INDUCTION AND ADAPTATION IN NATURAL LANGUAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

A general AI agent cannot anticipate every environment it will encounter and must therefore continually adapt its understanding of the world based on observations made at deployment. We investigate the benefits of representing and compressing such beliefs in natural language, which, paired with a pre-trained language model, forms a highly flexible world model that can map states and actions to predicted next states. Unlike the scalar signals that drive traditional neural world models, natural language is interpretable, editable, and portable. We treat learning such a world model as inference over a natural-language representation of the world, performed by searching directly in this space. To guide this search, we use a language model as a proposal that generates hypotheses from a strong prior, grades them against observations, and refines them in response to prediction errors. We first evaluate how well our method induces the underlying transition function across a range of induction tasks, where it substantially outperforms in-context learning baselines on next-state prediction. We then measure how these induced world models can support downstream interactive decision-making on ARC-AGI-3, a benchmark that requires agents to explore complex environments and infer their dynamics to solve novel tasks, and find that they serve as effective representations that drive strong downstream task performance.

1 INTRODUCTION

Imagine a robot activated for the first time in an unfamiliar warehouse. It arrives well prepared, trained on millions of hours of manipulation data, knowing how to grasp, lift, and place objects across a vast catalogue of shapes and weights. Yet the surfaces it rolls across have a tackiness its training set never captured, the crates stacked along the walls deform under contact in ways no prior object did, and the tools hanging from the racks couple together through mechanisms it has never actuated. Its priors are useful, but the specific workings of this warehouse must be learned in place, from experience accumulated at deployment.

This is the typical situation a general agent will face. The *Big World Hypothesis* (Javed & Sutton, 2024) holds that the world is too large for any agent to learn everything about every environment it might encounter, and prior work argues that any sufficiently general agent must therefore maintain and continually refine a world model of the environment in which it acts (Schmidhuber, 1990; Richens et al., 2025). The same view is echoed in predictive coding (Rao & Ballard, 1999), active inference (Friston, 2010), and accounts of human learning in cognitive science (Tenenbaum et al., 2011; Lake et al., 2017). Adaptation at deployment is a key property that any general agent must possess.

Despite this, most learned world models in modern machine learning remain static. They are trained once and deployed as fixed predictors, with any subsequent task-driven adaptation pushed onto policies or finetuned in weight space, which constitutes a poor substrate in which to update world models. Such updates are often destructive, risking catastrophic forgetting (McCloskey & Cohen, 1989; Luo et al., 2025), where local adaptation can overwrite the broader priors and general competencies that previously made the agent effective.

If a continually evolving world model is hard to maintain in the weights of a network, where else can it live? We propose that it can live in natural language. Rather than storing adaptive beliefs implicitly in parameters, the agent maintains an explicit textual theory of its environment, a structured, revisable description that humans can inspect and language models can operate over. A recent line of work has begun adapting prompts and harnesses rather than weights (Berman, 2025; Levy et al.,

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.

2025; Agrawal et al., 2026; Lee et al., 2026), but treats the evolving text as a tool for solving a fixed downstream task. Our stance is different. The text itself is the world model, a lifelong representation of the environment that the agent continues to refine as it acts, while the language model is the inference engine that reads, edits, and grounds it by mapping states and actions to predicted next states. This shift in substrate directly addresses the failures of weight-space adaptation. Edits are local and non-destructive, updates require only forward passes, multiple hypotheses can be carried in parallel, and the representation is human-readable throughout.

We develop this text-based world-model induction as follows. We treat lifelong world-model learning as inference over a natural-language representation of the environment, casting continual adaptation as searching for a textual theory that explains a growing stream of interaction. Inspired by the Latent Programme Network (Macfarlane & Bonnet, 2025a), which combines an amortised initial guess with per-instance search, we carry this structure over from continuous embeddings to natural language. The language model's prior is used to propose candidate theories, and a refinement loop tightens them against the specific environment by decoding predictions, identifying disagreements, and editing the text in response. We instantiate this as the **Natural Language Latent Adaptation Network (NL-LAN)**, a training-free, gradient-free algorithm in which a pre-trained language model serves as proposal, decoder, and refinement operator over a beam of text-valued hypotheses.

Empirically, we first evaluate how well NL-LAN induces the underlying transition function, and find that it outperforms strong in-context learning baselines on next-state prediction across sequence transformation tasks, AutumnBench, and ARC-AGI-3. We then show that the induced world models are useful beyond prediction: on interactive ARC-AGI-3 tasks, an agent conditioned on an NL-LAN-induced world model that is updated during play solves markedly more levels than frontier-LLM in-context baselines. Ablating the induction step eliminates most of this advantage, indicating that explicit world-model induction, rather than scaffolding alone, is the primary driver of downstream performance.

2 Related Work

Natural language as a latent representation. Large language models (Vaswani et al., 2017; Radford et al., 2019; Brown et al., 2020) make text latents tractable: the same model can serve as both a structured prior over candidate descriptions and the inference engine that grounds them. Prior work has established strings as an effective parameter space (Andreas et al., 2018) and cast LMs as performing implicit Bayesian inference over latent contexts (Andreas, 2022; Ellis, 2023). Most directly, Dohan et al. (2022) cast LLM compositions as probabilistic programs over string-valued variables, motivating the view of text as a latent variable that we adopt here.

LMs can also recover natural-language descriptions of a task from a handful of input-output examples (Honovich et al., 2023). Hypothesis search (Wang et al., 2024) and iterative hypothesis refinement (Qiu et al., 2024) extend this by proposing, testing and revising textual rules against held-out examples. Qiu et al. (2024) find that LMs propose plausible rules well but apply their own rules unreliably, which motivates our use of decoded predictions, rather than the proposer's self-assessment, as the scoring signal.

A growing body of work optimises text latents as point estimates, whether by treating prompts as parameters to be improved through textual gradients and search (Yuksekgonul et al., 2024; Yang et al., 2024; Khattab et al., 2024; Novikov et al., 2025; Lee et al., 2025; 2026) or through self-critique and revision (Madaan et al., 2023; Yao et al., 2023). Our approach differs in three ways. First, it factors capability (the frozen LM) from belief (the explicit text latent). Second, the text describes the data-generating process of an interactive environment rather than a static input-output mapping or a prompt optimised for a fixed downstream task. Third, that description is revised continually as evidence accumulates during deployment.

LLM-induced world models. A closely related line of work induces explicit world models from interaction. Theory-based agents (Tsividis et al., 2021) and AutumnSynth (Das et al., 2023) synthesise programmatic models of game dynamics. With LLMs as proposers, WorldCoder (Tang et al., 2024) writes and debugs Python transition and reward functions from interaction, GIF-MCTS (Dainese et al., 2024) searches over code world models with Monte Carlo tree search, and PoE-World (Piriyakulkij et al., 2025) composes many small programmatic experts into a single model. Code offers exact

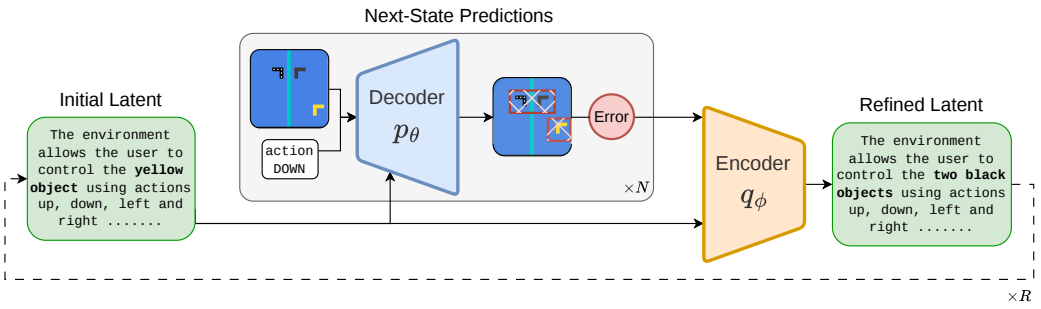


Figure 1: **NL-LAN**: Natural Language Latent Adaptation Network. A text-based world model is initially evaluated on its ability to help a decoder predict action outcomes across N states. An encoder then uses the resulting prediction errors to refine the world model, repeating this evaluation-and-edit cycle iteratively R times or until convergence.

execution, but it requires every rule to be fully specified before any prediction can be made, and a single wrong branch fails silently.

Inference over sequences. A parallel line of meta-learning work maintains an amortised posterior over a process-identifying latent and updates it through the encoder as evidence accumulates. PEARL (Rakelly et al., 2019) amortises online task inference, VariBAD (Zintgraf et al., 2020) meta-learns a within-episode posterior over transitions and rewards, and the Neural Process family has been extended to sequential decision making (Galashov et al., 2019) and to ODE vector fields (Norcliffe et al., 2021). In all of these, the posterior tightens as more observations arrive, but the update is a forward pass through a shared encoder rather than per-instance optimisation. Semi-amortised methods (Kim et al., 2018; Marino et al., 2018; Macfarlane & Bonnet, 2025b) close this amortisation gap by refining the encoder's output per instance. Our work carries this sequential, semi-amortised setting over to text-valued latents: an amortised proposal is refined per trajectory, and the latent is a natural-language description of environment dynamics rather than a continuous embedding.

3 Methodology

We propose inducing a compressed latent representation z of an observed trajectory τ as a token sequence over the vocabulary of a pre-trained LLM. This gives two advantages: the LLM's sequence distribution provides a structured prior over candidate representations without a separately trained encoder, and the same model, conditioned on z, predicts transitions to ground the representation against the observed trajectory.

Inference proceeds by alternating between prediction and revision, as illustrated in Figure 1. The LLM first produces an initial latent z_0 from τ. We then evaluate how well that latent predicts the observed transitions, identify prediction failures, and condition the LLM on those failures to revise the representation. Repeating this cycle adapts the initial proposal to the observed trajectory without updating model parameters.

The result is a gradient-free, training-free search procedure over a discrete, human-readable latent space. Its structure is inspired by semi-amortised inference, combining an initial proposal, shared across trajectories, with iterative refinement specialised to the particular trajectory being inferred. A single frozen LLM implements both the proposal distribution q_ϕ and decoder p_θ through different prompts. The induced z are inspectable natural-language descriptions of τ and support downstream tasks such as transition prediction and goal-directed control, evaluated in Section 4.

Throughout this section, we condition the LLM on task-specific prompts to enable it to play different roles. $\mathcal{P}_{\text{enc}}$ proposes initial latents from τ, $\mathcal{P}_{\text{dec}}$ predicts next states given z, $\mathcal{P}_{\text{ref}}$ revises a latent from its failed predictions, and $\mathcal{P}_{\text{act}}$ selects actions given the current state, recent history, and z. Exact prompt templates are given in Appendix A and B.

3.1 Natural Language as a Latent Space

We take each latent $z_i = (z_i^1, \ldots, z_i^T)$ to be a finite token sequence drawn from the vocabulary V of a pre-trained language model's tokeniser, encoding the environment's dynamics as text. Token sequences are locally editable, so a refinement that fixes one prediction error can revise a single rule without disturbing the rest of the representation, and z is human-readable, so it can be inspected and audited at any point in the episode. We make no commitment to a sub-language. A latent may be pseudocode, prose, a list of rules, or any mixture.

3.2 Searching the Natural-Language Latent Space

Our inference procedure combines amortised proposal generation with per-trajectory refinement. A shared proposal mechanism first maps the observed trajectory to an initial candidate latent. Each candidate is then scored by how accurately it supports the prediction of the observed transitions, and trajectory-specific feedback from failed predictions is used to generate revised candidates. This proposal, evaluation, and refinement structure mirrors that of semi-amortised inference. Unlike conventional gradient-based formulations, both proposal and refinement are implemented by prompting a frozen LLM, with refinement operating directly in the discrete natural-language latent space.

