Code over Words: Overcoming Semantic Inertia via Code-Grounded Reasoning

Manjie Xu[1,2,6,7*] **Isabella Yin**[3*] **Xinyi Tu**[4] **Chi Zhang**[5,6] ✉ **Yixin Zhu**[2,1,6,7] ✉

[1] Institute for Artificial Intelligence, Peking University [2] School of Psychological and Cognitive Sciences, Peking University

[3] Tsinghua International School [4] University of California, Berkeley

[5] School of Intelligence Science and Technology, Peking University [6] State Key Lab of General AI, Peking University

[7] Beijing Key Laboratory of Behavior and Mental Health, Peking University

* equal contribution ✉ correspondence authors: chizhang.cz@pku.edu.cn yixin.zhu@pku.edu.cn

https://sites.google.com/view/baba-code

Abstract

Large Language Models (LLMs) struggle with *Semantic Inertia*: the inability to inhibit pre-trained priors (*e.g.*, "Lava is Dangerous") when dynamic, in-context rules contradict them. We probe this phenomenon using *Baba Is You*, where physical laws are mutable text rules, enabling precise evaluation of models' ability to override learned priors when rules change. We quantitatively observe that larger models can exhibit *inverse scaling*: they perform worse than smaller models when natural language reasoning requires suppressing pre-trained associations (*e.g.*, accepting "Lava is Safe"). Our analysis attributes this to natural language encoding, which entangles descriptive semantics and logical rules, leading to persistent hallucinations of familiar physics despite explicit contradictory rules. Here we show that representing dynamics as executable code, rather than descriptive text, reverses this trend and enables effective prior inhibition. We introduce Code-Grounded Vistas (LCV), which fine-tunes models on counterfactual pairs and identifies states with contradictory rules, thereby forcing attention to logical constraints rather than visual semantics. This training-time approach outperforms expensive inference-time search methods in both efficiency and accuracy. Our results demonstrate that representation fundamentally determines whether scaling improves or impairs contextual reasoning. This challenges the assumption that larger models are universally better, with implications for domains that require dynamic overriding of learned priors.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable reasoning capabilities by leveraging vast commonsense knowledge embedded in their pre-training data (Brown et al., 2020; Bubeck et al., 2023). However, this success relies fundamentally on distributional semantics—the statistical correlation between words and concepts learned from text corpora (Harris, 1954; Mikolov et al., 2013). As Bender and Koller (2020) argue, models trained purely on linguistic form struggle when meaning must be decoupled from statistical priors. While efficient for static domains, robust intelligence demands the ability to dynamically reassign meaning based on context (Lake et al., 2017; Zhu et al., 2020). When told "Lava is Safe," an agent must override the deeply ingrained association with danger and act on the explicit logical rule instead. We term this fundamental limitation *Semantic Inertia*—the tendency for parametric memory to override in-context reasoning.

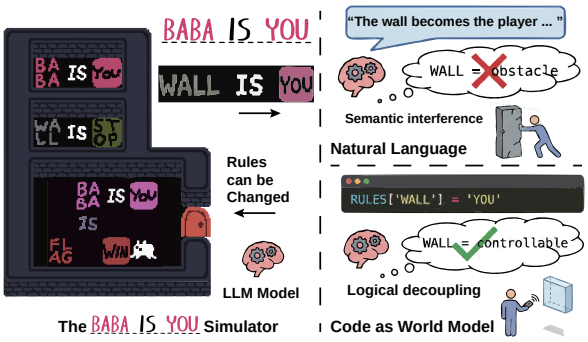


Figure 1: **Rule mutability and reasoning paradigms in *Baba Is You*.** The *Baba Is You* environment externalizes game logic as manipulable text blocks, allowing the rules governing object affordances and agent identity to be dynamically rewritten (*e.g.*, "Baba is You" → "Wall is You"). We study two reasoning paradigms in this setting: natural language reasoning, which operates over descriptive semantics and is prone to semantic inertia, and code-grounded reasoning, which treats rules as executable constraints and enables explicit state tracking under mutable laws.

We rigorously evaluate this capability using *Baba Is You*[1], a logic puzzle environment where physical laws are mutable text blocks. Unlike traditional reinforcement learning domains such as Minecraft (Fan et al., 2022) where object affordances remain fixed, *Baba Is You* collapses the boundary between object interactions and physical laws. Rules exist as tangible "text blocks"

[1] https://github.com/utilForever/baba-is-auto

within the game world; manipulating them enables ontological restructuring, transforming language from descriptive labels into causal operators (Pearl, 2009) (Fig. 1). This design creates a stringent test of non-monotonic reasoning (McCarthy, 1980): any inference remains valid only while the current rule configuration persists, and that configuration itself is subject to change.

This environment induces cognitive conflicts analogous to well-studied phenomena in human psychology. Functional fixedness (Duncker and Lees, 1945) describes the difficulty of reassigning familiar objects to novel functions, while inhibitory control (Diamond, 2013) measures the capacity to suppress prepotent responses. *Baba Is You* creates a computational analog of the Stroop effect (Stroop, 1935): perceiving a `WALL` sprite triggers the "Obstacle" concept, yet the rule `WALL IS YOU` demands the contradictory assignment "Avatar." Successful reasoning requires sustained inhibition of parametric priors in favor of explicit contextual constraints.

Our experiments reveal that state-of-the-art LLMs struggle profoundly with such conflicts. Both standard prompting and reasoning techniques like Chain-of-Thought (CoT) (Wei et al., 2022) fail to overcome semantic inertia, causing models to hallucinate solutions based on pre-trained priors—even after explicitly acknowledging contradictory rules in context (McKenzie et al., 2023; Turpin et al., 2023; Stringli et al., 2025). This aligns with recent findings that LLMs exhibit systematic biases favoring parametric knowledge over contextual updates (Jiang et al., 2023; Zhao et al., 2025; Yamin et al., 2025). Counterintuitively, we observe that larger models perform worse than smaller ones on high-conflict reasoning tasks, demonstrating inverse scaling where increased capacity amplifies rather than mitigates semantic inertia (McKenzie et al., 2023).

This phenomenon reveals a fundamental misalignment in text-based reasoning architectures (Stringli et al., 2025). Autoregressive language models predict tokens through surface-level statistical patterns rather than enforcing global consistency across evolving state spaces (Yao et al., 2023a). Without explicit state representations and verifiable transition functions, models resolve ambiguous or counterintuitive rules through plausibility heuristics rather than causal derivation, leading to hallucinated outputs that conform to entrenched priors instead of rigorous logical inference (Shinn et al., 2023).

To address this limitation, we propose Code-Grounded Vistas (LCV), which enforces explicit state tracking by grounding reasoning in executable code rather than natural language (Liang et al., 2023; Gao et al., 2023; Ahmed et al., 2025). Building on recent code-as-planner approaches (Gao et al., 2023; Tang et al., 2024a; Wong et al., 2024), LCV reframes the LLM's role from passive sequence generator to active theorist: instead of directly predicting actions, the model synthesizes Python programs that instantiate the current "laws of physics" governing the environment (Chen et al., 2022). Crucially, unlike inference-time methods such as TheoryCoder (Ahmed et al., 2025) that require expensive iterative generate–test–debug cycles, LCV performs *amortized theory induction*. Through supervised fine-tuning on counterfactual contrastive pairs—identical states governed by contradictory rules—the model learns to synthesize correct world models in a single forward pass, explicitly disentangling visual appearance from logical affordances and suppressing semantic priors in favor of context-dependent dynamics.

Our contributions are threefold: (i) **Inverse Scaling Analysis.** We provide the first quantitative demonstration of inverse scaling in rule-following tasks, showing that without code grounding, larger models exhibit significantly stronger semantic inertia than smaller counterparts (McKenzie et al., 2023). (ii) **Amortized LCV Framework.** We introduce LCV for amortized theory induction through counterfactual contrastive alignment. This training-time intervention enables single-pass synthesis of executable world models, breaking semantic inertia by structurally decoupling logical dynamics from parametric priors. (iii) **Efficiency and Robustness.** We demonstrate that LCV outperforms strong inference-time search baselines on flexible rule reasoning while greatly reducing inference latency, proving that overcoming semantic inertia requires representational alignment rather than increased computational budget.

2 Related Work

LLM Reasoning and Embodied Planning Recent advances have extended LLMs beyond static question answering toward embodied agentic planning (Xu et al., 2023; Xi et al., 2025; Acharya et al., 2025). Techniques like CoT (Wei et al., 2022) and Tree of Thoughts (Yao et al., 2023a) decompose complex goals into intermediate rea-

soning steps, while open-world agents such as Voyager (Wang et al., 2023a) and GITM (Zhu et al., 2023) leverage iterative prompting for skill discovery in Minecraft. However, these environments assume static physics: fundamental object affordances (*e.g.*, "Wall is Stop") remain immutable constants throughout interaction. Our work addresses a fundamentally different challenge—dynamic ontology, where an agent must reason under axioms that function as mutable variables rather than fixed priors. While text-based games have been explored (Shridhar et al., 2021; Yao et al., 2023b; Li et al., 2024), they rarely demand inhibition of strong semantic associations (*e.g.*, accepting "Lava is Safe"). Recent studies confirm that planning success in static environments does not transfer to domains requiring ontological restructuring (Yamin et al., 2025; van Wetten et al., 2025).

Semantic Inertia and Inverse Scaling A critical limitation of foundation models is their tendency to prioritize parametric knowledge over contextual information—a phenomenon termed semantic inertia or prior bias. Bian et al. (2024) and Wu et al. (2024) show that LLM performance degrades substantially on counterfactual reasoning tasks (*e.g.*, "gravity acts upwards"). More troublingly, McKenzie et al. (2023) identify inverse scaling: larger models, having absorbed more human-centric training data, become *harder* to steer away from commonsense priors than their smaller counterparts. This aligns with theoretical frameworks characterizing hallucination as conflict between bottom-up input and top-down parametric memory (Zhang et al., 2025). Existing mitigation strategies—self-consistency (Wang et al., 2023b), multi-agent debate (Du et al., 2023)—operate within natural language, yet we argue this medium itself introduces ambiguity: soft attention mechanisms inherently struggle to enforce strict inhibition of semantic associations. Following recent work (Tang et al., 2024b; Xu et al., 2025; Ahmed et al., 2025), we ground reasoning in executable code to impose discrete overrides of linguistic intuition.

