Outcome-Aware Agents: From Token Prediction to Action Consequence Modeling

Mallesh Madapathi[1]

[1]thinkingdbx pvt. ltd., Hyderabad, India , mallesh@thinkingdbx.com

May 25, 2026

Abstract

Large language models are increasingly deployed as agents. They write code that executes, run database migrations, navigate browsers, and operate on production systems. Their training objective, next-token prediction, contains no signal about what their actions actually do. A code agent that proposes `DROP TABLE` has not modeled the resulting database state. It has predicted a plausible token sequence. We argue this is the central reliability gap in current agent systems, and that closing it requires giving agents an internal forward model: given a candidate action, predict the outcome before executing. We propose the Action-Conditioned Latent Structural Causal Model (AC-LSCM) as one such mechanism. AC-LSCM maintains a small set of latent factors related by a learned sparse directed acyclic graph (DAG), and implements actions as structural interventions in the sense of Pearl's do-operator rather than as context concatenation. We evaluate AC-LSCM on synthetic structural causal models and report two findings. First, on a safety-critical agent planning task, AC-LSCM reduces safety violations by roughly $36\times$ relative to a Transformer baseline (mean 0.005 versus 0.180 across 13 seeds). Second, the architecture as originally specified is over-engineered: ablations show the do-operator and the abduction loop carry the result, while the NOTEARS DAG constraint and a contrastive hinge term are net-negative interventions that we recommend removing. We report the negative results in full and propose a simplified follow-up architecture.

Keywords: LLM agents, outcome modeling, causal inference, world models, action selection, structural causal models.

1 Introduction: Agents That Do Not Know What They Are Doing

A code agent generates and executes a database migration. The migration is syntactically valid. It follows recognizable patterns from training data, and matches the user's intent in natural language. It also silently drops a foreign key constraint that fifteen downstream services depend on. The model did not predict this consequence because the model does not predict consequences. It predicts tokens.

This is not hypothetical. As LLMs are wired into agent frameworks (tool-using assistants, autonomous coders, browser agents, data engineering copilots), they take actions that change real systems. The training objective remains $P(w_t \mid w_{<t})$. Nothing in that objective rewards an accurate model of what happens after the action is taken. An LLM agent acts the way a person speaks while sleepwalking: fluent, coherent, and not connected to the world it appears to be talking about.

The core thesis of this paper is straightforward. An AI that takes actions should know, in advance, what those actions will cause. Achieving this requires more than scaling token prediction. It requires giving agents an explicit internal forward model that, given a candidate action and the current state, predicts the resulting state, and using that model to filter, rank, or reject candidate actions before execution.

We propose one such architecture, the Action-Conditioned Latent Structural Causal Model (AC-LSCM), and a planning loop that uses it for outcome-aware action selection. The architecture draws on three established lines of work: action-conditioned latent world models [5, 6], causal representation learning [2], and continuous DAG optimization [3, 4]. The specific contribution is to combine them into an agent-facing mechanism with three capabilities that current LLM agents lack: forward simulation of outcomes, counterfactual evaluation across alternative actions, and verified action selection that rejects actions whose simulated outcomes violate goals or constraints.

Contributions. This paper makes four contributions. (1) An architecture specification combining a variational encoder, a DAG-structured transition operator that implements Pearl's do-operator, and a generative decoder. (2) A four-term training objective and a three-step counterfactual inference procedure (abduction, intervention, prediction). (3) Empirical evaluation on synthetic SCMs and an agent-task benchmark. On the agent task, AC-LSCM achieves roughly 36$\times$ fewer safety violations than a Transformer baseline. (4) Honest negative results from ablation: the do-operator carries the work, while the NOTEARS DAG constraint and a contrastive hinge term hurt more than they help at the scales tested. We recommend a simplified follow-up.

Scope. All experiments are on synthetic SCMs at K up to 20. We do not yet evaluate on language benchmarks or scale the architecture to LLM regimes. Section 10 is explicit about what this work does and does not establish.

2 The Agent Problem with Token Prediction

2.1 Why next-token training fails as an action policy

An autoregressive language model is trained to maximize $\log P(w_t \mid w_{<t})$ over a corpus. When such a model is deployed as an agent, the action a_t is just another token sequence sampled from this distribution. The model has no representation of the world state the action will modify, no representation of the state that will result, and no objective term that rewards accuracy of either. What it has is a representation of which action-tokens tend to follow which contexts in the training corpus.