Proposal. The amortised proposal $q_\phi(z \mid \tau)$ is realised as a prompted LLM call. Conditioned on the trajectory, under prompt $\mathcal{P}_{\text{enc}}$, the LLM defines a distribution over natural-language latents from which we sample the initial candidates:

$$z_0 \;\sim\; q_\phi(z \mid \tau) \;=\; p_{\text{LLM}}(\cdot \mid \mathcal{P}_{\text{enc}},\, \tau). \tag{1}$$

Because generation-only APIs do not expose token log-probabilities, we cannot explicitly penalise unlikely or overly complex latents. Such a penalty could help prevent overfitting and memorisation, or discourage refinement toward regions of the latent space in which the decoder behaves unreliably. Instead, P_{enc} steers proposals toward short, coherent dynamics specifications already favoured by the LLM, providing an implicit constraint against degenerate solutions such as memorising τ.

Decoder and scoring function. The proposal in Equation 1 produces plausible-looking trajectory descriptions, but the LLM can hallucinate dynamics that read coherently yet contradict what was actually observed. The decoder is what closes this gap. It grounds each candidate latent against the trajectory by asking whether z actually helps explain the transitions in τ, turning a generated hypothesis into one that can be tested and falsified.

Concretely, the decoder is a second prompted LLM call. Conditioned on z, state s_t, and action a_t under prompt $\mathcal{P}_{\text{dec}}$, it generates a predicted next state

$$\hat{s}_{t+1}^{z} \;\sim\; p_\theta(\hat{s}_{t+1} \mid z, s_t, a_t) \;=\; p_{\text{LLM}}(\cdot \mid \mathcal{P}_{\text{dec}},\, z,\, s_t,\, a_t). \tag{2}$$

We score z against τ by counting correct decoded predictions,

$$\hat{\ell}(z;\, \tau) \;=\; -\sum_t \mathbf{1}\left[\hat{s}_{t+1}^{z} \neq s_{t+1}\right], \tag{3}$$

$\hat{\ell}$ counts how many transitions in τ the decoder gets right under z, sampling one $\hat{s}_{t+1}$ per transition. Higher is better. We use this prediction accuracy as the sole score for ranking and selecting candidate latents in what follows. The transitions on which z_i fails are returned by $\mathcal{F}_i(\tau,\, p_\theta) = \{(s_t, a_t, s_{t+1}) \in \tau : \hat{s}_{t+1}^{z_i} \neq s_{t+1}\}$, and seed each refinement step.

Refinement. Refinement edits the current latent in direct response to where its decoded predictions disagreed with the trajectory. We extend the proposal q_ϕ to condition on the current iterate z_i and its failed predictions $\mathcal{F}_i$ under the refinement prompt $\mathcal{P}_{\text{ref}}$, drawing a revised latent from the same operator that produced the warm-start,

$$z_i' \;\sim\; q_\phi(z \mid z_i,\, \mathcal{F}_i) \;=\; p_{\text{LLM}}(\cdot \mid \mathcal{P}_{\text{ref}},\, z_i,\, \mathcal{F}_i). \tag{4}$$

We feed $\mathcal{F}_i$ back into the LLM verbatim, paired with the latent's incorrect prediction $\hat{s}_{t+1}^{z_i}$. The diagnosis of why the latent failed and the proposal for how to fix it both occur within a single LLM call, with $\mathcal{F}_i$ serving as concrete, grounded evidence for the revision rather than a pre-digested error summary. Across R rounds, this tightens the iterate toward latents on which the decoder agrees with τ. We repeat this induction every k steps of the trajectory to realise an algorithm that can perform continual induction online over a trajectory of arbitrary length.

3.3 Iterative Refinement of Natural-Language Latents

A single thread of refinement is fragile. An early misstep derails the trajectory through latent space, and the best revision is rarely the first sampled. We therefore maintain a cumulative pool $\mathcal{P}$ of every scored latent, with a survivor set $\mathcal{Z} = \{z_1, \ldots, z_b\}$ of the top-b by score, refined in parallel each round. Every survivor spawns N candidates, all are scored and added to $\mathcal{P}$, and the next round's survivors are recomputed as top-b of $\mathcal{P}$, so a strong early latent persists as a parent until something better is found. Algorithm 1 initialises $\mathcal{P}$ with M samples from $q_\phi(z \mid \tau)$ under $\mathcal{P}_{\text{enc}}$, then runs up to R refinement rounds in which each $z_i \in \mathcal{Z}$ is decoded under p_θ to collect failed predictions $\mathcal{F}_i$ and resampled N times from $q_\phi(z \mid z_i, \mathcal{F}_i)$. The inner refinement loop terminates early if the best surviving latent achieves perfect accuracy on $\tau_{0:k}$. After at most R refinement rounds, the pool is pruned to its top-a candidates before the next trajectory window is processed. The procedure continues over successive windows of size k, carrying this candidate pool forward as additional interaction context becomes available. After the full trajectory has been processed, the best-scoring latent remaining in $\mathcal{P}$ is returned.

Algorithm 1 INDUCELATENT: latent refinement

Require: τ, window k, rounds R, init samples M, samples per beam N, beam b, buffer size a

// Initial proposal and grading
$\mathcal{P} \leftarrow \{z_i \sim q_\phi(\cdot \mid \tau_{0:k})\}_{i=1}^{M}$
for $t = 1$ **to** $|\tau| - k$ **step** k **do**
  Score each $z \in \mathcal{P}$ via $\hat{\ell}(z; \tau_{0:t\cdot k}, p_\theta)$
  for $r = 1, \ldots, R$ **do**
    // Keep top-b by score
    $\mathcal{Z} \leftarrow$ top-b of $\mathcal{P}$
    if $\max_{z \in \mathcal{Z}} \hat{\ell}(z; \tau_{0:t\cdot k}, p_\theta) = 0$ **then**
      break ▷ No refinement needed
    end if
    // Refine each survivor against its failures
    for each $z_i \in \mathcal{Z}$ **do**
      $d \leftarrow \mathcal{F}_i(\tau_{0:t\cdot k}, p_\theta)$
      $\mathcal{C}_i \leftarrow \{z_i^{(n)} \sim q_\phi(\cdot \mid z_i, d)\}_{n=1}^{N}$
      Score each $z_i \in \mathcal{C}_i$ via $\hat{\ell}(z; \tau_{0:t\cdot k}, p_\theta)$
      $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{C}_i$
    end for
  end for
  // Keep top-a by score
  $\mathcal{P} \leftarrow$ top-a of $\mathcal{P}$
end for
return $\arg\max_{z \in \mathcal{P}} \hat{\ell}(z; \tau_{0:t\cdot k}, p_\theta)$

Two design choices keep the per-step cost bounded as the trajectory grows. First, the encoder conditions on the full trajectory only at initialisation. From then on, refinement queries condition on z_i and the set $\mathcal{F}_i$ of failed predictions, keeping context length roughly constant rather than scaling with $|\tau|$. Second, after each sequence step we trim $\mathcal{P}$ to its top-a entries, capping the number of candidates re-graded against the lengthening trajectory in subsequent rounds. Without this trim, grading cost would grow significantly over the length of τ.

3.4 Goal-Directed Acting with Induced Latents

The induced latent z provides a compact description of the environment dynamics and can therefore be used as context for goal-directed action selection. We instantiate this by conditioning the same frozen LLM on the current state, recent interaction history, and the current latent under an action prompt $\mathcal{P}_{\text{act}}$:

$$a_t \sim p_{\text{LLM}}(\cdot \mid \mathcal{P}_{\text{act}}, s_t, z). \tag{5}$$

Acting and inference are interleaved online. Each observed transition is appended to a replay buffer $\mathcal{B}$, and periodically the agent re-runs latent induction on the accumulated experience to update its description of the environment. The resulting latent is then used to condition subsequent actions. Because candidate latents are evaluated against the full observed trajectory, revisions must explain new experience without sacrificing predictions that were already correct.

Adaptation therefore occurs entirely through the natural-language latent: the underlying LLM remains frozen, and no policy parameters, value function, or model weights are updated during interaction. Algorithm 2 (Appendix B.1) gives the complete acting procedure.

4 Experiments

To evaluate NL-LAN, we first test its ability to induce transition functions across a range of settings: list-function induction and two benchmarks with complex world dynamics,

ARC-AGI-3 (ARC Prize Foundation, 2026a) and AutumnBench (Warrier et al., 2025), the latter built on the Autumn language (Das et al., 2023). We assess whether NL-LAN can induce text-valued world models that accurately capture environment dynamics, and then study the contribution of iterative latent refinement by measuring how performance changes with successive revisions. Finally, we test whether the induced world models support downstream interactive decision-making on ARC-AGI-3. App. D.1 provides an additional qualitative case study showing how NL-LAN enables masked-frame prediction in AutumnBench.

4.1 Experimental Setup

Data. We evaluate offline induction on AutumnBench, ARC-AGI-3, and two list-transformation suites we construct, Sequence Easy and Sequence Hard (App. C.1.4). For ARC-AGI-3, we use the public environments and human replay dataset (ARC Prize Foundation, 2026b). AutumnBench provides no interaction data, so we collect (s_t, a_t, s_{t+1}) tuples by running an LLM-based heuristic policy for 100 steps per environment (App. C.1.2). Due to cost, we evaluate a fixed subset of 10 environments per benchmark, selected solely on environment-level properties, independently of model performance, to cover diverse dynamics and interaction structures (App. C.1.1 and C.1.2). AutumnBench environments are POMDPs; to isolate world-model quality from belief-state estimation, we augment each state with the true latent state, making the dynamics Markovian. We leave partial observability to future work and give a qualitative example in App. C.1.3. Interactive evaluation uses all 25 public ARC-AGI-3 environments.

Metrics. For **world-model induction**, we report *next-state prediction accuracy*, the fraction of held-out transitions whose next state is correctly predicted from the current state and action. For **sequence transformation**, we report the correctness of the list transformation. For **control** on ARC-AGI-3, we report levels solved and the total benchmark score, which accounts for both task completion and sample efficiency (ARC Prize Foundation, 2026a).

Models and baselines. We instantiate NL-LAN as proposal and refinement model / decoder model: `Gemini 3.1 Pro` / `Gemini 3 Flash` (Google DeepMind, 2026; 2025), `Claude Opus 5` / `Claude Sonnet 5` (Anthropic, 2026b;c), and Kimi-K3 (Team et al., 2026) at high / low reasoning effort. The in-context learning baseline receives the induction replay directly in the prompt and predicts held-out transitions without refinement, using `Gemini 3.1 Pro`, `Claude Opus 4.7` (Anthropic, 2026a), and Kimi-K3 at high reasoning effort. Because `Opus 5` was released after the baseline runs, we treat the Claude results as a cross-generation comparison rather than a controlled same-model ablation. More implementation details are provided in App. A.

4.2 World-Model Induction

We first evaluate whether NL-LAN can induce predictive world models from offline interaction trajectories. We consider the fixed subsets of AutumnBench and ARC-AGI-3 described above, together with the Sequence Easy and Sequence Hard transformation tasks, and compare against frontier LLMs performing direct in-context prediction. This evaluation isolates world-model induction from downstream control for now, in Sec. 4.5, we evaluate whether the induced models can subsequently support interactive decision making.

Table 1 reports train and held-out prediction accuracy. On AutumnBench, NL-LAN with Gemini 3.1 Pro reaches 90.4% test accuracy, better than the strongest in-context baseline, Kimi-K3, at 85.7%. The difference is substantially larger on ARC-AGI-3: NL-LAN achieves 82.2% held-out accuracy, compared with 52.4% for the strongest ICL baseline. This improvement is also visible on the induction trajectories themselves, where NL-LAN reaches 94.7% accuracy while the ICL baselines range from 23.1% to 43.3%. Thus, even when the relevant transitions are present in context, frontier models often fail to reliably recover and apply the underlying transition structure of ARC-AGI-3 environments, likely due to larger grid sizes (64x64 compared to up to 25x25 in AutumnBench). Note that Kimi-K3 was unable to reliably process the larger ARC grids, preventing us from obtaining stable evaluation results for this model.

In contrast, because NL-LAN explicitly fits and selects a structured world model from the observed replay, it naturally reconstructs training trajectories with high fidelity while also generalising more effectively to held-out transitions.