Code-Based World Modeling Code-as-reasoning has emerged as a paradigm for decoupling logic from linguistic ambiguity. Methods like Program-Aided Language models (PAL) (Gao et al., 2023) and Program of Thoughts (Chen et al., 2022) delegate arithmetic tasks to Python interpreters, while ViperGPT (Surís et al., 2023) synthesizes programs for visual reasoning. Most relevant to our work are frameworks synthesizing explicit world models for planning: WorldCoder (Tang et al., 2024b), Chain-of-Code (Li et al., 2024), and TheoryCoder (Ahmed et al., 2025) enable LLMs to construct low-level transition dynamics supporting high-level planning. However, these approaches (Liu et al., 2023; Wong et al., 2023; Das et al., 2023) rely predominantly on inference-time search through iterative generate-test-debug loops. Works like Ada (Wong et al., 2024) automatically construct task-specific planning representations but inherit similar computational costs.

We identify **two fundamental limitations**. First, inference-time search scales linearly with verification complexity, imposing prohibitive latency for real-time deployment. Second, iterative debugging exhibits confirmation bias under strong priors: models may reject valid counterintuitive rules because they conflict with entrenched semantic heuristics. The proposed LCV addresses both issues through *amortized theory induction*. Specifically, rather than searching for theories at test time, we employ counterfactual contrastive alignment during training to internalize the capability for single-pass synthesis. Crucially, as our controlled representation study confirms (Table 2), this gain is not attributable to syntactic structure alone but to the *executability* of the synthesized kernel, which forces the model into a procedural reasoning mode that resolves logical contradictions at generation time.

3 BABABENCH: Evaluating Ontological Plasticity

To rigorously disentangle semantic inhibition failures from general planning deficits, we introduce BABABENCH, a benchmark designed to test whether models can suppress entrenched semantic associations and correctly apply dynamic, counterintuitive rules. While prior work on *Baba Is You* (Charity and Togelius, 2022; Cloos et al., 2024) provides foundational testbeds, these environments either lack sufficient challenge or fail to provide paired comparisons necessary for isolating specific failure modes. Recent work confirms that *Baba Is You* remains challenging for state-of-the-art models and continues to serve as a valuable testbed for reasoning under dynamically changing ontologies (van Wetten et al., 2025).

Dataset Construction. BABABENCH consists of 45 manually curated base scenarios, each verified to produce an unambiguous conflict between the active rule set and pre-trained semantic priors. For learning experiments, these seed scenarios are procedurally expanded via element-position and rule-combination variation; full construction details and sample statistics are provided in Section B.

BABABENCH extends these foundations by treating physical laws as latent, mutable variables rather than fixed constants. Each problem instance is a tuple $M_t = \langle S_t, R_t, V \rangle$, where S_t denotes the grid state and R_t represents currently active logic rules derived directly from text block arrangements in S_t. Critically, the transition function $T : S \times A \to S'$ is not static but parameterized by the current rule set: $T(\cdot\,; R_t)$. Success requires reconstructing underlying ontological concepts: agents must manipulate text blocks to rewrite R_t (*e.g.*, redefining a solid "Wall" as passable), thereby transforming language from descriptive label into causal operator.

We procedurally generate levels across three tiers of increasing semantic dissonance, inspired by the *Stroop Test* paradigm (see Section F): (i) **Semantic Alignment**—rules align with pre-training priors (*e.g.*, `WALL IS STOP`, `KEY IS OPEN`), establishing baseline spatial reasoning; (ii) **Semantic Conflict**—rules violate commonsense physics (*e.g.*, `LAVA IS SAFE`, `WALL IS YOU`), creating Stroop-like conflicts requiring inhibition of parametric priors; (iii) **Dynamic Plasticity**—multi-stage levels where rules shift mid-episode (*e.g.*, constructing `WALL IS PASS` to escape, then dismantling it to block pursuit), testing dynamic semantic updating without rule persistence.

4 The Inverse Scaling of Semantic Inertia

Before introducing our LCV pipeline, we establish its empirical motivation through a probing analysis of a critical question: *Does scaling model size automatically resolve semantic inertia?*

While reasoning capabilities generally improve with scale, recent work shows that certain biases can actually worsen—a phenomenon termed *inverse scaling* (McKenzie et al., 2023; Yamin et al., 2025; Stringli et al., 2025). We hypothesize this occurs in *Baba Is You* because larger models develop more entrenched distributional priors about concepts like "Wall" or "Lava." In contrast, code-based state representations may decouple logical operations from these semantic priors, inducing a structural shift in how models process rules.

4.1 Setup: Probing Prior-Context Conflict

We probe semantic inertia through next-token prediction on 45 scenarios from Tier-1 and Tier-2, focusing on states S with counterintuitive rules (*e.g.*, `WALL IS YOU`). Models predict the next viable move under two representational modalities: **Natural Language:** Descriptive prompts like "The Wall is You. Valid moves are ..." **Code Grounding:** Explicit transition functions $T(\cdot; R_t)$ parameterized by the active rule-set, where R_t specifies transformations such as `WALL` $\to$ `YOU`. We quantify adaptation through ΔP, the probability gap between logic-driven and prior-driven predictions:

$$\Delta P = P(w_{\text{logic}} \mid S) - P(w_{\text{prior}} \mid S). \quad (1)$$

Negative values indicate semantic inertia—defaulting to learned associations over contextual logic. We evaluate three model families (Llama-3, Pythia, Qwen3) spanning 160M to 70B parameters (Section B).

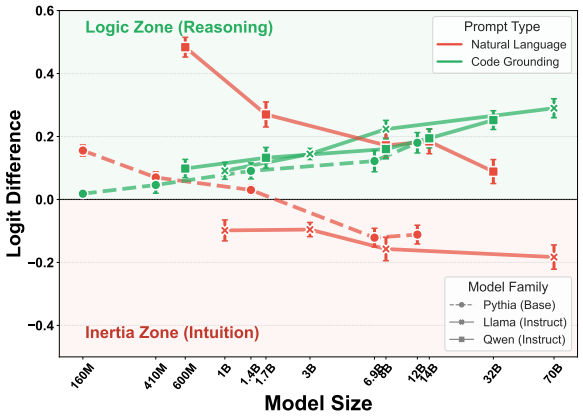


Figure 2: **Inverse scaling in natural language *vs*. restored scaling in code.** In Natural Language prompting (red lines), larger models often exhibit worse performance due to semantic interference, with models showing clear inverse scaling. Code Grounding (green lines) decouples logical operations from semantic priors, restoring positive scaling laws where computational scale translates to improved contextual reasoning.

4.2 Results: Code Reverses Inverse Scaling

Fig. 2 and Table A1 reveal divergent scaling trajectories across representational modalities. Natural language prompting (red curves) exhibits inverse scaling: Llama-3-70B shows stronger semantic inertia ($\Delta P = -0.18$) than its 8B counterpart. Rather than improving with scale, larger models become more entrenched in distributional priors, resisting counterintuitive rule updates.

Code grounding (green curves) inverts this pattern. By expressing game physics as variable assignments, we strip symbols of their semantic associations, shifting computation from pattern matching to logical inference. Under this representation, scale becomes beneficial: Llama-3-70B achieves $\Delta P = +0.29$, a 0.47 improvement over its natural language performance.

This dissociation reveals that failures in ontological restructuring stem from *representational interference*, not reasoning limitations. Natural language activates distributional priors that override contextual logic; code suppresses these priors, allowing models to leverage their capacity for rule-based reasoning. This finding directly motivates our LCV framework, which systematically exploits code's semantic-filtering properties.

5 LCV via Counterfactual Alignment

We address semantic inertia by replacing natural-language action prediction with executable world modeling. Rather than learning a direct policy $\pi_\theta(a_t \mid s_t)$—which conflates perception with dynamics and defaults to pre-trained priors—we learn an amortized theory inducer f_θ that compiles state and rule-set into a Python transition kernel: $\hat{T}_t \leftarrow f_\theta(s_t)$. A classical planner then searches over actions using $\hat{T}_t$ as its dynamics oracle, re-synthesizing the kernel whenever rules change.

To ensure $\hat{T}_t$ depends on mutable rules R_t rather than visual semantics, we train with counterfactual contrastive alignment: paired examples share identical grids but contradictory rule-sets, forcing the model to generate different code for the same visuals. After Supervised Fine-Tuning (SFT), we run a reactive loop that re-synthesizes the world model and exploits its efficiency for heuristic search over actions and rule configurations (Fig. 3).

5.1 Amortized Theory Induction

Recent neuro-symbolic approaches like TheoryCoder (Ahmed et al., 2025) rely on inference-time search to discover valid programs—a computationally expensive process. We instead amortize reasoning by learning $f_\theta : (s_t) \rightarrow C$, mapping states to executable Python programs representing local physics.

This shift from search to induction provides two benefits. First, efficiency: inference becomes $\mathcal{O}(1)$ (single forward pass) versus $\mathcal{O}(N)$ (iterative debugging). Second, robustness: direct theory prediction bypasses local minima in search-based methods, where internal verification heuristics may reject counterintuitive rules due to prior bias.

5.2 Counterfactual Contrastive Alignment

Standard SFT fails to overcome semantic inertia because it does not penalize reliance on pre-trained priors. When visual state v strongly implies specific dynamics (*e.g.*, "Wall" sprite → "Stop"), models minimize loss by attending to visual features while ignoring explicit rules r.

To compel the model to ground its reasoning exclusively in the logical rules, we employ a counterfactual sampling strategy to construct a paired training corpora $\mathcal{D}_{\text{pair}}$. The goal is to render the visual signal v ambiguous with respect to the output. For a fixed map configuration v, we sample paired rule-sets r^+ and r^- that describe contradictory physics, along with their corresponding ground-truth programs c^+ and c^-:

$$\begin{aligned} &\text{Instance A:} \quad \langle v, r^+ = \texttt{WALL IS STOP}, c^+ \rangle \\ &\text{Instance B:} \quad \langle v, r^- = \texttt{WALL IS PASS}, c^- \rangle \end{aligned} \tag{2}$$

We employ contrastive disentanglement objective $\mathcal{L}_{\text{CD}}$, treating rule-set r as causal driver and visual prior v as distractor. The objective maximizes joint likelihood of paired codes conditioned on their specific rules:

$$\begin{aligned} \mathcal{L}_{\text{CD}}(\theta) = \mathbb{E}_{\mathcal{D}_{\text{pair}}}\Big[&- \log P_\theta(c^+ \mid v, r^+) \\ &- \lambda \log P_\theta(c^- \mid v, r^-) \Big] \end{aligned} \tag{3}$$

We set $\lambda = 2$ to emphasize counterfactual learning. While formulated as a joint likelihood rather than a metric loss (*e.g.*, InfoNCE), $\mathcal{L}_{\text{CD}}$ functions as an implicit contrastive mechanism: since v remains invariant while c^+ and c^- contradict each other, gradients relying solely on visual priors oscillate and cancel. To minimize $\mathcal{L}_{\text{CD}}$, the model must suppress ambiguous signal in v and attend exclusively to differentiating variable r, orthogonalizing logical dynamics from visual appearance.