This is sufficient when both the action and its consequences are well represented in training data, and when the consequences are themselves textual. It is insufficient in three cases that show up constantly in agent deployments. First, when the action causes a state change in a system outside the text stream: filesystem, database, browser DOM, physical actuator. Second, when the training corpus does not contain the specific configuration the agent is currently acting on. Third, when the right action depends on the resulting state rather than on surface similarity to training contexts. In data engineering, agentic code execution, and operations work, all three are common. The result is a class of failure that is not hallucination in the usual sense (the model is not making up facts) but rather action without outcome modeling.

2.2 Concrete failure modes

The pattern shows up across deployed systems. Code agents emit destructive shell commands when the surface form of the task resembles a cleanup. SQL agents propose joins that match query

templates from training data but produce wrong results on the actual schema. Browser agents submit forms that look like the right form but operate on the wrong account. Data pipeline agents recommend transformations whose effect on downstream consumers they cannot articulate, because they never represented those consumers in the first place.

In each case, the proximate cause is the same. The agent selected an action based on pattern-matching against training data rather than against a model of what the action will do to the current system state.

2.3 What an outcome-aware agent needs

An agent that knows the consequences of its actions needs four things: a compact representation of the current world state $\mathbf{z}_t$; a transition model $p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, a_t)$ that predicts the next state given an action; an evaluation function $V(\mathbf{z}_{t+1})$ that scores predicted outcomes against goals and constraints; and a selection rule that uses simulated outcomes to choose, defer, or reject candidate actions.

The transition model is the part that next-token training does not provide. Building it so that it actually models causal consequences (not just correlations between action-tokens and successor-tokens) is the technical problem this paper addresses.

3 Pearl's Hierarchy as a Diagnostic

Pearl [1] distinguishes three levels of reasoning about a system. Association ($P(y \mid x)$) is about what is observed to co-occur. Intervention ($P(y \mid \mathrm{do}(x))$) is about what happens if a variable is forced to a value by an external agent. Counterfactual ($P(y_x \mid x', y')$) is about what would have happened in a specific realized case had the agent acted differently.

Autoregressive LLMs operate at the first level. They are estimators of the joint distribution of token sequences, and their conditionals are conditional on observed context, not on interventions. The do-operator is the mathematical device that distinguishes the second level from the first. $P(y \mid \mathrm{do}(x))$ severs the natural causes of x and substitutes the externally imposed action, then propagates consequences through the rest of the system.

For an agent, this is the difference between "users who run command X tend to be in situation Y afterward" (association) and "if I run command X, the system will be in situation Y afterward" (intervention). A token-prediction model can learn the first. Only an architecture with explicit interventional semantics can learn the second.

Dimension	**Token-prediction agent**	**Outcome-aware agent (AC-LSCM)**
What it models	Token distributions in context	World-state transitions under actions
Action handling	Action-tokens emitted from context	Structural intervention on state model
Causal rung	Association	Intervention and counterfactual
Failure mode	Plausible action, wrong outcome	Action filtered if simulated outcome is wrong

Table 1: Two agent paradigms.

4 The AC-LSCM Architecture

4.1 State representation

We represent the world state as K disentangled latent factors:

$$\mathbf{z}_t = (z_t^1, z_t^2, \ldots, z_t^K), \qquad z_t^i \in \mathbb{R}^{d_i}. \tag{1}$$

Each factor is intended to track a single mechanism in the environment. Disentanglement is encouraged (not guaranteed; see Section 10) by the structural transitions and the DAG sparsity constraint below.

4.2 Structural transitions

Transitions are governed by structural equations parameterized by neural networks f_i:

$$z_{t+1}^i \;:=\; f_i\big(\mathrm{Pa}(z_t^i),\; a_t^i,\; \epsilon_t^i\big), \qquad i \in \{1, \ldots, K\}, \tag{2}$$

where $\mathrm{Pa}(z_t^i) \subseteq \mathbf{z}_t$ is the parent set under the learned DAG, a_t^i is the action component targeting factor i (null if the action does not touch this factor), and $\epsilon_t^i \sim p(\epsilon)$ is an exogenous noise term. The parent structure comes from a learned adjacency matrix $A \in \mathbb{R}^{K \times K}$ trained jointly with the f_i.