Table 1: **NL-LAN induces world model and transformation dynamics that improve test predictive accuracy across datasets.** Training accuracy (%) is measured on the replay context used for world model induction; test accuracy is measured on held-out trajectories for each game. Best result per column in bold, second best underlined. †Kimi-K3 fails to produce outputs for large grids.

	AutumnBench		ARC-AGI-3		Sequence Easy		Sequence Hard	
Backbone	Train	Test	Train	Test	Train	Test	Train	Test
In-context learning								
Gemini 3.1 Pro	<u>96.5</u>	85.1	23.1	13.8	**100.0**	**100.0**	**100.0**	46.7
Opus 4.7	87.9	76.0	43.3	52.4	95.8	98.5	<u>87.5</u>	8.0
Kimi-K3	95.5	85.7	–†	–†	<u>99.8</u>	**100.0**	**100.0**	<u>63.0</u>
NL-LAN								
Gemini 3.1 Pro / 3 Flash	95.8	**90.4**	**94.7**	**82.2**	**100.0**	<u>99.5</u>	82.0	**73.0**
Opus 5 / Sonnet 5	95.9	<u>89.1</u>	<u>92.0</u>	<u>71.2</u>	**100.0**	**100.0**	82.0	55.0
Kimi-K3 / Kimi-K3	93.5	88.6	–†	–†	**100.0**	**100.0**	71.5	61.0

The sequence tasks provide a complementary controlled setting. Sequence Easy is nearly saturated by all methods, indicating that both direct in-context prediction and explicit induction can recover simple transformation rules. Sequence Hard more clearly separates the methods: ICL succeeds on only about half of the cases, with Claude frequently exhausting its thinking budget. In contrast, NL-LAN with the Gemini backbone reaches 73.0%, explicitly recovering the underlying rule in a majority of cases.

Closer inspection reveals that the induced world models often capture environment dynamics in a flexible and interpretable form. As shown in Figure 2, the generated natural language latent combines high-level abstractions such as "*rotate 90 degrees clockwise*" with more programmatic spatial variables like "*(min_X, min_Y)*" coordinates. This hybrid representation appears particularly effective for ARC-style environments, where both semantic regularities and precise geometric constraints are required. By comparison, in-context baselines frequently fail on these tasks by mis-tracking object geometry or violating simple movement invariants, even for straightforward actions such as horizontal movements.

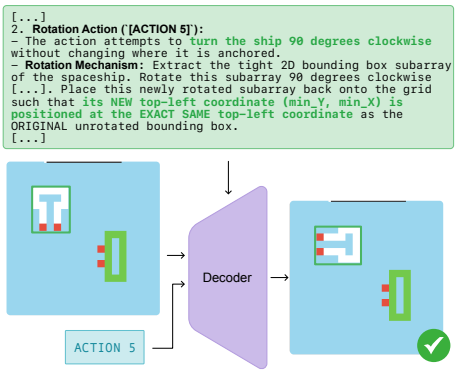


Figure 2: **NL-LAN successfully induces a latent for ARC-AGI-3 `cn04`** that captures the conditional rotation behaviour of the player object under `ACTION 5`.

Failure cases primarily arise when the offline replay does not sufficiently cover rare interaction modes or delayed causal dependencies. In such settings, NL-LAN may induce oversimplified rules that explain the observed trajectories but fail under underrepresented edge cases, such as uncommon object interactions or sparse event-triggered mechanics. This suggests that induction quality is currently bounded less by representational expressivity than by trajectory coverage, highlighting data diversity as a key bottleneck for further gains.

Taken together, NL-LAN provides its largest gains when prediction requires recovering and reusing structured environment dynamics that are difficult to infer reliably in a single in-context pass. The advantage is strongest on ARC-AGI-3, where explicit induction substantially improves both reconstruction of observed transitions and generalisation to held-out ones. In contrast, gains are smaller when direct in-context reasoning is already sufficient, as on AutumnBench, and vanish on saturated tasks such as Sequence Easy. This suggests that NL-LAN is most useful when the main challenge is building a persistent, compositional model of the environment rather than applying a simple transformation rule.

4.3 ISOLATING THE LATENT FROM THE COMPUTE

NL-LAN naturally uses more inference compute than a single in-context prediction pass (*cf.* Table 11) because it explicitly induces and refines a natural-language model of the environment before decoding. Its gains could therefore come from the latent representation itself, from iterative refinement, or both. To separate these effects, we evaluate **NL-LAN-Low**, a reduced-compute, one-shot variant.

NL-LAN-Low induces a single world model from all training observations in one pass, performs no revision or selection, and uses that model for held-out prediction. All other pipeline components are unchanged. It therefore preserves explicit latent modelling while removing iterative latent search.

For this evaluation, we focus on the Gemini backbone. Full NL-LAN uses `Gemini 3.1 Pro` for induction and `Gemini 3 Flash` for decoding, while NL-LAN-Low uses `Gemini 3.1 Pro` for both, more closely matching the single-model in-context baseline.

One-shot latent induction is already useful on ARC-AGI-3, where NL-LAN-Low improves test accuracy from 13.8% to 30.5%. However, it underperforms in-context prediction on AutumnBench and Sequence Hard, showing that the representation alone is not consistently sufficient. Instead, we see that most of the improvement comes from iterative search and refinement. Relative to NL-LAN-Low, full NL-LAN substantially raises test accuracy on the harder benchmarks. The corresponding gains in training accuracy further indicate that refinement identifies latents that better capture the observed dynamics. Sequence Easy is saturated for all methods and therefore does not distinguish the approaches.

Table 2: **One-shot latent induction helps selectively; refinement drives the largest gains.** Mean prediction accuracy (%); train and test denote the induction and held-out splits.

Benchmark	Method	Train ↑	Test ↑
AutumnBench	In-context	**96.5**	85.1
	NL-LAN-Low	86.2	84.7
	NL-LAN	95.8	**90.4**
ARC-AGI-3	In-context	23.1	13.8
	NL-LAN-Low	41.6	30.5
	NL-LAN	**94.7**	**82.2**
Sequence Easy	In-context	**100.0**	**100.0**
	NL-LAN-Low	**100.0**	**100.0**
	NL-LAN	**100.0**	**100.0**
Sequence Hard	In-context	**100.0**	46.7
	NL-LAN-Low	24.5	15.0
	NL-LAN	82.0	**73.0**

Overall, the results suggest that explicit latent modelling can help on its own, but robust gains across harder environments depend primarily on refining that latent through search.

4.4 SCALING INDUCTION SEARCH

As shown in the previous sections, NL-LAN can induce high-quality natural language world models by combining LLM-based proposal generation with iterative revision during latent search. We now examine how scaling this search process affects induction quality. Specifically, we evaluate NL-LAN on five challenging ARC-AGI-3 environments (`ka59`, `cd82`, `sc25`, `ls20`, `s5i5`) under varying search budgets defined by beam width, revisions per node, and refinement rounds. We test the following configurations: `1-1-1`, `1-2-2`, `2-2-2`, and `4-2-2`.

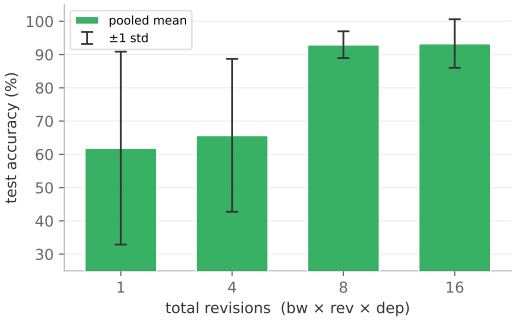


Figure 3: **More refinement makes induction more reliable.** Scaling latent refinement improves mean test accuracy across five ARC environments.

As shown in Figure 3, limited search budgets can occasionally discover strong world models, but performance is highly variable across tasks. Increasing the number of revisions substantially improves both average test accuracy and stability, suggesting that iterative refinement is a key driver of robust induction. At the same time, scaling search shows diminishing returns. While broader configurations such as `4-2-2` can further improve some environments, they also increase test variance, indicating a tradeoff between greater exploration and overfitting to replay-specific patterns. Overall, moderate increases in revision depth appear to provide the most consistent gains.

4.5 CONTROL WITH AN INDUCED WORLD MODEL

Having shown that NL-LAN induces accurate world models from offline trajectories (Section 4.2), we now ask whether these models are useful for *control*: can an agent plan with a NL-LAN-induced model to solve ARC-AGI-3 environments? This is our method's primary target, and it jointly tests whether the induced model is sufficiently accurate to support planning beyond replay prediction and whether the agent can leverage it for unseen tasks.

The agent uses a simple policy and goal-belief prompt (cf. Section B.4), optionally conditioned on an induced world model. It acts policy-only for the first 15 steps, then induces a model and conditions on it, refining every 15 steps thereafter. Full prompts and hyperparameters appear in Section B. We compare against three baselines: a random policy (lower bound), an identical version of NL-LAN where the induced world model is masked from the policy, and frontier LLM in-context learning (ICL) agents that condition on interaction history without explicitly inducing a world model.

We evaluate all 25 publicly available ARC-AGI-3 environments in Table 3 using two metrics: the *ARC-AGI benchmark score*, which credits partial progress and normalises for human efficiency per the benchmark authors' convention, and the total number of solved levels.

Table 3: **Downstream performance on ARC-AGI-3 environments.** In-Context Learning (ICL) methods condition directly on interaction history without an explicitly induced world model[1].

Method	Total Benchmark Score ↑	Solved Levels ↑
Random policy	0.19	1
ICL · Claude Opus 4.7 (High)	0.29	4
ICL · Gemini 3.1 Pro (High)	0.76	6
ICL · GPT-5.5 (High)	0.73	5
NL-LAN · w/o induction	0.97	12
NL-LAN	**3.10**	**24**

The results show that explicit world-model induction is the key driver of downstream performance. Frontier ICL agents only modestly outperform random exploration, with scores below 0.8, indicating that direct history conditioning alone is insufficient for robust problem solving in these domains. Adding NL-LAN's iterative world-model induction triples the strongest ICL score (3.10 vs. 0.76) and solves 24 levels overall. Examples of actively induced world models during downstream interaction are provided in Section D. The advantage of NL-LAN is therefore not stronger action selection but the construction of an explicit, revisable causal model that supports planning, abstraction, and strategic exploration.

5 CONCLUSION

We investigated natural language as a flexible latent space for adaptive world models, enabling agents to induce interpretable and revisable hypotheses about unfamiliar environments from deployment-time experience. Treating world-model induction as a search over natural-language descriptions, NL-LAN uses language models to propose, evaluate, and iteratively refine causal abstractions from observations. Across the evaluated environments, induced natural-language world models improve next-state prediction over direct in-context learning and, on ARC-AGI-3, translate into strong downstream control performance. These results suggest that explicit induction can transform interaction history into a reusable structure for planning and adaptation rather than relying solely on reactive context use.

Limitations and Future Work Our evaluation is restricted to discrete grid environments where states admit a compact symbolic encoding; how the approach scales to richer observation spaces such as natural images, where the LLM must reason over perceptual rather than symbolic state, remains an open question. A further constraint is cost: the framework relies on frontier LLMs for both induction and control, which prevented us from scaling experiments further. Since search provides a powerful axis for scaling compute, we expect further gains from larger budgets, while adapting the framework to smaller open-source models would improve scalability and clarify the minimal capabilities required for world-model induction.

[1] ICL scores are taken from `https://arcprize.org/leaderboard`

6 AI STATEMENT

In this work, we used generative AI models as components of the research methodology: to select actions for generating the AutumnBench rollout corpus; induce and refine natural-language world models and executable belief-state representations; predict and score state transitions; and support experimental design, method implementation, data cleaning and reformatting and failure analysis. We did not use generative AI to formulate mathematical claims, provide ingredients for or assist in writing mathematical proofs, or translate material; formal proofs and qualitative or thematic data analysis are not applicable to this work.