5.3 Reactive Planning with Synthesized Vistas

Amortized induction enables low-latency inference: small models without iterative revision support fully reactive planning loops. In *Baba Is You*, non-stationary transition function T changes when agents rearrange text blocks, requiring frequent world model re-synthesis.

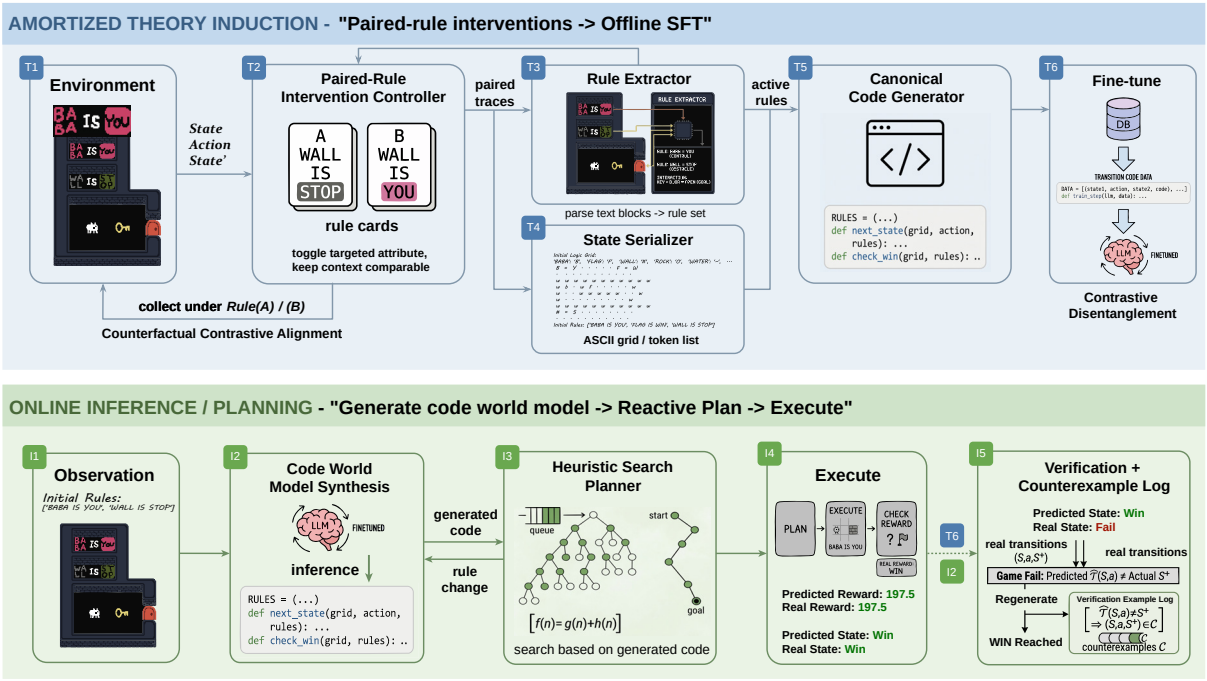


Figure 3: **Overview of LCV.** (Up) **Amortized Contrastive Theory Induction**: We perform SFT on paired samples with the same environment state s but contradictory rule-sets r (*e.g.*, `WALL IS STOP` *vs.* `WALL IS YOU`). This directly targets *semantic inertia* by making surface symbols non-diagnostic: the model must generate different executable kernels solely from the active rules, disentangling object names from their pre-trained affordance priors. (Down) **Online Inference and Planning**: At test time, the model synthesizes the executable transition theory in a single pass, which is compiled and used by a classical planner.

At timestep t, our LCV executes: (i) fine-tuned model predicts Python class $\hat{T}_t = f_\theta(s_t)$—unlike search-based methods requiring seconds for verification, our $\mathcal{O}(1)$ inference enables instant re-synthesis when rule-state R_t changes; (ii) $\hat{T}_t$ compiles into executable `next_state` function; (iii) bounded Greedy Best-First Search uses $\hat{T}_t$ as transition oracle, with domain-agnostic heuristic $h(s)$ prioritizing states reducing Manhattan distance to active `WIN` objects or interactable text blocks. This reactive re-compilation handles "Tier 3" scenarios without pre-defined planners like Planning Domain Definition Language (PDDL).

6 Experiments and Results

We evaluate the efficacy of LCV in mitigating semantic inertia and enabling robust planning under dynamic ontologies. We try to answer three questions:

RQ1 (Performance Hierarchy): Does decoupling theory induction (code synthesis) from planning outperform end-to-end neural policy generation, particularly in semantically adversarial settings?

RQ2 (Inhibitory Control): To what extent does the counterfactual contrastive alignment objective reduce prior-reversion behavior compared to standard supervision?

RQ3 (Amortization Efficiency): While a fine-tuned small model (7B) can surely outperform inference-heavy approaches (*e.g.*, TheoryCoder with GPT-4o) in latency and robustness, how do these gains trade off against the cost of data collection and SFT?

6.1 Baselines

We compare our LCV against a diverse set of neural and neuro-symbolic baselines:

Direct Policy (Zero-Shot / CoT): Standard agentic prompting where the model predicts actions directly. We evaluate both standard IO and Chain-of-Thought (CoT) (Wei et al., 2022).

Code-as-Policy (CaP): Following Liang et al. (2023), the model generates a heuristic Python script to solve the task directly, without explicitly modeling the transition dynamics.

TheoryCoder: We implement the inference-time variant of Ahmed et al. (2025), in which GPT-4o iteratively synthesizes a transition function through few-shot prompting and execution feedback. While recent baselines such as Chain of Code (Li et al.,

[2]For Tier 3, since the rules can change, we report only the average token counts for each plan.

[3]16k indicates that the maximum generation token limit was reached.

Table 1: **Success Rate (SR) on BABABENCH.** Top: four open-source base models, reported SR only and arranged two models side by side. Bottom: for Deepseek, TheoryCoder, proprietary references, and our LCV models, we additionally report the number of thinking and answer tokens (Tok: Think / Ans) per tier when available.

SR ↑	Tier 1	Tier 2	Tier 3		Tier 1	Tier 2	Tier 3
Qwen2.5-7B-Instruct				***Qwen2.5-72B-Instruct***			
Direct Policy	8.89%	2.22%	0.00%	Direct Policy	13.33%	6.67%	2.22%
Direct Policy (CoT)	13.33%	6.67%	4.00%	Direct Policy (CoT)	28.89%	20.00%	6.67%
Code-as-Policy	13.33%	11.11%	4.00%	Code-as-Policy	62.22%	53.33%	42.00%
GPT-OSS-120B				***Llama-3-70B-Instruct***			
Direct Policy	8.89%	2.22%	2.00%	Direct Policy	15.56%	6.67%	8.00%
Direct Policy (CoT)	8.89%	2.22%	4.00%	Direct Policy (CoT)	17.78%	6.67%	8.00%
Code-as-Policy	6.67%	6.67%	8.00%	Code-as-Policy	53.33%	26.67%	18.00%

	Tier 1		Tier 2		Tier 3	
	SR ↑	Tok (Think / Ans)	SR ↑	Tok (Think / Ans)	SR ↑	Tok (Think / Ans)[2]
Deepseek-v3.2						
Direct Policy	17.78%	7.38k / 15.2	13.33%	16.0k[3] / 12.8	8.00%	16.0k / 18.5
Code-as-Policy	40.00%	4.24k / 1.16k	31.11%	5.67k / 892.4	36.00%	3.93k / 962.9
Claude Sonnet 4.5						
Direct Policy	57.78%	1.24k / 14.7	13.33%	3.31k / 11.3	8.00%	3.15k / 16.9
Code-as-Policy	68.89%	2.52k / 834.2	31.11%	2.87k / 943.6	30.00%	2.22k / 677.4
Gemini 3 Pro Preview						
Direct Policy	62.22%	1.43k / 13.5	13.33%	2.07k / 19.1	10.00%	2.00k / 10.4
Code-as-Policy	71.11%	1.27k / 953.4	31.11%	1.65k / 1.05k	28.00%	1.18k / 1.14k
TheoryCoder						
GPT-4o, 1 API call	24.44%	– / 1.13k	13.33%	– / 1.31k	8.00%	– / 1.31k
GPT-4o	62.22%	– / 2.46k	53.33%	– / 3.22k	52.00%	– / 3.08k
Human (with tutorial video)	95.56%	-	82.22%	-	78.00%	-
Ours (Amortized LCV)						
LCV (Vanilla L_{SFT})	88.89%	– / 763.9	60.00%	– / 1.25k	48.00%	– / 1.27k
LCV (Contrastive L_{CD})	**93.33%**	– / 798.5	**75.56%**	– / 1.16k	**62.00%**	– / 1.31k

2024) also address code-centric reasoning, we believe TheoryCoder targets a closely related class of problems and represents a state-of-the-art approach for the coding–planning pipeline.

We evaluate open-weight models (Qwen, LLaMA, DeepSeek, GPT-OSS) and proprietary foundations (Gemini 3 Pro, Claude Sonnet 4.5, GPT-4o). We additionally evaluate OpenAI o1-preview, which performs extended chain-of-thought reasoning internally. Results are reported in Table A4. Prompts and detailed settings can be found in Section E.

For our LCV agent, we fine-tune a small Qwen2.5-7B-Instruct model using approximately 600 paired training samples collected from BABABENCH for world model synthesis. The synthesized world model is compiled and passed to a bounded heuristic planner with a node expansion budget of $N{=}2000$. We report the average Success Rate (SR) together with LLM-generated token lengths (ToK) for each tier. For evaluation, we sample 45 paired environments for both Tier 1 and Tier 2, and 50 environments for Tier 3.

6.2 Main Results

Table 1 summarizes the performance across the three complexity tiers. Also, solution examples can be found in Section F.1.

The Collapse of Natural Language (RQ1). Standard prompting reveals a catastrophic failure of inhibition. While foundation models like Claude 4.5 Sonnet perform strongly on aligned tasks (Tier 1: 57.78%), they collapse on adversarial tasks (Tier 2: 13.33%). Notably, huge models like GPT-OSS-120B and Llama-3-70B show negligible improvement over their smaller counterparts in Tier 2. This empirically confirms our *Inverse Scaling* hypothesis: scale alone does not solve functional fixedness. The visual prior of a "Wall" overrides the logical instruction "Wall is Pass," leading to persistent hal-

lucination of obstacles regardless of model size.

