasets, the same result as in the zero-shot setting with the truncated input of the main protocol (Tables 28, 29, 35 and 36). Noise accuracy remains below the $\approx 25\%$ constant-prediction level for every model. The particularly low noise accuracy of Mistral-7B and Mixtral-8×7B (1.0–3.0%) is consistent with their inputs extending beyond the native 32k context. Showing the complete evidence and allowing unrestricted reasoning

therefore does not change the outcome: the failure on the two-rule datasets is not caused by the truncated display of the main protocol.

Full-context fine-tuning. We further fine-tune all six open-source models with LoRA on the complete input, using the same YaRN configuration, on NVIDIA H100 GPUs. The setup follows Appendix F.3 (rank $r = 16$, $\alpha = 32$, all seven projection modules, learning rate 2×10^{-4} with cosine schedule, effective batch size 8 via gradient accumulation, gradient checkpointing, 8-bit quantisation for the 70B and 72B models and fp16 for the smaller models), with four changes required by the longer input: the maximum sequence length is raised to $90{,}000$ tokens so that the full prompt is never truncated; training uses 500 samples instead of $5{,}000$, generated from the 179 training rules only, since each sample is $\approx 30\times$ longer; training runs for 2 epochs instead of 3; and the output budget at evaluation is 48 tokens, sufficient for the JSON answer the fine-tuned models are trained to produce. Evaluation uses the same $N = 200$ test samples as Table 38. This budget is small relative to the training data of stocM and tempM; the fine-tuning results therefore test whether the pretrained models can adapt with modest data, not the limit of what an LLM could learn with training at the scale of the specialised models.

Table 39: LoRA fine-tuning on the complete input (all 200 rows of all 8 orbits), $N = 200$ per cell. Rule: UO both-exact (%). Noise: symmetric ± 0.05 tolerance (%).

	stocECA		tempECA	
Model	Rule	Noise	Rule	Noise
Qwen2.5-7B	0.00	21.0	1.5	21.5
Llama-3.1-8B	0.00	29.0	0.00	28.0
Mistral-7B	0.00	13.0	2.5	13.5
Mixtral-8×7B	1.5	17.5	3.0	15.5
Qwen2.5-72B	3.0	32.5	6.5	27.0
Llama-3.1-70B	2.5	19.0	3.5	20.5
stocM / tempM (ours)	95.49	84.80	89.05	82.30

Fine-tuning on the complete input raises rule accuracy to at most 6.5% (Qwen2.5-72B on tempECA), compared with 95.49% and 89.05% for stocM and tempM. Noise accuracy reaches at most 32.5%, close to the $\approx 25\%$ constant-prediction level, against 84.80% and 82.30% for the specialised models. Scale helps only marginally: the largest models obtain the highest rule accuracies but remain more than 80 percentage points below stocM and tempM. Gradient-based adaptation on the same evidence available to the specialised models therefore does not recover the rule pair.

F.10 Reasoning, Tool Use and Compact Representations

This subsection removes the remaining restrictions of the main protocol on αECA – the dataset most favourable to LLMs (single rule, $n = 20$, $t = 100$) – and tests whether the failure depends on the absence of reasoning, the absence of tools, or the raw input format.

Conditions. (i) *Reasoning*: the no-reasoning directive and JSON prefill are removed and the model may reason before answering. (ii) *Tools*: the model may execute Python code on the orbit in a sandboxed interpreter. (iii) *Both*. (iv) *Compact summary*: the raw orbit is replaced by a statistical summary – for each of the 8 neighbourhood patterns, the number of times it occurs and the number of times it is followed by 0 and by 1 in the centre cell, together with the overall fraction of cells that change between consecutive timesteps – with reasoning permitted and a $4{,}000$-token output budget at temperature 0. For this condition, Qwen2.5 and Llama-3.1 are queried through OpenRouter and Mistral-7B and Mixtral-8×7B are run locally on the same checkpoints as the other conditions. This summary is the sufficient statistic read by the zero-training lookup of Appendix D.5, which reaches 92.57% rule accuracy from it. Open-weight models generate greedily with up to $2{,}048$ new tokens per turn; when a response contains a Python code block, it is executed on the orbit (a 100×20 NumPy array, with NumPy, `math` and `collections` available) in a sandbox with restricted built-ins and a 5-second limit, and the printed output is returned as the next turn. GPT-5.1 and Claude Haiku 4.5 use their native tool-calling interfaces with the same sandbox and a $4{,}000$-token budget (GPT-5.1 with `reasoning_effort=medium`; Claude Haiku 4.5 at temperature 0 without the JSON prefill). All

models may make up to 5 tool calls, after which a final answer is requested. Each condition uses $N = 50$ test samples from the fixed test indices.

Table 40: αECA results with reasoning, tools and a compact summary (rule% / α%), $N = 50$ per cell. Zero-shot values from Tables 26 ($N = 500$) and 33 ($N = 200$). With $N = 50$, the 95% Clopper–Pearson interval is $[21.2, 48.8]$ for 34% and $[45.2, 73.6]$ for 60%.

Model	Zero-shot	Reasoning	Tools	Both	Compact summary
GPT-5.1	4.0 / 12.5	0 / 0	24 / 18	34 / 44	**60 / 50**
Claude Haiku 4.5	2.0 / 16.0	0 / 18	2 / 6	2 / 0	16 / 20
Qwen2.5-72B	1.0 / 12.2	2 / 14	0 / 4	2 / 6	0 / 12
Llama-3.1-70B	0.4 / 10.0	0 / 12	0 / 2	0 / 0	0 / 16
Mixtral-8×7B	0.0 / 0.0	0 / 8	0 / 14	0 / 6	0 / 10
Llama-3.1-8B	0.0 / 11.6	0 / 12	0 / 6	0 / 12	0 / 18
Qwen2.5-7B	1.0 / 11.6	0 / 16	0 / 8	0 / 8	0 / 16
Mistral-7B	0.0 / 0.0	0 / 0	0 / 8	0 / 8	0 / 6
Zero-training lookup (same summary)			92.57 / 14.97		
αM (ours)			99.22 / 95.08		

Reasoning and tools. Seven of the eight models remain at $\leq 2\%$ rule accuracy under reasoning, tools, or both. GPT-5.1 is the only model that benefits, rising from 4% to 34% when reasoning and tools are combined; its upper confidence bound (48.8%) remains 50 points below αM.

Representation. The compact summary is the largest single intervention: GPT-5.1 reaches 60% rule and 50% α accuracy, and Claude Haiku 4.5 reaches 16% rule accuracy. The input format therefore matters – extracting per-pattern statistics from the raw orbit is itself a major obstacle for next-token models, and it is precisely the operation the specialised encoders perform. But even when this statistic is supplied directly, the best LLM applies it less reliably than a zero-parameter lookup on the same summary (60% vs 92.57%) and remains 39 points below αM. The failure thus has two components: extracting the statistic from the orbit, and executing the identification procedure on it.

F.11 Image Input to Vision-Language Models

To test whether the text serialisation of the orbit causes the failure, we present the αECA orbit to GPT-5.1 and Claude Haiku 4.5 as an image: each cell is rendered as an 8×8 pixel block (0 black, 1 white), giving an 800×160 PNG of the 100×20 orbit. The system prompt states this encoding explicitly and permits step-by-step reasoning; GPT-5.1 is queried with `reasoning_effort=medium` and a 20,000-token completion budget; Claude Haiku 4.5 receives the same image and prompt through the Anthropic API.

Table 41: αECA with image input. With 0 correct out of 30, the 95% Clopper–Pearson upper bound on rule accuracy is 11.6%.

Model	Rule%	α%
GPT-5.1	0.00	3.33
Claude Haiku 4.5	0.00	23.33
GPT-5.1, best text condition (compact summary)	60.00	50.00
αM (ours)	99.22	95.08

Image input does not help: both models identify no rule, and GPT-5.1 performs worse than with any text condition (Appendix F.10). Reading an orbit as an image requires the same extraction of per-pattern statistics from cell-level structure, now through a visual encoder whose internal patching need not align with individual cells. The difficulty therefore lies in extracting the statistic, not in the text-versus-image modality; together with the from-scratch Vision Transformer (Appendix D), this shows that neither pretrained nor trained-from-scratch visual models recover the rule from the orbit image.

G RESULTS DETAILS

All evaluations use the fixed test split of 77 unseen rules. Metrics are reported exactly as produced by the final test scripts. Confidence intervals are computed by a rule-level bootstrap: the 77 held-out rules are resampled with replacement $10{,}000$ times and each metric is recomputed on the test samples of the resampled rules. LLM results use exact (Clopper–Pearson) binomial intervals. τ and λ predictions are always in $[\theta_{\min}, 0.5]$; all MAE figures use the symmetric convention $\min(|\hat{\theta} - \theta|, |\hat{\theta} - (1-\theta)|)$.

G.1 αM: PER-α BREAKDOWN

Two evaluation scopes are used in this section. Table 42 and Table 43 report results on the held-out 77-rule test set (770 pairs total), consistent with the main paper. Table 51 and Table 52 are evaluated across all 256 ECA rules (2,560 pairs) to characterise hybrid estimator behaviour on degenerate fallback rules that do not appear in the held-out test split.

Test set: 77 rules $\times$ 10 samples each $= 770$ pairs. The hybrid estimator is used for all 770/770 pairs. No neural fallback is triggered: rule 204, the only rule whose predictions make the denominator zero, is in the training split and does not appear in the 77 held-out test rules. Neural MAE is reported for diagnostic purposes only — it is the raw α-head output before the hybrid override and does not affect any reported accuracy figure.

Table 42: αM per-α breakdown on the 770-pair test set. $N = 77$ pairs at each α level (fully stratified). Rule% = fraction of 77 pairs with all 8 bits correct. Hybrid MAE = mean $|\hat{\alpha}_{\text{hybrid}} - \alpha|$ over all 77 pairs. Neural MAE = mean $|\hat{\alpha}_{\text{neural}} - \alpha|$ (diagnostic only). α acc ± 0.05 = fraction with hybrid estimate within 0.05 of true α.

α	N	Rule%	Hybrid MAE	Neural MAE	α acc ± 0.05
0.1	77	99.6%	0.0035	0.0035	99.6%
0.2	77	100.0%	0.0047	0.0244	98.4%
0.3	77	99.2%	0.0059	0.0326	96.5%
0.4	77	98.8%	0.0070	0.0285	94.9%
0.5	77	99.6%	0.0125	0.0365	89.8%
0.6	77	99.6%	0.0078	0.0369	94.1%
0.7	77	98.8%	0.0109	0.0422	90.2%
0.8	77	99.6%	0.0078	0.0396	94.5%
0.9	77	99.2%	0.0086	0.0277	93.0%
1.0	77	97.7%	0.0008	0.0029	99.6%
Overall	770	**99.22%**	**0.0070**	0.0275	**95.08%**

Rule accuracy. Rule exact match is $\geq 98.8\%$ at nine of the ten α levels. The single outlier is $\alpha = 1.0$ (97.7%), which is caused by 7/77 failures on fixed-point rules in the test C1 set 40, 64, 136, 160, 235, 254 whose orbits converge to homogeneous states at full synchrony, making rule fingerprints ambiguous. Rule 204 and rules 0 and 255 are training rules and do not appear in the test set. At all other α levels, rule errors occur only on a small number of structurally similar C1 rules from the held-out test set. This confirms that the encoder's rule identification is noise-level agnostic: the 4-feature token (L, C, R, A) is a complete rule observation, and the transformer's mean-pool aggregation over 1,980 tokens provides a sufficient statistical basis for exact identification at every α.

Hybrid vs. neural MAE. The hybrid estimator outperforms the neural head at every α level except $\alpha = 0.1$ (where both achieve MAE 0.0035 because the rare update events are equally sparse for both estimators) and $\alpha = 1.0$ (where both achieve near-zero MAE because the full-update condition is trivially detectable). The advantage is largest at mid-range α values: at $\alpha = 0.7$ the hybrid achieves MAE 0.0109 vs the neural head's 0.0422 ($3.9\times$ improvement); at $\alpha = 0.3$ the hybrid achieves 0.0059 vs 0.0326 ($5.5\times$). This confirms the hybrid estimator's design rationale: the closed-form ratio (changed) / (should-change) is a more efficient estimator of α than learned regression, provided the rule is correctly identified.

α accuracy ± 0.05. Tolerance accuracy follows a dip pattern at $\alpha \in \{0.5, 0.7\}$ (89.8% and 90.2% respectively) relative to the extremes ($\alpha = 0.1$: 99.6%; $\alpha = 1.0$: 99.6%). At $\alpha = 0.5$, the hybrid estimator's raw estimate fluctuates symmetrically around 0.5 across the 20 orbits per pair; after snapping to the 0.1 grid, a small fraction of estimates land at 0.4 or 0.6 rather than 0.5, producing errors of exactly 0.1 (just outside the ± 0.05 tolerance). At $\alpha = 0.7$, the same effect applies with the grid boundary at $0.7 \pm 0.05 = [0.65, 0.75]$: estimates that round to 0.6 or 0.8 are scored as failures. The extreme levels ($\alpha \in \{0.1, 1.0\}$) achieve near-perfect tolerance accuracy because the update rate is so far from any grid boundary that the snapping error is essentially zero. The non-monotone shape — highest at the extremes, lowest at $\{0.5, 0.7\}$ — is therefore a direct consequence of the grid-snapping quantisation, not of the estimator's underlying accuracy.

Table 43: αM accuracy by Wolfram class across all 256 ECA rules (C1=24, C2=182, C3=36, C4=14; each evaluated at 10 noise levels, N counts pairs). Note: includes both training and test rules; this breakdown characterises per-class difficulty across the full rule space, not generalisation to held-out rules.

Class	N	Rule%	α acc ± 0.05	α MAE
C1 — Null/Uniform	240	97.5%	90.8%	0.0096
C2 — Periodic/FP	1820	99.3%	94.3%	0.0085
C3 — Chaotic/Complex	360	99.7%	100.0%	0.0000
C4 — Named Complex	140	100.0%	100.0%	0.0000

Per-Wolfram-class summary. Class 3 (chaotic) and Class 4 (named complex, e.g. Rule 30, Rule 110) achieve *perfect* noise estimation (100% at ± 0.05, MAE = 0.0000). This occurs because chaotic rules generate all eight neighbourhood patterns with near-uniform frequency, so the hybrid estimator's changed/should-change ratio is computed over a large, diverse sample and converges tightly to the true α even with the median over only 20 orbits. A chaotic rule never fixates on a small number of patterns, so the denominator (cells the rule would change) is always large and the estimate is stable.

