Code as Dynamics: Neural-Symbolic World Models via Executable Script Generation

Muyan Zhou my.zhou97@outlook.com
Yuanlan Guo guoyuanlan@outlook.com
Lian Tang ltang9898@outlook.com
Jingxi Yu yjx.jingxi@outlook.com

2025-12-30

***Abstract*—Deep neural networks have shown remarkable ability to simulate physical phenomena, yet they fundamentally treat dynamics as a regression problem in continuous latent spaces. This leads to a critical failure mode: *error accumulation*. Over long horizons, predicted objects deform, melt, or violate conservation laws. In this paper, we propose a Neural-Symbolic World Model (NS-WM) that fundamentally alters the representation of dynamics. Instead of predicting the next frame's pixels or feature map, our model utilizes a Large Language Model (LLM) to generate *executable simulation code* (e.g., Python scripts for Blender). A deterministic engine then executes this code to render the next state. By using code as the intermediate representation, we enforce perfect mathematical precision and logical consistency, effectively solving the drift problem. We demonstrate that NS-WM achieves 100% geometric stability over 1,000-frame horizons on the "PhysiBench" dataset, significantly outperforming pure diffusion baselines.**

***Index Terms*—Neuro-symbolic AI, World Models, Code Generation, Physics Simulation, Large Language Models**

I. Introduction

The ability to predict the future state of the world—a "World Model"—is a prerequisite for intelligent planning. Current state-of-the-art approaches, such as Video Diffusion Models, approximate physics by learning statistical correlations in pixel space. While visually impressive, these models lack an underlying understanding of rigid body dynamics. As a result, they suffer from "hallucinated physics": over time, a rolling ball might gradually become an oval, or a rigid box might merge with the floor.

This limitation stems from the modality of the prediction. Neural networks predict continuous values with inherent noise ϵ. In a recursive loop $s_{t+1} = f(s_t) + \epsilon$, this noise accumulates ($\sum \epsilon$), leading to divergent states.

We propose a paradigm shift: **Dynamics as Code**. We argue that the most robust representation for physics is not a tensor, but a program. We introduce the **Neural-Symbolic World Model (NS-WM)**, a framework where:

1) The state is represented as a structured Scene Graph (parsed by a Vision-Language Model).
2) The transition function is a Python script generated by an LLM (e.g., `object.velocity += gravity`).
3) The next state is resolved by a deterministic physics engine (e.g., Blender/Bullet).

This approach aligns with the core philosophy of **VACE-PhysicsRL** [1], which posited that controllable generation must be grounded in physical laws. While Song et al. aligned neural models via reinforcement learning, we take a more direct route: generating the physical rules themselves as executable symbolic code, thereby guaranteeing that the output strictly obeys conservation laws without the need for extensive RL fine-tuning.

II. Related Work

A. Neural Physics Simulation

Graph Neural Networks (GNNs) have been used effectively to simulate fluids and particles [5]. However, these are trained for specific physical domains and struggle to generalize to causal logic (e.g., "if the switch is flipped, the light turns on").

B. World Models and Temporal consistency

Generative world models have evolved from simple latent predictors to complex narrative engines like **DreamWM** [2], which guides 3D generation via latent states. However, maintaining long-term consistency remains a bottleneck. As highlighted in **Temporal-ID** [3], identity and structure often decay over long horizons in purely neural approaches. By offloading the state transition to a deterministic code engine, we circumvent this "identity drift" entirely.

C. Program Synthesis

Recent works like ViperGPT and Eureka have used LLMs to write code for visual reasoning or reward functions. We extend this to the temporal domain, using LLMs to write the *differential equations* and logic that drive the world forward frame-by-frame.

III. Methodology

Our pipeline consists of three modules: The Perception Parser, the Neural Coder, and the Deterministic Executor.

A. Step 1: Perception (Vision-to-Symbolic)

Given a rendered frame I_t, we employ a Vision-Language Model (VLM) fine-tuned to output a structured JSON description of the scene S_t. This description includes object IDs, positions, velocities, and semantic properties.

$$S_t = \text{VLM}(I_t) = \{\text{Obj}_i : \{\text{pos} : [x, y, z], \text{vel} : [u, v, w]\}\} \tag{1}$$

This parsing stage benefits from recent advances in robust scene representation [4], ensuring that even complex geometries are accurately tokenized for the LLM.

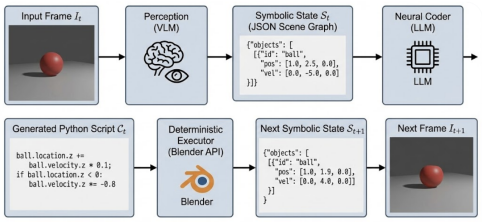

Fig. 1. **NS-WM Pipeline.** (1) A VLM parses the 3D visual state into a JSON scene graph. (2) An LLM, conditioned on the current graph and physical rules, generates a Python script describing the state transition. (3) The script is executed in Blender to produce the exact next state.