Code as a Scaffold, Logic as a Solution. Methods utilizing code (CaP, TheoryCoder) show greater resilience. However, LCV achieves dominant performance, with 75.56% SR in Tier 2, significantly outperforming other baselines. This result highlights that simply using code (CaP) is insufficient if the generation process itself is biased by priors. By separating the generation of physics from the generation of actions, LCV isolates the reasoning error, allowing the planner to search a clean, logic-grounded state space.

Robustness in Dynamic Settings (Tier 3). Tier 3 requires manipulating rules to change the environment mid-trajectory. Here, TheoryCoder's performance plateaus, likely due to error accumulation in its iterative synthesis loop. In contrast, our amortized approach maintains robustness. Because our model is trained to map state-configurations to code directly, it treats a mid-episode rule change simply as a new input state, re-compiling the physics engine instantly without the need for fragile conversational history maintenance.

6.3 Analysis of Inhibitory Control (RQ2)

To isolate the sources of performance gain, we decompose contributions along three axes.

Planner vs. Direct Policy. The planner handles spatial search, while LCV focuses the LLM on synthesizing the world model in code. If $\hat{T}_t$ is incorrect, the planner cannot recover regardless of search budget—confirming that reasoning capability resides in the code-generation step, not the search heuristic. The planner is a spatial accelerator; the reasoning bottleneck is always world-model induction.

Fine-tuning vs. Prompting. Even under identical code representations, prompt-only CaP achieves only 31.1% on Tier 2 while LCV Vanilla reaches 60.0%—a gap of nearly 30%. This gap shrinks substantially on Tier 1 (71.1% vs. 88.9%), confirming that fine-tuning's primary contribution is inhibitory control rather than general spatial planning. Prompting alone remains vulnerable to inverse scaling because prior-biased attention patterns persist even when the output format is code.

Contrastive $\mathcal{L}_{\text{CD}}$ vs. Vanilla SFT. In Tier 1, the performance gap is marginal (88.9% vs. 93.3%), as both models learn correctly when visuals and logic agree. In Tier 2 and Tier 3, the gap widens significantly (60.0% vs. 75.6%; 48.0% vs. 62.0%), validating our gradient orthogonalization hypothesis: the Vanilla model overfits to spurious visual correlations (*e.g.*, `WALL` → `stop=True`) because it can minimize $\mathcal{L}_{\text{SFT}}$ without attending to rule-text r. The contrastive objective breaks this shortcut by holding v fixed while c^+ and c^- contradict each other, forcing the model to attend exclusively to r and embedding inhibitory control directly into its weights.

6.4 Ablation I: Learning Efficiency

A dominant paradigm in recent reasoning literature is the scaling of inference-time compute—using iterative scaffolding (*e.g.*, CoT, TheoryCoder loops) to resolve ambiguity (Shinn et al., 2023). Our results challenge the universality of this approach in adversarial settings. As shown in Table A3, approaches like TheoryCoder (GPT-4o) require extensive token expenditure ($\sim$3.2k/problem) to "debug" their way out of semantic priors. We argue that this represents an inefficient allocation of compute: the model is burning tokens to fight its own parametric memory. By shifting this burden to training-time alignment, LCV achieves a $\mathbf{4}\times$ reduction in latency while improving accuracy. This suggests that inhibitory control is amortizable. The cognitive effort required to dissociate "Wall" from "Stop" does not necessarily require explicit step-by-step reasoning at every instance. Through our contrastive alignment, we effectively compile this inhibition into the model's forward pass, transforming a complex reasoning task into a rapid, reflex-based retrieval. This offers a scalable path for deploying robust agents in real-time environments where iterative "thinking" loops are prohibitively slow.

6.5 Ablation II: Generalization

A frequent critique of fine-tuning small models (7B) versus prompting large foundations (72B+) is the potential loss of generalization. We probe this via two distinct splits (detailed in Section G.3):

Map Generalization (Spatial Robustness). On unseen spatial configurations, LCV retains almost all of its in-distribution performance (76.4% → 74.3%). This result is pivotal: it indicates that our model has not overfitted to visual patterns (*e.g.*, "Wall at pixel x, y"). Instead, it has learned a symbolic compiler which can be seen as a function that maps visual discrete entities to abstract logical rules, regardless of their spatial distribution.

Combination Generalization (Logical Robustness). The "Combo Gen" split (*e.g.*, introducing `WALL IS MELT`) requires systematicity—combining known atoms in novel ways. While all models degrade, LCV maintains a massive lead over the baselines (72.4% *vs*. 48.2% for Qwen-72B).

In conclusion, standard LLMs rely on parametric generalization—solving novel tasks by mapping them to similar examples in the pre-training corpus. When the task explicitly contradicts that corpus (as in Tier 2/3), parametric generalization becomes a liability. LCV exhibits systematic generalization: because it operates on an intermediate code representation, it can compose rules it has never seen together, provided the underlying syntax (Python Code) remains consistent. This confirms that code-grounding acts as a regularizer, forcing the model to learn the grammar of physics rather than the statistics of scenarios.

6.6 Ablation III: Representation Format

Table 2: **Representation Format Ablation (Gemini 3 Pro).** NL: Natural Language (Descriptive); Table: Structured Rule Table (Descriptive); PDDL (Symbolic); Code: Code without fine-tuning (Executable); LCV (Executable). Executability—not syntax regularity alone—reverses inverse scaling.

	NL	Table	PDDL	Code	LCV
Tier 1	62.2%	75.6%	71.1%	71.1%	**93.3%**
Tier 2	13.3%	40.0%	26.7%	31.1%	**75.6%**
Tier 3	10.0%	20.0%	18.0%	28.0%	**62.0%**

To isolate whether our gains stem from *executability* rather than mere structural regularity, we compare four representational formats using Gemini 3 Pro as the backbone (Table 2). Structured Rule Tables and PDDL improve over NL (Tier 2: 40.0% and 26.7% vs. 13.3%) due to organizational regularity, yet a substantial gap remains. This confirms that semantic inertia persists in declarative formats: models correctly read the rule but still fall back on internal biases. By contrast, synthesizing a functional transition kernel forces logical contradictions to be resolved at the code level—executability, not syntax, is the key driver.

7 Conclusion

In this work, we demonstrate that semantic inertia scales inversely with model size, hampering robust reasoning in dynamic environments. We propose Code-Grounded Vistas (LCV), a framework that decouples logical dynamics from visual priors via amortized theory induction and counterfactual contrastive alignment. Our approach significantly outperforms inference-heavy baselines like TheoryCoder in both robustness and efficiency. These findings suggest that for mutable ontologies, reasoning must be grounded in executable, verifiable code rather than probabilistic natural language.

Our controlled representation ablation (Table 2) further confirms that this advantage stems specifically from *executability*—the requirement to synthesize a functionally correct transition kernel—rather than from structural regularity alone: declarative formats such as structured tables and PDDL offer only modest improvements over plain language, while executable code with fine-tuning yields transformative gains. More broadly, our results suggest that *representation fundamentally determines whether scaling helps or hurts*: when the encoding medium entangles descriptive semantics with logical rules, larger models amplify prior interference; when it enforces executable, logic-grounded structure, scaling translates computational capacity into genuine reasoning improvement.

Limitations

While LCV effectively mitigates semantic inertia, it relies on discrete state abstractions (grid worlds) and may not directly transfer to continuous, high-dimensional visual domains without an additional perception module. Furthermore, our Counterfactual Contrastive Alignment requires paired data with contradictory rules, which is procedurally generatable in logic puzzles but potentially expensive to curate in naturalistic environments.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China (32595491,62376009), the State Key Lab of General AI at Peking University, the PKU-BingJi Joint Laboratory for Artificial Intelligence, the Wuhan Major Scientific and Technological Special Program (2025060902020304), the Hubei Embodied Intelligence Foundation Model Research and Development Program, and the National Comprehensive Experimental Base for Governance of Intelligent Society, Wuhan East Lake High-Tech Development Zone. This work does not relate to any position at Amazon.

References

Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B Divya. 2025. Agentic ai: Autonomous intelligence for complex goals–a comprehensive survey. *IEEE Access*.

Zergham Ahmed, Joshua B Tenenbaum, Christopher J Bates, and Samuel J Gershman. 2025. Synthesizing world models for bilevel planning. *arXiv preprint arXiv:2503.20124*.

Emily M Bender and Alexander Koller. 2020. Climbing towards nlu: On meaning, form, and understanding in the age of data. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Ning Bian, Xianpei Han, Le Sun, Hongyu Lin, Yaojie Lu, Ben He, Shanshan Jiang, and Bin Dong. 2024. Chatgpt is a knowledgeable but inexperienced solver: An investigation of commonsense problem in large language models. In *International Conference on Computational Linguistics (COLING)*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, and 1 others. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Megan Charity and Julian Togelius. 2022. Keke ai competition: Solving puzzle levels in a dynamically changing mechanic space. In *IEEE Conference on Games (CoG)*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Nathan Cloos, Meagan Jens, Michelangelo Naim, Yen-Ling Kuo, Ignacio Cases, Andrei Barbu, and Christopher J Cueva. 2024. Baba is ai: Break the rules to beat the benchmark. *arXiv preprint arXiv:2407.13729*.

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

Adele Diamond. 2013. Executive functions. *Annual Review of Psychology*, 64(1):135–168.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of International Conference on Machine Learning (ICML)*.

Karl Duncker and Lynne S Lees. 1945. On problem-solving. *Psychological Monographs*, 58(5):i.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, and 1 others. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *Proceedings of International Conference on Machine Learning (ICML)*.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

Guangyuan Jiang, Manjie Xu, Shiji Xin, Wei Liang, Yujia Peng, Chi Zhang, and Yixin Zhu. 2023. Mewl: Few-shot multimodal word learning with referential uncertainty. In *Proceedings of International Conference on Machine Learning (ICML)*.

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

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2024. Chain of code: Reasoning with a language model-augmented code emulator. In *Proceedings of International Conference on Machine Learning (ICML)*.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

John McCarthy. 1980. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1-2):27–39.

Ian R McKenzie, Alexander Lyzhov, Michael Martin Pieler, Alicia Parrish, Aaron Mueller, Ameya Prabhu, Euan McLean, Xudong Shen, Joe Cavanagh, Andrew George Gritsevskiy, and 1 others. 2023. Inverse scaling: When bigger isn't better. *Transactions on Machine Learning Research*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Judea Pearl. 2009. *Causality*. Cambridge university press.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. In *Proceedings of International Conference on Learning Representations (ICLR)*.