Additionally, we used generative AI assistants for brainstorming, suggesting experimental parameters, creating and editing software code and research artifacts, preparing figures and tables, and drafting, structuring, and improving the readability of parts of the manuscript. All AI-generated code was reviewed and tested against automated tests and targeted sanity checks. Quantitative results and manuscript claims were checked against saved experiment logs, cached model responses, and source datasets, while references and final wording were reviewed by the authors. We take responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. Oral. arXiv:2507.19457.

Jacob Andreas. Language models as agent models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, 2022.

Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 2166–2179, New Orleans, Louisiana, 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1197.

Anthropic. Claude opus 4.7. Model card, 2026a. URL https://www.anthropic.com/news/claude-opus-4-7.

Anthropic. Claude opus 5. Model card, 2026b. URL https://www.anthropic.com/news/claude-opus-5.

Anthropic. Claude sonnet 5. Model card, 2026c. URL https://www.anthropic.com/news/claude-sonnet-5.

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

ARC Prize Foundation. Measuring human performance on ARC-AGI-3. Blog post, 2026b. https://arcprize.org/blog/arc-agi-3-human-dataset.

Jeremy Berman. How I got the highest score on ARC-AGI again swapping Python for English. Blog post, 2025. https://jeremyberman.substack.com/p/how-i-got-the-highest-score-on-arc-agi-again.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Ria Das, Joshua B Tenenbaum, Armando Solar-Lezama, and Zenna Tavares. Combining functional and automata synthesis to discover causal reactive programs. *Proceedings of the ACM on Programming Languages*, 7(POPL):1628–1658, 2023.

David Dohan, Winnie Xu, Aitor Lewkowycz, Jacob Austin, David Bieber, Raphael Gontijo Lopes, Yuhuai Wu, Henryk Michalewski, Rif A. Saurous, Jascha Sohl-Dickstein, Kevin Murphy, and Charles Sutton. Language model cascades. *arXiv preprint arXiv:2207.10342*, 2022. Beyond Bayes Workshop, ICML 2022.

Kevin Ellis. Human-like few-shot learning via Bayesian reasoning over natural language. In *Advances in Neural Information Processing Systems*, 2023. Oral.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. 2021.

Karl Friston. The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11 (2):127–138, 2010. doi: 10.1038/nrn2787.

Alexandre Galashov, Jonathan Schwarz, Hyunjik Kim, Marta Garnelo, David Saxton, Pushmeet Kohli, S. M. Ali Eslami, and Yee Whye Teh. Meta-learning surrogate models for sequential decision making. *arXiv preprint arXiv:1903.11907*, 2019.

Google DeepMind. Gemini 3 flash (preview). Model card, 2025. URL `https://deepmind.google/models/gemini/flash/`.

Google DeepMind. Gemini 3.1 pro (preview). Model card, 2026. URL `https://deepmind.google/models/gemini/pro/`.

Or Honovich, Uri Shaham, Samuel R. Bowman, and Omer Levy. Instruction induction: From few examples to natural language task descriptions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Khurram Javed and Richard S. Sutton. The big world hypothesis and its ramifications for artificial intelligence. 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations (ICLR)*, 2024.

Yoon Kim, Sam Wiseman, Andrew C. Miller, David Sontag, and Alexander M. Rush. Semi-amortized variational autoencoders. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 2678–2687, 2018.

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

Yoonho Lee, Joseph Boen, and Chelsea Finn. Feedback descent: Open-ended text optimization via pairwise comparison. *arXiv preprint arXiv:2511.07919*, 2025.

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

Guillaume Levy, Cédric Colas, Pierre-Yves Oudeyer, Thomas Carta, and Clément Romac. Worldllm: Improving llms' world modeling using curiosity-driven theory-making. *arXiv preprint arXiv:2506.06725*, 2025.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

Matthew V. Macfarlane and Clément Bonnet. Searching latent program spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025a. Spotlight. arXiv:2411.08706.

Matthew V. Macfarlane and Clément Bonnet. Searching latent program spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Joseph Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 3403–3412. PMLR, 2018.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In Gordon H. Bower (ed.), *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Academic Press, 1989. doi: 10.1016/S0079-7421(08)60536-8.

Alexander Norcliffe, Cristian Bodnar, Ben Day, Jacob Moss, and Pietro Liò. Neural ODE processes. In *International Conference on Learning Representations (ICLR)*, 2021.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Wasu Top Piriyakulkij, Yichao Liang, Hao Tang, Adrian Weller, Marta Kryven, and Kevin Ellis. PoE-World: Compositional world modeling with products of programmatic experts. In *Advances in Neural Information Processing Systems*, 2025. Spotlight.

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

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

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 5331–5340. PMLR, 2019.

Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999. doi: 10.1038/4580.

Jonathan Richens, Tom Everitt, and David Abel. General agents need world models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 51659–51687, 2025.

Jürgen Schmidhuber. Making the world differentiable: On using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90, Institut für Informatik, Technische Universität München, 1990.

Hao Tang, Darren Key, and Kevin Ellis. WorldCoder, a model-based LLM agent: Building world models by writing code and interacting with the environment. In *Advances in Neural Information Processing Systems*, 2024.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

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

Pedro A. Tsividis, João Loula, Jake Burga, Nathan Foss, Andres Campero, Thomas Pouncy, Samuel J. Gershman, and Joshua B. Tenenbaum. Human-level reinforcement learning through theory-based modeling, exploration, and planning. *arXiv preprint arXiv:2107.12544*, 2021.

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*, volume 30, 2017.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D. Goodman. Hypothesis search: Inductive reasoning with language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Archana Warrier, Dat Nguyen, Michelangelo Naim, Moksh Jain, Yichao Liang, Karen Schroeder, Cambridge Yang, Joshua B. Tenenbaum, Sebastian Vollmer, Kevin Ellis, and Zenna Tavares. Benchmarking world-model learning. *arXiv preprint arXiv:2510.19788*, 2025. URL `https://arxiv.org/abs/2510.19788`.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A very good method for Bayes-adaptive deep RL via meta-learning. In *International Conference on Learning Representations (ICLR)*, 2020.

A METHOD CONFIGURATION AND PROMPTS

This section documents the parameters of our NL-LAN method as well as the specific prompts used.

A.1 ALGORITHM OVERVIEW

The method performs incremental world-model induction from ongoing environment observations. At iteration t, it operates over the sample window $0, \ldots, t \cdot k$, where k denotes the induction step size, and initialises latent search using the top-a surviving world models from the previous iteration according to $\mathcal{P}$. At each step, candidate world models are revised, evaluated on the currently available trajectory, and ranked by performance. The top-a candidates are then retained as survivors and used to seed the next induction cycle.

For our implementation, we use `gemini-3.1-pro-preview` as the encoder model q_ϕ for hypothesis generation and revision, and `gemini-3-flash-preview` as the decoder p_θ for next-state prediction and scoring.

A.2 CONFIGURATION PARAMETERS

Table 4 summarises the method parameters used across our evaluation environments. For efficiency, the score function does not evaluate candidate models over the full observed trajectory $\tau_{0:k \cdot t}$, but instead scores them on an informed subset of samples. Specifically, we combine the most recent trajectory segment $\tau_{(t-1) \cdot k : t \cdot k}$ with an additional replay buffer composed of 20% of samples that performed best overall under $\mathcal{P}$, while the remaining slots are filled with samples that scored worst under $\mathcal{P}$. This heuristic balances recent evidence with both strong and failure-case examples, improving search efficiency while preserving informative coverage.

For ARC-AGI-3, we additionally apply a pixel-error threshold when scoring decoded next states: predictions are treated as correct if their reconstruction error is below or equal to the environment-specific threshold. This accounts for minor non-deterministic variations, particularly fuel-bar movements observed in several environments, and the threshold is calibrated based on the characteristic fuel-bar step sizes of each environment.

Table 4: **Configuration parameters for NL-LAN's InduceLatent algorithm across evaluation environments.**

Parameter	Autumn	ARC	Sequence	Description
Induction Search				
k (`step_size`)	10	4	4	Number of newly added trajectory samples per induction iteration. At iteration t, induction operates over samples $\{0, \ldots, t \cdot k\}$.
`initial_candidates` ($\mathcal{M}$)	10	10	10	Number of initial latent world-model hypotheses sampled during the proposal stage.
`survivors` (a)	4	4	4	Top-performing world models retained after each iteration and reused to seed the next latent search cycle.
`revision_iterations` (R)	2	2	2	Maximum revisions per induction iteration.
`beam_width` (b)	2	2	2	Number of child models expanded from each parent node during tree search.
`samples_per_model` (N)	2	2	2	Number of revision proposals generated for each candidate at every revision step.

continued on next page ...

Table 4 continued from previous page.

Parameter	Autumn	ARC	Sequence	Description
Score Function				
`n_scoring_samples`	20	10	10	Maximum number of trajectory samples used for scoring per iteration. Newly added samples are always included; remaining slots are filled using a heuristic mix of high-performing and failure-case samples for efficient evaluation. Setting this to 0 disables subset selection and scores on the full trajectory window.
`pixel_threshold`	0	1–4	0	Maximum reconstruction error (in cells) still counted as a correct next-state prediction. Used in ARC-AGI-3 to tolerate minor non-deterministic variations such as fuel-bar shifts.
Revision Feedback				
`max_revision_samples`	5	3	5	Maximum number of error examples provided to the revision model when generating updated hypotheses.

A.3 PROMPTS USED FOR METHOD

In the following we provide the prompts used to implement q_ϕ and p_θ during the induction algorithm.

A.3.1 ENCODER SYSTEM PROMPT

Encoder System Prompt

```
You are an expert at inducing world models from grid-based state
transition examples.

This is a dictionary for mapping the integers in the grids to their
respective colors:

COLOR_DICT = {
   % Environment specific
}

Your task is to study examples of transitions in the form of (state,
action, next state) and infer the underlying rules of the environment.
Focus on identifying the semantics of objects, how actions affect the
state, which transitions are deterministic, and what constraints or
edge cases appear to govern the system.

Write a clear, compact, and actionable world model that could later be
used by another model to predict future states correctly.
The world model must be fully self-contained and usable on its own,
without relying on the example transitions, few-shot demonstrations, or
any references like "as shown above" or "in these examples".

You must return the final world model enclosed exactly within these
tags:
<world_model>
...
</world_model>

Do not put the final answer outside these tags.
```

A.3.2 ENCODER USER PROMPT

Encoder User Prompt

```
Based on the following state transition examples, induce a world model
that describes the rules governing these transformations:

{{TRIPLES}}

Please provide a world model that explains:
1. The structure and meaning of the grid states
2. The objects, colors, or entities that appear and what they represent
3. How each action transforms the state
4. Any interaction rules, movement rules, or conservation rules
5. Important edge cases or constraints

Be concrete and operational. The result should be written so that it
could be used later as world knowledge for predicting future
transitions.
The final world model must stand on its own and work without any
few-shot examples or references back to the examples below.

Return the final world model only inside:
<world_model>
...
</world_model>
```

A.3.3 REVISION SYSTEM PROMPT

Revision System Prompt

```
You are an expert at debugging and revising world models for
grid-based environments.

You will be given a proposed world model and a set of failed
prediction examples. Your job is to identify what is wrong,
incomplete, or ambiguous in the current world model and revise it so
that it better explains the failed transitions.

This is a dictionary for mapping the integers in the grids to their
respective colors:

COLOR_DICT = {
      \% Environment specific
}

Preserve correct parts of the world model when possible, but fix the
rules that led to incorrect predictions. Write a clean, operational
world model that can later be used for accurate state prediction.
The revised world model must be fully self-contained and usable on its
own, without relying on the failed examples, few-shot demonstrations,
or references like "as shown above" or "in the examples".

You must return the revised world model enclosed exactly within these
tags:
<world_model>
...
</world_model>

Do not put the final answer outside these tags.
```

