SELF-EVOLVING TASK DISCOVERY AND CONSISTENCY REPAIR IN DROID-GROUNDED GENERATIVE WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Controllable generative world models make it possible to iterate on manipulation behaviors without repeatedly running real robots. In practice, the bottleneck is often not only "how to generate" but how to keep an automated loop tethered to the same assumptions that make classical simulation pipelines reliable: realistic initial states, consistent task semantics, and careful data hygiene. We present **SelfEvoWM**, a generate–verify–repair loop built on Ctrl-World (Guo et al., 2025) that explicitly interfaces with a simulation chain. The loop (i) grounds goal proposals by retrieving DROID (Khazatsky et al., 2024) anchors as simulation-ready initial states, (ii) uses a VLM critic to audit physical consistency and localize failure modes, and (iii) automatically constructs targeted simulation environments to generate supplemental data that repairs weak regions of the world model. Rather than claiming a finished benchmark, this workshop paper focuses on the system design and early failure modes we observed when wiring a world model into an automated simulation pipeline, including retrieval collapse, contact-level artifacts, and sensitivity of VLM judgments to phrasing and viewpoints. We hope this provides a concrete starting point for integrating generative world models with end-to-end simulation stacks for robot learning.

1 INTRODUCTION

Robot learning has long relied on simulation pipelines for scale: define tasks, sample initial conditions, roll out trajectories, and turn them into training signal. Recent generative simulation systems push this idea further by synthesizing tasks and data at scale (Wang et al., 2023). In parallel, controllable world models offer a different abstraction: instead of rendering from a hand-coded engine, a learned model can generate rollouts conditioned on actions and goals (Guo et al., 2025).

When we try to connect these two threads into an end-to-end automated loop, two issues show up quickly. First, task proposal tends to become prompt-driven and brittle: small changes in wording can produce different "tasks," and the loop starts optimizing for what the prompt suggests rather than what the environment supports. Second, self-generated rollouts can drift in subtle ways. They may look plausible frame-by-frame while violating basic physical constraints across time, which is especially damaging when the loop treats its own output as supervision.

This workshop paper asks a pragmatic question: can a controllable world model be embedded into a simulation chain that looks and behaves like a classical pipeline—goal $\rightarrow$ initial state $\rightarrow$ rollouts $\rightarrow$ training signal—while keeping the loop stable without manual curation?

We propose **SelfEvoWM**, a loop built on Ctrl-World (Guo et al., 2025) and designed to plug into a simulation stack rather than replace it. The key is to treat grounding and hygiene as first-class: *grounding* comes from retrieval of real anchors that serve as simulation-ready initial states; *hygiene* comes from conservative consistency auditing; and, crucially, low-consistency cases are not only filtered out but also used to trigger a *repair step* that automatically builds targeted simulation environments and generates supplemental data for the world model's weak spots.

Contributions. This workshop paper contributes:

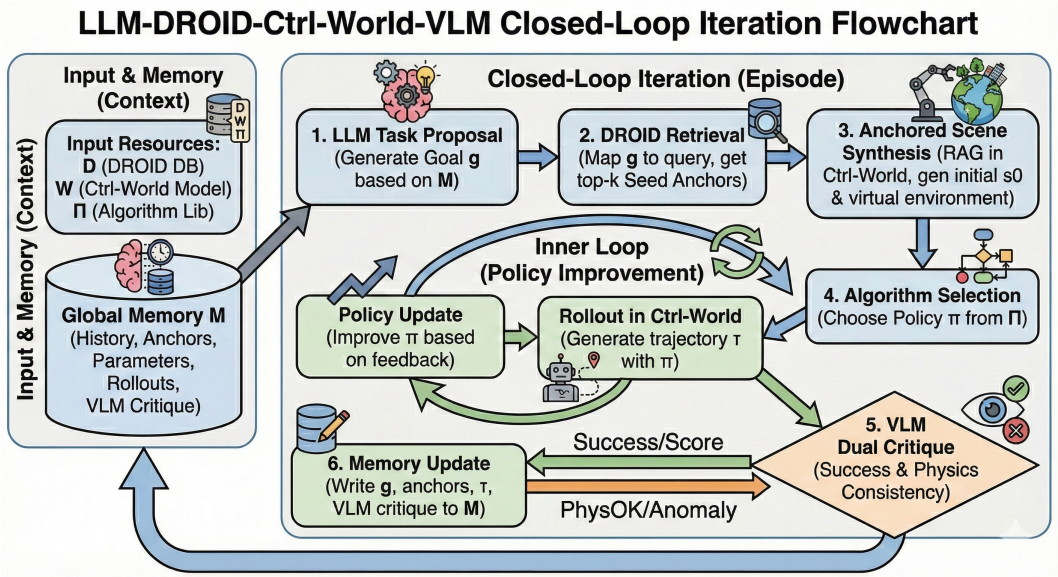

Figure 1: SelfEvoWM as a simulation chain. Goals are proposed, DROID (Khazatsky et al., 2024) anchors are retrieved as simulation-ready initial states, Ctrl-World (Guo et al., 2025) generates controllable rollouts, and a VLM critic audits success and physical consistency. Low-consistency cases trigger failure localization and targeted simulation environment construction to generate supplemental data for repairing weak regions of the world model.

- **A simulation-chain view of world-model training loops.** We frame self-evolving world-model learning as an end-to-end pipeline with explicit interfaces for goal proposal, initialization, rollout, verification, and logging.
- **DROID-grounded initialization for controllable generation.** We retrieve anchors as simulation-ready initial states to reduce prompt-only drift and keep rollouts closer to real data distributions.
- **Consistency-driven repair via targeted simulation.** We use a VLM critic to localize where the world model breaks (e.g., contact and object persistence), then automatically construct focused simulation environments to generate supplemental data that repairs those failure modes.
- **Early system-building observations.** We summarize failure modes we encountered (retrieval collapse, contact artifacts, and VLM sensitivity), which are easy to miss when describing such loops abstractly.

2 METHOD

2.1 PROBLEM FORMULATION

SelfEvoWM turns a controllable world model into a self-evolving system that can be slotted into a simulation chain. We assume access to: (i) a large-scale real-world dataset $\mathcal{D}$ (DROID (Khazatsky et al., 2024)), (ii) a controllable world model $\mathcal{W}$ (Ctrl-World (Guo et al., 2025)), (iii) an LLM controller $\mathcal{L}$ for proposing goals and meta-decisions, and (iv) a VLM critic $\mathcal{V}$ for verification and failure localization.

At episode e, the pipeline follows the usual simulation pattern: *goal* $\rightarrow$ *initialization* $\rightarrow$ *rollout* $\rightarrow$ *buffer*. The key difference is that verification is not the end: low-consistency rollouts also drive a repair step, where the system constructs targeted simulation environments and generates supplemental data to improve the world model in the regions where it breaks.

Algorithm 1 SelfEvoWM generate–verify–repair loop.

Require: Dataset $\mathcal{D}$, world model $\mathcal{W}$, memory $\mathcal{M}$, controller $\mathcal{L}$, critic $\mathcal{V}$
Hyperparameters: anchors k, inner-loop steps T, thresholds τ_s (success), τ_c (consistency)

for episode $e = 1, 2, \ldots$ **do**
  Propose goal $g \leftarrow \mathcal{L}(\mathcal{M})$
  Retrieve anchors $\{a_i\}_{i=1}^{k} \leftarrow \mathrm{Retrieve}_k(g, \mathcal{D})$
  Condition the world model to construct a controllable environment around $\{a_i\}_{i=1}^{k}$
  for $t = 1, \ldots, T$ **do**
    Roll out $\tau \leftarrow \mathcal{W}(g, \{a_i\}_{i=1}^{k})$
    Critic feedback $s, c \leftarrow \mathcal{V}(\tau, g)$ ▷ s: success, c: consistency
    if $c < \tau_c$ **then**
      Localize failure $\phi \leftarrow \mathrm{Localize}(\mathcal{V}, \tau, g)$
      Build repair env $\mathcal{E}_\phi \leftarrow \mathrm{ConstructSim}(\{a_i\}_{i=1}^{k}, \phi)$
      Generate supplemental data $\mathcal{B}_\phi \leftarrow \mathrm{Simulate}(\mathcal{E}_\phi)$
      Refresh world model $\mathcal{W} \leftarrow \mathrm{Update}(\mathcal{W}, \mathcal{B}_\phi)$
      discard τ and continue
    end if
    if $s \geq \tau_s$ **then**
      mark success and break
    end if
    Store rollout $\mathrm{Add}(\mathcal{M}, g, \{a_i\}_{i=1}^{k}, \tau, s, c)$
  end for
  Update memory summary $\mathcal{M} \leftarrow \mathrm{Summarize}(\mathcal{M})$
end for

2.2 System overview

Figure 1 gives the pipeline view, and Algorithm 1 summarizes the loop. The design follows two constraints familiar from classical simulation pipelines: **(i) initialization matters**—the loop is only as good as the distribution of starting states; and **(ii) data hygiene matters**—once inconsistent rollouts enter the buffer, the loop amplifies them. SelfEvoWM adds a third constraint that becomes visible in practice: **(iii) filtering is not enough**—the loop benefits from a repair path that actively strengthens the generator where it fails.