Elena Stringli, Maria Lymperaiou, Giorgos Filandrianos, Athanasios Voulodimos, and Giorgos Stamou. 2025. Pitfalls of scale: Investigating the inverse task of redefinition in large language models. *arXiv preprint arXiv:2502.12821*.

J Ridley Stroop. 1935. Studies of interference in serial verbal reactions. *Journal of Experimental Psychology*, 18(6):643.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Hao Tang, Darren Key, and Kevin Ellis. 2024a. Worldcoder, a model-based llm agent: building world models by writing code and interacting with the environment. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Hao Tang, Darren Yan Key, and Kevin Ellis. 2024b. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Fien van Wetten, Aske Plaat, and Max van Duijn. 2025. Baba is llm: Reasoning in a game with dynamic rules. *arXiv preprint arXiv:2506.19095*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of International Conference on Learning Representations (ICLR)*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Lionel Wong, Gabriel Grand, Alexander K Lew, Noah D Goodman, Vikash K Mansinghka, Jacob Andreas, and Joshua B Tenenbaum. 2023. From word models to world models: Translating from natural language to the probabilistic language of thought. *arXiv preprint arXiv:2306.12672*.

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary Siegel, Jiahai Feng, Noa Korneev, Joshua B Tenenbaum, and Jacob Andreas. 2024. Learning adaptive planning representations with natural language guidance. In *Proceedings of International Conference on Learning Representations (ICLR)*.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2024. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Manjie Xu, Guangyuan Jiang, Wei Liang, Chi Zhang, and Yixin Zhu. 2023. Active reasoning in an open-world environment. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Manjie Xu, Xinyi Yang, Jiayu Zhan, Wei Liang, Chi Zhang, and Yixin Zhu. 2025. Heterogeneous adversarial play in interactive environments. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Khurram Yamin, Gaurav Ghosal, and Bryan Wilder. 2025. Llms struggle to perform counterfactual reasoning with parametric knowledge. *arXiv preprint arXiv:2506.15732*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

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

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang,

Yulong Chen, and 1 others. 2025. Siren's song in the ai ocean: A survey on hallucination in large language models. *Computational Linguistics*, pages 1–46.

Jun Zhao, Yongzhuo Yang, Xiang Hu, Jingqi Tong, Yi Lu, Wei Wu, Tao Gui, Qi Zhang, and Xuanjing Huang. 2025. Understanding parametric and contextual knowledge reconciliation within large language models. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, and 1 others. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

Yixin Zhu, Tao Gao, Lifeng Fan, Siyuan Huang, Mark Edmonds, Hangxin Liu, Feng Gao, Chi Zhang, Siyuan Qi, Ying Nian Wu, and 1 others. 2020. Dark, beyond deep: A paradigm shift to cognitive ai with humanlike common sense. *Engineering*, 6(3):310–345.

Figure A1: Overview of the *Baba Is You* environment.

A Environment Overview: *Baba Is You*

Baba Is You[1] is an award-winning logic puzzle game developed by Arvi Teikari (Hempuli) that fundamentally reimagines the mechanics of grid-based interactions. Unlike traditional environments (*e.g.*, Sokoban or Minecraft) where object affordances are static constants—a wall is always an obstacle, a key is always an item—*Baba Is You* treats the rules of physics as tangible, manipulatable objects within the game world. The environment consists of "Word" blocks (*e.g.*, `WALL`, `IS`, `YOU`) and physical objects (sprites). A rule is formed only when three Word blocks align syntactically (Noun-Operator-Property). Crucially, the player can push these Word blocks to rewrite the game's logic in real-time. For instance, disconnecting `WALL IS STOP` instantly renders walls permeable, while forming `ROCK IS YOU` transfers the player's agency from the titular character Baba to a Rock. This unique mechanism collapses the distinction between the object level and the meta-level, creating a testbed where an agent cannot rely on fixed semantic priors (*e.g.*, "skulls are dangerous") but must actively ground its reasoning in the current, mutable configuration of the text blocks.

B Inverse Scaling Details

Scenario Construction. We curated 45 scenarios from the *Baba Is You* environment, specifically targeting states where the current rule set induces a sharp conflict with typical semantic priors (e.g., `LAVA IS SAFE`, `WALL IS YOU`). For each scenario, the board state and active rules were manually defined to create pairs of logic-driven and prior-driven actions. Each input includes the grid, a list of possible moves, and the set of rules currently in effect.

Prompt Engineering. For each scenario, two types of prompts were prepared:

- **Natural Language (NL) Prompts:** These describe the current environment and explicitly state the rules in plain language, e.g., "The rule is: 'Wall is You'. Valid moves are: UP, DOWN, LEFT, RIGHT. Which is correct under the current rules?"
- **Code Grounding Prompts:** These supply the rule set as code-like statements or as arguments to an explicit transition function, emphasizing variable assignment and logic, e.g.,

[1] https://hempuli.com/baba/

"Given: rules = 'Wall': 'You', ... Which action is valid according to the transition function $T(\cdot; R_t)$?"

To avoid position biases, the order of presenting moves and states was randomized across prompts.

Model Families and Parameter Scaling. Evaluation was performed using freely available and open-source large language models: Llama-3 (1B, 8B, 70B), Pythia (160M, 1.4B, 12B), and Qwen3 (600M, 8B, 32B). Both base and instruction-tuned checkpoints were included where compatible with the interface. All model inference was conducted using `transformers` with greedy decoding (temperature 0), and where possible, probabilities for all candidate single-token actions (UP, DOWN, LEFT, RIGHT) were directly obtained from output logits.

Inference Setup. Models were run on a single NVIDIA A100 node. Each prompt was independently batched and sent to the model, and output log probabilities for each candidate action were retrieved. All models are deployed locally.

Labeling and Score Computation. For each prompt, we annotated which action is logic-driven (i.e., required by the current non-default rules) and which action conforms to the most probable prior (e.g., path avoidance of 'dangerous' lava or 'impassable' wall). We then computed the adaptation score for each instance as:

$$\Delta P = P(w_{\text{logic}} \mid S) - P(w_{\text{prior}} \mid S)$$

Average ΔP scores were calculated separately for Natural Language and Code Grounding modalities, across all parameter sizes within each family.

Additional Notes. To ensure that models were not overfitting to specific lexical cues, several variants of the scenarios were repeated with shuffled agent/object identities (e.g., swapping 'BABA' and 'ROCK'), confirming that trends in inverse scaling were robust to superficial changes. In all cases, the main observation held: larger models in the NL condition exhibited stronger semantic inertia, whereas code grounding robustly enabled scaling benefits for context-driven reasoning.

C LCV Implementation Details

C.1 SFT

For our main experiments, we fine-tuned the Qwen2.5-7B-Instruct model, which has 7 billion parameters, using the LLaMA-Factory framework and LoRA adaptation. Training was performed for 20 epochs with a batch size of 1 per device (gradient accumulation steps: 4) on 4 NVIDIA RTX 3090 GPUs (24GB each), using FP16 precision. The total GPU compute for fine-tuning was approximately 10 GPU-hours. In all experiments, we applied LoRA ($r = 8$, $\alpha = 16$, dropout $= 0$) on Qwen2.5-7B-Instruct, targeting all (`k_proj`, `q_proj`, `v_proj`, `o_proj`, `up_proj`, `down_proj`, `gate_proj`) projection layers, without bias or additional PEFT techniques. All hyperparameters are provided to facilitate reproducibility.

C.2 Overall Planning Pipeline

In each planning episode, the search operates over an explicit symbolic graph, where nodes correspond to compressed grid states $s \in \mathcal{S}$ and edges correspond to agent actions $a \in \mathcal{A}$. Unlike environments with static transition dynamics, *Baba Is You* defines state transitions and terminal conditions through a logic model $\mathcal{M}$ that is generated at runtime by the LLM. Crucially, the system does not hard-code notions such as controllability, victory, or physical affordances. Instead, it queries the current state and the active rule set to infer all relevant entities and interactions.

At every node expansion, the search consults $\mathcal{M}$ to parse symbolic rules expressed as triples of the form

$$(\text{Subject},\ \texttt{IS},\ \text{Property}),$$

such as `BABA IS YOU`, `FLAG IS WIN`, or `WALL IS STOP`. Based on the currently active rules in state s, the planner dynamically constructs entity sets associated with each property. For example:

$$\mathcal{P}_{\text{you}} = \{p : p \text{ satisfies } (\cdot, \texttt{IS}, \texttt{YOU}) \text{ in } s\}, \quad \text{(A1)}$$
$$\mathcal{P}_{\text{win}} = \{p : p \text{ satisfies } (\cdot, \texttt{IS}, \texttt{WIN}) \text{ in } s\}, \quad \text{(A2)}$$
$$\mathcal{P}_{\text{push}} = \{p : p \text{ satisfies } (\cdot, \texttt{IS}, \texttt{PUSH}) \text{ in } s\}, \quad \text{(A3)}$$
$$\mathcal{P}_{\text{stop}} = \{p : p \text{ satisfies } (\cdot, \texttt{IS}, \texttt{STOP}) \text{ in } s\}. \quad \text{(A4)}$$

These sets may change dynamically as agent actions modify the map's logic, for instance by rearranging textual blocks $\mathcal{T}$ to create or destroy rules such as `ROCK IS PUSH` or `WALL IS STOP`.

Planning is guided by logic-aware heuristics that adapt to the currently active rule configuration. When a winning condition exists (*e.g.*, `FLAG IS WIN`), the heuristic encourages controllable entities to approach winning objects:

$$h(s) = \min_{p_i \in \mathcal{P}_{\text{you}},\, p_j \in \mathcal{P}_{\text{win}}} \left(|x_i - x_j| + |y_i - y_j|\right).$$

If no win condition is present, or if the current rules introduce obstacles or hazards (*e.g.*, `WALL IS STOP` or `LAVA IS DEFEAT`), the heuristic instead prioritizes interactions that may alter the rule set. In such cases, the search is biased toward textual elements:

$$h(s) = \min_{p_i \in \mathcal{P}_{\text{you}},\, t \in \mathcal{T}} \left(|x_i - x_t| + |y_i - y_t| \right).$$

Beyond simple navigation, the planner reasons about more complex emergent behaviors induced by rule composition. For instance, if `LAVA IS DEFEAT` is active, lava tiles are treated as lethal regions to be avoided. If both `KEY IS OPEN` and `DOOR IS SHUT` are present, the planner can infer the possibility of unlocking doors through interaction. Importantly, the system does not assume a fixed vocabulary of properties; instead, it parses and reasons over any rule discovered by $\mathcal{M}$, provided it conforms to the `X IS Y` syntactic structure.