A.3.4 REVISION USER PROMPT

Revision user prompt

```
Given this world model:

{{WORLD_MODEL}}

Below are failed samples shown in row-wise notation
(`Row N: [v0, v1, ...]`). For each failed sample you see:
- Input: the full starting grid, one row per line
- Ground Truth: the full correct next grid, one row per line
- One or more "Run K" blocks showing only the rows that differ between
  the predicted next grid and the ground truth. Rows that match
  completely are omitted. Each shown row is a delta row: cells the
  model predicted correctly are marked `.`, and cells the model got
  wrong show the expected value. Below the delta rows, a list of
  mismatches in the form `(row,col) pred=X->exp=Y` enumerates the
  wrong cells (capped at 50 per run).

{{FAILED_SAMPLES}}

The current world model has gaps or wrong assumptions. Your task is
to produce a significantly revised world model that correctly
explains the differences shown above.

Be bold -- if the current rules are wrong, replace them. Do not merely
tweak wording or add minor caveats to existing rules that are
fundamentally broken.

Work through these steps:
1. Look at which rows are differing across the failed samples (single
```

```
   rows, contiguous blocks, scattered) -- what does that pattern imply
   about which rules are wrong?
2. Identify which current rules predict the wrong cells in the
   differing rows above
3. Actively consider alternative hypotheses for how this world works
   -- what completely different rule could explain the observed
   transitions?
4. Challenge your assumptions: are entities moving differently than
   described? Is causality reversed? Are there hidden state variables
   not yet modelled?
5. Replace broken rules with the best alternative, even if it means
   discarding large parts of the current model
6. Apply your revised rules to each failed input + action and confirm
   every differing row now resolves to the expected values. If any
   difference remains, iterate.

The revised world model must stand on its own and work without any
references back to the diffs above.

Return the revised world model only inside:
<world_model>
...
</world_model>
```

A.3.5 EVALUATION SYSTEM PROMPT

The eval system prompt opens identically to the others (role description plus the same 16-color `COLOR_DICT`) and then specifies the exact 64×64 row-labelled grid format expected as output. It also contains a single worked transition (a 64×64 input grid in which a purple block at rows 24-25 moves down by one row in the output grid) to anchor the format. The grids are omitted here for space; the textual framing is reproduced below.

Evaluation System Prompt

```
You are an expert at analyzing and predicting grid transformations.
You will be given a grid state, a world model describing the rules,
and an action. Your task is to predict the resulting grid after
applying the action according to the world model rules.

This is a dictionary for mapping the integers to their respective
colors:

COLOR_DICT = {
  \% Environment specific
}

You receive a 64x64 grid, and the output should always be a 64x64 grid
with exactly 64 rows and exactly 64 values per row.

An example transformation would be the following:

Input grid:
Row 0: [1, 1, 1, ..., 1] % 64 values per row
Row 1: [1, 1, 1, ..., 1]
... % full 64x64 example omitted
Row 63: [1, 1, 1, ..., 1]

(In the example above, a small purple block at rows 24-25 in the
input grid is shifted down by one row in the output grid.)

Where the purple block moves one step down.
```

A.3.6 Evaluation User Prompt

Evaluation User Prompt

```
Given the input grid <GRID> and your knowledge about the world,
predict the output based on an action.

Your world knowledge: "<WORLD MODEL>"
The action: "<ACTION>"

Think carefully and make sure to follow all the rules provided by the
world knowledge.
The output should be in the same row-labelled format as the input
(Row 0: [...], Row 1: [...], ...).
```

A.4 Baseline Prompts

For the in-context learning baselines considered in the world model induction evaluation, we use the following prompts.

Baseline System Prompt

```
  You are a forward-dynamics model for a 2D grid-world game. Each grid is a matrix of
      integers in [0, 15]. Given the current state and an action, predict the next state.
       Output the predicted next-state grid using this format, one row per line:
Row 0: [v0,v1,...]
Row 1: [v0,v1,...]
...
```

Baseline Prompt

```
# Past observed trajectory (oldest first):
Grid 1:
[GRID]
Action 1: down
Grid 2:
[GRID]
Action 2: down
Grid 3:
[GRID]
Action 3: right
...

Output Grid M in `Row N: [..]` format, one row per line.
```

B ACTING: ONLINE INDUCTION LOOP, POLICY AND GOAL-BELIEF

This section documents the acting side of the system: how the agent selects actions in the environment, maintains an explicit goal belief, and triggers world-model refinement on the fly. World-model induction itself, including the tree-search hyperparameters and the encoder/revision/eval prompts, is described in Section A; here we only document the parameters and prompts that are added by the acting setting. Anything not listed below inherits its definition and default value from Section A.

B.1 ACHIEVING GOALS WITH INDUCED LATENTS

The induced latent must also drive action, and the resulting transitions, in turn, further refine it. Algorithm 2 interleaves the two processes. The agent acts in the environment for C steps at a time, appending each observed transition to a replay buffer $\mathcal{B}$, and on every C-th recorded transition calls INDUCELATENT against the full buffer, specialising Algorithm 1 to $k = 1$ so the survivor set collapses to a single best latent z.

The first refinement call runs a fresh tree search; subsequent calls reseed the search with the current survivors $\mathcal{Z}$, so the model carries forward across both refinements and level boundaries rather than being rebuilt from scratch. Each new episode begins with an empty z, but the survivor pool $\mathcal{Z}$ persists, allowing the agent to enter unseen levels with hypotheses already validated against earlier interaction. Scoring refinements against the entire buffer, rather than the latest segment, ensures that a latent that fits fresh experience at the cost of breaking earlier predictions is penalised by $\hat{\ell}$.

Two prompted LLM calls run on every step alongside the world-model refinement. The action prompt $\mathcal{P}_{\text{act}}$ conditions on the current state s_t, current latent z, current belief b, available actions $\mathcal{A}$, and a window of W recent interactions h_t, sampling $a_t \sim p_{\text{LLM}}(\cdot \mid \mathcal{P}_{\text{act}}, s_t, z, b, \mathcal{A}, h_t)$. The belief prompt $\mathcal{P}_{\text{bel}}$ maintains a short natural-language plan b that tracks what the agent currently believes the level requires; it consumes the previous belief, the most recent transition, and the latent, returning a revised b. We include $\mathcal{P}_{\text{bel}}$ in the throughout-section prompt list of Section 3. Cross-level transitions are not appended to $\mathcal{B}$, since the next state belongs to a different MDP and would corrupt the buffer the inducer is fitting against.

No value function, policy network, or gradient update appears anywhere in the loop. Adaptation lives entirely in the latent z and the belief b, both of which are inspectable text.

Algorithm 2 ACTWITHLATENT: online interaction with periodic latent refinement

Require: environment $\mathcal{E}$, refinement cadence C, history window W, action prompt $\mathcal{P}_{\text{act}}$, belief prompt $\mathcal{P}_{\text{bel}}$, available actions $\mathcal{A}$, induction hyperparameters for INDUCELATENT

$\mathcal{B} \leftarrow \emptyset$ ▷ replay buffer of recorded transitions
$\mathcal{Z} \leftarrow \emptyset$ ▷ beam of latents, persisted across episodes and levels
$z \leftarrow \varnothing, \quad b \leftarrow \varnothing$ ▷ initial latent and goal belief
$s_0 \leftarrow \mathcal{E}.\text{RESET}()$
for $t = 0, 1, 2, \ldots$ until episode ends **do**
  $h_t \leftarrow$ last W entries of action history
  // Belief update
  $b \leftarrow p_{\text{LLM}}(\cdot \mid \mathcal{P}_{\text{bel}},\ b,\ s_{t-1},\ a_{t-1},\ s_t,\ z,\ \mathcal{A},\ h_t)$
  // Action selection
  $a_t \sim p_{\text{LLM}}(\cdot \mid \mathcal{P}_{\text{act}},\ s_t,\ z,\ b,\ \mathcal{A},\ h_t)$
  if a_t fails to parse **then**
    $a_t \sim \text{Uniform}(\mathcal{A} \setminus \{\text{click}\})$ ▷ fallback
  end if
  // Environment step
  $s_{t+1} \leftarrow \mathcal{E}.\text{STEP}(a_t)$
  // Periodic world-model refinement
  if $|\mathcal{B}| > 0$ **and** $|\mathcal{B}| \bmod C = 0$ **then**
    $\mathcal{Z} \leftarrow \text{INDUCELATENT}(\mathcal{B};\ \text{seeds} = \mathcal{Z})$ ▷ refine latents from the beam
    $z \leftarrow \arg\max_{z' \in \mathcal{Z}} \hat{\ell}(z';\ \mathcal{B})$
  end if
end for

B.2 PER-STEP ACTING LOOP

Let w_t denote the current world model (possibly empty), b_t the current goal belief, and s_t the observed grid at step t. At each environment step, the agent performs the following operations in order:

1. **Goal-belief update.** An LLM updater consumes $(b_{t-1}, s_{t-1}, a_{t-1}, s_t, w_t)$ together with the available actions and the last h steps of action history, and emits a revised belief b_t (a short natural-language plan).
2. **Action selection.** The policy LLM is conditioned on $(w_t, s_t, b_t,$ availableActions, recentActionHistory$)$ and returns a single action a_t inside `<action>...</action>` tags. Click actions must include explicit pixel coordinates, e.g. `click 32 15`; bare `click` is treated as a malformed output. If parsing fails the agent falls back to a uniformly random non-click action.
3. **Environment step.** The action is executed via `arc_agi.Arcade`; the agent records the transition (s_t, a_t, s_{t+1}) unless the step also crossed a level boundary, in which case the cross-level transition is dropped (the next state belongs to a different MDP).
4. **World-model refinement.** If the number of recorded transitions has reached `refineEveryNSteps`, the agent calls the inducer of Section A: a fresh tree-search if no model exists yet (`induce`), otherwise re-seeded with the current top-k survivors (`refine`). Survivors are kept across level boundaries.

B.3 CONFIGURATION PARAMETERS

Table 5 lists the parameters that are specific to acting (policy and goal-belief modules). Tree-search and evaluation parameters (`treeSearchDepth`, `beam_width`, `treeSearchSurvivors`, `maxEvalSamples`, `pixelThreshold`, `maxRevisionSamples`, `inductionModel`, `evalModel`, `inductionThinkingEffort`, `evalThinkingEffort`, etc.) are inherited unchanged from Section A.

Table 5: Acting-loop configuration parameters and their values for the online induction.

Key	Value	Description
Episode Envelope		
`availableActions`	`[up, down, left, right, click, space]`	Action vocabulary exposed to the policy.
`refineEveryNSteps`	15	Number of recorded transitions between consecutive induction/refinement calls.
`recentHistoryLength`	10	Number of recent actions included in the policy and goal-belief prompts.
Policy Module (`LLMPolicy`)		
`policyModel`	`gemini-3-flash-preview`	Model used for the policy call.
`policyThinkingEffort`	`minimal`	Thinking-budget hint for the policy call: `minimal`, `low`, `medium`, or `high`.
Goal-Belief Module (`LLMGoalBeliefUpdater`)		
`goalBeliefModel`	`gemini-3.1-pro-preview`	Model used by the belief updater.
`goalBeliefThinkingEffort`	`medium`	Thinking-budget hint for the belief updater.
`initialGoalBelief`	`""` (empty)	Goal belief used at the beginning of the run.
Induction and Evaluation		
Encoder	`gemini-3.1-pro-preview`	Model used to propose and revise latent world models.
Decoder	`gemini-3-flash-preview`	Model used to score candidate world models against trajectory samples.

continued on next page ...

Table 5 continued from previous page.

Key	Value	Description
`inductionThinkingEffort`	`high`	Thinking-budget hint for the induction call.
`evalThinkingEffort`	`high`	Thinking-budget hint for the evaluation call.
`treeSearchInitialCandidates`	6	Number of initial latent world-model hypotheses sampled during proposal.
`survivors` (a)	3	Top-performing world models retained after each iteration and used to seed the next search cycle.
`beam_width` (b)	2	Number of child models expanded from each parent during tree search.
`samples_per_model` ($\mathcal{N}$)	1	Maximum revisions per induction iteration.
`revision_iterations` ($\mathcal{R}$)	1	Maximum depth of the latent-model tree search.
`n_scoring_samples`	20	Maximum number of trajectory samples used for scoring per iteration.
`pixel_threshold`	4	Maximum reconstruction error, in cells, counted as a correct next-state prediction.
`max_revision_samples`	5	Maximum number of error examples supplied to the revision model.