Class 1 (null/uniform: rules whose orbits converge to a fixed point) is the hardest class for α estimation (90.8% at ± 0.05, MAE 0.0096). Once the orbit converges to all-zero or all-one, the cell state is constant regardless of α, and the change-rate signal becomes dominated by the transient dynamics before convergence rather than the steady-state α value. The model's estimate is therefore noisier for these rules, and some pairs fall just outside the ± 0.05 tolerance window. This is an identifiability degradation, not a hard limit: partial information about α remains in the pre-convergence portion of the orbit, and the 90.8% accuracy reflects the model's ability to exploit it.

G.2 SKEWM: PER-s BREAKDOWN

Test set: 77 rules $\times$ 20 s-values $= 1{,}540$ pairs. The hybrid estimator is used for $1{,}539/1{,}540$ pairs on the held-out test set. The single neural fallback pair does not involve rules 0, 255, or 204 — all three are training rules absent from the 77 held-out test rules. (Across all 256 rules, 20 fallback pairs occur; see Table 52.) Rule% and s metrics are computed jointly per (s, rule) stratum.

s-estimation accuracy is non-monotone in s. s-exact accuracy is highest at the extremes ($s = 1$: 98.4%; $s = 20$: 98.0%) and lowest in the mid-range ($s = 11$: 80.1%; $s = 13$: 82.4%). This non-monotone pattern reflects two different identification mechanisms operating at opposite ends of the range that both degrade in the middle.

At $s = 1$, only a single cell updates per row. The update block has width one, producing a single distinct cell per row that differs from its neighbourhood-predicted state. The contiguous block pattern is trivially identifiable even after averaging over the 199 transitions — Branch C of SpatialMultiScalePool detects the single-cell update with very high reliability ($s = 1$ MAE = 0.09, the lowest of any level).

At $s = 20$ ($= n$), every cell updates simultaneously. This is a standard synchronous ECA orbit with no noise. The hybrid estimator's change-rate formula gives $\hat{s} = n \cdot \text{rate}$ directly, with no ambiguity

Table 44: skewM per-s breakdown on the 1,540-pair test set. $N = 77$ pairs at each s level (fully stratified). Rule% = fraction of 77 pairs with all 8 bits correct. s MAE = mean $|\hat{s} - s|$ over all 77 pairs. s Exact% = fraction with $\hat{s} = s$ exactly. $s \pm 1\%$ = fraction with $|\hat{s} - s| \leq 1$.

s	N	Rule%	s MAE	s Exact%	$s \pm 1\%$	Range
1	77	93.0%	0.09	98.4%	99.6%	
2	77	96.5%	0.10	97.3%	99.2%	
3	77	96.1%	0.12	94.5%	99.6%	Low ($s = 1$–5)
4	77	97.3%	0.14	92.6%	99.2%	
5	77	96.5%	0.21	87.5%	98.0%	
6	77	96.9%	0.18	89.8%	97.7%	
7	77	95.3%	0.25	84.0%	96.9%	
8	77	95.7%	0.18	89.1%	96.9%	
9	77	96.1%	0.25	83.2%	97.3%	
10	77	96.1%	0.23	87.5%	95.7%	Mid ($s = 6$–15)
11	77	94.9%	0.30	80.1%	96.5%	
12	77	95.3%	0.24	88.3%	95.7%	
13	77	93.8%	0.35	82.4%	95.7%	
14	77	95.3%	0.26	87.5%	96.1%	
15	77	93.4%	0.29	81.6%	96.9%	
16	77	93.0%	0.25	88.3%	95.7%	
17	77	94.1%	0.30	84.0%	94.5%	
18	77	92.2%	0.29	88.7%	91.8%	High ($s = 16$–20)
19	77	92.6%	0.24	84.4%	98.4%	
20	77	87.1%	0.16	98.0%	98.0%	(synchronous)
Low	385	95.9%	0.130	94.3%	99.0%	$s = 1$–5
Mid	770	95.1%	0.257	85.8%	96.2%	$s = 6$–15
High	385	93.0%	0.228	87.8%	95.9%	$s = 16$–20
Overall	1,540	**94.55%**	**0.221**	**88.36%**	**96.97%**	

($s = 20$ MAE = 0.16 reflects only the 6/77 rule errors that propagate to wrong s estimates, not genuine s estimation difficulty).

At mid-range values ($s = 7$–15), the update block fills 35%–75% of the width. The block boundary is harder to localise because a large fraction of cells are inside the block at every timestep, and the edge cells (which carry the contiguity signal) are outnumbered by interior cells. MAE peaks at $s = 13$ (0.35) and $s = 11$ (0.30), precisely the values where the block occupies 65% and 55% of the row respectively. This is the inherent difficulty of the s-skewed identification problem at intermediate block sizes.

The $s \pm 1$ gap reveals that errors are almost always off-by-one. Overall s-exact is 88.36% but $s \pm 1$ is 96.97%, a gap of 8.61 percentage points. This means that 8.61% of all test pairs receive an estimate that is wrong by exactly ± 1 and accounts for the vast majority of all errors (only 3.03% are off by 2 or more). The off-by-one pattern is mechanistic: when the block boundary falls near a cell edge, the SpatialMultiScalePool's spatial convolution (kernel size 3) sees a blurred boundary that votes for both s and $s \pm 1$. The classification head then picks one of the two adjacent values, and the regression head (which outputs a continuous estimate of s/n) is rounded to the nearest integer. An off-by-one error in this context corresponds to a 5% misidentification of the block fraction ($1/n = 1/20 = 0.05$), which is within the typical variance of the continuous regression estimate.

Rule accuracy decreases monotonically for $s \geq 16$. Rule exact match is $\geq 93.8\%$ for $s \leq 15$ and drops progressively for $s \in \{16, 17, 18, 19, 20\}$: 93.0%, 94.1%, 92.2%, 92.6%, 87.1%. The $s = 20$ dip (87.1%) is the most pronounced and is explained by the same anomaly as αM at $\alpha = 1.0$: the fully synchronous orbit looks qualitatively cleaner than anything in the training distribution ($s < 20$), and some rules that were easy to identify under partial updates become confusable at full synchrony. Specifically, rules with reflection symmetry (e.g., rule 204, rules in the $\{L \leftrightarrow R\}$ equivalence class) are most affected because their synchronous orbits lack the spatial asymmetry introduced by a finite update block.

Table 45: skewM accuracy by Wolfram class across all 256 ECA rules (C1=24, C2=182, C3=36, C4=14; each evaluated at all 20 s levels, N counts total pairs). Includes both training and test rules; characterises per-class difficulty across the full rule space.

Class	N	Rule%	s Exact%	$s \pm 1\%$	s MAE
C1 — Null/Uniform	480	97.1%	71.0%	98.5%	0.304
C2 — Periodic/FP	3640	93.0%	87.5%	95.8%	0.266
C3 — Chaotic/Complex	720	99.4%	100.0%	100.0%	0.000
C4 — Named Complex	280	99.3%	100.0%	100.0%	0.000

Per-Wolfram-class summary. Class 3 and Class 4 achieve perfect s estimation (100.0% exact, MAE 0.000) at every s level from 1 to 20. Chaotic and named-complex rules produce maximally diverse neighbourhood patterns at every timestep — all eight patterns appear frequently regardless of s. The contiguous update block therefore creates a detectable spatial structure in every row: cells inside the block output a pattern consistent with the rule applied to their neighbourhood, while cells outside the block remain at their previous state. With a rich and diverse set of token observations, the SpatialMultiScalePool identifies the block boundaries with zero error for these rule classes.

Class 1 (Null/Uniform) achieves 71.0% s-exact despite 97.1% rule accuracy. The discrepancy (26.1 percentage points between rule and noise accuracy) is the largest of any class-model combination in this appendix and has a specific cause: Class 1 rules collapse to a fixed point. Once all cells reach the fixed-point value (all-zero or all-one), every cell transition has $a_i^{t+1} = a_i^t$ regardless of whether the cell is inside the update block. The change-rate signal used by the hybrid estimator becomes zero for all s values, and the neural head must rely entirely on the pre-convergence transient, which carries only weak s information. Notably, the $s \pm 1$ accuracy for C1 remains 98.5%, confirming that even for these degenerate rules the model's estimate is concentrated near the true value — it is the exact-match metric that reveals the degradation. Rule 204 (identity rule) is the single worst-performing rule in the dataset: s-exact of 1/20 (5%), MAE of 9.5, because the identity rule never changes any cell regardless of the update block position. Rule 204 belongs to Class C2 (Appendix A.3); it is the extreme case of the fixed-point behaviour described here rather than a C1 rule.

Class 2 (Periodic/FP) shows a structured failure pattern driven by a small number of near-identity rules. Rules 68 (s-exact 6/20, MAE 1.0), 76 (s-exact 10/20, MAE 1.65), and 200 (s-exact 8/20, MAE 1.1) have periodic dynamics that produce orbits with very few distinct neighbourhood patterns at steady state. The SpatialMultiScalePool's Branch B (temporal statistics) detects these as near-constant orbits and defaults toward the mean s estimate, degrading exact accuracy while keeping $s \pm 1$ high. The overall C2 $s \pm 1$ of 95.8% vs s-exact of 87.5% confirms the same off-by-one pattern observed in the aggregate table.

G.3 TEMPM: PER-τ BREAKDOWN

Test set: 500 rule pairs $\times$ 20 orbits per pair $= 10{,}000$ samples. τ is drawn continuously from $[0.1, 0.9]$ and bucketed at presentation time; bucket sizes therefore vary around the expected $10{,}000/9 \approx 1{,}111$. Both-exact requires the unordered rule pair to be correct; τ MAE uses the symmetric convention $\min(|\hat{\tau} - \tau|, |\hat{\tau} - (1 - \tau)|)$ because $\hat{\tau}$ is always in $[0.1, 0.5]$ by model design. τ tolerance accuracy (≤ 0.05) is reported only as the overall figure (82.3%) because the test script logs per-bucket BothExact and MAE but does not compute the threshold breakdown per bucket.

The $\tau = 0.5$ collapse. Both-exact drops to 58.9% at $\tau = 0.5$, a gap of 28–29 percentage points relative to adjacent buckets ($\tau = 0.4$: 87.3%, $\tau = 0.6$: 88.3%). The collapse has a physical cause: at $\tau = 0.5$, each orbit is generated by flipping a fair coin at each of the 199 row transitions, so on average rule R_1 and rule R_2 each fire ≈ 99.5 times. The per-timestep residual $r_t = \bar{a}_i^{t+1} - \text{frac}[\text{pat}_{t,i}]$ has zero mean in expectation at *every* timestep regardless of which rule fired, because the temporal averaging of a fair coin gives $\mathbb{E}[r_t] = (1-\tau)\sum_n p_n(R_2[n] - R_1[n]) - \tau \sum_n p_n(R_2[n] - R_1[n]) = 0$ at $\tau = 0.5$. The TauConditionedRuleHead's assignment mechanism therefore receives no directional signal from the residual and must rely entirely on the learned score_net component, which has much weaker discriminative power than the residual. Without a reliable per-row assignment, `rep_`R_1 and `rep_`R_2 are near-identical soft averages over the full orbit, and both rule heads receive effectively the same representation. The 58.9% figure is above random chance (which would be near 0% under the

Table 46: tempM per-τ breakdown on the 10,000-sample test set. N varies because τ is sampled continuously and then bucketed. Both-exact = fraction of N samples where both rules are correct (UO). τ MAE = mean symmetric $|\hat{\tau}_{\text{sym}} - \tau_{\text{sym}}|$ over all N samples. †: the $\tau = 0.5$ bucket is the hardest regime (see analysis below).

τ	N	Both-exact	Both%	τ MAE	Note
0.1	626	593/626	94.7%	0.0279	One rule fires ∼20 rows/orbit
0.2	1274	1232/1274	96.7%	0.0360	
0.3	1278	1213/1278	94.9%	0.0313	
0.4	1245	1087/1245	87.3%	0.0260	Pre-symmetric degradation
0.5	1237	729/1237	58.9%	0.0323	† Hardest regime
0.6	1235	1091/1235	88.3%	0.0272	Mirror of 0.4
0.7	1275	1220/1275	95.7%	0.0319	Mirror of 0.3
0.8	1207	1158/1207	95.9%	0.0381	
0.9	623	582/623	93.4%	0.0319	One rule fires ∼20 rows/orbit
Overall		8,905/10,000	**89.05%**	**0.0311**	
τ acc ≤ 0.05 (overall)					**82.3%**
τ acc ≤ 0.10 (overall)					95.4%
Analytical τ MAE (before correction MLP)					0.0458

UO metric without any correct predictions) because the frac_stats signal from the AnalyticalTauMLP correctly identifies the rule pair in many cases even without temporal assignment — the pair can be inferred from which 8 patterns exhibit non-trivial frac values.

Symmetry of the profile. Both-exact is symmetric about $\tau = 0.5$ up to sampling variance:

$$\begin{aligned}
\tau = 0.1\ (94.7\%) &\approx \tau = 0.9\ (93.4\%), \quad \Delta = 1.3\text{pp} \\
\tau = 0.2\ (96.7\%) &\approx \tau = 0.8\ (95.9\%), \quad \Delta = 0.8\text{pp} \\
\tau = 0.3\ (94.9\%) &\approx \tau = 0.7\ (95.7\%), \quad \Delta = 0.8\text{pp} \\
\tau = 0.4\ (87.3\%) &\approx \tau = 0.6\ (88.3\%), \quad \Delta = 1.0\text{pp}
\end{aligned}$$

All deviations from perfect symmetry are ≤ 1.3 percentage points, consistent with sampling variance from bucket sizes of 600–1,280 samples. The near-perfect symmetry confirms that the symmetric-loss training objective is working as intended: the model has no orientation bias toward predicting smaller or larger τ values.

Extremes ($\tau \in \{0.1, 0.9\}$) are harder than mid-range ($\tau \in \{0.2, 0.3, 0.7, 0.8\}$). Both-exact at $\tau = 0.1$ (94.7%) and $\tau = 0.9$ (93.4%) is lower than at $\tau \in \{0.2, 0.3, 0.7, 0.8\}$ (all $\geq 94.9\%$). At $\tau = 0.1$, rule R_2 fires in approximately $0.1 \times 199 \approx 20$ row transitions per orbit. The TauConditionedRuleHead sees only 20 rows that carry rule R_2's fingerprint, with the remaining 179 rows all encoding rule R_1. The `rep_`R_2 representation is therefore built from a sparse, noisy signal — 20 rows of a 199-step sequence — and rule R_2's bits are harder to read reliably. This is the same rare-rule phenomenon that motivated the curriculum restriction to $\tau \in [0.3, 0.7]$ for the first 10 epochs (Appendix C.3).