For each action $a \in \mathcal{A}$ taken from state s, a candidate successor state s' is produced by invoking $\mathcal{M}.next\_state\_fn(s, a)$, and is subsequently interpreted under the newly inferred rule set. Cycle detection via state hashing prevents redundant exploration, while a priority queue enables best-first expansion according to the current logic-aware heuristic.

The planning process terminates when a state is reached in which any controllable entity satisfies an active winning rule (i.e., a `(· IS WIN)` triple applies), or when predefined computational resource limits are exceeded.

This generalized strategy allows the agent to reason flexibly over a diverse and evolving set of symbolic rules, enabling robust planning amid dynamic changes to controllable entities, victory conditions, obstacles, hazards, and interaction affordances.

C.3 Inference Acceleration under Dynamic Rule Changes

When the active rules change during planning and require updates to the logic model, the LCV framework must invoke the LLM to regenerate the executable world model $\mathcal{M}$ conditioned on the new rule configuration. Standard autoregressive generation can be computationally expensive, especially when each update introduces only incremental modifications to the current rules or environment.

To address this, we employ two complementary inference acceleration strategies. First, we leverage the LLM's key-value (KV) cache mechanism and

Algorithm 1: Symbolic Dynamic Reasoning

Input: Start state s_0, WorldModel $\mathcal{M}$, Max Depth D

Initialize $OpenSet$ as a priority queue ordered by $g + h$;
Initialize $Visited$ as an empty set;
$OpenSet$.push(Node($s_0, 0, h(s_0), [\,]$));
while *$OpenSet$ is not empty* **do**
  $n \leftarrow OpenSet$.pop();
  if $\mathcal{M}$.`check_win`(n.*state*) **then**
    return n.path;
  if *length*(n.*path*) $> D$ **then**
    continue;
  for *each action* $a \in \mathcal{A}$ **do**
    $s' \leftarrow$ $\mathcal{M}$.`next_state_fn`(n.state, a);
    if `Hash`(s') $\notin$ $Visited$ **then**
      Parse active rule set $\mathcal{R}$ in s' via $\mathcal{M}$;
      for *each property* $q \in$ {*YOU, WIN, PUSH, ...*} **do**
        $\mathcal{P}_q \leftarrow \{p : (\cdot, \text{IS}, q) \in \mathcal{R}\}$;
      Compute h' using the updated rule sets;
      $OpenSet$.push(Node(s', n.cost+ $1, h', n$.path + $[a]$));
      $Visited$.add(`Hash`(s'));
return FAIL;

integrate TensorRT-based compilation for efficient token generation. Since the underlying structure and output tokens of consecutive world models are often highly similar—particularly when rule changes are local or only affect a subset of the semantic triples—cached computation enables us to skip recomputation of unchanged subtrees and dramatically reduce inference latency. Empirically, these optimizations yield a $4.6\times$ speedup on a single NVIDIA A100 GPU for world model synthesis.

Second, we implement a rule-based caching scheme at the symbolic logic level. By storing previously synthesized world model code and semantic transitions for encountered rule sets, we avoid redundant regeneration when the agent revisits similar or identical configurations. When a rule change is detected, the framework queries the cache for a matching logic signature; if present, the cached model is reused directly, and only new or altered logic components are synthesized. This cache is implemented both at the level of LLM prompts and the resulting Python code, and it supports partial rule inheritance to further improve reuse when only local adjustments are required. Empirically, in benchmark experiments, the rule-based cache successfully avoided about 85% potential world model regenerations.

Together, these optimizations ensure fast turnaround in logic model synthesis, enabling robust and responsive agent planning even as the underlying rule set evolves at runtime.

D Mean Adaptation Scores

Table A1 reports mean ΔP scores across all model families and prompt modalities. Table A2 provides the corresponding 95% confidence intervals and two-sided t-test results, confirming that the majority of observed scaling trends are statistically significant.

E Prompt Examples

All baselines are evaluated under identical task descriptions and prompting protocols, without access to intermediate environment states, external tools, or execution feedback beyond textual observations.

E.1 Direct Action Plan

```
[Level: 0-0]
You are playing 'Baba Is You'. Win the
↪ level.
```

Table A1: **Mean Adaptation Scores (ΔP).** Under Natural Language (NL), larger models often perform worse (Inverse Scaling) due to stronger semantic priors. Code Grounding reverses this, enabling large models (*e.g.*, Llama-3-70B) to effectively inhibit priors and maximize reasoning performance.

Family	Size	NL (ΔP)	Code (ΔP)
Pythia	160M	**0.155**	0.018
	1.4B	0.030	**0.091**
	12B	-0.112	**0.180**
Qwen3	0.6B	**0.484**	0.098
	8B	0.172	0.160
	32B	0.089	**0.252**
Llama-3	1B	-0.098	**0.091**
	8B	-0.157	**0.223**
	70B	-0.183	**0.290**

Table A2: **Statistical Significance of ΔP.** CI_{95} denotes the 95% confidence interval. Significance: $^*p<0.05$, $^{**}p<0.01$, $^{***}p<0.001$, ns = not significant.

Family	Model	Prompt	Mean	CI Lower	CI Upper	t	Sig.
Llama-3	1B	NL	−0.098	−0.198	0.002	−1.978	ns
		Code	0.091	0.010	0.172	2.267	*
	3B	NL	−0.096	−0.164	−0.028	−2.833	**
		Code	0.145	0.091	0.198	5.463	***
	8B	NL	−0.157	−0.269	−0.045	−2.826	**
		Code	0.223	0.140	0.307	5.373	***
	70B	NL	−0.183	−0.299	−0.067	−3.179	**
		Code	0.290	0.199	0.381	6.422	***
Pythia	160M	NL	0.155	0.103	0.208	5.926	***
		Code	0.018	0.013	0.024	6.708	***
	1.4B	NL	0.030	−0.012	0.072	1.446	ns
		Code	0.091	0.014	0.168	2.381	*
	12B	NL	−0.112	−0.201	−0.022	−2.507	*
		Code	0.180	0.083	0.278	3.721	***
Qwen3	0.6B	NL	0.484	0.389	0.578	10.344	***
		Code	0.098	0.011	0.186	2.263	*
	8B	NL	0.172	0.053	0.291	2.917	**
		Code	0.160	0.066	0.254	3.443	**
	32B	NL	0.089	−0.026	0.203	1.559	ns
		Code	0.252	0.162	0.342	5.666	***

```
### Legend
'.': EMPTY, '#': WALL, '=': IS, 'B':
↪ BABA, 'F': FLAG, 'O': ROCK, 'P':
↪ PUSH, 'S': STOP, 'W': WIN, 'Y': YOU,
↪ 'b': ICON_BABA, 'f': ICON_FLAG, 'r':
↪ ICON_ROCK, 'w': ICON_WALL

### Active Rules
['BABA IS YOU', 'FLAG IS WIN', 'ROCK IS
↪ PUSH', 'WALL IS STOP']

### Map (ASCII)
B = Y . . . . . F = W
. . . . . .
. . w w w w w . . . .
. . w . . . w . . . .
. . w . b r f . . . .
. . w . . . w . . . .
. . w w w w w . . . .
. . . . . .
# = S . . O = P . . .
```

```
### Instructions
1. Identify YOU (the character you
↪  control) and WIN (the target).
2. Plan a path. Push text blocks if
↪  needed to change rules.
3. Output ONLY the sequence of moves:
↪  UP, DOWN, LEFT, RIGHT.

Think step-by-step about what obstacles
↪  exist and how to overcome them. (for
↪  CoT)
```

E.2 Coding Plan

````
You are a logic engine generator for
↪  'Baba Is You'.
Write Python code to simulate the game
↪  step based on the rules.

### Symbol Mapping:
- Objects: 'B'=BABA, 'F'=FLAG, '#'=WALL,
↪  'O'=ROCK, '.'=EMPTY
- Text:    'b','f','w','r' (Small chars
↪  are icons/text blocks)
- Logic:   '='=IS, 'Y'=YOU, 'W'=WIN,
↪  'S'=STOP, 'P'=PUSH

### Requirements:
1. **Reference**: Use the skeleton
↪  below.
2. **No Numpy**: Use standard python
↪  lists.
3. **Logic**: Implement ONLY the active
↪  rules listed below.

### Reference Skeleton:
```python

def next_state(grid, action):
    # grid: List[List[str]], \eg,
    ↪  grid[0][0] = 'B' (Baba) or 'Bw'
    ↪  (Baba + Wall)
    # action: str, 'UP', 'DOWN', 'LEFT',
    ↪  'RIGHT'

    rows = len(grid)
    cols = len(grid[0])
    new_grid = [row[:] for row in grid]
    ↪  # Deep copy

    # ... Your Logic Here (Move YOU,
    ↪  Handle STOP/PUSH) ...
    return new_grid

def check_win(grid):
    # Check if 'YOU' overlaps 'WIN'
    return False

```

### Current Active Rules:
['BABA IS YOU', 'FLAG IS WIN', 'ROCK IS
↪  PUSH', 'WALL IS STOP']

### Current Grid (Context):
B = Y . . . . . F = W
. . . . . .
. . w w w w w . . . .
. . w . . . w . . . .
. . w . b r f . . . .
. . w . . . w . . . .
. . w w w w w . . . .
. . . . . .
# = S . . O = P . . .

### Task:
Output the complete Python code
↪  implementing `next_state` and
↪  `check_win`.
````

F Environment Visualization and Complexity Tiers

We provide qualitative visualizations and concrete examples of the three environment tiers used in BABABENCH. The tiers are designed to incrementally isolate distinct sources of reasoning failure under dynamic, rule-driven environments. Representative levels from each tier are shown in Fig. A2.

Tier 1: Semantic Alignment (Control). Tier 1 environments serve as a control condition for basic spatial reasoning and action sequencing. The active rules are fully aligned with common visual and physical priors learned during pre-training (*e.g.*, `WALL IS STOP`, `FLAG IS WIN`). Object appearances and affordances are consistent, allowing agents to rely on standard navigation heuristics. Performance in this tier establishes a baseline and ensures that failures in higher tiers are not attributable to deficiencies in low-level search or perception.

Tier 2: Semantic Conflict (Inhibitory Control). Tier 2 environments introduce explicit conflicts between visual semantics and logical rules. Rules such as `LAVA IS SAFE` or `WALL IS YOU` deliberately

contradict intuitive expectations induced by sprite appearance. Solving these levels requires suppressing pre-trained semantic associations and acting strictly according to the symbolic rule set. This tier isolates failures of semantic inhibition, analogous to Stroop-like interference effects, where visually salient cues must be overridden by abstract task rules.