4.3 Action as intervention, not concatenation

This is the architectural commitment that distinguishes AC-LSCM from a generic action-conditioned world model. When action a_t^i is non-null at factor i, the transition f_i does not take $\mathrm{Pa}(z_t^i)$ as input. The natural causes are severed and the intervention value is substituted. This is Pearl's do-operator made explicit at the level of the computation graph. A standard sequence model instead concatenates the action embedding into the input and lets the network learn whether to attend to it, which conflates association and intervention.

4.4 The three trainable modules

The system has three components. An encoder $q_\phi(\mathbf{z}_t \mid x_t)$ maps observations to latent factors. A decoder $p_\theta(x_t \mid \mathbf{z}_t)$ reconstructs observations and provides a grounding signal. A transition operator $p_\psi(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathrm{do}(a_t))$ implements the structural equations with interventional semantics.

5 Training Objective

$$\mathcal{L}_{\text{Total}} = \beta_1 \mathcal{L}_{\text{Recon}} + \beta_2 \mathcal{L}_{\text{Transition}} + \beta_3 \mathcal{L}_{\text{Causal}} + \beta_4 \mathcal{L}_{\text{Contrastive}}. \tag{3}$$

The reconstruction loss anchors the latent to observations:

$$\mathcal{L}_{\text{Recon}} = \mathbb{E}_{\mathbf{z}_{t+1} \sim q_\phi(\cdot \mid x_{t+1})}\big[-\log p_\theta(x_{t+1} \mid \mathbf{z}_{t+1})\big]. \tag{4}$$

The transition loss trains the forward model to match the next-state posterior:

$$\mathcal{L}_{\text{Transition}} = \mathbb{E}_{x_t, a_t, x_{t+1}}\Big[D_{\mathrm{KL}}\big(q_\phi(\mathbf{z}_{t+1} \mid x_{t+1}) \,\|\, p_\psi(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathrm{do}(a_t))\big)\Big]. \tag{5}$$

The DAG loss prevents factors from collapsing into a fully connected, mutually entangled blob:

$$\mathcal{L}_{\text{Causal}} = \lambda_1 \|A\|_1 + \lambda_2\big(\mathrm{tr}(e^{A \odot A}) - K\big). \tag{6}$$

The first term encourages sparsity. The second is the NOTEARS [3] characterization of acyclicity, which is zero if and only if A is a DAG. The log-determinant characterization of DAGMA [4] is an alternative and we compare them empirically below.

The counterfactual contrastive loss uses ground-truth counterfactuals where available:

$$\mathcal{L}_{\text{Contrastive}} = \left\|\tilde{\mathbf{z}}_{t+1} - \mathbf{z}^{\text{cf}}_{t+1}\right\|_2^2 \; + \; \max\bigl(0,\; \gamma - \|\tilde{\mathbf{z}}_{t+1} - \mathbf{z}_{t+1}\|_2^2\bigr), \tag{7}$$

where $\tilde{\mathbf{z}}_{t+1} \sim p_\psi(\cdot \mid \mathbf{z}_t, \text{do}(\tilde{a}_t))$ is the model's counterfactual prediction. The first term supervises against ground truth. The hinge term keeps counterfactual and factual trajectories from collapsing into the same prediction. As we report below, the hinge term turns out to be the wrong design.

6 Counterfactual Inference

For a query of the form "we observed (x_t, a_t, x_{t+1}); what would x_{t+1} have been under action $\tilde{a}_t$?", the model executes Pearl's three-step procedure.

Abduction. Recover the realized noise consistent with the observed outcome:

$$\hat{\epsilon}_t \sim q_\phi(\epsilon_t \mid x_t, a_t, x_{t+1}). \tag{8}$$

Intervention. Replace a_t with $\tilde{a}_t$, holding all parameters and the inferred noise fixed.