Episode workflow. Each episode consists of:

1. **Goal proposal** ($\mathcal{L}$): propose a goal g from memory $\mathcal{M}$.
2. **Initialization** (retrieval): retrieve anchors $\{a_i\}_{i=1}^{k}$ from $\mathcal{D}$ as candidate initial states.
3. **Rollout** ($\mathcal{W}$): condition the world model on $(g, \{a_i\}_{i=1}^{k})$ to generate a controllable scenario neighborhood.
4. **Verification** ($\mathcal{V}$): audit success and physical consistency.
5. **Repair** (simulation chain): when consistency is low, localize the failure, construct a targeted simulation environment, generate supplemental data, and refresh the world model.

2.3 Semantic-Anchored Scene Synthesis via DROID (Khazatsky et al., 2024) Retrieval

Why anchoring helps. RoboGen-style generative simulation emphasizes automatic synthesis of tasks and data (Wang et al., 2023), but the loop still needs a stable way to sample starting states. In a learned world model, this becomes even more important: once initialization drifts, the model can remain visually coherent while gradually leaving the support of the training distribution.

Retrieval-anchored initialization. Given a goal g, we retrieve top-k anchors $\{a_i\}_{i=1}^{k}$ from DROID (Khazatsky et al., 2024) using vision-language similarity. Each anchor a_i acts as a simulation-ready initial state: it contains a short observation window (frames or keyframes), the corresponding robot state (e.g., end-effector pose and joint configuration), and lightweight metadata

when available. We use anchors as constraints for controllable generation: conditioned on (g, a_i), the world model generates rollouts in a local neighborhood of the anchor. The neighborhood is implemented as bounded perturbations (e.g., small pose changes, viewpoint shifts, stochastic dynamics), which increases diversity without turning the loop into prompt-only free generation.

Keeping the retrieval distribution healthy. In preliminary runs, we often saw retrieval collapse: semantically similar goals map to near-duplicate anchors, which inflates short-horizon success but reduces diversity later. A simple mitigation is to remove near-duplicates and enforce marginal coverage (e.g., clustering in embedding space and sampling across clusters). We treat this as part of the simulation chain, not an optional add-on.

Optional asset enrichment. When anchors are missing task-relevant objects, we augment the scenario with generated assets using EmbodiedGen (Wang et al., 2025), then let the world model explore variations around the anchored state.

2.4 Simulation-Chain Driven Physical Consistency Refinement

Motivation. In self-evolving loops, the main long-run risk is not a single failed rollout but silent buffer corruption: subtle physical defects accumulate, get reinforced by self-training, and gradually dominate the generated distribution. Contacts and object persistence are common pain points in early implementations.

VLM-based failure localization. We treat the VLM critic as more than a pass/fail gate. When a rollout fails the consistency check, we ask the critic for structured cues about *where* the violation begins and *what* it involves (e.g., the objects implicated, the interaction type such as penetration or slippage, and the temporal window). To reduce spurious localization, we query multiple frames (and viewpoints when available) and use agreement as a stability filter before acting on the signal.

Automatic repair environment construction. Given localized cues ϕ, the system builds a focused simulation environment $\mathcal{E}_\phi$ that reproduces the failure mode with minimal degrees of freedom: a relevant initial state (often derived from the retrieved anchor), the subset of objects involved, and a small set of perturbations (poses, friction/contact parameters, viewpoints, motion primitives) that reliably elicit the inconsistency. This mirrors standard debugging practice in simulation: isolate the failure into a reproducible "unit test" rather than sampling more generic rollouts.