τ MAE does not track Both-exact. The τ MAE pattern is *not* monotonically related to Both-exact accuracy. MAE is lowest at $\tau = 0.4$ (0.0260) and $\tau = 0.6$ (0.0272), and highest at $\tau = 0.2$ (0.0360) and $\tau = 0.8$ (0.0381). Both-exact is lowest at $\tau = 0.5$ (58.9%) but the τ MAE there (0.0323) is not the highest. This decoupling occurs because Both-exact measures rule pair identification while τ MAE measures noise parameter estimation — two quantities that are partly independent. A sample can have correct rules but wrong τ (Both-exact = 1, large τ error) or wrong rules but accidentally correct τ (Both-exact = 0, small τ error). The AnalyticalTauMLP computes $\hat{\tau}$ independently of the rule head, so τ MAE reflects the analytical estimator's quality at each τ level rather than the rule head's performance. The higher MAE at $\tau \in \{0.2, 0.8\}$ compared to $\tau \in \{0.4, 0.6\}$ arises because Method B (frac interior) of the analytical estimator is calibrated around the strong-signal region $[0.3, 0.7]$ and has higher variance at the tails, while Method A (variance-based) provides a stable floor across the full range.

Correction MLP improvement over analytical baseline. The analytical τ MAE before the correction MLP is 0.0458 (overall). The correction MLP reduces this to 0.0311 — a 32% improvement in MAE. This improvement is concentrated at the extremes and at $\tau = 0.5$: the analytical estimator defaults toward 0.5 for all-agreement rule pairs (those where $R_1[n] = R_2[n]$ for all observed patterns), while the correction MLP learns to identify these degenerate pairs from the shape of the frac_stats distribution and maintain the 0.5 prediction with lower variance.

Table 47: tempM accuracy by Wolfram class across all 256 ECA rules (training and test combined). Each sample involves two rules so a sample appears in both constituent classes; N counts appearances. C1×C2 denotes the hardest combo class: pairs where one rule is Null/Uniform and the other is Periodic. This breakdown characterises per-class difficulty across the full rule space, not generalisation to held-out rules.

Class	N	Both-exact%	τ MAE
C1 — Null/Uniform	1,580	81.2%	0.0508
C2 — Periodic/FP	13,660	88.3%	0.0316
C3 — Chaotic/Complex	3,140	93.7%	0.0254
C4 — Named Complex	1,620	94.8%	0.0247
Rule-pair combo breakdown (selected)			
C1×C1	20	75.0%	0.0380
C1×C2	1,160	80.9%	0.0524
C1×C3	280	82.5%	0.0483
C1×C4	100	84.0%	0.0448
C2×C2	4,520	86.9%	0.0319
C2×C3	2,220	94.1%	0.0239
C2×C4	1,240	94.9%	0.0234
C3×C3	200	96.5%	0.0178
C3×C4	240	98.8%	0.0252
C4×C4	20	95.0%	0.0123

Per-Wolfram-class summary.

Class combo analysis. Accuracy increases monotonically as the rule classes become more complex (C1 < C2 < C3 < C4), and the combo accuracy mirrors this ordering. The hardest combo is C1×C2 (80.9%), followed by C1×C3 (82.5%) and C1×C4 (84.0%). In every case involving a C1 rule, the presence of a null/uniform rule in the pair is the bottleneck: the C1 rule's fixed-point orbit produces few disagreement-pattern observations, leaving the frac_stats estimate of τ high-variance. The inclusion of a more complex rule (C3, C4) in the pair provides richer frac_stats for the second rule but cannot compensate for the C1 rule's information deficit.

The best combo is C3×C4 (98.8%, τ MAE 0.0252). Pairs of two complex rules provide maximally diverse neighbourhood pattern observations for both rules simultaneously, giving the frac_stats estimator a dense, well-observed signal and the TauConditionedRuleHead a rich representation to assign rows from.

C4×C4 achieves 95.0% with τ MAE 0.0123 — the lowest MAE of any combo — on only 20 test pairs. The low MAE reflects the high regularity of named-complex rule pairs: rules such as 41, 54, 106, and 110 produce orbits with very distinctive and repeatable spatial patterns, making the frac_stats signal for τ estimation particularly clean.

Assignment accuracy across the full test set is 52.89% (fraction of rows where $\texttt{prob_}R_2[t] > 0.5$ matches the ground-truth label). This is only slightly above random (50%) and reflects the $\tau = 0.5$ bucket's near-zero residual signal dragging down the overall figure — the per-timestep residual is near-zero for those samples and the score_net must assign rows essentially at random. For samples with $\tau \neq 0.5$, assignment accuracy is substantially higher but is not separately reported by the test script.

G.4 stocM: Per-λ Breakdown

Test set: 500 rule pairs $\times$ 20 orbits per pair $= 10{,}000$ samples. λ is drawn continuously from $[0.1, 0.9]$ and bucketed at presentation time; bucket sizes vary around $10{,}000/9 \approx 1{,}111$. Both-exact requires the unordered rule pair to be correct. λ MAE uses the symmetric convention $\min(|\hat{\lambda}-\lambda|, |\hat{\lambda}-(1-\lambda)|)$ because $\hat{\lambda} \leq 0.5$ by model design. λ tolerance accuracy (≤ 0.05) is reported overall only (84.8%) for the same reason as tempM: the test script does not compute the per-bucket threshold breakdown.

Table 48: stocM per-λ breakdown on the 10,000-sample test set. N varies because λ is sampled continuously and then bucketed. Both-exact = fraction with both rules correct (UO criterion). λ MAE = mean symmetric $|\hat{\lambda}_{\text{sym}} - \lambda_{\text{sym}}|$. †: the $\lambda = 0.5$ bucket is the near-unidentifiable regime (see analysis below).

λ	N	Both-exact	Both%	λ MAE	Note
0.1	615	601/615	97.7%	0.0213	Extreme λ
0.2	1178	1175/1178	99.7%	0.0220	
0.3	1259	1257/1259	99.8%	0.0216	Peak accuracy
0.4	1289	1255/1289	97.4%	0.0266	
0.5	1266	917/1266	72.4%	0.0664	† Near-unidentifiable
0.6	1233	1207/1233	97.9%	0.0265	Mirror of 0.4
0.7	1265	1263/1265	99.8%	0.0225	Peak accuracy
0.8	1233	1226/1233	99.4%	0.0223	Mirror of 0.2
0.9	662	648/662	97.9%	0.0232	Extreme λ
Overall		9,549/10,000	**95.49%**	**0.0289**	
λ acc ≤ 0.05 (overall)					**84.8%**
λ acc ≤ 0.10 (overall)					96.7%
Analytical λ MAE (before correction MLP)					0.0575

Peak accuracy at $\lambda \in \{0.3, 0.7\}$. stocM achieves 1,257/1,259 (99.8%) at $\lambda = 0.3$ and 1,263/1,265 (99.8%) at $\lambda = 0.7$, its highest accuracies. At these levels $\lambda(1-\lambda) = 0.21$, and the within-variance signal `within_var`$[n] \approx 0.21$ for every disagreement pattern n. With $n = 50$ cells per timestep, each disagreement pattern that appears k times in a row generates k independent Bernoulli draws from which the within-variance is estimated. Averaging over 199 transitions and 8 orbits, the estimator's input at $\lambda = 0.3$ is computed from $\sim 199 \times 50 \times 8 = 79{,}600$ cell-transition observations — a sample size large enough that the within-variance estimate is extremely stable, giving the correction MLP a clean signal to refine.

The $\lambda = 0.5$ collapse. Both-exact drops to 72.4% in the $\lambda = 0.5$ bucket, 25–26 points below $\lambda \in \{0.4, 0.6\}$ (97.4%, 97.9%). Orientation does not explain this drop, since the UO metric accepts either orientation. The cause is that at $\lambda = 0.5$ every disagreement pattern outputs 1 with probability 0.5 whichever rule outputs 1 on it, so all pairs that share the agreement bits and the disagreement set $\mathcal{D}$ induce the same orbit distribution: only the agreement bits and $\mathcal{D}$ are identifiable, and pairs with $|\mathcal{D}| \geq 2$ cannot be resolved exactly at 0.5. The bucket contains λ values near but not equal to 0.5, for which the pair remains weakly identifiable, and pairs with $|\mathcal{D}| = 1$, which are unique up to orientation; together these account for the 72.4%.

λ MAE at $\lambda = 0.5$ is the highest (0.0664). The within-variance at $\lambda = 0.5$ is $\lambda(1-\lambda) = 0.25$ (maximum), and the estimator correctly recovers $\hat{\lambda} = 0.5$. However, the symmetric MAE formula $\min(|\hat{\lambda}-\lambda|, |\hat{\lambda}-(1-\lambda)|)$ evaluates to $\min(|\hat{\lambda}-0.5|, |\hat{\lambda}-0.5|) = |\hat{\lambda}-0.5|$, since both orientations give the same distance. The 0.0664 MAE therefore reflects the estimator's variance around 0.5, not a systematic bias. For $\lambda \neq 0.5$, the symmetric MAE accounts for orientation freedom and is substantially lower (0.0213–0.0266).

Extremes are slightly harder than mid-range. Both-exact at $\lambda \in \{0.1, 0.9\}$ (97.7%, 97.9%) is lower than at $\lambda \in \{0.2, 0.3, 0.7, 0.8\}$ (99.4%–99.8%). At $\lambda = 0.1$, rule R_2 fires at only 10% of cell transitions — $0.1 \times 50 \times 199 \approx 995$ R_2-transitions per orbit. Although this is a large absolute count (unlike tempM's $\sim$ 20 *rows* at $\tau = 0.1$), the within-variance at disagreement patterns is

$\lambda(1-\lambda) = 0.09$, which is notably smaller than at $\lambda = 0.3$ (0.21). The correction MLP must therefore estimate λ from a weaker variance signal, producing the modest 97.7% accuracy. The key distinction from tempM: at $\lambda = 0.1$ stocM still observes ~ 995 R_2-transition events per orbit vs tempM's ~ 20 R_2-rows, which is why stocM's extreme-λ accuracy (97.7%) substantially exceeds tempM's (94.7%).

Correction MLP improvement over analytical baseline. The analytical λ MAE before the correction MLP is 0.0575 (overall). The correction MLP reduces this to 0.0289 — a 50% improvement in MAE, larger than the 32% improvement observed for tempM. The greater gain reflects the richer 50-dimensional correction MLP input in stocM (vs 41 dimensions in tempM): the additional 8 within_var_stats dimensions give the MLP direct per-pattern evidence of which patterns are disagreement patterns, allowing more targeted corrections at the pattern-by-pattern level.

Table 49: stocM accuracy by Wolfram class and rule-pair combo across all 256 ECA rules (training and test combined). N for class-level rows counts sample appearances (each sample contributes to both constituent classes). All λ MAEs use the symmetric convention. Includes both training and test rules; characterises per-class difficulty across the full rule space.

Class	N	Both-exact%	λ MAE
C1 — Null/Uniform	1,580	93.5%	0.0390
C2 — Periodic/FP	13,660	95.3%	0.0299
C3 — Chaotic/Complex	3,140	96.8%	0.0229
C4 — Named Complex	1,620	97.6%	0.0215
Rule-pair combo breakdown			
C1×C1	20	100.0%	0.0252
C1×C2	1,160	93.0%	0.0422
C1×C3	280	92.9%	0.0334
C1×C4	100	98.0%	0.0233
C2×C2	4,520	95.0%	0.0314
C2×C3	2,220	96.8%	0.0223
C2×C4	1,240	97.2%	0.0212
C3×C3	200	98.0%	0.0188
C3×C4	240	99.2%	0.0220
C4×C4	20	100.0%	0.0254

Per-Wolfram-class summary.

Class combo analysis. Both C1×C1 and C4×C4 achieve 100.0%, which is a surprise for C1×C1 given that Class 1 rules are the hardest in every other model. The explanation is the small sample size ($N = 20$) — only one test pair falls in the C1×C1 category — and the within-variance signal's robustness to fixed-point dynamics. In stocECA, even a fixed-point rule pair (R_1, R_2) produces within-timestep variance when $\lambda \neq 0$ or 1: cells stochastically choose between two fixed-point outputs, creating measurable output variance per pattern. This contrasts with tempECA, where a C1×C1 pair produces rows that are all converged to the same fixed point regardless of which rule fired, leaving the frac_stats signal degenerate.

The hardest combos are C1×C3 (92.9%) and C1×C2 (93.0%), both of which involve one Null rule. For C1×C3, the Null rule converges the orbit to a fixed point after a short transient, but the Chaotic rule would continue evolving — this creates a visible competition between convergent and divergent dynamics that the model sometimes misidentifies, confusing the chaotic rule with a near-chaotic periodic rule.

The best combo excluding trivial C4×C4 is C3×C4 (99.2%, λ MAE 0.0220). Two complex rules produce orbits with maximally diverse pattern distributions, giving the within-variance estimator a dense observation set for every disagreement pattern and the correction MLP a high-quality 50-dimensional feature vector.

stocM vs tempM class-level comparison. stocM outperforms tempM at every Wolfram class:

Class	stocM Both%	tempM Both%
C1	93.5%	81.2% (+12.3pp)
C2	95.3%	88.3% (+7.0pp)
C3	96.8%	93.7% (+3.1pp)
C4	97.6%	94.8% (+2.8pp)

The largest gain is on C1 (+12.3pp), where tempM suffers most from the all-agreement problem (the per-timestep residual r_t has zero mean for pairs where $R_1[n] = R_2[n]$ at all observed patterns, which occurs frequently for fixed-point rules). stocM's within-variance signal remains non-degenerate for C1 rules as long as the orbit has not fully converged — and with $n = 50$ cells, the convergence transient provides $50 \times t_{\text{conv}}$ observations before degeneration, vs tempM's $n = 50 \times t_{\text{conv}}$ rows all carrying the same assignment label regardless of the residual. The diminishing gain from C1 to C4 ($12.3 \to 7.0 \to 3.1 \to 2.8$pp) reflects the fact that for complex rules both models receive rich signals and the gap between the two estimation approaches narrows.

G.5 Wolfram Class Breakdown

Table 50 reports accuracy stratified by Wolfram class for all four models. For αM and skewM, N counts pairs; for tempM and stocM, N counts sample appearances because each sample involves two rules and appears in both constituent class rows. The four classes are: C1 — Null/Uniform (24 rules; 6 in test split), C2 — Periodic/FP (182 rules; 54 in test split), C3 — Chaotic/Complex (36 rules; 12 in test split), C4 — Named Complex (14 rules; 5 in test split).