Prediction. Roll the structural equations forward and decode:

$$\mathbf{z}^*_{t+1} = f_\psi(\mathbf{z}_t, \tilde{a}_t, \hat{\epsilon}_t), \tag{9}$$

$$x^*_{t+1} = p_\theta(x_{t+1} \mid \mathbf{z}^*_{t+1}). \tag{10}$$

Reusing the same noise across factual and counterfactual evaluation is what makes the comparison counterfactual rather than merely interventional.

7 The Outcome-Aware Planning Loop

The architecture gives the agent a forward simulator. Algorithm 1 describes how the agent uses it for action selection. This is the link between AC-LSCM as a model and AC-LSCM as an agent component.

Three properties are worth noting. Safety filtering happens before execution, on simulated outcomes, not on action text. Deferral is a first-class output rather than a low-confidence prediction. Prediction error after execution provides a training signal for the world model and a flag that the agent's outcome predictions for this kind of action should not be trusted.

8 Experimental Setup

Compute. All experiments ran on a single NVIDIA Tesla T4 (16 GB VRAM, fp32) on a Google Cloud preemptible instance, with checkpointing to survive preemptions. Total compute for the results reported here was roughly 40 GPU hours.

Synthetic SCM environments. We generated structural causal models with known ground-truth DAGs and known counterfactual outcomes. Four graph structures (chain, fork, collider, Erdős-Rényi random) and node counts $K \in \{10, 20\}$. Each environment produces $(x_t, a_t, x_{t+1}, \epsilon_t, x^{\text{cf}}_{t+1}, \tilde{a}_t)$ tuples with the same noise reused across factual and counterfactual rollouts. We trained at $N = 10{,}000$ samples for the main results and ran an $N = 2{,}000$ control to isolate the contribution of data volume versus architectural choices.

Algorithm 1 Outcome-aware action selection

Require: Current observation x_t, candidate action set $\mathcal{A}_t$, goal predicate G, safety predicate S, value function V

$\mathbf{z}_t \leftarrow q_\phi(\mathbf{z}_t \mid x_t)$
$\mathcal{C} \leftarrow \emptyset$
for each $a \in \mathcal{A}_t$ **do**
    $\mathbf{z}_{t+1}^{(a)} \leftarrow p_\psi(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathrm{do}(a))$
    if $S(\mathbf{z}_{t+1}^{(a)})$ holds **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{(a, \mathbf{z}_{t+1}^{(a)})\}$
    end if
end for
if $\mathcal{C} = \emptyset$ **then**
    return DEFER
end if
return $\arg\max_{(a,\mathbf{z}')\in\mathcal{C}} \big[\, V(\mathbf{z}') + \mathbb{1}[G(\mathbf{z}')] \,\big]$

Baselines. Three baselines share the same encoder and decoder as AC-LSCM but differ in the transition model. *Vanilla VAE* uses a single MLP transition with the action concatenated to the input. *Dreamer-style* uses a recurrent transition cell. *Small Transformer* treats (x_t, a_t) as a token sequence and predicts x_{t+1}. The Transformer is the closest analog to a token-prediction agent and is the target of the thesis claim.

Metrics. Intervention MSE (held-out prediction error under $\mathrm{do}(x)$), counterfactual MSE (held-out error after abduction-intervention-prediction), structural Hamming distance between learned and true DAGs, the NOTEARS constraint value at convergence, and abduction recovery error. For the agent task we report goal achievement rate, safety violation rate, and appropriate deferral rate on a setup of 100 episodes per seed: 80 with a safe-and-goal-reachable option and 20 with no safe action available.

All reported numbers are mean and standard deviation across three seeds, except the agent task on ER K=10 where we ran 13 seeds (the original three, five extensions, and a five-seed N=2k control).

9 Results

9.1 Agent task: the headline result

Table 2 reports the agent-task evaluation on ER K=10. AC-LSCM achieves roughly 36 times fewer safety violations than the Transformer baseline when averaged across all 13 with-fixes seeds (0.005 versus 0.180), and 12 of those 13 seeds produce zero safety violations.

The Transformer's goal rate of 0.800 is not a result of the baseline being weak. It is the task ceiling: 80 of 100 episodes have a safe action that achieves the goal, and any predictor with MSE below roughly 0.05 will rank that action first. AC-LSCM's nine working seeds also tie this ceiling. The actual difference between the two models is what happens on the 20 no-safe-action episodes. The Transformer acts in 18 of those 20 episodes and violates safety in most. AC-LSCM defers in all 20 on its working seeds.

