CS590: Socially Cognizant Robotics
Deliverable 1

Group Members: Rohit Bellam (rsb204), Karim Smires (ks1686)

February 9th, 2026

1 Selected Paper

Title: Dream to Control: Learning Behaviors by Latent Imagination
Authors: Danijar Hafner, Timothy Lillicrap, Jimmy Ba, Mohammad Norouzi
Venue: ICLR 2020
Paper: `https://arxiv.org/pdf/1912.01603`
Code: `https://github.com/danijar/dreamer`
Our Fork: `https://github.com/ks1686/dreamer`

2 Summary

Dreamer is a model-based reinforcement learning agent that learns long-horizon behaviors from high-dimensional image observations by "dreaming" in a latent space. The problem is formulated as a POMDP, and the architecture relies on a Recurrent State Space Model (RSSM) to learn a world model. This dynamics model includes a representation model that encodes observations into compact latent states, a transition model that predicts future latent states without generating images, and a reward model. By predicting future states and rewards, the agent decouples representation learning from behavior learning, enabling it to learn policy and value functions purely from imagined trajectories.

Behavior learning in Dreamer uses an actor-critic approach where the action model outputs tanh-transformed Gaussian actions and the value model estimates expected rewards beyond the imagination horizon using $V\lambda$ returns. A key novelty is the use of stochastic backpropagation to propagate analytic value gradients back through the learned dynamics, avoiding high-variance derivative-free optimization. Evaluated on 20 tasks from the DeepMind Control Suite, Dreamer achieves state-of-the-art data efficiency, surpassing methods like PlaNet and D4PG while maintaining comparable asymptotic performance.

3 Implementation Details

We utilized the open-source TensorFlow 2 implementation provided by the authors. The setup process involved several steps to ensure compatibility with our modern Linux environment.

3.1 Environment Setup

To avoid dependency conflicts with system packages, we created a specialized Python 3.8 virtual environment. Key dependencies included: `tensorflow-gpu==2.2.0` (required for the legacy codebase), `dm_control` for the simulation environments, and helper libraries like `termcolor` and `ruamel.yaml` for logging and configuration.

3.2 Challenges and Solutions

The primary challenge encountered was a missing system dependency for `ffmpeg`, which the code uses to generate video summaries of the agent's performance. The original code failed with a "Broken pipe" error when attempting to write improved GIF summaries.

Solution: We downloaded a static build of `ffmpeg` and placed it in a local `.tools/` directory within the project. We then updated the system `PATH` environment variable to include this directory before running the training script.

4 Results

We executed the Dreamer agent on the `dmc_walker_walk` task for an initial 5,000 steps to verify the installation and data collection pipeline. This phase corresponds to the "prefill" stage where the agent acts randomly to populate the replay buffer before model training begins.

4.1 Quantitative Metrics

As expected for a random policy, the returns fluctuated between 30 and 50 (Figure 1). This confirms the environment is correctly providing rewards and the agent is interacting with it, establishing a baseline for future learning.

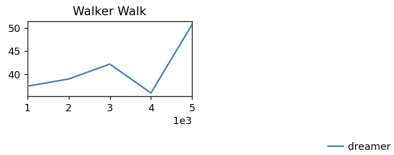

Figure 1: Returns over the first 5,000 steps (prefill phase) on `dmc_walker_walk`. The values reflect the random policy used for initial data collection.

4.2 Qualitative Analysis

The training pipeline successfully generated all necessary artifacts, including `metrics.jsonl` for performance tracking. Additionally, `events.out.tfevents` files were created for TensorBoard. Finally, the system rendered video summaries of the agent's interactions, confirming the `ffmpeg` integration. This successful execution of the prefill phase demonstrates that the complex software stack (TensorFlow 2.2, MuJoCo, DeepMind Control) is correctly configured.

5 Conclusion

We have successfully set up and verified the Dreamer codebase. The agent effectively interacts with the simulated environment, collects data, and logs performance metrics. While the 5,000-step run was insufficient for the agent to learn a high-performing policy (which typically requires millions of steps), it confirms the reproducibility of the implementation environment and readiness for full-scale experiments.