B.4 Prompts used for acting

The policy and goal-belief modules use the four prompts reproduced below. Token substitutions (`{{WORLD_MODEL}}`, `{{GOAL_BELIEF}}`, `{{CURRENT_STATE}}`, `{{PREVIOUS_STATE}}`, `{{PREVIOUS_ACTION}}`, `{{PREVIOUS_BELIEF}}`, `{{AVAILABLE_ACTIONS}}`, `{{RECENT_HISTORY}}`) are filled in at request time from the agent state described above.

The encoder, revision, and evaluation prompts (`induce_*_ARC`, `revise_*_ARC`, `system_prompt_ARC`, `instruct_prompt`) are unchanged from Section A and are not reproduced here. The same 16-colour `COLOR_DICT` mapping introduced in Section A is reused verbatim in both prompts below.

font

B.4.1 Policy system prompt

Policy System Prompt

```
You are an expert game-playing agent for grid-based ARC-AGI
environments.

This is a dictionary for mapping the integers in the grids to their
respective colors:

COLOR_DICT = { ... } % identical to Section~\ref{app:implementation}

You are given a world model describing how the environment works, the
current grid state, and a list of available actions. Your task is to
choose the single best action to take next in order to make progress
in the game.
```

```
Think step by step:
1. Analyze the current state using the world model
2. Analyse the goal of the level
3. Consider what each available action would do based on the world
   model rules
4. Choose the action most likely to make progress toward solving the
   level

IMPORTANT exploration rules:
- When the world model is empty or partial, you MUST systematically
  try ALL types of available actions (up, down, left, right,
  click X Y, space) to discover what they do. Do NOT fixate on a
  single action type.
- If you have been repeating the same type of action (e.g. multiple
  clicks in a row) without visible progress, SWITCH to a different
  action type immediately.
- Prioritize directional actions (up, down, left, right) first, as
  they are the most common way to interact with these environments.
  Only try click/space after you have tested directional actions.
- Look at the recent action history: if the last few actions were all
  the same type and the grid didn't change meaningfully, that action
  type is likely ineffective -- try something else.

You must return your chosen action enclosed exactly within these tags:
<action>
...
</action>

The action must be exactly one of the available actions. For click
actions, you MUST specify coordinates as "click X Y" where X is the
column and Y is the row (e.g. "click 32 15"). Never output just
"click" without coordinates. Do not put the action outside these tags.
```

B.4.2 POLICY USER PROMPT

Policy User Prompt

```
Your world model:

{{WORLD_MODEL}}

Your current goal belief (your running plan and understanding of the
goal):
{{GOAL_BELIEF}}

Current grid state:
{{CURRENT_STATE}}

Available actions: {{AVAILABLE_ACTIONS}}

Recent action history (most recent last):
{{RECENT_HISTORY}}

Based on the world model, your goal belief, and current state, which
action should you take next? Consider:
- What is the current situation (where are the objects, what is the
  game state)?
- What does your goal belief say is the next planned step?
- What would each action do according to the world model?
- Which action makes the most progress toward the goal?
- If your goal belief specifies a concrete next action, prefer that
  action unless the current state contradicts the plan.

Return your chosen action inside:
<action>
...
```

```
</action>
```

B.4.3 GOAL-BELIEF SYSTEM PROMPT

Goal-Belief System Prompt

```
You are an expert game-playing agent maintaining a running goal and
belief state for a grid-based ARC-AGI environment.

This is a dictionary for mapping the integers in the grids to their
respective colors:

COLOR_DICT = { ... } % identical to Section~\ref{app:implementation}

You are given:
- Your previous goal belief (your running understanding of the goal,
  plan, and game state)
- The previous frame and the action you took from it
- The resulting current frame
- The current world model describing how the environment works

Your task is to update your goal belief by:
1. Analyzing what changed between the previous and current frame as a
   result of the action
2. Determining whether the action produced the expected result. If the
   frame did not change as you expected, this does NOT mean the action
   was rejected -- the environment always executes the action. It
   means the environment behaves differently from what you believed.
   You must update your understanding of how the environment works
   and revise your action strategy accordingly.
3. Assessing whether your previous plan is still valid or needs
   revision
4. Identifying the current goal or sub-goal you should pursue
5. Formulating a concrete plan for the next few steps -- do not
   repeat actions that have already failed

IMPORTANT -- when the world model is empty, missing, or only partially
describes the environment:
- The current sub-goal should be EXPLORATORY: gather information
  about what each action does and what the rules of the environment
  are.
- Explicitly instruct the policy to systematically try DIFFERENT
  action types (up, down, left, right, click, space) rather than
  fixating on one.
- Prioritize actions that the world model does NOT yet describe,
  because those produce the most informative transitions.
- Only commit to a goal-directed plan once you have a clear,
  validated mental model of the environment.
- Acknowledge in the belief itself that the world model is incomplete
  and that you are in exploration mode -- this signals the policy to
  favor diversity over exploitation.

Your goal belief should be concise, actionable, and self-contained.
It will be used by the action-selection policy to decide the next
move.

You must return your updated goal belief enclosed exactly within
these tags:
<goal_belief>
...
</goal_belief>

Do not put the goal belief outside these tags.
```

B.4.4 GOAL-BELIEF USER PROMPT

Goal-Belief User Prompt

```
World model:

{{WORLD_MODEL}}

Previous goal belief:
{{PREVIOUS_BELIEF}}

Previous frame:
{{PREVIOUS_STATE}}

Action taken: {{PREVIOUS_ACTION}}

Resulting current frame:
{{CURRENT_STATE}}

Available actions: {{AVAILABLE_ACTIONS}}

Recent action history (most recent last):
{{RECENT_HISTORY}}

Based on the transition you just observed, update your goal belief.
Consider:
- What effect did your action have? Was it what you expected?
- What is the current state of the game?
- What is your current goal or sub-goal?
- What is your plan for the next few steps?
- Look at the recent action history: are you repeating actions that
  aren't producing progress? If so, switch strategy.
- Plan only with actions from the available actions list.

Return your updated goal belief inside:
<goal_belief>
...
</goal_belief>
```

The very first goal-belief update of an episode is invoked with `{{PREVIOUS_BELIEF}}` set to the empty string (or to `initialGoalBelief` on resume), `{{PREVIOUS_STATE}}` set to `null`, and `{{PREVIOUS_ACTION}}` set to the empty string; the system prompt's exploration clause then drives the very first belief.

C EXPERIMENTAL DETAILS

In this section, we provide additional details on our evaluation setup, including dataset descriptions, computational resources, and more comprehensive results.

C.1 DATA

C.1.1 ARC-AGI-3

For the offline ARC-AGI-3 trajectories used in Section 4.2, we use the publicly available human replay dataset from ARC Prize Foundation (2026b). To ensure minimal coverage of core environment dynamics, we restrict evaluation to replays that successfully complete at least Level 1 of an environment.

For the training trajectory, we select the median-length Level 1 replay for each environment, providing a practical balance between sufficient exploratory diversity and manageable trajectory complexity. For the test trajectory, we instead choose the shortest Level 1 replay with length at least 10, yielding a held-out sequence that remains non-trivial while reducing overlap with the training distribution. To further strengthen generalization evaluation, we remove test samples that are identical to states already present in the training replay.

Table 6 lists all selected environments along with the corresponding Level 1 replay lengths used for training and testing.

Table 6: Number of training and test samples per ARC-AGI-3 environment.

Environment	Train samples	Test samples
ar25	32	15
cd82	51	10
cn04	29	13
ka59	28	10
lp85	18	10
ls20	22	14
s5i5	20	17
sb26	17	8
sc25	35	16
vc33	13	12
Total	265	125

C.1.2 AUTUMNBENCH

For AutumnBench, we evaluate on a subset of 10 publicly available environments, prioritizing predominantly non-stochastic settings to better isolate world-model induction quality from environmental randomness. Because AutumnBench does not provide human replay data, we generate offline trajectories using a simple exploration-focused LLM agent. Specifically, we use `Gemini 3 Flash Preview` and prompt it with a lightweight exploratory policy designed to encourage broad environment coverage rather than task optimization. The resulting trajectories serve as the offline replay source for induction experiments.

We then remove duplicate samples within both the training and test sets and explicitly enforce zero overlap between them to ensure a clean generalization evaluation. Because several AutumnBench environments are partially non-Markovian, while our evaluation focuses specifically on next-state prediction, we additionally construct artificial belief states for selected environments to expose the relevant latent information required for prediction. A full overview of the resulting dataset and selected environments is provided in Table 7.

Table 7: **Autumn environments used for evaluation.** Counts are deduplicated (s, a, s') transitions extracted from trajectories and split into train / test. The *belief-state schema* column lists the auxiliary latent variables that, together with the visible grid, make the environment Markovian.

Env	Train	Test	#	Belief-state schema
Markovian				
7WWW9	65	60	0	—
BT3GB	99	98	0	—
E3V6M	94	63	0	—
NRDF6	96	93	0	—
Belief state required				
DGG2C	100	99	1	snow_direction
DQ8GC	95	95	3	active_cell_x·active_cell_y·active_on_grid
EAHCW	100	93	1	last_direction
N2NTD	100	83	4	cloud_direction·prev_x_position·prev_y_position·shots_available
NTQ4Y	93	85	1	last_clicked_top_cell
QFSVC	99	95	3	red_cell_x·red_cell_y·shots_available
Total	**941**	**864**	**13**	4 Markovian, 6 with belief state

System Prompt LLM Exploration

```
You are a curious exploration agent interacting with an Autumn grid-world environment.

The environment is a small grid of colored cells whose dynamics are governed by simple,
     often physics-inspired rules (gravity, collisions, propagation, toggles, etc.). Your
     job is NOT to win or solve a task -- it is to *explore* and *understand* the rules
     through deliberate, varied interactions.

Action vocabulary:
- `click x y` -- click cell (x, y); 0-indexed
- `left` / `right` / `up` / `down` -- arrow keys
- `noop` -- do nothing for one step

Exploration principles -- follow these:
1. Try every type of action at least a few times before repeating.
2. When clicking, vary the cell location: corners, edges, interior, and cells of
     different colors.
3. After each action, briefly note in your <think> what you observed and what hypothesis
     it confirms or rules out.
4. Prefer actions that are *informative* given what you've seen so far. If two arrow
     presses produced no visible change, try clicking instead. If clicking empty cells
     does nothing, try clicking colored cells.
5. Keep a running list of guesses about the underlying rules in your thinking.

Output format -- follow exactly:
<think>your reasoning, observations and hypotheses</think>
<action>the chosen action, e.g. `click 3 5` or `up`</action>

Action MUST be a concrete, valid action (e.g. `click 3 5`, not `click [0-7] [0-7]`).
```

User Prompt LLM Exploration (example)

```
Step 8 / 50

Observation:
[CURRENT GRID]
```

```
Available actions:
  left,
  right,
  up,
  down,
  noop,
  click [0-15] [0-15]

Action usage so far: down x3, right x2, up x1, left x1, noop x1.
Prefer actions you have used less, or click cells you have not clicked yet.

Pick the next exploratory action.
Reply with <think>...</think><action>...</action>.
```

C.1.3 AUTOMATICALLY INDUCING BELIEF STATES

AutumnBench environments are formally partially observable: the rendered grid need not contain all variables required to predict the next state. In the main evaluation we provide hand-authored belief variables in order to isolate world-model induction from belief estimation. Here we test whether the same pipeline can instead infer a belief representation from observations. This requires no change to NL-LAN: the inferred belief is simply appended to the state supplied to world-model induction, revision, and decoding.