Model	Goal Rate	Safety Violations	Appropriate Deferral
Small Transformer (n=3)	0.800 ± 0.000	0.180 ± 0.022	0.100 ± 0.108
AC-LSCM, all seeds (n=13)	0.554 ± 0.369	$\mathbf{0.005 \pm 0.019}$	$\mathbf{0.985 \pm 0.053}$
AC-LSCM, working seeds (n=9)	0.800 ± 0.000	$\mathbf{0.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$

Table 2: Agent-task evaluation on ER K=10. Transformer's 0.800 goal rate is the task ceiling given the episode mix (80 with safe goal action, 20 with none). Working AC-LSCM seeds tie this ceiling while eliminating safety violations.

9.2 Failure modes

Four of the 13 AC-LSCM seeds did not produce a usable planner despite normal training-time MSE. The failure modes break into two distinct patterns. Three seeds (23%) converged to what we call a passive policy: the model never selects the goal action, picking some safe non-goal action every time. This produces zero safety violations and perfect appropriate deferral but zero goal achievement. One seed (8%) showed a value-function mis-rank: the predicted next-state value for some safe-but-non-goal action exceeded the value for the direct goal action, leading to occasional unsafe picks.

Both failure modes have intervention MSE in the same range (0.053 to 0.066) as the working seeds. The pathology is at the planning-time value-ranking step, not at training-time prediction. Even on the failure seeds, AC-LSCM's safety violation rate (maximum 0.07) stays below the Transformer's mean (0.18). The safety claim is robust to the training instability. The goal-rate claim is conditional on training succeeding, which is currently a 9/13 (69%) event.

9.3 Attribution: architectural fixes versus data volume

Round 1 of this work, with $N = 2{,}000$ and an earlier version of the contrastive loss, produced high safety violations (0.267) and zero appropriate deferral. The improvements reported above came alongside two changes: an architectural fix (clean ground-truth counterfactual supervision plus a DAG curriculum warm-up), and 5× more data. To attribute the win to one or the other, we ran a control: five seeds with the architectural fixes at $N = 2{,}000$.

Configuration	N	Fixes	Goal Rate	Safety Viol.	Approp. Deferral
Round 1 baseline (3 seeds)	2k	no	0.510	0.267	0.000
N=2k control (5 seeds)	2k	yes	0.480	**0.000**	**1.000**
Round 2 main (8 seeds)	10k	yes	0.600	**0.009**	**0.975**

Table 3: Attribution control: the architectural fixes account for the safety and deferral improvement. Going from Round 1 to the N=2k control (same data, fixes added) eliminates safety violations and lifts deferral to perfect. Going from N=2k control to Round 2 (same fixes, 5× data) does not move the agent-task metrics further.

The architectural fixes alone account for the safety improvement. Going from Round 1 to the N=2k control drops safety violations from 0.267 to 0.000 and lifts appropriate deferral from 0.000 to 1.000. The additional five-fold data increase between the N=2k control and the main run leaves agent-task metrics essentially unchanged. Intervention MSE improves modestly with more data (0.060 to 0.057, a 5% reduction). Counterfactual MSE actually has a slightly higher mean at N=10k due to one high-CF seed, within seed-to-seed variance.

9.4 Intervention and counterfactual MSE

Table 4 reports MSE on the synthetic environments. On raw counterfactual MSE, AC-LSCM is worse than the baselines on three of four graphs, often by a factor of two or more. This contradicts our pre-registered hypothesis that the explicit counterfactual machinery would translate into lower CF MSE.

Model	Chain	Fork	Collider	ER K=10
Small Transformer	0.046 / 0.049	0.033 / 0.034	0.031 / 0.032	0.038 / 0.039
Vanilla VAE	0.031 / 0.032	0.030 / 0.030	0.025 / 0.025	0.028 / 0.029
Dreamer-style	0.032 / 0.034	0.029 / 0.029	0.025 / 0.024	0.028 / 0.029
AC-LSCM	0.101 / 0.064	0.057 / 0.177	0.046 / 0.052	0.060 / 0.130

Table 4: Intervention MSE / Counterfactual MSE on synthetic environments. Lower is better. AC-LSCM does not win on raw MSE.