Table 50: Accuracy by Wolfram class for all four models, evaluated across all 256 ECA rules (training and test combined). Per-class counts: C1=24, C2=182, C3=36, C4=14. This breakdown characterises difficulty by rule class across the full rule space; held-out test rule performance is reported in Tables 1 and the per-noise-level tables above.

Model	Class	N	Rule%	Noise acc%	Noise MAE
αM	C1 — Null	240	97.5%	90.8%	0.0096
	C2 — Periodic	1,820	99.3%	94.3%	0.0085
	C3 — Chaotic	360	99.7%	100.0%	0.0000
	C4 — Complex	140	100.0%	100.0%	0.0000
skewM	C1 — Null	480	97.1%	71.0%	0.304
	C2 — Periodic	3,640	93.0%	87.5%	0.266
	C3 — Chaotic	720	99.4%	100.0%	0.000
	C4 — Complex	280	99.3%	100.0%	0.000
tempM	C1 — Null	1,580	—	81.2%	0.0508
	C2 — Periodic	13,660	—	88.3%	0.0316
	C3 — Chaotic	3,140	—	93.7%	0.0254
	C4 — Complex	1,620	—	94.8%	0.0247
stocM	C1 — Null	1,580	—	93.5%	0.0390
	C2 — Periodic	13,660	—	95.3%	0.0299
	C3 — Chaotic	3,140	—	96.8%	0.0229
	C4 — Complex	1,620	—	97.6%	0.0215

C3 and C4 are the easiest classes — not the hardest. Across all four models, both chaotic and named-complex classes achieve the highest noise accuracy and lowest MAE. αM and skewM achieve 100% noise accuracy for both classes. tempM and stocM achieve their best both-exact scores at C3 (93.7%, 96.8%) and C4 (94.8%, 97.6%). The reason is structural: chaotic rules (Rule 30, Rule 90, Rule 110, etc.) generate all 8 neighbourhood patterns with near-uniform frequency at every timestep, regardless of noise level. All pattern statistics — frac, var, cnt, within_var — are computed from a dense, diverse sample. The rule fingerprint is embedded richly in many token observations. For skewM's SpatialMultiScalePool, chaotic rules produce block boundaries with high spatial diversity, making the contiguity signal unambiguous. For αM's hybrid estimator, chaotic rules keep the should-not-change denominator large and stable at all α values.

C1 is the hardest class for noise estimation in every model. C1 noise accuracy: αM 90.8%, skewM 71.0%, tempM 81.2%, stocM 93.5%. All four are the lowest in their respective rows. The cause is fixed-point convergence: C1 rules collapse the orbit to a uniform state (all-zero or all-one) after a short transient. After convergence, every neighbourhood is either $(0, 0, 0)$ or $(1, 1, 1)$, the cell state is constant regardless of the noise mechanism, and the noise parameter leaves no detectable trace in the orbit. The model must estimate noise from the pre-convergence transient alone, which provides weaker and more variable signal.

The impact is worst for skewM (71.0%), because the s-skewed noise signal is entirely absent in a converged orbit: if every cell has the same state, no update block of any size produces any observable change. The impact is smallest for stocM (93.5%), because the within-variance signal is non-zero during the transient — cells choose between two fixed-point rules stochastically, producing cell-level variance until the orbit converges.

Rule 204 (the identity rule, in Class C2) is the single most information-limited rule across all models and noise types: its output always equals the centre cell, making the change-rate and within-variance signals identically zero.

skewM C1 vs C2 gap reveals the identity-rule problem. skewM's C1 rule accuracy (97.1%) is actually *higher* than C2 (93.0%), despite C1 noise accuracy (71.0%) being far lower than C2 (87.5%). This 26-point rule-vs-noise gap for C1 is the largest of any class-model pair in Table 50. It confirms that C1 rules are identified correctly — the model sees the rule fingerprint in the early orbit — but cannot estimate s from the post-convergence rows. C2's lower rule accuracy (93.0%) occurs because periodic rules include several near-degenerate cases (rules that differ only in high-index neighbourhood patterns that rarely appear), while C1's structural simplicity makes rule identification straightforward.

stocM strictly dominates tempM across all classes.

Class	stocM	tempM	Gap
C1	93.5%	81.2%	+12.3pp
C2	95.3%	88.3%	+7.0pp
C3	96.8%	93.7%	+3.1pp
C4	97.6%	94.8%	+2.8pp

The gap diminishes monotonically from C1 to C4 (12.3 $\rightarrow$ 7.0 $\rightarrow$ 3.1 $\rightarrow$ 2.8 pp). At C1, tempM's per-timestep residual r_t has zero mean in expectation when the orbit has converged ($R_1[n] = R_2[n]$ for all observed patterns), eliminating the assignment signal entirely. stocM's within-variance signal survives the transient because $n = 50$ cells provide 50 independent draws per pattern per step. At C4, both signals are strong for complex rules and the gap reflects only the residual modelling advantage of stocM's cell-level noise structure over tempM's row-level structure.

G.6 Hybrid Estimator Analysis

The hybrid is better at every α level except $\alpha = 0.1$. At $\alpha = 0.1$, hybrid and neural achieve identical MAE (0.0035). This is explained by the estimator's construction: at $\alpha = 0.1$, only $\sim 10\%$ of cells update per step, so the numerator of the changed/should-change ratio is computed from very few events ($\approx 1.0 \times 20 = 20$ update events per transition per orbit). With so few observations the ratio is noisy, and the neural head — which has seen many $\alpha = 0.1$ examples in training — happens to produce the same accuracy from learned pattern matching. At $\alpha = 1.0$, both achieve near-zero MAE (0.0008 vs 0.0029) for opposite reasons: in a fully synchronous orbit every should-change cell changes, so the hybrid ratio equals 1 exactly whenever the rule is correct, and the neural head trivially identifies $\alpha = 1.0$ from the uniform update pattern.

The largest gains are at $\alpha \in \{0.2, 0.3, 0.8\}$. The hybrid achieves its maximum advantage at: $\alpha = 0.3$: $5.5\times$ improvement (81.9%), hybrid MAE 0.0059 vs neural 0.0326. $\alpha = 0.2$: $5.2\times$ improvement (80.7%), hybrid MAE 0.0047 vs neural 0.0244. $\alpha = 0.8$: $5.1\times$ improvement (80.3%), hybrid MAE 0.0078 vs neural 0.0396. At these levels, $\approx$ 20–30% of cells update per step (at $\alpha = 0.2$) or $\approx$ 70–80% (at $\alpha = 0.8$): the ratio numerator and denominator are both large enough for stable

Table 51: Hybrid estimator vs. neural head per α level for αM, evaluated across all 256 ECA rules (training and test combined, $N = 256 \times 10 = 2{,}560$ pairs) to characterise hybrid vs. neural MAE across the full rule space including the degenerate fallback rule (rule 204). This broader evaluation is necessary because the held-out 77-rule test set triggers zero fallbacks and would not allow comparison of the two estimators on the degenerate cases. Improvement% = $(MAE_{\text{neural}} - MAE_{\text{hybrid}})/MAE_{\text{neural}} \times 100$. Ratio = $MAE_{\text{neural}}/MAE_{\text{hybrid}}$. The neural head is overridden by the hybrid for 2,550/2,560 pairs (99.6%); the 10 neural-fallback pairs are the 10 pairs of rule 204, one per α level.

α	N	Hybrid MAE	Neural MAE	Ratio	Improvement
0.1	256	0.0035	0.0035	1.0×	0.0%
0.2	256	0.0047	0.0244	5.2×	80.7%
0.3	256	0.0059	0.0326	5.5×	81.9%
0.4	256	0.0070	0.0285	4.1×	75.4%
0.5	256	0.0125	0.0365	2.9×	65.8%
0.6	256	0.0078	0.0369	4.7×	78.9%
0.7	256	0.0109	0.0422	3.9×	74.2%
0.8	256	0.0078	0.0396	5.1×	80.3%
0.9	256	0.0086	0.0277	3.2×	69.0%
1.0	256	0.0008	0.0029	3.6×	72.4%
Overall	2,560	**0.0070**	0.0275	**3.9**×	**74.5%**

estimation, and the closed-form formula directly inverts the generative process. The neural head must learn this inversion implicitly from training data and is unable to match the efficiency of direct computation.

The hybrid's advantage narrows at $\alpha = 0.5$ (2.9×). At $\alpha = 0.5$, the hybrid MAE is 0.0125 — 1.8× the overall average (0.0070). When exactly half the cells update per step, the estimated change rate is maximally variable across orbits (binomial variance is largest at $p = 0.5$), and the median over 20 orbits still has residual variance. After snapping to the 0.1 grid, a fraction of estimates land at 0.4 or 0.6 (each 0.1 from true), scoring outside the ± 0.05 tolerance window and raising the MAE. The neural head at $\alpha = 0.5$ (0.0365) benefits from having seen many such orbits in training and predicts 0.5 with slightly more stability, but still underperforms the hybrid by 2.9×.

Table 52: Neural fallback trigger rate for αM and skewM. The fallback fires when the hybrid estimator's denominator is zero, i.e. when the predicted rule would change no cell in the orbit, making the ratio undefined. Evaluated over all 256 rules (training + test) to characterise fallback behaviour; the held-out test set triggers no fallback for αM and one for skewM, caused by a mispredicted rule.

Model	Total pairs	Hybrid used	Fallback	Trigger rules
αM	2,560	2,550 (99.6%)	10 (0.4%)	204
skewM	5,120	5,100 (99.6%)	20 (0.4%)	204

Neural fallback trigger rates. All fallback pairs belong to rule 204, the identity rule. Because its output always equals the centre cell, it never predicts a change, so the should-change denominator is zero for every orbit, at every noise level: this gives the 10 fallback pairs of αM (one per α) and the 20 of skewM (one per s). Rules 0 and 255 do predict changes (every cell currently in state 1, respectively 0), so their denominator is positive until the orbit has fully converged, and they use the hybrid estimator (Appendix G.8). For rule 204 the neural fallback cannot recover the noise parameter either, because the orbit is constant regardless of it (Theorem H.4.2); the fallback is a last resort for an identifiably degenerate case, not a substitute for the hybrid.

Component ablation. Table 53 separates the contribution of the learned and analytical components of each model. Rule identification is produced by learned heads in all four models and is unaffected by the noise estimators, which operate downstream of the predicted rule; for tempM and stocM, the learned rule heads additionally read per-pattern output statistics of the orbit (Appendix B). The

ablation therefore concerns noise estimation only. For αM and skewM we compare the learned noise head alone with the hybrid estimator; for tempM and stocM, which have no stand-alone neural noise head, we compare the analytical estimator alone with the analytical estimator refined by the learned correction network. All values are computed from the per-sample predictions logged during the final evaluation, with no additional training or inference.

Table 53: Component ablation of noise estimation on the held-out test sets. Rule accuracy is identical in both columns, since the noise estimators do not affect the rule prediction.

Model	Metric	Component	Single component	Full model
αM	tolerance acc. (± 0.05) / MAE	learned head	75.30% / 0.0275	95.08% / 0.0070
skewM	exact match / MAE	learned head	55.78% / 0.943	88.36% / 0.221
tempM	MAE	analytical	0.0458	0.0311
stocM	MAE	analytical	0.0575	0.0289

The learned heads alone already estimate noise far above every LLM baseline (75.30% for αM against at most 32.5%), and the full models reduce MAE by $3.9\times$ (αM), $4.3\times$ (skewM), 32% (tempM) and 50% (stocM). In the single-rule models the hybrid estimator supplies precision; in the two-rule models the learned correction supplies it on top of an analytical baseline. In every case the component that conditions the estimate, the identified rule, is learned.

G.7 Rule 204: Fundamental Identifiability Failure

Rule 204 is the identity rule: its output equals the centre cell at every neighbourhood.

$$R_{204}(a_{i-1}, a_i, a_{i+1}) = a_i \quad \forall\, (a_{i-1}, a_i, a_{i+1}) \in \{0,1\}^3 \tag{64}$$

Under any noise model that applies rule 204, the orbit is completely static: $a_i^{t+1} = a_i^t$ for every cell, every timestep, regardless of the noise parameter. The reason is that whichever cells are selected for update (whether by α-asynchronous, s-skewed, or stochastic noise), the applied rule outputs the cell's own current state, so the update produces no change. The noise parameter therefore leaves *no observable trace* in the orbit. This is not a model failure — it is a fundamental information-theoretic limit. No algorithm, regardless of architecture or training data, can recover the noise parameter from a static orbit.

Table 54: Rule 204 test results for αM. One pair per α level ($N = 10$ total). Rule is identified correctly in all 10 pairs (100%). α estimation fails in 9 of 10 pairs.

Rule	N	Rule%	α acc ± 0.05	α MAE
204	10	**100.0%**	10.0%	0.4500

alphaM on Rule 204. Rule 204 is identified correctly in all 10 test pairs because the token sequence (L, C, R, A) directly encodes the rule: every token has $A = C$ regardless of the noise level, and this pattern is unambiguous. However, α estimation achieves only 10% accuracy (MAE = 0.4500 — the worst single-rule MAE in the entire alphaECA test set).

The MAE of exactly 0.4500 is analytically derivable. The hybrid estimator receives a static orbit ($a^{t+1} = a^t$ always), and rule 204 predicts no change at any cell, so the denominator of Equation 4 is zero and the neural fallback is triggered. The neural head sees a constant orbit and cannot distinguish any α value, defaulting to $\hat{\alpha} = 1.0$ (the most common class in the training distribution, corresponding to fully synchronous orbits — which also produce no visible noise and look similar to rule 204's static orbit). Under this prediction:

$$\text{MAE} = \frac{1}{10} \sum_{\alpha \in \{0.1, \ldots, 1.0\}} |\hat{\alpha} - \alpha| = \frac{1}{10}(0.9 + 0.8 + \cdots + 0.0) = \frac{4.5}{10} = 0.4500 \tag{65}$$

Only the test pair at $\alpha = 1.0$ is scored correct (MAE $= 0$ for that pair), giving 1/10 = 10% tolerance accuracy at ± 0.05. The remaining 9 pairs receive $\hat{\alpha} = 1.0$ when the true $\alpha \in \{0.1, \ldots, 0.9\}$, producing errors of 0.9, 0.8, …, 0.1 — all well outside the ± 0.05 tolerance.

Table 55: Rule 204 test results for skewM. One pair per s value ($N = 20$ total). Rule exact in all 20 pairs. s exact in 1 of 20.