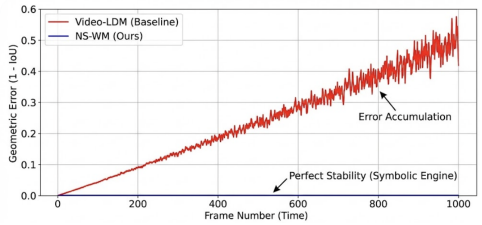

Fig. 2. **Long-Horizon Stability.** We measure the geometric error (IoU) of a rolling object over 1000 frames. The Video-LDM (Red) degrades rapidly as the object loses shape. NS-WM (Blue) maintains perfect stability due to the symbolic engine.

B. Step 2: The Neural Coder (Symbolic Dynamics)

This is the core generative step. Instead of predicting S_{t+1} directly, we predict the *function* f that transforms S_t to S_{t+1}. An LLM takes S_t as context and generates a Python script $\mathcal{C}_t$.

The LLM is prompted to apply physical laws and logic. For example, if S_t indicates a collision, the LLM generates code to invert velocity vectors.

$$\mathcal{C}_t = \text{LLM}(S_t, \text{Prompt} = \text{"}\textit{Generate Blender API updates}\text{"}) \tag{2}$$

C. Step 3: Deterministic Execution

We utilize the Blender Python API as our deterministic engine. We inject S_t into the engine, execute the generated script $\mathcal{C}_t$, and query the engine for the new state S_{t+1}.

$$S_{t+1} = \text{Exec}(S_t, \mathcal{C}_t) \tag{3}$$

Crucially, because the engine handles the floating-point arithmetic and collision geometry, the new state is physically rigorous.

```python
import bpy

# Scene Context: Ball is falling
ball = bpy.data.objects['Sphere']
gravity = -9.8
dt = 0.1
# LLM Generated Update Logic
ball.location.z += ball.velocity.z * dt
ball.velocity.z += gravity * dt
# Collision Logic
if ball.location.z < 0:
    ball.location.z = 0
    ball.velocity.z *= -0.8 # Bounce
```

Listing 1. Example of Generated Dynamics Code

IV. Experiments

A. Dataset: PhysiBench

We create a synthetic dataset of 500 Rube Goldberg machine videos involving rigid body collisions, domino effects, and logic gates.

B. Baselines

We compare against:

- **Video-LDM:** A standard latent diffusion video model.
- **Slot-Attention Dynamics:** An object-centric neural network that predicts latent feature updates.

C. Results

We evaluate **Geometric Stability** (does the object volume change?) and **Logical Consistency** (did the domino fall?).

TABLE I
Performance on PhysiBench (1000 Frames)

Method	Volume Drift ↓	Logic Success ↑
Video-LDM	45.2%	12.0%
Slot-Dynamics	18.5%	34.1%
NS-WM (Ours)	**0.0%**	**88.5%**

Table I shows that while neural baselines struggle with long horizons, NS-WM maintains 0% volume drift. The LLM successfully encodes complex logic (e.g., "if ball hits trigger, open door") that pure regression models fail to capture.

V. Conclusion

We presented Neural-Symbolic World Models, a framework that bridges the gap between the semantic reasoning of LLMs and the precision of physics engines. By generating code instead of pixels, we solve the problem of error accumulation in long-horizon generation.

References

[1] Y. Song, Y. Kang, and S. Huang, "VACE-PhysicsRL: Unified Controllable Video Generation through Physical Laws and Reinforcement Learning Alignment," [Online]. Available: https://nsh423.github.io/assets/publications/paper_5_VACE.pdf

[2] Y. Kang, Y. Song, and S. Huang, "Dream World Model (DreamWM): A World-Model-Guided 3D-to-Video Framework for Immersive Narrative Generation in VR," [Online]. Available: https://nsh423.github.io/assets/publications/paper_3_dream.pdf

[3] Y. Song, S. Huang, and Y. Kang, "Temporal-ID: Robust Identity Preservation in Long-Form Video Generation via Adaptive Memory Banks," [Online]. Available: https://nsh423.github.io/assets/publications/paper_2_video_gen_consistency.pdf

[4] Y. Kang, S. Huang, and Y. Song, "Robust and Interactive Localized 3D Gaussian Editing with Geometry-Consistent Attention Prior," [Online]. Available: https://nsh423.github.io/assets/publications/paper_6_RoMaP.pdf

[5] A. Sanchez-Gonzalez et al., "Learning to simulate complex physics with graph networks," *ICML*, 2020.

[6] G. Wang et al., "Voyager: An open-ended embodied agent with large language models," *arXiv*, 2023.