There is one structural pattern in the AC-LSCM numbers that the baselines do not show. On chain ($0.064 < 0.101$), on collider (0.052 vs 0.046, roughly equal), and on ER K=20 (CF 0.130 vs Int 0.220), counterfactual MSE is lower than or comparable to intervention MSE. For every baseline, the two are essentially equal because the baselines have no special counterfactual procedure and a CF prediction is just an Int prediction with a different action input. The AC-LSCM asymmetry is a fingerprint of the abduction mechanism doing real work, even where the encoder and decoder struggle with high-dimensional latent recovery. The DAG constraint converged exactly to zero on every AC-LSCM seed across all configurations, and structural Hamming distance on ER K=10 was 8.0 with zero variance: the learned graph is precise.

9.5 Ablations: what is actually pulling weight

The ablation results are the most uncomfortable finding in this paper, and we report them in full. Table 5 ablates each loss term and the do-operator on ER K=10.

Variant	Int MSE	CF MSE	SHD
AC-LSCM (full)	0.060 ± 0.003	0.130 ± 0.020	8.0
no $\mathcal{L}_{\text{Causal}}$ ($\beta_3 = 0$)	$\mathbf{0.033 \pm 0.001}$	$\mathbf{0.068 \pm 0.011}$	68.7
no $\mathcal{L}_{\text{Contrastive}}$ ($\beta_4 = 0$)	0.058 ± 0.002	0.077 ± 0.005	8.0
no do-operator	0.060 ± 0.002	0.138 ± 0.022	8.0
DAGMA instead of NOTEARS	0.033 ± 0.001	0.075 ± 0.009	36.3

Table 5: Ablations on ER K=10. Three of four ablations improve CF MSE. Only removing the do-operator makes things worse. The DAG loss and the contrastive hinge are net-negative interventions at this scale.

Three findings stand out. First, removing the do-operator is the only ablation that hurts. It raises CF MSE from 0.130 to 0.138. This isolates the do-operator and the abduction loop as the working components of the architecture. Second, removing the DAG loss gives the best CF MSE we have measured for AC-LSCM (0.068, a 48% improvement over the full model). The cost is that structural Hamming distance jumps to 68.7, near random. The model is not learning structure without the constraint, but the structure also is not helping prediction. Third, removing the contrastive hinge improves CF MSE from 0.130 to 0.077. The likely culprit is the hinge margin:

when factual and counterfactual trajectories are naturally close, pushing them apart by a fixed margin produces spurious gradients.

DAGMA shows an interesting middle ground. It gives nearly identical MSE to the no-DAG-loss ablation (Int 0.033, CF 0.075) but recovers partial structure (SHD 36.3, between the NOTEARS-precise 8 and the structureless 69). The DAGMA log-determinant term does not converge to zero in our setup (it stays around 30+), but the soft pressure is enough to bias the encoder toward partial disentanglement without crushing the loss surface.

9.6 What this means for the architecture

The ablation results, taken together, point to a simpler architecture as the right next step. Keep the encoder, decoder, transition operator, and the do-operator semantics. Keep the supervised counterfactual loss. Remove the contrastive hinge. Replace the hard NOTEARS constraint with a soft DAGMA-style regularizer, or drop the structural loss entirely if structure recovery is not a paper deliverable. We have not yet retrained under this simplified configuration. That is the most important follow-up.

10 Limitations and Open Questions

Training instability. Roughly 31% of seeds (4 of 13) failed to produce a usable planner despite normal MSE. The failure is at the planning-time value-ranking step. The simplified architecture proposed above may or may not fix this. Adding a validation-time planning diagnostic during training, so degenerate value functions are caught before being evaluated, is a near-term follow-up.

Scale. Experiments here are at K up to 20, with small MLPs and modest training samples. Whether the DAG constraint, the disentanglement pressure, and the contrastive supervision all scale to high-dimensional latent spaces is unresolved. The negative results on the DAG loss already suggest it may not be the right tool even at K=20.