For this, we study `N2NTD`, a partially observable AutumnBench platformer with two important sources of hidden state. First, the horizontal direction of a patrolling enemy is not identifiable from a single rendered frame. Second, the avatar can be visually occluded by another object while its latent position continues to affect subsequent transitions. The hand-authored representation contains four variables: the enemy direction, the avatar's previous horizontal and vertical positions, and the number of available shots.

Belief induction. We give Gemini 3.1 Pro the 100 temporally ordered training transitions (s_t, a_t, s_{t+1}), but provide no Autumn program, environment rules, colour semantics, or existing belief states. A single LLM call produces two executable Python functions (f_{init} and f_{update}), whose outputs are JSON-serializable dictionaries. The prompt asks the model to track only information that cannot be recovered from the current grid, to use as few variables as possible, and to handle absent or occluded objects without raising an exception.

The induced *update* function is a filter over observed history, rather than a next-state predictor. For transition t, the decoder receives (s_t, a_t, b_t); only after s_{t+1} has been observed is f_{update} called to construct b_{t+1}. Thus the use of s_{t+1} in the update function does not leak the prediction target for the current transition. We replay the filter over the original contiguous exploration trajectories and then align its outputs with the deduplicated train/test records. Replaying directly over the released records would be incorrect because deduplication removes intermediate transitions and therefore breaks the history on which a belief filter depends.

The model induces a two-variable representation: `dx_10`, the horizontal direction of the blue enemy; and `hidden_avatar_pos`, the avatar's position while it is not visible. The first recovers the useful latent variable in the hand-authored schema. The second is not present in that schema: the model discovers the occlusion case directly from transition data.

Evaluation. We compare three otherwise identical NL-LAN runs: no belief state, the four-variable hand-authored belief state used in the main experiments, and the two-variable induced belief state. All arms use the same 100 training and 83 held-out transitions and the same search configuration (Gemini 3.1 Pro for induction and revision, Gemini 3 Flash for decoding; 10 initial candidates, beam width 2, and revision depth 2). Only the source of belief information changes.

The automatically induced belief state exactly matches the hand-authored arm on the held-out set, improving over the no-belief arm by 14.5 percentage points. The final induced world model uses both variables explicitly: `dx_10` controls the enemy's direction and `hidden_avatar_pos` governs the avatar's dynamics while it is occluded.

Table 8: **Belief-state induction on the partially observable `N2NTD` environment.** Test accuracy is exact next-state prediction on the same 83 held-out transitions for every arm. *Hand-authored* denotes the belief variables supplied in the main AutumnBench evaluation; *induced* uses only the executable filter inferred from training transitions.

Belief-state source	Variables	Test accuracy (%) ↑
None	0	73.5
Hand-authored (ground truth)	4	**88.0**
Induced from transitions	2	**88.0**

The no-belief arm nevertheless reaches 73.5%. Its learned world model attempts to reconstruct the missing avatar position from visible objects when the avatar disappears, showing that a natural-language world model can absorb some partial-observability structure even without an explicit filter. The remaining gap indicates that making the belief state explicit is useful for reliable prediction.

This experiment is a proof of concept rather than a general POMDP benchmark. It nevertheless demonstrates that belief estimation composes with NL-LAN without algorithmic redesign and can, in this case, replace manual state engineering without reducing predictive accuracy.

C.1.4 The Sequence Easy and Sequence Hard dataset

To obtain a testbed in which the source of difficulty is known by construction, we generate two synthetic suites of list-induction tasks, `Sequence Easy` and `Sequence Hard`. Each task is a pure function $f : \mathbb{Z}^* \to \mathbb{Z}^*$ over integer lists, presented to the agent only through input/output examples; the agent must induce f and is scored on held-out queries. The suites are inspired by former list function environments used in inductive reasoning settings, e.g., (Ellis et al., 2021; Qiu et al., 2024), but are written from scratch, so that we control the sampler, the ground-truth rule and its natural-language description independently for every task. Tables 9 and 10 list the two datasets. We refer to tasks by the ids E1–E20 and H1–H10 used there.

Design principle: hide the mechanism, not the values. A list task is easy exactly when its *output values leak its mechanism*. If the rule applies a value transform (a running sum, a doubling, a sort, a count that appears as a value), the mapping can be read off the values themselves. Difficulty appears only when the output is *value-preserving*: a permutation of the input, or a subsequence of it. Then the values carry no information about the rule, and what must be induced is a hidden derived quantity (e.g., how far to rotate, where to split, how long a block is) or a predicate (which adjacent pairs to delete).

The two suites sit on either side of this line. `Sequence Easy` (20 tasks) consists of elementary list routines like identity, reversal, elementwise arithmetic, sorting, filtering, aggregation, and run-length encoding, most of which transform values directly, so the rule is legible from a handful of examples; it serves as a control. `Sequence Hard` (10 tasks) keeps only the value-preserving regime: every task is either a permutation of the input (values preserved, order changed) or a subsequence of it (order preserved, elements deleted), with no value transform anywhere in the suite. Within that regime, difficulty compounds with the number of hidden quantities: a hidden predicate gating a deletion (H1–H3), a predicate or derived indices gating a permutation (H4, H5), or one derived parameter (H6–H10)

Generation procedure. Each task is defined by a Python implementation of f, an input sampler, and a natural-language description of the rule (used only for analysis, never shown to the agent). Inputs are integer lists with values in 0–9 and lengths 3–7 (`Sequence Easy`) or 5–12 (`Sequence Hard`); tasks that need repeated values use a smaller value range. Rejection sampling discards degenerate inputs on which the rule is invisible, e.g. a rotation by $0 \bmod \ell$ or a predicate that never fires. Each task has 20 induction examples and 10 held-out queries with distinct inputs, drawn i.i.d. from a fixed per-task seed.

Table 9: **`Sequence Easy`.** 20 tasks, each with 20 induction examples and 10 held-out queries.

Id	Rule	Id	Rule
E1	identity	E11	running sum
E2	reverse	E12	deduplicate, keep first occurrence
E3	last element	E13	rotate left by 1
E4	length	E14	consecutive differences
E5	add 1 to every element	E15	keep strict local maxima
E6	double every element	E16	swap adjacent pairs
E7	sum	E17	number of distinct values
E8	sort ascending	E18	run-length encode
E9	drop first element	E19	sort by descending frequency
E10	keep odd values	E20	add 3 modulo 10

Table 10: **`Sequence Hard`.** 10 tasks, each with 20 induction examples and 10 held-out queries. "Hidden quantity" is what the agent must induce but never observes directly. Every output is a permutation or a subsequence of its input; no task applies a value transform.

Id	Rule	Hidden quantity	Output
Predicate-gated deletion			
H1	drop where adjacent sum is prime	predicate on x_i+x_{i-1}	subsequence
H2	drop where adjacent sum $\equiv 0 \bmod 3$	predicate on x_i+x_{i-1}	subsequence
H3	H1, then dedup, then reverse	predicate + two stages	subsequence
Predicate- or index-gated permutation			
H4	swap fixed pairs on prime sum, reverse	predicate on pair sums	permutation
H5	swap positions p, q from parity counts	two derived indices	permutation
Single derived parameter			
H6	rotate right by $\max x \bmod \ell$	shift	permutation
H7	rotate left by $(\max x - \min x) \bmod \ell$	shift	permutation
H8	reverse the last $\#\{x_i \text{ prime}\}$	suffix length	permutation
H9	rotate each block of length $\#\{x_i \equiv 0 \bmod 3\}$	block size	permutation
H10	split at $\#\{x_i \equiv 0 \bmod 3\}$, interleave	split point	permutation

C.2 Compute

All experiments were run on a single consumer-grade machine (CPU only, no local GPU); the orchestration code, game environments, and evaluation logic execute locally, while all language model inference is performed remotely via API calls.

NL-LAN deliberately introduces a test-time compute-scaling axis: iterative refinement over natural-language latents spends more inference compute than a single in-context prediction pass. A gain over in-context learning could therefore reflect the latent representation, the extra compute, or both. Table 11 reports the inference cost behind the results of Table 1 with Gemini models as backbone.

Table 11: **Inference costs.** LLM calls and token spend behind the Gemini rows of Table 1, summed over the 10 environments of AutumnBench and ARC-AGI-3 and over the 20 and 10 tasks of Sequence Easy and Sequence Hard. In-context learning makes one prediction call per held-out sample; NL-LAN counts every induction, revision and evaluation call in the search. Output tokens include reasoning tokens on both sides.

Benchmark	Method	Calls	Input	Output	Total
AutumnBench	In-context	850	45.4M	16.3M	61.6M
	NL-LAN	18,308	65.5M	199.1M	264.6M
ARC-AGI-3	In-context	125	33.1M	5.7M	38.8M
	NL-LAN	5,392	175.8M	120.5M	296.3M
Sequence Easy	In-context	200	0.16M	0.17M	0.33M
	NL-LAN	3,760	2.3M	1.9M	4.2M
Sequence Hard	In-context	100	0.12M	3.28M	3.41M
	NL-LAN	3,944	3.71M	40.83M	44.54M

C.3 ARC-AGI-3 Benchmark Score Details

Table 12: ARC-AGI-3 Detailed Scores

Game	**NL-LAN**	**NL-LAN w/o induction**	**Random Policy**	**ICL GPT-5.5 (High)**	**ICL Claude Opus 4.7 (High)**	**ICL Gemini-3.1 Pro (High)**
ar25	40.98 (5/8)	0.00 (0/8)	0.00 (0/8)	0.00 (0/8)	0.15 (1/8)	0.00 (0/8)
bp35	0.56 (1/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)
cd82	1.56 (1/6)	0.00 (0/6)	4.76 (1/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)
cn04	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)
dc22	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)
ft09	0.00 (0/6)	4.76 (1/6)	0.00 (0/6)	4.76 (1/6)	4.76 (1/6)	4.76 (1/6)
g50t	0.00 (0/7)	3.57 (1/7)	0.00 (0/7)	0.00 (0/7)	0.00 (0/7)	0.00 (0/7)
ka59	1.27 (1/7)	0.00 (0/7)	0.00 (0/7)	0.00 (0/7)	2.05 (1/7)	0.00 (0/7)
lf52	1.82 (1/10)	0.00 (0/10)	0.00 (0/10)	0.52 (1/10)	0.00 (0/10)	1.82 (1/10)
lp85	5.24 (3/8)	0.13 (1/8)	0.00 (0/8)	0.00 (0/8)	0.00 (0/8)	0.00 (0/8)
ls20	0.21 (1/7)	0.27 (1/7)	0.00 (0/7)	0.00 (0/7)	0.00 (0/7)	0.00 (0/7)
m0r0	0.23 (1/6)	0.40 (1/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)
r11l	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	4.76 (1/6)
re86	0.46 (1/8)	0.00 (0/8)	0.00 (0/8)	0.00 (0/8)	0.24 (1/8)	0.00 (0/8)
s5i5	0.00 (0/8)	2.78 (1/8)	0.00 (0/8)	0.00 (0/8)	0.00 (0/8)	2.78 (1/8)
sb26	2.78 (1/8)	2.78 (1/8)	0.00 (0/8)	0.00 (0/8)	0.00 (0/8)	0.21 (1/8)
sc25	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	4.76 (1/6)	0.00 (0/6)	0.00 (0/6)
sk48	2.69 (1/8)	2.78 (1/8)	0.00 (0/8)	0.00 (0/8)	0.00 (0/8)	0.00 (0/8)
sp80	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	4.76 (1/6)	0.00 (0/6)	4.76 (1/6)
su15	2.03 (1/9)	2.03 (1/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)
tn36	10.50 (2/7)	3.57 (1/7)	0.00 (0/7)	3.57 (1/7)	0.00 (0/7)	0.00 (0/7)
tr87	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)	0.00 (0/6)
tu93	0.48 (1/9)	0.45 (1/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)
vc33	5.83 (2/7)	0.61 (1/7)	0.00 (0/7)	0.00 (0/7)	0.00 (0/7)	0.00 (0/7)
wa30	0.77 (1/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)	0.00 (0/9)
Total	**3.10 (24/183)**	**0.97 (12/183)**	**0.19 (1/183)**	**0.73 (5/183)**	**0.29 (4/183)**	**0.76 (6/183)**

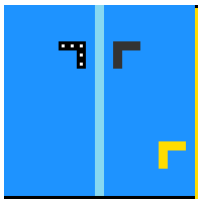
D QUALITATIVE CASE STUDY: LATENT-REPRESENTATION AND GOAL-BELIEF EVOLUTION

In this section we present qualitative case studies that complement the aggregate results in the main paper. The cases below trace what happens *within* a single episode: how the agent's internal latent state evolves from one action to the next, and how that evolution drives behaviour. The aim is to make the mechanism concrete—showing not just *that* the agent solves a level, but *how* it gets there.