Targeted supplemental data generation. The constructed environments are used to generate supplemental trajectories that emphasize the failure mode, not just average-case behavior. We place these samples into a dedicated buffer and periodically refresh the world model with them (or use them to recalibrate the generator), aiming to improve consistency specifically in the region where the model breaks. This detect–construct–refresh pathway is a current focus of our work: gating keeps the buffer clean, while targeted repair makes the generator better.

2.5 Autonomous Loop Closure with VLM-Based Consistency Gating

Motivation. Automated loops fail quietly when they start training on their own mistakes. For learned rollouts, a small rate of inconsistent trajectories can be enough to poison the buffer over time.

VLM as a conservative verifier. We use a VLM critic to provide two signals: a goal-conditioned success score s and a physical-consistency score c. The critic is treated as noisy, so we make it conservative: we query multiple time slices (and viewpoints when available) and aggregate the outputs with a simple rule such as majority vote with a minimum threshold. Only rollouts that pass the gate are stored and used for downstream learning or analysis.

When to give up. If a goal repeatedly fails consistency checks even with anchored initialization and targeted repair, we down-weight or discard it as out-of-support for the current world model. This mirrors a standard simulation practice: tasks that cannot be executed reliably under the current simulator are not used as training targets.

3 PRELIMINARY FINDINGS AND LIMITATIONS

SelfEvoWM is meant to behave like a simulation chain: it proposes goals, samples initial states, generates rollouts, and decides what to keep. In preliminary runs over a small set of automatically proposed goals, we found that performance is often determined less by the headline loop and more by a few recurring failure modes.

Retrieval can overfit the loop. Anchoring makes rollouts more plausible, but retrieval quality dominates exploration. Near-duplicate anchors boost early success but narrow the state distribution, making later "new" goals look solvable when they are actually repeats. Diversity-aware sampling in embedding space helped, but it introduces its own tuning knobs.

Contacts are where realism breaks first. Even when motion is smooth, contact-rich manipulation exposes inconsistencies: slight penetration, floating, or slippage accumulates across frames. These artifacts are easy to miss visually and easy to exploit accidentally during learning. Anchoring reduces catastrophic drift but does not fix contact fidelity; consistency auditing and targeted repair are both useful.

VLM judgments are sensitive. The critic is useful as a filter and a localizer, but it is not stable under phrasing or viewpoint changes. Underspecified goals lead to high variance across paraphrases, and borderline cases can flip under small camera changes. This is partly why we use the critic conservatively and rely on agreement across queries before triggering repair.

Repair loops introduce their own knobs. A detect–construct–refresh pipeline is powerful, but it adds practical questions: how often to trigger repair, how large the repair buffer should be, and how to avoid overfitting the world model to a narrow set of "unit tests". We also observed cases where localization was correct but incomplete (e.g., contact issues that depend on a longer temporal context), which led to repair environments that were reproducible yet not representative. These are the kinds of engineering details that matter for stability, and they are a main focus of ongoing work.

What we would measure next. A fuller study should quantify (i) anchoring quality versus diversity under a fixed retrieval budget, (ii) repair effectiveness (consistency improvement on localized modes versus side effects elsewhere), and (iii) the precision–recall trade-off of gating and localization, including disagreement across views or paraphrases. These measurements would clarify when simulation-chain integration helps and where the loop still breaks.

4 CONCLUSION

We presented **SelfEvoWM**, a generate–verify–repair loop that integrates a controllable world model into a simulation chain: goals are proposed, DROID (Khazatsky et al., 2024) anchors provide simulation-ready initialization, a VLM critic audits and localizes physical inconsistencies, and low-consistency cases trigger targeted simulation environment construction to generate supplemental data for repairing weak regions of the world model. As a workshop contribution, we emphasize the system design and the early failure modes that arise when such loops are implemented end-to-end, and we hope this helps future work build more reliable automated pipelines for robot learning.

REFERENCES

Yanjiang Guo, Lucy Xiaoyang Shi, Jianyu Chen, and Chelsea Finn. Ctrl-world: A controllable generative world model for robot manipulation. *arXiv preprint arXiv:2510.10125*, 2025.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe,

Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, Vitor Guizilini, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Muhammad Zubair Irshad, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. Droid: A large-scale in-the-wild robot manipulation dataset. 2024.

Xinjie Wang, Liu Liu, Yu Cao, Ruiqi Wu, Wenkang Qin, Dehui Wang, Wei Sui, and Zhizhong Su. Embodiedgen: Towards a generative 3d world engine for embodied intelligence, 2025.

Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023.