Task ceiling. The agent task as currently constructed has a task ceiling of 0.800 on goal rate because 80% of episodes have a trivially-rankable goal action. Working AC-LSCM seeds tie the Transformer baseline at this ceiling. To discriminate between models that are at-ceiling versus above-ceiling, future runs should raise the no-safe-action fraction and add upstream-effect distractors that make the optimal action harder to identify.

Identifiability. The encoder is not guaranteed to recover the true disentangled factors. Many latent representations explain the same data up to permutation and rotation [2]. We do not solve this problem.

Counterfactual supervision. The contrastive loss, when present, requires ground-truth counterfactuals. This restricts AC-LSCM training to synthetic, simulated, or oracle-supervised environments. Extension to purely factual observational data is future work.

No language-domain evaluation. All experiments are on synthetic SCMs. We have not yet evaluated AC-LSCM on language-based causal benchmarks (CLadder [7], CounterBench [8]) or on agent benchmarks that involve text-based actions. Doing so is the most important external-validity step.

11 Related Work

Action-conditioned latent world models. Ha and Schmidhuber [5] established the template. The Dreamer line [6] scaled it to challenging environments. AC-LSCM differs in adding an explicit DAG over latent factors and treating actions as interventions rather than concatenated inputs. Our

ablation results suggest the DAG component is not pulling its weight at the scales tested, which weakens that distinction in practice.

Causal representation learning. Schölkopf and colleagues [2] survey the program of learning representations aligned with causal mechanisms. AC-LSCM is a concrete instantiation oriented toward agent decision-making.

Continuous DAG optimization. NOTEARS [3] reformulated DAG learning as continuous optimization. DAGMA [4] improved numerical stability via a log-determinant characterization. Our results favor DAGMA as a soft regularizer over NOTEARS as a hard constraint.

Causal reasoning in LLMs. CLadder [7] and CounterBench [8] document that current LLMs perform poorly on formal causal reasoning, especially counterfactuals. These benchmarks motivate the problem AC-LSCM targets, and we have not yet evaluated on them.

12 Conclusion

LLM agents act on the world without modeling what their actions cause. Closing this gap requires giving agents an explicit forward model with interventional semantics, not just scaling token prediction. We have proposed AC-LSCM as one such mechanism. On a safety-critical agent task, AC-LSCM eliminates safety violations on 12 of 13 seeds and reduces the mean violation rate by roughly 36× relative to a Transformer baseline. The architecture as we originally proposed it is over-engineered: ablations show the do-operator and abduction loop carry the result, while the structural and contrastive components are net-negative interventions at the scales tested. We recommend a simplified follow-up and have been explicit about what this work does not establish: language-domain evaluation, scale, and the training instability that still affects roughly a third of seeds.

Acknowledgments

The author thanks the broader open research community working on causal representation learning, world models, and agent safety, whose published artifacts and benchmarks made this work possible.

Code and Reproducibility

Code, configs, and per-seed result JSONs will be released alongside the preprint. All experiments ran on a single NVIDIA Tesla T4 in fp32 with deterministic seeding and checkpointing. Total compute budget was under 50 GPU-hours.

References

[1] J. Pearl, *Causality: Models, Reasoning, and Inference*, 2nd ed. Cambridge University Press, 2009.

[2] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio, "Towards causal representation learning," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 612–634, 2021.

[3] X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing, "DAGs with NO TEARS: continuous optimization for structure learning," in *Advances in Neural Information Processing Systems*, vol. 31, 2018, pp. 9472–9483.

[4] K. Bello, B. Aragam, and P. Ravikumar, "DAGMA: learning DAGs via M-matrices and a log-determinant acyclicity characterization," in *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[5] D. Ha and J. Schmidhuber, "World models," arXiv preprint arXiv:1803.10122, 2018.

[6] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," arXiv preprint arXiv:2301.04104, 2023.

[7] Z. Jin, Y. Chen, F. Leeb, L. Gresele, O. Kamal, Z. Lyu, K. Blin, F. Gonzalez, M. Kleiman-Weiner, M. Sachan, and B. Schölkopf, "CLadder: assessing causal reasoning in language models," in *Advances in Neural Information Processing Systems*, 2023.

[8] Y. Chen, V. K. Singh, J. Ma, and R. Tang, "CounterBench: a benchmark for counterfactuals reasoning in large language models," arXiv preprint arXiv:2502.11008, 2025.