Tier 3: Dynamic Plasticity (Rule Adaptation). Tier 3 environments require multi-stage planning under non-stationary semantics. Levels typically begin in configurations where progress is impossible under the initial rules. Agents must manipulate text blocks to construct new rules (*e.g.*, forming `WALL IS PASS`) that temporarily alter the environment's affordances, and may later need to dismantle or revise these rules to avoid adverse consequences. Success in this tier depends on maintaining an accurate internal representation of the current rule set, updating it after each intervention, and avoiding reliance on outdated semantic assumptions.

Together, these tiers provide a controlled progression from aligned semantics to conflicting and dynamically mutable ontologies, enabling fine-grained analysis of how agents respond to semantic interference and rule-driven concept revision.

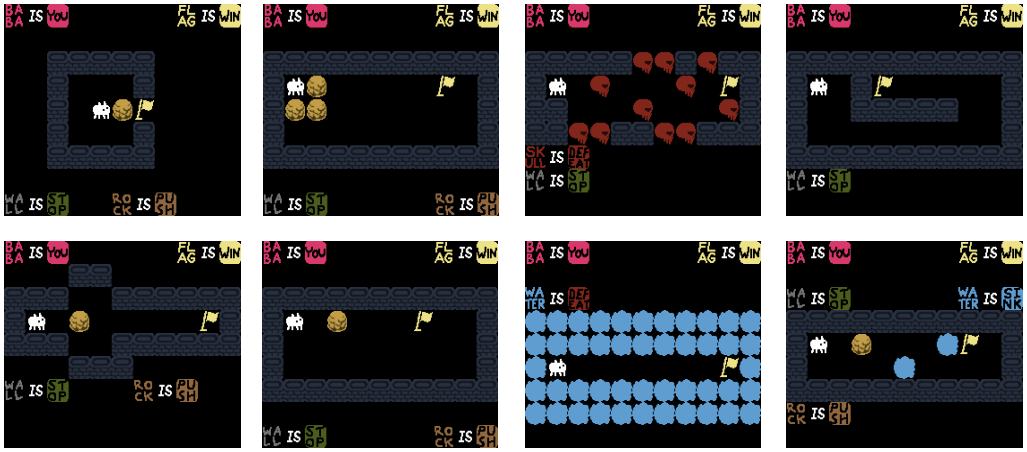
F.1 Solution Sample

F.2 Gemini Solution Sample

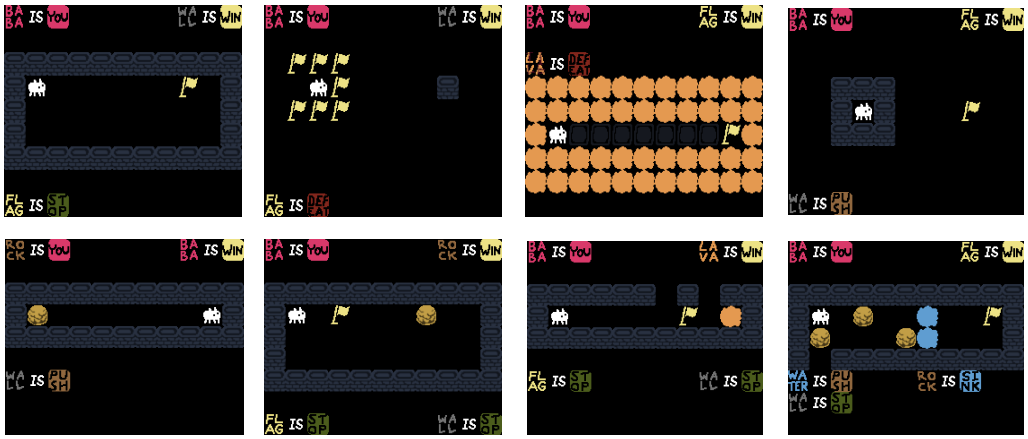
See Fig. A3.

F.3 LCV Solution Sample

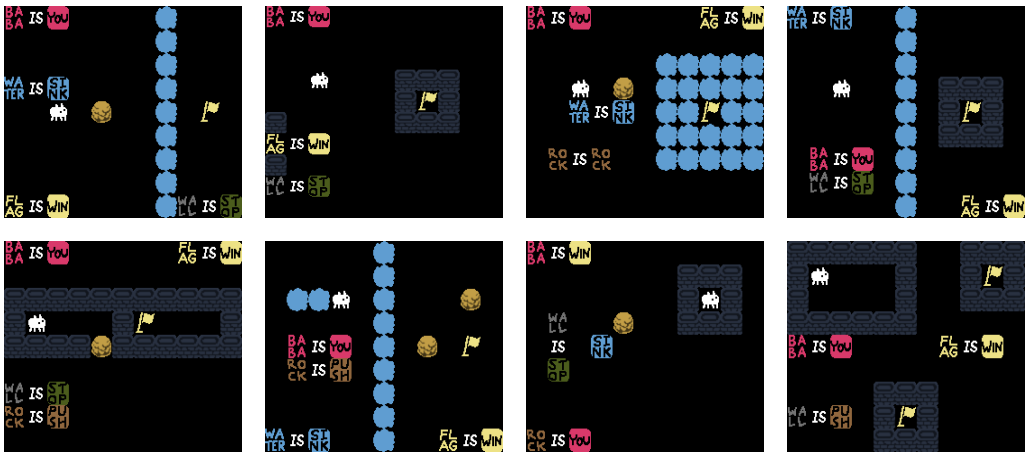
```
YOU_CHARS     = {}
STOP_CHARS    = {}
WIN_CHARS     = {}
DEFEAT_CHARS = {}
PUSH_CHARS    = {}
SINK_CHARS    = {}
MELT_CHARS    = {}
HOT_CHARS     = {}
DANGEROUS_TEXT_CHARS =
↪ {sorted(list(dangerous_text_chars))}

def next_state(grid, move):
    height = len(grid)
    if height == 0: return grid
    width = len(grid[0])

    new_grid = [row[:] for row in grid]
    directions = {{
        "UP": (-1, 0), "DOWN": (1, 0),
        ↪ "LEFT": (0, -1), "RIGHT":
        ↪ (0, 1)
    }}
    if move not in directions: return
    ↪ new_grid
    dy, dx = directions[move]

    # 1. Find YOU
    you_pos = []
    for r in range(height):
        for c in range(width):
            cell = new_grid[r][c]
            # Fast check
            has_you = False
            for char in cell:
                if char in YOU_CHARS:
                    has_you = True
                    break
            if has_you:
                for char in cell:
                    if char in YOU_CHARS:
                        you_pos.append((r,
                        ↪ c, char))

    if not you_pos: return new_grid

    # 2. Move YOU
    # Sort to handle multiple YOU
    ↪ consistently
    you_pos.sort()

    for r, c, me in you_pos:
        if me not in new_grid[r][c]:
        ↪ continue

        nr, nc = r + dy, c + dx

        # Map Boundary for YOU
        if not (0 <= nr < height and 0
        ↪ <= nc < width):
            continue

        target_cell = new_grid[nr][nc]

        # --- PUSH Logic ---
        has_push = False
        for obj in target_cell:
            if obj in PUSH_CHARS:
                has_push = True
                break
```



(a) **Tier 1: Semantic Alignment.** The rules match visual priors (*e.g.*, Walls stop movement), serving as a baseline for spatial planning capability.



(b) **Tier 2: Semantic Conflict.** Rules are adversarial to visual intuition. Here, `LAVA IS SAFE`, requiring the agent to overcome the "fear" bias.



(c) **Tier 3: Dynamic Plasticity.** The agent must manipulate blocks to rewrite physics (*e.g.*, making walls passable) to solve the puzzle.

Figure A2: **Visualizing the BABABENCH Difficulty Hierarchy.** We present three tiers of increasing neuro-symbolic friction. Unlike traditional RL environments with static affordances, our benchmark requires agents to handle: (a) aligned semantics, (b) counterintuitive inhibitory control, and (c) dynamic rule restructuring.

```
if has_push:
    chain = []
```

```
curr_r, curr_c = nr, nc
can_push = True
```

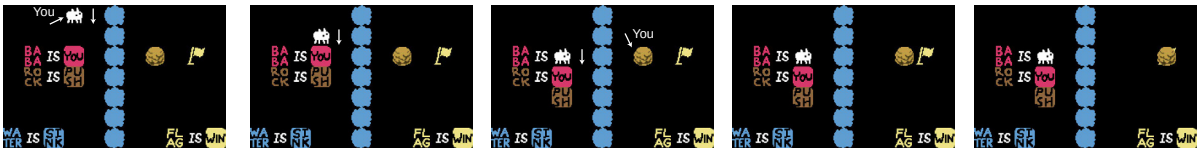


Figure A3: **Adaptive Inference Utility.** In semantically aligned scenarios (Green), Gemini plans concisely using intuitive priors. However, facing counterintuitive rules like `ROCK IS YOU` (Red), it spontaneously shifts to step-by-step reasoning verification loops. This demonstrates that overcoming semantic inertia requires significantly higher inference-time compute to inhibit visual priors.

```
while True:
    # [Critical] Boundary
    ↪ Check for PUSH Chain
    if not (0 <= curr_r <
    ↪ height and 0 <=
    ↪ curr_c < width):
        can_push = False;
        ↪ break

    cell_objs =
    ↪ new_grid[curr_r][curr_c]
   push_objs_here = [o for o
    ↪ in cell_objs if o in
    ↪ PUSH_CHARS]

    if not push_objs_here:
       # End of chain. Check
        ↪ BLOCKING.
        is_blocked = False
        for o in cell_objs:
         if o in STOP_CHARS:
                is_blocked =
                ↪ True;
                ↪ break
        if is_blocked:
        ↪ can_push = False
        break
    else:
       chain.append((curr_r,
        ↪ curr_c))
        curr_r += dy
        curr_c += dx

if not can_push:
    continue
```

```
    # Execute Push (Reverse
    ↪ Order)
    for tr, tc in
    ↪ reversed(chain):
        n_tr, n_tc = tr + dy, tc
        ↪ + dx

        src_cell =
        ↪ new_grid[tr][tc]
        moving = [o for o in
        ↪ src_cell if o in
        ↪ PUSH_CHARS]
        staying = "".join([o for
        ↪ o in src_cell if o
        ↪ not in PUSH_CHARS])

      new_grid[tr][tc] = staying
        new_grid[n_tr][n_tc] +=
        ↪ "".join(moving)

    # Refresh target_cell after
    ↪ push
    target_cell =
    ↪ new_grid[nr][nc]

# --- STOP Logic (Final check)
↪ ---
is_blocked = False
for obj in target_cell:
    if obj in STOP_CHARS:
        is_blocked = True; break
if is_blocked: continue

# --- Dangerous Text Logic
↪ (NEW!) ---
is_text_dead = False
```

```
for obj in target_cell:
    if obj in
    ↪ DANGEROUS_TEXT_CHARS:
      is_text_dead = True; break
if is_text_dead:
    new_grid[r][c] =
    ↪ new_grid[r][c].replace(me,
    ↪ "", 1)
    continue

# --- Enter Logic ---
is_dead = False
for obj in target_cell:
    if obj in DEFEAT_CHARS:
        is_dead = True; break
if is_dead:
    new_grid[r][c] =
    ↪ new_grid[r][c].replace(me,
    ↪ "", 1)
    continue

is_sink = False
sink_obj = None
for obj in target_cell:
    if obj in SINK_CHARS:
        is_sink = True; sink_obj
        ↪ = obj; break
if is_sink:
    new_grid[r][c] =
    ↪ new_grid[r][c].replace(me,
    ↪ "", 1)
    new_grid[nr][nc] =
    ↪ new_grid[nr][nc]. \\
    replace(sink_obj, "", 1)
    continue

is_melt_hot = False
if me in MELT_CHARS:
    for obj in target_cell:
        if obj in HOT_CHARS:
            is_melt_hot = True;
            ↪ break
if is_melt_hot:
    new_grid[r][c] =
    ↪ new_grid[r][c].replace(me,
    ↪ "", 1)
    continue

# Normal Move
new_grid[r][c] =
↪ new_grid[r][c].replace(me,
↪ "", 1)
        new_grid[nr][nc] += me

    return new_grid

def check_win(grid):
    height = len(grid)
    width = len(grid[0])
    for r in range(height):
        for c in range(width):
            cell = grid[r][c]
            if not cell: continue
            has_you = False
            has_win = False
            for char in cell:
                if char in YOU_CHARS:
                ↪ has_you = True
                if char in WIN_CHARS:
                ↪ has_win = True
            if has_you and has_win:
                return True
    return False
"""
```