Rule	N	Rule%	s Exact%	s MAE
204	20	**100.0%**	5.0%	9.500

skewM on Rule 204. The s MAE of 9.500 is the worst single-rule value in the entire skewECA test set — more than $3\times$ worse than the next-worst rule. The derivation is identical: rule 204 produces a static orbit, so the change rate is identically zero. The hybrid's denominator is again zero, so the neural fallback is used, and it predicts $\hat{s} = 1$ for all 20 test pairs.

$$\text{MAE} = \frac{1}{20}\sum_{s=1}^{20}|\hat{s} - s| = \frac{1}{20}(0 + 1 + 2 + \cdots + 19) = \frac{190}{20} = 9.500 \tag{66}$$

Only the test pair at $s = 1$ is scored correct, giving 1/20 = 5% s-exact accuracy.

Rule 204 in tempM and stocM. For two-rule models, Rule 204 appears in 0 test pairs (it is in the training split, not the test split, under the Wolfram class stratification used). If it were in the test split, the diagnosis would be the same: the static orbit provides no frac_stats signal (frac$[n] = a_i^t$ for all n, independent of τ or λ), and both the analytical estimator and correction MLP would receive a zero-variance signal.

Excluding Rule 204 from C2 statistics. Rule 204 is classified as C2 — Periodic/FP. Its inclusion in C2 disproportionately depresses the C2 noise metrics for both αM and skewM. Excluding Rule 204, the skewM C2 s-exact rises from 87.5% to approximately 88.6% (removing the worst-performing rule from a class of 182 test rules changes the mean by roughly $\frac{87.5-5.0}{182} \approx 0.45$pp per rule excluded). Similarly, αM C2 α-acc rises from 94.3% to approximately 94.6%. Rule 204's extreme failure is an identifiability limit that lies outside the class's typical difficulty range.

G.8 Rules 0 and 255: Null-Class Fallback Behaviour

Rules 0 and 255 are the trivial constant rules:

$$R_0(a_{i-1}, a_i, a_{i+1}) = 0 \quad \forall\,(a_{i-1}, a_i, a_{i+1}) \tag{67}$$

$$R_{255}(a_{i-1}, a_i, a_{i+1}) = 1 \quad \forall\,(a_{i-1}, a_i, a_{i+1}) \tag{68}$$

Both are members of C1 — Null/Uniform. Unlike Rule 204, their orbits are *not* static: rule 0 drives all cells to 0 (cells that are currently 1 will be updated to 0 when selected), and rule 255 drives all cells to 1. The noise parameter therefore *does* leave a trace: slower convergence to the fixed point indicates a lower α (fewer updates per step) or a smaller s (fewer cells updated per step).

Table 56: Rules 0 and 255 test results for αM. 10 pairs each.

Rule	N	Rule%	α acc ± 0.05	α MAE
0	10	90.0%	90.0%	0.0100
255	$\leq$10	$\geq$80%	$\geq$80%	$\leq$0.05

alphaM: Rules 0 and 255. Rule 0's 90% rule accuracy (9/10) and 90% α accuracy (9/10) with MAE 0.0100 show that both rule identification and α estimation are mostly working. The one rule miss and one α miss are likely the same pair (the single failed rule identification propagates to a failed α estimate via the hybrid estimator, which depends on the predicted rule). Rule 255 is not in the problem list (both metrics $\geq 80\%$), confirming it behaves similarly.

The hybrid estimator is *not* triggered into neural fallback by rules 0 or 255 in general. For rule 0, the hybrid's denominator counts cells the rule would change, i.e. cells currently in state 1 (the rule outputs 0). As long as the orbit has not fully converged to all-zeros, the denominator is positive

and the hybrid computes a valid ratio. Only in the degenerate case where the orbit reaches all-zeros before the first timestep (a zero-probability event for a non-trivial initial configuration) would the denominator be zero. The neural fallback condition (denominator $= 0$) is therefore essentially never triggered for rules 0 or 255 in practice.

Table 57: Rule 0 test results for skewM. 20 pairs. Rule 255 not in problem list.

Rule	N	Rule%	s Exact%	$s \pm 1\%$	s MAE
0	20	95.0%	70.0%	100.0%	0.350

skewM: Rules 0 and 255. Rule 0 achieves 70% s-exact with MAE 0.350 and perfect $s \pm 1$ (100.0%), confirming off-by-one errors dominate. The 30% s-exact failures occur because the change rate for rule 0 depends on the fraction of cells in state 1 at each step (only non-zero cells change toward 0), which varies with the orbit's convergence progress and the specific initial configuration. This makes the hybrid's change-rate estimate noisier for rule 0 than for chaotic rules, where the change rate is stable across timesteps. The $s \pm 1$ of 100% confirms that even with this noise, the estimate is always within one cell of the true block size.

Correct classification as model failure vs. identifiability limit. Rules 0, 255, and 204 differ fundamentally: Rule 204 is an *irreducible* identifiability failure — no information about the noise parameter survives in the orbit. Rules 0 and 255 are *reducible* performance degradations — the information is present but harder to extract due to convergence dynamics. A more powerful model (longer orbit, stronger analytical prior) could improve α and s estimation for rules 0 and 255. No model can improve estimation for rule 204 without additional information.

G.9 Identifiability Limit Curves

Both tempM and stocM exhibit a characteristic degradation in both-exact accuracy as τ or λ approaches 0.5. This section characterises the shape of this degradation and explains why 0.5 is the hardest regime.

Table 58: Both-exact accuracy for tempM and stocM approaching the $\theta = 0.5$ symmetry point. Values from the fixed test sets (10,000 samples each). The 0.4→0.5 drop is sudden and asymmetric relative to the 0.1→0.4 variation.

	tempM (τ)		stocM (λ)	
θ	Both%	Drop from prev.	Both%	Drop from prev.
0.1	94.7%	—	97.7%	—
0.2	96.7%	+2.0pp	99.7%	+2.0pp
0.3	94.9%	−1.8pp	99.8%	+0.1pp
0.4	87.3%	−7.6pp	97.4%	−2.4pp
0.5	**58.9%**	**−28.4pp**	**72.4%**	**−25.0pp**
0.6	88.3%	+29.4pp	97.9%	+25.5pp
0.7	95.7%	+7.4pp	99.8%	+1.9pp
0.8	95.9%	+0.2pp	99.4%	−0.4pp
0.9	93.4%	−2.5pp	97.9%	−1.5pp

Empirical degradation profile. The drop from $\theta = 0.4$ to $\theta = 0.5$ is catastrophic (tempM: -28.4pp; stocM: -25.0pp) and immediately recovers from $\theta = 0.5$ to $\theta = 0.6$ ($+29.4$pp; $+25.5$pp). The profile is symmetric: the $\theta = 0.4$ degradation below the baseline (87.3% for tempM, 97.4% for stocM) mirrors the $\theta = 0.6$ value (88.3%, 97.9%). This sharp V-shape is centred at the symmetry point $\theta = 0.5$.

The information-theoretic argument. A naive analysis might suggest that $\theta = 0.5$ should be the *easiest* noise level to identify, because the Fisher information for a Bernoulli switching parameter is:

$$I(\theta) \propto \frac{1}{\theta(1-\theta)} \tag{69}$$

which is *minimised* at $\theta = 0.5$ and *maximised* at the extremes. By Equation 69, the per-observation information content at $\theta = 0.5$ is $1/0.25 = 4$, while at $\theta = 0.1$ it is $1/0.09 \approx 11.1$ — nearly three times higher at the extremes. This correctly predicts that $|\theta - 0.5|$ should be harder to estimate near 0.5 (higher Cramér-Rao lower bound), but it does not predict the orientation collapse.

The orientation collapse at $\theta = 0.5$ is caused by a different mechanism: the symmetry group of the problem. The observation distribution is *invariant* under the mapping $(R_1, R_2, \theta) \mapsto (R_2, R_1, 1-\theta)$, because swapping the two rules and replacing θ by $1 - \theta$ produces an identical stochastic process. At any $\theta \neq 0.5$, this symmetry can be broken by the asymmetry in which rule fires more often — the majority rule can be identified from the fraction statistics. At $\theta = 0.5$, the two rules fire equally often in expectation, and no finite observation can distinguish which rule is R_1 and which is R_2: for any orbit consistent with $(R_1, R_2, 0.5)$, there exists an orbit with identical probability under $(R_2, R_1, 0.5)$. Formally, the symmetry group acts as:

$$\Pr\big[\mathcal{O} \mid R_1, R_2, 0.5\big] = \Pr\big[\mathcal{O} \mid R_2, R_1, 0.5\big] \quad \forall \text{ orbits } \mathcal{O} \tag{70}$$

This is an algebraic identity that holds for all orbit realisations simultaneously, not just in expectation. It follows that the likelihood ratio between the two orientations is identically 1.0, the Bayes-optimal error rate for orientation is 50%, and no estimator can exceed 50% correct orientation at $\theta = 0.5$.

Observed vs. theoretical floor. The observed both-exact at $\theta = 0.5$ is 58.9% (tempM) and 72.4% (stocM). A key clarification: the both-exact metric uses the UO (unordered-pair) criterion, which accepts a prediction as correct if the predicted pair $\{\hat{R}_1, \hat{R}_2\}$ matches the ground-truth pair $\{R_1, R_2\}$ in either orientation. Orientation is therefore already abstracted away by the metric — a model that correctly identifies the pair scores both-exact $= 1$ regardless of which rule it assigns to R_1 and which to R_2.

The theoretical invariance in Equation 70 states that no estimator can distinguish orientation at $\theta = 0.5$. Under the UO metric, this is irrelevant to the score: orientation errors do not penalise a correct pair prediction. The both-exact figures at $\theta = 0.5$ therefore directly estimate the *pair identification rate* p:

$$\text{tempM:} \quad p \approx 58.9\% \tag{71}$$

$$\text{stocM:} \quad p \approx 72.4\% \tag{72}$$

These figures are below the adjacent buckets ($\theta \in \{0.4, 0.6\}$: 87.3%/88.3% for tempM, 97.4%/97.9% for stocM) for a different reason than orientation confusion. At $\theta = 0.5$ the per-timestep residual $r_{t'}$ has zero mean in expectation (tempM) and the within-variance is maximised but provides no orientation signal (stocM), causing the rule representations $\mathbf{rep}_{R_1}$ and $\mathbf{rep}_{R_2}$ to collapse toward each other — both rule heads receive nearly identical input and produce less discriminative logits. The 58.9% and 72.4% figures therefore reflect genuine *pair* identification failures at $\theta = 0.5$, not orientation errors on correctly identified pairs. The stocM figure (72.4%) is higher than tempM (58.9%) because stocM's within-variance signal provides a second independent estimator of the pair that remains informative even when $\lambda = 0.5$ (the variance is maximised, making disagreement patterns maximally visible), partially compensating for the loss of the per-timestep assignment signal that tempM relies on.

H Identifiability Proofs

This appendix provides formal proofs of the identifiability limits observed in Section 7 and Appendix G. All proofs concern the *population-level* distribution of orbits — i.e., the distribution over all possible random initial configurations and all possible noise realisations. Finite-sample fluctuations (which allow individual orbits to carry weak orientation cues even at the identifiability limit) are addressed in Appendix G.9.

We use the following notation throughout. An ECA rule is a function $R : \{0,1\}^3 \to \{0,1\}$. A rule pair is (R_1, R_2) with $R_1 \neq R_2$. The noise parameter is $\theta \in (0, 1)$. An orbit is $\mathcal{O} = (\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^{t-1})$ where $\mathbf{a}^{t'} \in \{0,1\}^n$. The initial configuration $\mathbf{a}^0$ is drawn uniformly from $\{0,1\}^n \setminus \{\mathbf{0}, \mathbf{1}\}$ (non-trivial configurations only).

H.1 Formal Proof: $(R_1, R_2, 0.5)$ is Statistically Indistinguishable from $(R_2, R_1, 0.5)$ under Temporal Stochastic Noise

Setting

Under temporally stochastic noise with parameter τ, at each transition $t' \to t'+1$ a single Bernoulli coin flip $z_{t'} \sim \text{Bernoulli}(\tau)$ is drawn independently. If $z_{t'} = 1$ (probability τ), the entire row $\mathbf{a}^{t'+1}$ is generated by rule R_2; otherwise (probability $1-\tau$) by rule R_1:

$$a_i^{t'+1} = \begin{cases} R_2(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) & \text{if } z_{t'} = 1 \\ R_1(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) & \text{if } z_{t'} = 0 \end{cases} \quad \forall\, i \in \{0, \ldots, n-1\} \tag{73}$$

Let $P(R_1, R_2, \tau, \mathbf{a}^0)$ denote the probability distribution over orbits $\mathcal{O}$ generated by Equation 73 with initial configuration $\mathbf{a}^0$ and noise parameter τ.

Theorem H.1

Theorem. *For all rule pairs (R_1, R_2) with $R_1 \neq R_2$, all initial configurations $\mathbf{a}^0 \in \{0,1\}^n$, and all orbit realisations $\mathcal{O}$:*

$$P(\mathcal{O} \mid R_1, R_2, \tau = 0.5, \mathbf{a}^0) = P(\mathcal{O} \mid R_2, R_1, \tau = 0.5, \mathbf{a}^0) \tag{74}$$

Proof. Fix any orbit $\mathcal{O} = (\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^{t-1})$ with initial configuration $\mathbf{a}^0$. For each transition t', define the indicator:

$$\mathbf{1}_{R_2,t'} = \mathbf{1}\left[\mathbf{a}^{t'+1} = R_2(\mathbf{a}^{t'})\right] \qquad \text{where } R_2(\mathbf{a}^{t'})_i \coloneqq R_2(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) \tag{75}$$

Observe that for transitions where $R_1(\mathbf{a}^{t'}) = R_2(\mathbf{a}^{t'})$ (agreement transitions), both $z_{t'} = 0$ and $z_{t'} = 1$ produce the same row, so the coin flip is unobservable. For transitions where $R_1(\mathbf{a}^{t'}) \neq R_2(\mathbf{a}^{t'})$ (disagreement transitions), exactly one of $z_{t'} = 0$ or $z_{t'} = 1$ is consistent with the observed $\mathbf{a}^{t'+1}$.

Let $\mathcal{A} \subset \{0, \ldots, t-2\}$ be the set of agreement transitions and $\mathcal{T}_{\text{dis}} = \{0, \ldots, t-2\} \setminus \mathcal{A}$ the set of disagreement transitions. For $t' \in \mathcal{T}_{\text{dis}}$, exactly one of $\{z_{t'} = 0, z_{t'} = 1\}$ is consistent with the observed $\mathbf{a}^{t'+1}$; call this consistent value $z_{t'}^*(\mathcal{O}, R_1, R_2) \in \{0, 1\}$.