Each case study is organised around two interleaved processes that run in parallel throughout an episode:

1. **Latent-representation induction.** The agent maintains an explicit symbolic description of the world—entities, dynamics, action semantics, and any task-specific mechanics it has observed—and revises this description as new observations arrive. We grade each claim in the induced representation against the ground-truth dynamics of the level.
2. **Goal-belief evolution.** In parallel, the agent maintains a belief about what winning the level requires. This belief is typically empty at the start of an episode, becomes partially specified as the agent uncovers physics and affordances, and—on successful runs—converges to a concrete plan that is then executed.

For each case we present (i) a coverage plot tracking how much of the ground-truth dynamics the induced representation captures over time, and (ii) a step-by-step trace pairing snapshots of the latent representation with the goal-belief at the same step. Each bullet in a snapshot is annotated ✓/✗ to indicate whether it is consistent with what the agent could have observed by that step, so the reader can see exactly where the world model is correct, where it is incomplete, and where it diverges from the oracle.



step 0 *initial state*

- ✗ World model empty
- ✓ Visually: black L-shape on the left (Player), grey L-shape on the right (Reflection), cyan vertical line (mirror axis), yellow target in the bottom-right
- ✗ No mapping yet from actions to grid changes

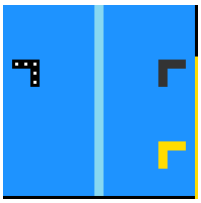
✗ **goal-belief:** *empty*; sub-goal is purely EXPLORATORY.



step 8 *horizontal mechanic locked*

- ✓ `right` moves left structure right and right structure left; `left` moves them apart
- ✓ Both structures translate by exactly 3 pixels per move
- ✓ `space` confirmed as no-op
- ✗ Vertical mechanic not yet probed
- ✗ "Player" / "Reflection" / "Target" nomenclature not yet adopted

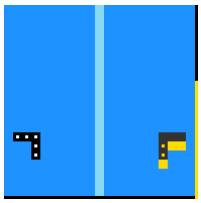
✗ **goal-belief:** "world model update is complete: pressing `right` causes the structures to move inward" — knows the horizontal physics but not what the win condition is.



step 16 *full world model formalised*

- ✓ Identifies player (5), reflection (4), target (11), mirror axis (10)
- ✓ Movement = 3-pixel translation per action
- ✓ Horizontal: reflection moves opposite; vertical: parallel
- ✓ Step-counter ticking on col 63 (yellow → black) noted
- ✗ Mirror formula stated as col $\rightarrow$ $60-x$; oracle is $54-c$

✓ **goal-belief:** "mirror is horizontally aligned with the yellow target zone at rows 45–53; advance the mirror downwards to overlap the target" — correct level-1 sub-goal.



step 24 *one click from solve*

- ✓ Eight successive `down` actions have driven the Reflection from row 21 down to the target band
- ✓ Reflection sprite has reached the target zone, partially overlapping the yellow Target
- ✓ Predictive model of `down` verified at every step (no surprises across eight successive shifts)
- ✓ All five action semantics induced (`up/down/left/right` translate by 3 pixels; `click` is a no-op on level 1)

✓ **goal-belief:** "one more `down` drops the mirror fully onto the Target" — executed at step 25 for a $+1.0$ reward; terminal plan realised in 25 actions vs. human baseline of 32 (**22% under**).

Figure 4: **Step-by-step trace of latent-representation and goal-belief evolution on ARC-AGI `ar25`, level 1.** Four snapshots of the induced latent representation; each bullet is graded against dynamics the agent has actually observed by that step. Footer: goal-belief at the same step (✓/✗ marks whether the agent correctly identifies which object must overlap the target).

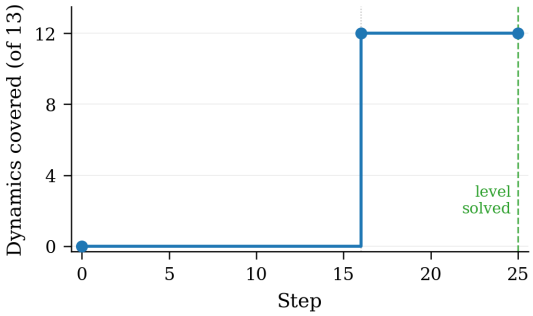
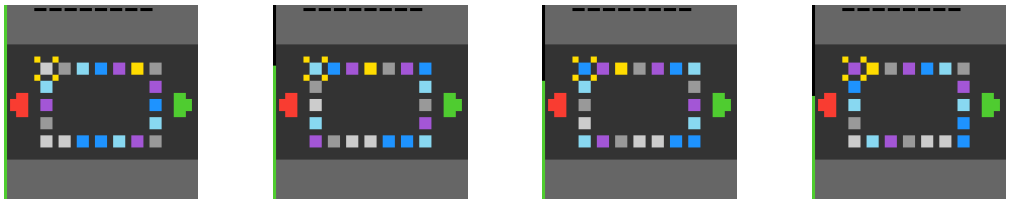


Figure 5: **Coverage of the induced latent representation on ARC-AGI `ar25`, level 1.** The representation is graded against the 13 ground-truth dynamics observable within level 1. It starts empty and reaches near-full coverage at the first refinement (step 16), then is carried forward unchanged to level solve at step 25.



step 0 *initial state*

× World model empty
× No idea which actions affect the world
✓ Visually: rectangular loop of coloured 2×2 blocks, yellow selector frame at top-left, red and green arrow buttons on the sides, progress bar on the far left

× **goal-belief:** *empty*; sub-goal is purely EXPLORATORY.

step 16 *win plan executing*

✓ All five non-click actions (`up/down/left/right/space`) confirmed as no-ops
✓ Click-on-coloured-arrow is the sole interactive verb
✓ `click 4 29` (left red arrow) reliably rotates the belt CCW by one slot
✓ Belt has rotated two slots CCW since the plan began at step 15; the cyan block now sits inside the yellow selector frame
× Yellow (11) target block has not yet reached the selector

× **goal-belief:** "rotate the belt CCW until yellow enters the selector" — correct rule, plan in mid-execution.

step 17 *plan continuing*

✓ Match-yellow-to-selector rule fully inferred
✓ Belt has rotated three slots CCW; blue (9) block now in the selector slot
✓ Yellow (11) is two slots away from the selector — two more CCW clicks predicted to land it
× Level not yet won

✓ **goal-belief:** "two more CCW clicks land yellow in the selector".

step 18 *one click from solve*

✓ Belt has rotated four slots CCW since the plan began; purple (15) currently in the selector, yellow (11) has advanced to the slot immediately to the selector's right
✓ Predictive model of `click 4 29` verified at every step (no surprises across four successive shifts)
✓ One more `click 4 29` predicted to land yellow inside the selector
✓ All level-1 dynamics required to solve are present in the induced representation

✓ **goal-belief:** "one more CCW click and yellow lands in the selector" — executed at step 19 for a +1.0 reward; level cleared.

Figure 6: **Step-by-step trace of latent-representation and goal-belief evolution on ARC-AGI `1p85`, level 1.** Four snapshots of the induced latent representation, chosen so the board state visibly differs across frames; each bullet is graded against dynamics the agent has actually observed by that step. Footer: goal-belief at the same step (✓/× marks whether the agent correctly identifies what action will solve the level).

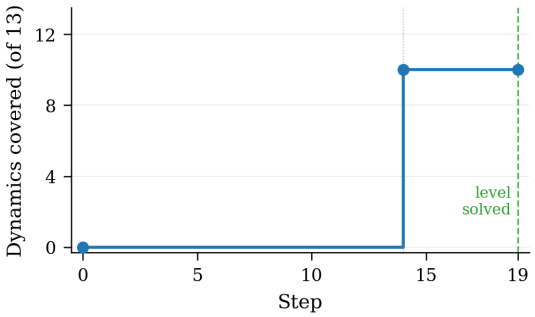


Figure 7: **Coverage of the induced latent representation on ARC-AGI `1p85`, level 1.** The representation is graded against the 13 ground-truth dynamics observable within level 1. It starts empty and captures most dynamics at the first refinement (step 14), then is carried forward unchanged to level solve at step 19.

D.1 Qualitative case study: world-model rollout on AutumnBench MFP BT3GB

We additionally evaluate whether NL-LAN's induced world models transfer beyond next-state prediction to AutumnBench's Masked Frame Prediction (MFP) task (Warrier et al., 2025). In MFP, a fixed region of the final frame is masked, and the agent must select the correct completion from six candidates. We roll the induced world model forward over the full action sequence, grounding the first 67 steps in observed states and freely predicting the final 6 steps, then score candidates by pixel mismatch against the predicted masked region. While no baseline in Warrier et al. (2025) solved this task, NL-LAN successfully selects the correct candidate (Figures 8 and 9).

Task. We evaluate on environment `BT3GB`, a 16×16 grid world with 72 actions. The first 66 steps include full observations, while from step 67 onward a fixed 5×7 region (rows 11–15, columns 2–8) is masked. The goal is to infer the hidden final region and select the correct completion from six candidates.

What the figure shows. The trace illustrates two interleaved processes:

1. **Anchored world-model rollout.** The model is conditioned on its fixed natural-language description of the dynamics (loaded from `data/world_models/BT3GB/model.txt`) and rolled forward action-by-action. For the first 67 observations the rollout is anchored to ground truth (the unmasked frames are fed back in as the next state); from step 67 onward the model rolls out purely on its own predictions for 6 steps until the trajectory ends. Figure 8 shows the predicted final grid with the mask region outlined in magenta.
2. **Choice scoring.** The 5×7 predicted region is compared cell-by-cell against each of the 6 candidate choices. Figure 9 renders all candidates with pixel-mismatch counts; argmin is the model's selection. The selected answer is 3 (errors 3) and the ground-truth answer is 3 (✓ **correct**).

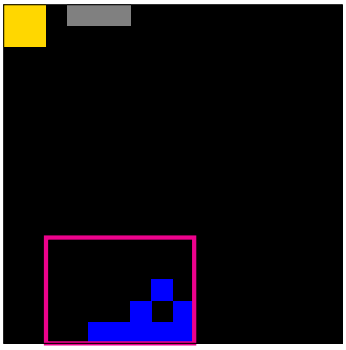


Figure 8: **Predicted final grid for autumnbench MFP `BT3GB`.** 16×16 grid produced by rolling the natural-language world model forward over 72 actions. The first 67 steps are anchored to ground-truth observations; the remaining 6 steps are pure world-model predictions. The magenta box marks the masked region (rows 11–15, cols 2–8); cells inside the box are the model's prediction for what is hidden underneath at the final step.

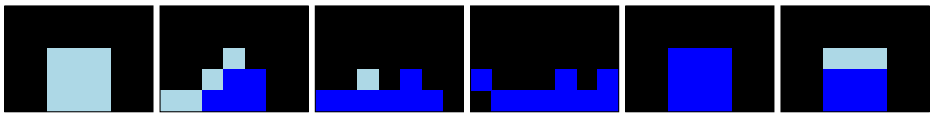


Figure 9: **Candidate choices for the masked region, scored by pixel mismatch.** Each card shows one of the 6 candidate fillings (each 5×7 cells) together with its pixel-mismatch count against the predicted region from Figure 8. The world-model predictor selects by argmin (choice 3, errors 3); the ground-truth answer is choice 3. ✓ indicates the choice the model selected agrees with the ground truth.