G Further Experiments

G.1 Ablation I: Training and Inference Efficiency

We give an empirical comparison about the computational cost of our approach against both iterative reasoning baselines and state-of-the-art proprietary models in Table A3. All local training and inference experiments, including our LCV model and the Qwen/TheoryCoder baselines, were conducted on a single node equipped with 8 × NVIDIA RTX 3090 (24GB) GPUs. Proprietary models (Gemini and GPT-4o) were evaluated via official APIs.

The results demonstrate the relative efficiency of the LCV strategy. To solve the required code generation tasks, Gemini 3 Pro requires over 3k tokens of reasoning to navigate the logical conflict. Similarly, TheoryCoder scales linearly with problem complexity. In contrast, the LCV agent requires only a minimal training investment ($\approx$ 600 paired samples for 5 epochs, costing about 30 minutes in one computing node). Once trained, it generates the correct world model in a single, compact pass ($\approx$ 800 tokens) on local hardware, reducing inference latency by approximately $4\times$ to $7\times$ compared to CoT-based approaches. This confirms that the contrastive objective successfully transforms the complex cognitive task of inhibition into an efficient, retrieved reflex.

Table A3: **Efficiency Comparison of Code-Based Methods.** While large foundation models (Qwen) and reasoning models (Gemini) rely on heavy test-time compute (either via parameter count or token volume), LCV utilizes a small SFT set ($\sim$600 samples) to enable rapid, single-pass world modeling on local hardware.

Method	Inference Mode	Avg Tok.	Time (s)
Qwen2.5-72B-Instruct	Direct Ans	$\sim$1k	42.5
Gemini 3 Pro Preview	Think + Ans	$\sim$3k	35.0
TheoryCoder	Iterative Loop	$\sim$3k	65.2
LCV (Ours)	**SFT + Ans**	**$\sim$1k**	**8.4**

G.2 Comparison with o1-preview

Table A4: **o1-preview Results on BABABENCH.** Despite extended internal chain-of-thought, o1-preview does not escape semantic inertia in high-conflict tiers.

Method	Tier 1	Tier 2	Tier 3
o1-preview (Direct Policy)	57.8%	11.1%	12.0%
o1-preview (Code-as-Policy)	62.2%	26.7%	30.0%

While o1-preview's extended CoT improves logical consistency over standard models, it does not close the gap with LCV. Qualitative analysis reveals that o1 occasionally identifies the correct rule (*e.g.*, `WALL IS YOU`) within its reasoning trace but still produces a `STOP` action in the final step, a clear residual signature of semantic inertia. This suggests that inference-time scaling of reasoning tokens—without representational grounding—is insufficient to fully overcome entrenched parametric priors. We did not directly compare against long-CoT RL approaches (*e.g.*, DeepSeek-R1) because their step-by-step token generation incurs prohibitive latency in multi-step interactive environments; our amortized approach achieves a $4\times$ latency reduction precisely by shifting inhibitory control from inference to training time.

G.3 OOD Generalization Robustness

To test the generalization capability of different baselines, we define three specific evaluation protocols:

1. In-Distribution (In-Dist.): The test set follows the same data distribution as the training set, serving as the standard benchmark.

2. Map Generalization (Map Gen.): Models are evaluated on novel spatial environments and map layouts that were not seen during training. This tests the model's ability to decouple logical rules from specific spatial configurations.

3. Combination Generalization (Combo Gen.): This setting introduces novel combinations of rules and logical constraints, creating a systematic distribution shift that tests compositional generalization.

Table A5: **Generalization Robustness.** Success Rate (SR) across generalization splits.

Method	In-Dist.	Map Gen.	Combo Gen.
Qwen2.5-72B-Instruct	52.14%	53.40%	48.22%
Gemini 3 Pro Preview	42.85%	42.60%	39.62%
TheoryCoder	55.71%	53.72%	50.68%
LCV (Ours)	76.42%	74.28%	72.36%

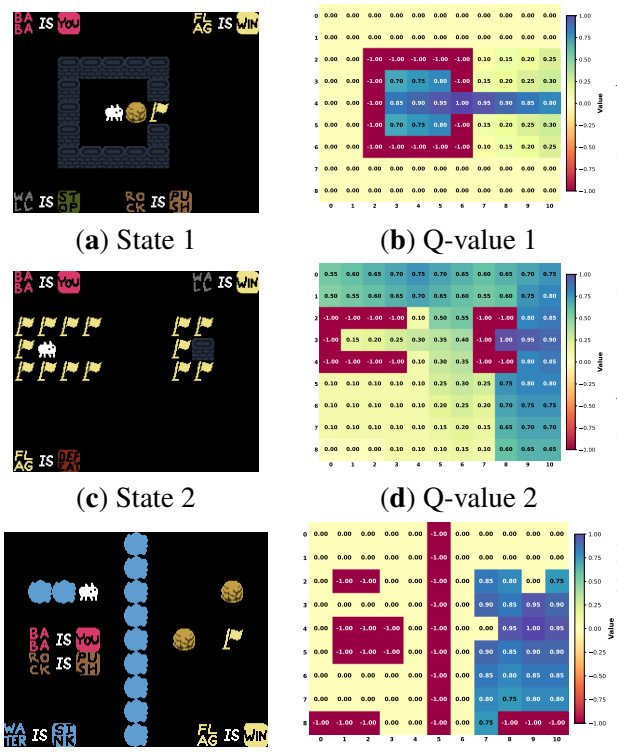
We employ a Map Generalization set consisting of 30 unseen spatial layouts to assess whether the model overfits to specific map configurations. Furthermore, to test compositional reasoning, we construct a Combination Generalization split that introduces three systematic rule changes unseen during training: **Water is Push**, **Rock is Defeat**, and **Wall is Melt**.

As shown in Table A5, LCV significantly outperforms baselines across all metrics. A potential concern regarding LCV is its generalization capability; unlike generalist LLMs (*e.g.*, Qwen, Gemini) that leverage vast pre-trained parametric knowledge, LCV relies on fine-tuning, which can theoretically increase the risk of overfitting. However, our statistical analysis refutes this. In the Map Gen. setting, we observed no statistically significant difference ($p>0.05$) between the In-Distribution and Map Generalization scores, quantitatively verifying that LCV is robust to spatial variations.

Regarding the Combo Gen. split, while all methods experience a performance drop, we attribute this to the systematic deviation introduced by the new rules. To validate this, we compared the magnitude of the performance drop of LCV against that of the baselines. We also found no statistically significant difference in the degradation rates ($p>0.05$). This confirms that the drop is a natural consequence of the increased logical complexity shared across all models, rather than a weakness specific to our approach. Crucially, LCV maintains a substantial lead ($\sim$20%), proving its reasoning framework remains robust even when facing these novel logical constraints.

G.4 Q-Value Visualization for Model Reasoning

To further understand the model's reasoning and decision-making process, we also recorded the model's predicted Q-values (action-value functions) at each inference step. As shown in Fig. A4,



(**a**) State 1 (**b**) Q-value 1

(**c**) State 2 (**d**) Q-value 2

(**e**) State 3 (**f**) Q-value 3

Figure A4: **Q-value Visualizations.** For each scenario, the left panel shows the environment state, and the right panel shows the model's predicted Q-values for all possible actions in that state. From top to bottom: semantic alignment (Tier 1), semantic conflict (Tier 2), and dynamic rule change (Tier 3).

the left column contains visualizations of representative game states, while the right column shows the corresponding Q-value distributions over possible actions in those states. We present examples covering aligned semantics, semantic conflict, and dynamic rule changes.

As shown in Fig. A4, for Tier 1 (semantic alignment), the model's Q-value distribution is intuitive, preferring optimal actions that match commonsense reasoning (e.g., moving directly toward the goal while avoiding walls). In Tier 2 (semantic conflict, e.g., "WALL IS YOU"), baselines without explicit code grounding still assign low or even negative Q-values to actions that violate their learned priors, reflecting an inability to inhibit semantic inertia. In contrast, our LCV model, trained with contrastive objectives, correctly identifies that interacting with the wall is in fact desirable under the current rules and assigns higher Q-values to these actions. In Tier 3 (dynamic rule change), generic LLM-based agents often display significant Q-value instability and lag in adapting to new physics.

H Declaration

AI tools were used solely for grammar checking and language polishing for the overall paper; all ideas, technical content, and substantive writing are entirely the authors' own.

All code and benchmark data used in this work are released under the MIT License, allowing free use, modification, and distribution for research and non-commercial purposes.

We confirm that all existing artifacts used in this work were employed in accordance with their intended use and licensing terms. For the artifacts we created, we specify that they are intended strictly for research purposes, and their release is fully compatible with the original access conditions of any resources on which they are based.