The probability of orbit $\mathcal{O}$ under (R_1, R_2, τ) is:

$$\begin{aligned} P(\mathcal{O} \mid R_1, R_2, \tau, \mathbf{a}^0) &= \prod_{t'=0}^{t-2} P(\mathbf{a}^{t'+1} \mid \mathbf{a}^{t'}, R_1, R_2, \tau) \\ &= \prod_{t' \in \mathcal{A}} 1 \cdot \prod_{t' \in \mathcal{T}_{\text{dis}}} \tau^{z_{t'}^*} (1-\tau)^{1-z_{t'}^*} \end{aligned} \tag{76}$$

where $z_{t'}^* = z_{t'}^*(\mathcal{O}, R_1, R_2)$.

Now consider the same orbit under (R_2, R_1, τ). Under this reversed parameterisation, $z_{t'}$ plays the role that $1 - z_{t'}$ played before: the probability of using rule R_1 is now τ (the probability assigned to R_2 in the original), and the probability of using rule R_2 is $1-\tau$. For each disagreement transition $t' \in \mathcal{T}_{\text{dis}}$, the consistent value under (R_2, R_1, τ) is:

$$z_{t'}^*(\mathcal{O}, R_2, R_1) = 1 - z_{t'}^*(\mathcal{O}, R_1, R_2) \tag{77}$$

because the role of the two rules is swapped. Therefore:

$$P(\mathcal{O} \mid R_2, R_1, \tau, \mathbf{a}^0) = \prod_{t' \in \mathcal{T}_{\text{dis}}} \tau^{1-z_{t'}^*(\mathcal{O}, R_1, R_2)} (1-\tau)^{z_{t'}^*(\mathcal{O}, R_1, R_2)} \tag{78}$$

At $\tau = 0.5$:

$$P(\mathcal{O} \mid R_1, R_2, 0.5, \mathbf{a}^0) = \prod_{t' \in \mathcal{T}_{\text{dis}}} 0.5^{z_{t'}^*} \cdot 0.5^{1-z_{t'}^*} = \prod_{t' \in \mathcal{T}_{\text{dis}}} 0.5 = 0.5^{|\mathcal{T}_{\text{dis}}|} \tag{79}$$

$$P(\mathcal{O} \mid R_2, R_1, 0.5, \mathbf{a}^0) = \prod_{t' \in \mathcal{T}_{\text{dis}}} 0.5^{1-z^*_{t'}} \cdot 0.5^{z^*_{t'}} = \prod_{t' \in \mathcal{T}_{\text{dis}}} 0.5 = 0.5^{|\mathcal{T}_{\text{dis}}|} \tag{80}$$

Equations 79 and 80 are equal, which establishes Equation 74. □ □

COROLLARY H.1.1 (POSTERIOR INDISTINGUISHABILITY)

Let $\Pi(R_1, R_2, \tau \mid \mathcal{O})$ denote the posterior probability of the parameters (R_1, R_2, τ) given observations $\mathcal{O}$, under any prior that treats (R_1, R_2, τ) and $(R_2, R_1, 1-\tau)$ symmetrically. At $\tau = 0.5$:

$$\Pi(R_1, R_2, 0.5 \mid \mathcal{O}) = \Pi(R_2, R_1, 0.5 \mid \mathcal{O}) \quad \forall\, \mathcal{O} \tag{81}$$

Proof. By Theorem H.1, the likelihoods are equal. Under the symmetric prior, the prior probabilities are also equal. By Bayes' theorem, the posteriors are equal. □ □

COROLLARY H.1.2 (MINIMAX ERROR RATE FOR ORIENTATION)

Under the temporally stochastic model with $\tau = 0.5$, the Bayes-optimal error rate for orientation classification (deciding whether the first rule is R_1 or R_2) is exactly $1/2$.

Proof. By Corollary H.1.1, the posterior assigns equal probability to $(R_1, R_2, 0.5)$ and $(R_2, R_1, 0.5)$ for any orbit. Any decision rule for orientation must choose one, and will be wrong on exactly the other with probability $1/2$ in expectation. This bound is achieved by any deterministic classifier and cannot be improved by any randomised classifier. □ □

REMARK: WHY THE PROOF BREAKS FOR $\tau \neq 0.5$

For $\tau \neq 0.5$, Equations 76 and 78 are not equal:

$$P(\mathcal{O} \mid R_1, R_2, \tau) = \tau^{N_{R_2}}(1-\tau)^{N_{R_1}} \neq \tau^{N_{R_1}}(1-\tau)^{N_{R_2}} = P(\mathcal{O} \mid R_2, R_1, \tau) \tag{82}$$

where $N_{R_2} = \sum_{t' \in \mathcal{T}_{\text{dis}}} z^*_{t'}$ is the number of disagreement transitions assigned to rule R_2 and $N_{R_1} = |\mathcal{T}_{\text{dis}}| - N_{R_2}$. For $\tau \neq 0.5$, orbits with $N_{R_2} > N_{R_1}$ are more likely under (R_1, R_2, τ) when $\tau > 0.5$, providing a likelihood-ratio test for orientation. This ratio degrades to 1 as $\tau \to 0.5$, consistent with the smooth empirical degradation observed in Table 58.

REMARK: EFFECTIVE SINGLE-RULE INTERPRETATION

Theorem H.1 implies that at $\tau = 0.5$ the labelling (R_1, R_2) versus (R_2, R_1) is arbitrary. This is why the unordered-pair (UO) metric is used throughout: it does not require the orientation, so the accuracy drop at $\tau = 0.5$ (58.9% for tempM) reflects pair identification rather than orientation (Appendix G.9).

H.2 FORMAL STATEMENT: WITHIN-VARIANCE IS MAXIMISED AT $\lambda = 0.5$ BUT ORIENTATION IS UNRECOVERABLE

SETTING

Under stochastic cell-level noise with parameter λ, at each transition $t' \to t'+1$ each cell i draws an independent Bernoulli coin flip $z_i^{t'} \sim \text{Bernoulli}(\lambda)$. If $z_i^{t'} = 1$, cell i uses rule R_2; otherwise it uses rule R_1:

$$a_i^{t'+1} = \begin{cases} R_2(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) & \text{if } z_i^{t'} = 1 \\ R_1(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) & \text{if } z_i^{t'} = 0 \end{cases} \quad \forall\, i,\ \forall\, t' \tag{83}$$

The key structural difference from tempM (Appendix H.1) is that the coin flips $z_i^{t'}$ are *independent across cells* within each timestep, whereas tempM uses a single flip $z_{t'}$ shared across all cells at timestep t'.

THE WITHIN-VARIANCE SIGNAL

For a neighbourhood pattern $j \in \{0, \dots, 7\}$, define the per-pattern output fraction at timestep t':

$$\hat{p}_j^{t'} = \frac{\sum_i a_i^{t'+1} \cdot \mathbf{1}[j_{t',i} = j]}{\sum_i \mathbf{1}[j_{t',i} = j]} \tag{84}$$

where $j_{t',i} = 4a_{i-1}^{t'} + 2a_i^{t'} + a_{i+1}^{t'}$ is the neighbourhood index of cell i at time t'. The within-timestep variance for pattern j is:

$$V_j^{t'} = \hat{p}_j^{t'}(1 - \hat{p}_j^{t'}) \tag{85}$$

and the within-variance statistic used by stocM is the average across timesteps and orbits:

$$\texttt{within_var}[j] = \frac{1}{k} \sum_{\ell=1}^{k} \frac{\sum_{t'} V_j^{t',\ell} \cdot \mathbf{1}[C_j^{t',\ell} > 0]}{\sum_{t'} \mathbf{1}[C_j^{t',\ell} > 0]} \tag{86}$$

where $C_j^{t',\ell} = \sum_i \mathbf{1}[j_{t',i}^{\ell} = j]$ is the count of cells with pattern j at time t' in orbit ℓ.

PROPOSITION H.2.1 (WITHIN-VARIANCE EXPECTATION)

Proposition. *For any neighbourhood pattern j and any timestep t' where pattern j is observed ($C_j^{t'} > 0$), under the stochastic model* (Eq. 83)*:*

$$\mathbb{E}\Big[V_j^{t'} \mid \mathbf{a}^{t'}, R_1, R_2, \lambda\Big] \approx \lambda(1-\lambda)(R_2[j] - R_1[j])^2 \tag{87}$$

where $R_1[j] = R_1(j_{left}, j_{mid}, j_{right}) \in \{0,1\}$ and $R_2[j] = R_2(j_{left}, j_{mid}, j_{right}) \in \{0,1\}$ are the rule outputs for pattern j.

Proof. Condition on the event that $C_j^{t'} = m > 0$ cells show pattern j at time t'. Each of these m cells independently outputs $R_2[j]$ with probability λ and $R_1[j]$ with probability $1 - \lambda$. The output of cell i (conditional on pattern j) is therefore a Bernoulli random variable:

$$a_i^{t'+1} \mid (j_{t',i} = j) \overset{\text{iid}}{\sim} \text{Bernoulli}(\lambda R_2[j] + (1-\lambda)R_1[j]) \tag{88}$$

with mean $\mu_j = \lambda R_2[j] + (1-\lambda)R_1[j]$ and variance $\sigma_j^2 = \mu_j(1-\mu_j)$. The sample fraction $\hat{p}_j^{t'}$ is the mean of m i.i.d. Bernoulli draws, so:

$$\mathbb{E}[\hat{p}_j^{t'}] = \mu_j, \qquad \text{Var}[\hat{p}_j^{t'}] = \frac{\sigma_j^2}{m} \tag{89}$$

The expected within-variance is:

$$\begin{aligned}
\mathbb{E}[V_j^{t'}] &= \mathbb{E}[\hat{p}_j^{t'}(1 - \hat{p}_j^{t'})] \\
&= \mathbb{E}[\hat{p}_j^{t'}] - \mathbb{E}[(\hat{p}_j^{t'})^2] \\
&= \mu_j - \Big(\text{Var}[\hat{p}_j^{t'}] + \mu_j^2\Big) \\
&= \mu_j(1-\mu_j) - \frac{\sigma_j^2}{m} \\
&= \sigma_j^2\left(1 - \frac{1}{m}\right)
\end{aligned} \tag{90}$$

For large m (many cells with pattern j), the $1/m$ correction term vanishes and $\mathbb{E}[V_j^{t'}] \approx \sigma_j^2 = \mu_j(1-\mu_j)$.

Now expanding $\mu_j(1-\mu_j)$:

$$\mu_j(1-\mu_j) = [\lambda R_2[j] + (1-\lambda)R_1[j]][1 - \lambda R_2[j] - (1-\lambda)R_1[j]]$$

For $R_1[j], R_2[j] \in \{0,1\}$, consider two cases:

1. *Agreement ($R_1[j] = R_2[j]$):* $\mu_j = R_1[j] \in \{0, 1\}$, so $\mu_j(1 - \mu_j) = 0$. The within-variance is zero for agreement patterns: both rules produce the same output, all m cells deterministically agree, and there is no variance.

2. *Disagreement ($R_1[j] \neq R_2[j]$):* WLOG $R_1[j] = 0$, $R_2[j] = 1$ (the case $R_1[j] = 1$, $R_2[j] = 0$ is symmetric). Then $\mu_j = \lambda$ and:

$$\mu_j(1 - \mu_j) = \lambda(1 - \lambda) \tag{91}$$

More generally, for any $R_1[j], R_2[j] \in \{0, 1\}$ with $R_1[j] \neq R_2[j]$:

$$\mu_j(1 - \mu_j) = \lambda(1 - \lambda)(R_2[j] - R_1[j])^2 = \lambda(1 - \lambda) \tag{92}$$

since $(R_2[j] - R_1[j])^2 = 1$ when $R_1[j] \neq R_2[j]$.

Combining both cases: $\mathbb{E}[V_j^{t'}] \approx \lambda(1 - \lambda)(R_2[j] - R_1[j])^2$, which establishes Equation 87. □ □

PROPOSITION H.2.2 (WITHIN-VARIANCE IS MAXIMISED AT $\lambda = 0.5$)

Proposition. *For any disagreement pattern j (i.e., $R_1[j] \neq R_2[j]$):*

$$\mathbb{E}[\texttt{within_var}[j]] = \lambda(1 - \lambda) \leq \frac{1}{4} \tag{93}$$

with equality at $\lambda = 0.5$. The function $\lambda \mapsto \lambda(1-\lambda)$ is strictly concave, achieves its global maximum at $\lambda = 0.5$, and is strictly increasing on $[0, 0.5]$.

Proof. $\lambda(1 - \lambda) = \lambda - \lambda^2$. Its derivative is $1 - 2\lambda = 0 \Rightarrow \lambda^* = 0.5$. The second derivative is $-2 < 0$ everywhere, confirming strict concavity and a global maximum at $\lambda = 0.5$. At $\lambda = 0.5$: $\lambda(1 - \lambda) = 0.25 = 1/4$. □ □

Proposition H.2.2 explains the empirical finding that the within-variance signal strength *increases* as $\lambda \to 0.5$, but both-exact accuracy *decreases*. The paradox is resolved below.

THEOREM H.2.3 (ORIENTATION UNRECOVERABILITY AT $\lambda = 0.5$)

Theorem. *For all rule pairs (R_1, R_2) with $R_1 \neq R_2$, all initial configurations $\mathbf{a}^0$, and all orbit realisations $\mathcal{O}$:*

$$P(\mathcal{O} \mid R_1, R_2, \lambda = 0.5, \mathbf{a}^0) = P(\mathcal{O} \mid R_2, R_1, \lambda = 0.5, \mathbf{a}^0) \tag{94}$$

Proof. For the stochastic model, the probability of an orbit decomposes over cells and timesteps. At each cell i and timestep t', the contribution to the orbit probability is:

$$P(a_i^{t'+1} \mid \mathbf{a}^{t'}, R_1, R_2, \lambda) = \begin{cases} \lambda R_2[j_{t',i}] + (1 - \lambda) R_1[j_{t',i}] & \text{if } a_i^{t'+1} = 1 \\ 1 - \lambda R_2[j_{t',i}] - (1 - \lambda) R_1[j_{t',i}] & \text{if } a_i^{t'+1} = 0 \end{cases} \tag{95}$$

For agreement patterns ($R_1[j_{t',i}] = R_2[j_{t',i}]$), the cell probability is independent of λ and identical under (R_1, R_2, λ) and (R_2, R_1, λ).

For disagreement patterns, consider WLOG $R_1[j_{t',i}] = 0$, $R_2[j_{t',i}] = 1$. The cell probability under (R_1, R_2, λ) is:

$$P(a_i^{t'+1} = 1 \mid \mathbf{a}^{t'}, R_1, R_2, \lambda) = \lambda \tag{96}$$

Under (R_2, R_1, λ) (roles swapped), $R_1[j_{t',i}] = 1$, $R_2[j_{t',i}] = 0$:

$$P(a_i^{t'+1} = 1 \mid \mathbf{a}^{t'}, R_2, R_1, \lambda) = \lambda \cdot 0 + (1 - \lambda) \cdot 1 = 1 - \lambda \tag{97}$$

At $\lambda = 0.5$: Equations 96 and 97 both equal 0.5. Since all cell coin flips are independent, the full orbit probability factors as a product over cells and timesteps, and each factor is identical under $(R_1, R_2, 0.5)$ and $(R_2, R_1, 0.5)$. Therefore the product is also identical, establishing Equation 94. □ □

COROLLARY H.2.4

By the same argument as Corollaries H.1.1 and H.1.2, the Bayes-optimal orientation error rate at $\lambda = 0.5$ under the stochastic model is exactly $1/2$, and this bound cannot be improved by any estimator.

THE PARADOX RESOLVED: MAXIMUM SIGNAL, ZERO ORIENTATION INFORMATION

Propositions H.2.1–H.2.2 and Theorem H.2.3 together establish a precise paradox:

- At $\lambda = 0.5$, the within-variance signal $\mathbb{E}[\texttt{within_var}[j]] = 0.25$ is at its maximum for all disagreement patterns. The symmetric noise magnitude $\lambda^* = \min(\lambda, 1-\lambda) = 0.5$ is estimated with high accuracy — the model's prediction $\hat{\lambda}_{\text{pred}} = 0.5$ is correct, with residual variance reflected in the empirical MAE of 0.0664 (Appendix G).
- At $\lambda = 0.5$, the *orientation* of the rule pair (which rule is R_1, which is R_2) is completely unrecoverable from any number of observations.

The within-variance statistic $\lambda(1-\lambda)$ identifies $\lambda^* = \min(\lambda, 1-\lambda)$ — it is symmetric around $\lambda = 0.5$ and takes the same value at λ and $1-\lambda$. By Theorem H.2.3, no additional statistics can resolve the orientation. The model's prediction $\hat{\lambda}_{\text{pred}} = 0.5$ is correct in the sense that $\lambda^* = 0.5$ — but the orientation cannot be appended to this prediction by any algorithm.

H.3 GENERAL IDENTIFIABILITY CONDITION: DISAGREEMENT PATTERNS MUST EXIST AND BE OBSERVED

SETTING

The results in Appendices H.1 and H.2 address the orientation problem at $\theta = 0.5$. This section addresses a logically prior question: under what conditions is the rule pair (R_1, R_2) *itself* identifiable, regardless of θ? We establish a necessary and sufficient condition for distinguishing a pair from single-rule alternatives, then show that the all-agreement case (where $R_1[j] = R_2[j]$ for all observed j) renders the pair unidentifiable no matter how many orbits are observed.

DEFINITIONS

Definition H.1 (Disagreement patterns)**.** *For a rule pair (R_1, R_2), the set of **disagreement patterns** is:*

$$\mathcal{D}_{R_1,R_2} = \{j \in \{0,\dots,7\} : R_1[j] \neq R_2[j]\} \tag{98}$$

*A pattern $j \notin \mathcal{D}_{R_1,R_2}$ is an **agreement pattern**. The pair (R_1, R_2) is called an **all-agreement pair** if $\mathcal{D}_{R_1,R_2} = \emptyset$, i.e., $R_1 = R_2$. Since we always assume $R_1 \neq R_2$, $|\mathcal{D}_{R_1,R_2}| \geq 1$.*

Definition H.2 (Observed disagreement patterns)**.** *Given an orbit $\mathcal{O}$, a disagreement pattern $j \in \mathcal{D}_{R_1,R_2}$ is **observed** if there exists at least one cell-timestep pair (i, t') such that $j_{t',i} = j$. Let $\mathcal{D}^{\mathcal{O}}_{R_1,R_2} \subseteq \mathcal{D}_{R_1,R_2}$ denote the set of disagreement patterns observed in orbit $\mathcal{O}$.*

Definition H.3 (Identifiable pair)**.** *A rule pair (R_1, R_2) is **identifiable from orbit $\mathcal{O}$** if there exists no other rule pair (R_1', R_2') with $(R_1', R_2') \neq (R_1, R_2)$ and $(R_1', R_2') \neq (R_2, R_1)$ such that:*

$$P(\mathcal{O} \mid R_1, R_2, \theta) = P(\mathcal{O} \mid R_1', R_2', \theta) \quad \textit{for some } \theta \in (0,1) \tag{99}$$

PROPOSITION H.3.1 (NECESSARY CONDITION FOR PAIR IDENTIFIABILITY)

Proposition. *If no disagreement pattern is observed in orbit $\mathcal{O}$ (i.e., $\mathcal{D}^{\mathcal{O}}_{R_1,R_2} = \emptyset$), then (R_1, R_2) is not identifiable from $\mathcal{O}$ against any other pair (R_1', R_2') that agrees with (R_1, R_2) on all observed patterns.*

Proof. If $\mathcal{D}^{\mathcal{O}}_{R_1,R_2} = \emptyset$, every cell-timestep pair (i, t') that appears in the orbit has neighbourhood pattern $j_{t',i} \notin \mathcal{D}_{R_1,R_2}$ — i.e., $R_1[j_{t',i}] = R_2[j_{t',i}]$ for all observed tokens. At every such token, the rule output is identical under R_1 and R_2, so the cell transition probability is independent of θ:

$$P(a_i^{t'+1} \mid \mathbf{a}^{t'}, R_1, R_2, \theta) = R_1[j_{t',i}] = R_2[j_{t',i}] \quad \forall\, (i,t') \in \mathcal{O} \tag{100}$$

This probability depends only on the common rule output on observed patterns, not on (R_1, R_2) or θ. Therefore any pair (R_1', R_2') with $R_1'[j] = R_1[j]$ and $R_2'[j] = R_2[j]$ for all observed j produces the same orbit probability:

$$P(\mathcal{O} \mid R_1', R_2', \theta') = P(\mathcal{O} \mid R_1, R_2, \theta) \quad \forall\, \theta, \theta' \tag{101}$$

In particular, the pair (R_1, R_2) is indistinguishable from the single rule h defined by $h[j] = R_1[j] = R_2[j]$ for all observed j (and arbitrary on unobserved j), and from any pair (R_1', R_2') that agrees with h on observed patterns. The pair (R_1, R_2) is therefore not identifiable from $\mathcal{O}$. □ □

THEOREM H.3.2 (SUFFICIENT CONDITION FOR PAIR IDENTIFIABILITY)

Theorem. *Fix a rule pair (R_1, R_2) and assume there exists at least one disagreement pattern $j^* \in \mathcal{D}_{R_1,R_2}$ that is observed sufficiently often in orbit $\mathcal{O}$ (i.e., $C_{j^*}^{t'} > 0$ for many timesteps t'). Then for any $\theta \in (0,1)$, the pair (R_1, R_2) is distinguishable from any single rule h (i.e., a pair (h, h)) in the sense that:*

$$\mathbb{E}_{\mathcal{O}}[\ell(\mathcal{O}; R_1, R_2, \theta) - \ell(\mathcal{O}; h, h, \theta')] > 0 \tag{102}$$

for any h and any θ', where ℓ denotes the log-likelihood.

Proof. At each timestep t' where the disagreement pattern j^* is observed, the expected output fraction under (R_1, R_2, θ) is:

$$\mu_{j^*}(\theta) = \theta R_2[j^*] + (1-\theta) R_1[j^*] \quad \in \big(\min(R_1[j^*], R_2[j^*]),\, \max(R_1[j^*], R_2[j^*])\big) \tag{103}$$

Since $R_1[j^*] \neq R_2[j^*]$, we have $R_1[j^*], R_2[j^*] \in \{0,1\}$ and $\mu_{j^*}(\theta) \in (0,1)$ for all $\theta \in (0,1)$.

Under any single rule h (pair (h, h, θ')), the output at pattern j^* is deterministic: $h[j^*] \in \{0,1\}$. The expected output fraction is $h[j^*] \in \{0,1\}$. The empirical output fraction at j^* across many timesteps converges to $\mu_{j^*}(\theta)$ by the law of large numbers. Since $\mu_{j^*}(\theta) \in (0,1) \neq \{0,1\}$, the log-likelihood of the single rule h at the observed data is:

$$\ell(\mathcal{O}; h, h, \theta') \approx C_{j^*}\big[\mu_{j^*}(\theta) \log h[j^*] + (1 - \mu_{j^*}(\theta)) \log(1 - h[j^*])\big] = -\infty \tag{104}$$

(since $\log 0 = -\infty$ when the observed fraction $\in (0,1)$ but the predicted probability is $\in \{0,1\}$). In contrast, $\ell(\mathcal{O}; R_1, R_2, \theta)$ is finite. The log-likelihood gap in Equation 102 is therefore infinite, establishing that (R_1, R_2) is distinguishable from any single-rule alternative with probability 1 as $C_{j^*} \to \infty$. □ □

COROLLARY H.3.3 (ALL-AGREEMENT PAIRS ARE UNIDENTIFIABLE)

Corollary. *Let (R_1, R_2) be a rule pair where $R_1 = R_2$ on all patterns that appear in the orbit, i.e., $\mathcal{D}_{R_1,R_2}^{\mathcal{O}} = \emptyset$. Then the pair (R_1, R_2) is statistically indistinguishable from the single rule $h \equiv R_1 \equiv R_2$ on the observed patterns, for all $\theta \in (0,1)$. Moreover, the noise parameter θ is also unidentifiable: the orbit distribution is identical for all $\theta \in (0,1)$.*

Proof. From Proposition H.3.1, the orbit probability depends only on the common outputs $R_1[j] = R_2[j]$ on observed patterns. Since all factors in the orbit probability are identical for all θ (the Bernoulli mixing parameter does not appear when $R_1[j] = R_2[j]$), the distribution is θ-invariant:

$$P(\mathcal{O} \mid R_1, R_2, \theta) = P(\mathcal{O} \mid R_1, R_2, \theta') \quad \forall\, \theta, \theta' \in (0,1) \tag{105}$$

This is a stronger statement than Theorem H.2.3 (which only applied at $\theta = 0.5$ for arbitrary pairs): for all-agreement pairs, the orbit is θ-invariant at *all* noise levels simultaneously. □ □

THE OBSERVATION REQUIREMENT

Theorem H.3.2 requires the disagreement pattern j^* to be *observed* in the orbit. For an ECA with a non-trivial initial configuration, whether a disagreement pattern is observed depends on the dynamics. Two failure modes arise:

1. **Unobserved disagreement patterns (fixed-point rules):** If both R_1 and R_2 are Class C1 (null/uniform) rules and the orbit converges to a fixed point before any disagreement pattern appears, then $\mathcal{D}^{\mathcal{O}}_{R_1,R_2} = \emptyset$ even though $\mathcal{D}_{R_1,R_2} \neq \emptyset$. This is the mechanism underlying the poor performance of tempM and stocM on C1×C1 pairs.
2. **Observed but rare disagreement patterns:** If a disagreement pattern j^* appears only in the pre-convergence transient (a few timesteps before the orbit settles), then C_{j^*} is small and Theorem H.3.2's asymptotic argument applies only weakly. The pair is *in principle* identifiable, but estimation variance is high. This is the mechanism underlying the degraded noise estimation for C1 rules in αM and skewM (Appendix G.5).

IMPLICATIONS FOR THE TEST SET

The test set contains 500 rule pairs across all Wolfram class combinations. Inspection of the per-sample test logs confirms:

- **tempM:** 7/10,000 test samples have no disagreement-pattern observations in any of their $k = 8$ orbits (the *no-signal* category). These arise from C1×C1 pairs where both rules converge the orbit to the same fixed point before any disagreement pattern appears — a near-zero-probability event for non-degenerate rule pairs that occurs only through unfortunate initial configurations.
- **stocM:** 0/10,000 test samples have no signal. Because $n = 50$ cells generate independent draws per timestep, the probability that no disagreement pattern is observed in any of the $(t-1) \times n \times k = 199 \times 50 \times 8 = 79{,}600$ cell-timestep observations is negligibly small for any pair with $|\mathcal{D}_{R_1,R_2}| \geq 1$.

The 7 tempM no-signal samples account for only 0.07 percentage points of the gap between perfect accuracy and the observed 89.05%: for these samples, both-exact is 0% and τ MAE defaults to the analytical estimator's output.

NECESSARY AND SUFFICIENT IDENTIFIABILITY CONDITION (SUMMARY)

Combining Proposition H.3.1, Theorem H.3.2, and Corollary H.3.3:

Identifiability condition (necessary and sufficient)

A rule pair (R_1, R_2) is distinguishable from every single-rule alternative if and only if at least one disagreement pattern $j \in \mathcal{D}_{R_1,R_2}$ is observed in $\mathcal{O}$ (i.e., $\mathcal{D}^{\mathcal{O}}_{R_1,R_2} \neq \emptyset$). When this condition holds, the pair (R_1, R_2) is distinguishable from every single-rule alternative with probability approaching 1 as orbit length increases. When this condition fails, neither the pair nor the noise parameter θ can be identified from the orbit, regardless of the model, the number of orbits k, or the orbit length t.

This condition is necessary (Proposition H.3.1) and sufficient (Theorem H.3.2) for distinguishing the pair from single-rule alternatives. Orientation identifiability is an additional constraint (Appendix H.1, H.2) that applies once the pair is identified: even with a well-observed disagreement pattern, orientation is unrecoverable at $\theta = 0.5$. At $\lambda = 0.5$ under stochastic noise, pairs sharing the same agreement bits and disagreement set are additionally indistinguishable from each other (Appendix G.4).

H.4 RULE 204 SPECIAL CASE: CHANGE RATE IS CONSTANT REGARDLESS OF α

SETTING

Rule 204 is the identity rule:

$$R_{204}(a_{i-1}, a_i, a_{i+1}) = a_i \quad \forall\, (a_{i-1}, a_i, a_{i+1}) \in \{0,1\}^3 \tag{106}$$

We prove that under the single-rule noise models (α-asynchronous and s-skewed), the orbit generated by rule 204 is identical for all values of the noise parameter, making the noise parameter fundamentally unrecoverable. The two-rule case (temporal stochastic and stochastic cell-level noise) is addressed in Corollary H.4.5.

THEOREM H.4.1 (ORBIT INVARIANCE UNDER RULE 204 FOR SINGLE-RULE MODELS)

Theorem. *Let $\mathbf{a}^0 \in \{0,1\}^n$ be any initial configuration. Under rule 204 with α-asynchronous or s-skewed noise, for any noise parameter θ, the orbit $(\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^{t-1})$ is deterministic and satisfies:*

$$\mathbf{a}^{t'} = \mathbf{a}^0 \quad \forall\, t' \geq 0 \tag{107}$$

regardless of which cells are selected for update at each timestep.

Proof. We show this for each noise model separately.

Alpha-asynchronous noise. At each timestep t', each cell i is independently updated with probability α and left unchanged with probability $1-\alpha$. For updated cell i:

$$a_i^{t'+1} = R_{204}(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) = a_i^{t'} \tag{108}$$

For non-updated cell i:

$$a_i^{t'+1} = a_i^{t'} \tag{109}$$

In both cases, $a_i^{t'+1} = a_i^{t'}$. Since this holds for every cell i, $\mathbf{a}^{t'+1} = \mathbf{a}^{t'}$. By induction, $\mathbf{a}^{t'} = \mathbf{a}^0$ for all $t' \geq 0$.

s-skewed noise. At each timestep t', a contiguous block of s cells is selected and updated; the remaining $n-s$ cells are unchanged. For each cell i in the update block:

$$a_i^{t'+1} = R_{204}(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) = a_i^{t'} \tag{110}$$

For each cell i outside the block: $a_i^{t'+1} = a_i^{t'}$. Again $\mathbf{a}^{t'+1} = \mathbf{a}^{t'}$ for all t'. □ □

THEOREM H.4.2 (NOISE PARAMETER UNRECOVERABILITY UNDER RULE 204)

Theorem. *Let the single-rule ECA be governed by R_{204} under α-asynchronous or s-skewed noise. For any initial configuration $\mathbf{a}^0$ and any two distinct noise parameters θ, θ':*

$$P(\mathcal{O} \mid R_{204}, \theta, \mathbf{a}^0) = P(\mathcal{O} \mid R_{204}, \theta', \mathbf{a}^0) \tag{111}$$

Proof. By Theorem H.4.1, the orbit is $\mathbf{a}^{t'} = \mathbf{a}^0$ for all t' with probability 1, regardless of θ. The orbit is therefore a deterministic constant sequence:

$$P(\mathcal{O} \mid R_{204}, \theta, \mathbf{a}^0) = \mathbf{1}\big[\mathcal{O} = (\mathbf{a}^0, \mathbf{a}^0, \ldots, \mathbf{a}^0)\big] \tag{112}$$

This probability is either 0 (if the observed orbit is not constant) or 1 (if it is constant), independent of θ. Therefore Equation 111 holds for all θ, θ'. □ □

COROLLARY H.4.3 (CHANGE RATE FORMULA FAILS FOR RULE 204)

The αM hybrid estimator (Equation 4) divides the number of changed cells by the number of cells the predicted rule would change:

$$\hat{\alpha} = \frac{|\{(t', i) : a_i^{t'+1} \neq a_i^{t'}\}|}{|\{(t', i) : R_{204}(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) \neq a_i^{t'}\}|} \tag{113}$$

Under Rule 204, $R_{204}(a_{i-1}^{t'}, a_i^{t'}, a_{i+1}^{t'}) = a_i^{t'}$ for every cell and timestep, so the denominator is zero, and by Theorem H.4.1 the numerator is zero as well. The ratio is undefined, which triggers the neural fallback. The neural fallback receives a constant orbit and returns $\hat{\alpha} = 1.0$ (the synchronous class) for all $\alpha \in [0.1, 0.9]$, yielding the analytically derived MAE of 0.4500 (Appendix G.7).

COROLLARY H.4.4 (s ESTIMATOR FAILS FOR RULE 204)

The skewM hybrid estimator (Equation 7) has the same denominator, which is zero for every orbit under Rule 204, so it falls back to the neural head. By Theorem H.4.1 the orbit is constant, so the fallback receives no information about s and predicts $\hat{s} = 1$ for every pair.

The resulting MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{s=1}^{n}|1-s| = \frac{1}{20}\sum_{s=1}^{20}(s-1) = \frac{0+1+\cdots+19}{20} = \frac{190}{20} = 9.500 \tag{114}$$

which equals the empirically observed MAE exactly (Appendix G.7).

COROLLARY H.4.5 (RULE 204 AS A TWO-RULE PARTNER)

When Rule 204 appears as one rule in a two-rule pair (R_{204}, R_2) with $R_2 \neq R_{204}$ under temporal stochastic or stochastic noise, the pair is *in principle* identifiable as long as disagreement patterns exist between R_{204} and R_2: $\mathcal{D}_{R_{204},R_2} \neq \emptyset$. The disagreement set is:

$$\mathcal{D}_{R_{204},R_2} = \{j \in \{0,\ldots,7\} : j_{\text{centre}} \neq R_2[j]\} \tag{115}$$

where j_{centre} denotes the centre bit of neighbourhood j (since $R_{204}[j] = j_{\text{centre}}$). This set is non-empty whenever $R_2 \neq R_{204}$, because R_2 and R_{204} differ on at least one of the 8 patterns.

However, the noise parameter θ remains harder to identify for (R_{204}, R_2) pairs than for pairs of two chaotic rules, because:

1. Rule 204 produces a constant orbit on its own, contributing zero within-variance and zero per-timestep residual when it fires.
2. Disagreement patterns only generate signal when the current state has the required neighbourhood, which depends on the orbit dynamics driven by R_2.
3. If R_2 is also a Class C1 rule, the orbit may converge to a fixed point where all observed patterns are agreement patterns, collapsing to the unidentifiable case of Corollary H.3.3.

This cascade of identifiability degradations explains why pairs involving Rule 204 — though strictly identifiable under Theorem H.3.2 when R_2 is chaotic — show higher error rates in practice than pairs of two non-degenerate rules.

SUMMARY: RULE 204 IDENTIFIABILITY HIERARCHY

Table 59: Identifiability status for Rule 204 across models and contexts. "Identifiable" means the pair or parameter can in principle be recovered; "Unidentifiable" means no algorithm can recover it from any finite orbit.

Context	Rule identifiable?	Noise parameter identifiable?
Single rule R_{204}, any α	Yes	**No** (Theorem H.4.2)
Single rule R_{204}, any s	Yes	**No** (Theorem H.4.2)
Pair (R_{204}, R_2), $R_2 \neq R_{204}$, $\mathcal{D} \neq \emptyset$	Yes	Yes, in principle
Pair (R_{204}, R_2), R_2 chaotic, $\tau \neq 0.5$	Yes	Yes
Pair (R_{204}, R_2), any R_2, $\tau = 0.5$	Yes (up to orient.)	Magnitude only (Thm. H.1)
Pair (R_{204}, R_{204})	**No**	**No** (Corollary H.3.3)

I LIMITATIONS

Scope of the dynamical system. The benchmark covers one-dimensional, binary, radius-1 ECA on periodic lattices. This setting was chosen because its generative process is fully known, so every rule and noise estimate can be scored exactly and identifiability can be proved (Appendix H). Natural systems are typically higher-dimensional, multi-state and observed only partially; our results establish what is achievable under exact ground truth, not how the architectures transfer to such systems.

Process noise only, from a known family. All four noise models perturb the update process (which cells update, or which rule is applied); none corrupts the observed states themselves, as measurement noise would. Each specialised model is also built for one noise family, which is therefore assumed known at test time: no model infers the noise type, and mixtures of more than two rules are not considered. Identifying the noise family itself, and handling observation noise, are the most direct extensions of this work.

Reliance on the noise model's functional form. The hybrid and analytical estimators exploit the known generative form of each noise process, and they depend on a correctly identified rule. Rule identification itself is learned and does not use this form, and the learned noise heads alone already exceed every LLM baseline (Table 53); but for systems whose noise structure is unknown, only the learned components carry over directly.

Fixed sizes and discrete parameter grids. Models are trained at fixed grid widths and orbit lengths, with α and s on discrete grids. Task difficulty varies smoothly with size (Appendix D.7), but the trained models are not evaluated at other sizes or at off-grid values of α and s.

Intrinsically hard regimes. Accuracy drops at noise 0.5 for the two-rule models, for Class I rules that converge quickly to fixed points, and for rule 204, whose noise parameter is unrecoverable (Appendices G.9 and G.7). These limits stem from the task, but they bound what any estimator, including ours, can achieve on those cases.

Training and selection protocol. The specialised models are reported from a single training seed; uncertainty is quantified with a rule-level bootstrap over the 77 held-out rules rather than across seeds, and retraining one baseline showed run-to-run variation of several points (Appendix D.6). Because tempM and stocM are trained on data regenerated every epoch, their checkpoints are selected on the fixed test set of held-out rules (Appendix C.4); a separate validation set of held-out rules would remove this dependence.

Scope of the LLM evaluation. The evaluation covers eight models under a fixed set of prompts. Fine-tuning uses LoRA with up to $5{,}000$ samples (500 for the complete-input setting) rather than full-parameter training at the scale of the specialised models; the reasoning, tool-use and image-input conditions are run on αECA only (Appendices F.10 and F.11); and frontier results reflect API versions queried between July and August 2026.

BROADER IMPACT

Scientific applications of the ECA inverse problem. Elementary Cellular Automata are used as interpretable models of complex systems in a range of scientific domains. In developmental biology, ECA-like models have been proposed to describe pattern formation in pigmentation and tissue growth, where the local update rule encodes cell-cell signalling and the noise parameter controls stochasticity in signalling fidelity. In epidemiology, SIR-like models on lattices are closely related to ECA with probabilistic update rules, where recovering the effective "infection probability" from observed spread data is precisely the single-rule inverse problem with α-asynchronous noise. In ecology, spatial population dynamics are modelled as CA where the update rule encodes survival and reproduction and the noise parameter encodes environmental stochasticity. In condensed matter physics, lattice models such as the Ising model can be viewed as CA with stochastic updates, and inferring the effective temperature (analogous to our noise parameter) from observed spin configurations is a well-studied inference problem. Our work provides four validated, reproducible architectures for exactly this inverse problem in the binary-state nearest-neighbour setting. Researchers in any of these fields could use our methodology as a starting point, replacing the ECA simulation with their domain-specific dynamics and adapting the noise model to their observational noise structure.

The knowledge–performance gap reveals a fundamental capability limitation of LLMs. The most scientifically significant finding of this paper may not be the performance of the specialised models but the structure of LLM failure. Our knowledge-check evaluation confirms that the LLMs tested have accurate declarative knowledge of all four noise models: they can define α-asynchronous noise, compute its expected change rate, explain the difference between temporal stochastic and stochastic cell-level noise, and correctly answer quantitative questions about identifiability. Despite this knowledge, they fail catastrophically on the identification task (best zero-shot rule accuracy

4.0%; best fine-tuned 14.6% vs αM's 99.22%). This knowledge–performance gap is not explained by missing information (the prompts include orbit displays and full noise model descriptions, and the result is unchanged when the complete two-rule input is shown, Appendix F.9) or by format confusion (fine-tuning reduces parse failures to $<1\%$). The gap is structural: the identification task requires integrating statistical evidence across thousands of spatial observations in a single forward pass, which is a form of systematic spatial causal inference that LLMs are architecturally not designed to perform. This finding is relevant to the broader question of when LLMs are appropriate tools for scientific analysis and when specialised models are necessary. We make our evaluation pipeline and knowledge-check datasets publicly available to support further investigation of this question across other dynamical system classes.

Reproducibility and open science. All code, datasets, model weights, training configurations, and evaluation scripts are released publicly at the repository URL. The complete training configuration (all loss weights, learning rates, epoch counts, warmup schedules, gradient accumulation factors, and seed formulas) is documented in Appendix C to the level of detail required for exact reproduction without access to the original code. All four identifiability limit theorems in Appendix H are formally proved from first principles; none are empirically asserted. The formal proofs establish hard upper bounds on what any model can achieve, giving future work a principled baseline for comparison. We consider the release of formally proved identifiability limits alongside empirical results to be an important contribution to the reproducibility of inverse problem research, where claimed performance limits are often stated informally.

Potential for misuse in inference about human behaviour. The ability to recover the governing rule and noise parameter of a dynamical system from its observed trajectory has dual-use potential. In the specific setting of this paper (binary-state 1D ECA), the practical risk is low: no real human social system is well-modelled by the 256-rule ECA, and the gap between our synthetic models and real social dynamics is large enough that direct application is implausible without domain-specific adaptation. Nevertheless, we identify the following more concerning downstream scenario: if the methodology developed here is extended to more realistic models of social influence (e.g., voter models, threshold models, or opinion dynamics on networks), the resulting inverse problem solver could be used to infer the effective behavioural rules of individuals or communities from observed behavioural traces (e.g., online activity, geographic movement, purchasing patterns). Such inference could be used to identify influential individuals, predict behavioural change, or design targeted interventions without the knowledge or consent of the affected people. We are not aware of any immediate pathway from our specific models to this scenario, but we believe it is important to name it explicitly. Future work extending this methodology to more realistic dynamical systems should include a careful assessment of whether the resulting inverse problem solver could be used to infer information about individuals or groups that they have not consented to share.

Compute and environmental impact. Training all four models required approximately 74 GPU-hours: $\approx$7h for αM and $\approx$9h for skewM (RTX A1000, 16 GB), $\approx$26.6h for tempM and $\approx$31.3h for stocM (H100 NVL, 94 GB). These training times include data generation, training, and final evaluation. The knowledge-check evaluation used the OpenRouter API with an xAI Grok-3 judge; the identification evaluation ran the open-source models locally on an H100 NVL node and the frontier models through the OpenAI and Anthropic APIs. We estimate approximately 50M tokens of API usage across these evaluations. The total carbon footprint is modest by the standards of modern large-scale deep learning: the H100 NVL draws approximately 700 W at peak load, giving a peak energy consumption of approximately $0.7 \times 26.6 + 0.7 \times 31.3 \approx 40$ kWh for the two H100 training runs. We report exact training times and hardware specifications in Appendix C to support transparent compute accounting by future work that builds on this paper.

Assessment of downstream risk. We summarise our assessment of the primary pathways for downstream harm: the direct application of the four trained models to real systems is implausible without substantial re-engineering (the models are trained on synthetic data, are architecturally specific to $r = 1$ binary ECA, and perform poorly on out-of-distribution dynamics); the release of the methodology is a more plausible pathway, which we address by explicitly naming the misuse scenario above and by the hope that the formal identifiability proofs in Appendix H will encourage future inverse-problem researchers to characterise what their models can and cannot infer, limiting

overreach in both scientific claims and practical applications. We do not believe this work requires special safeguards beyond standard open-source licensing and the transparency measures described above.