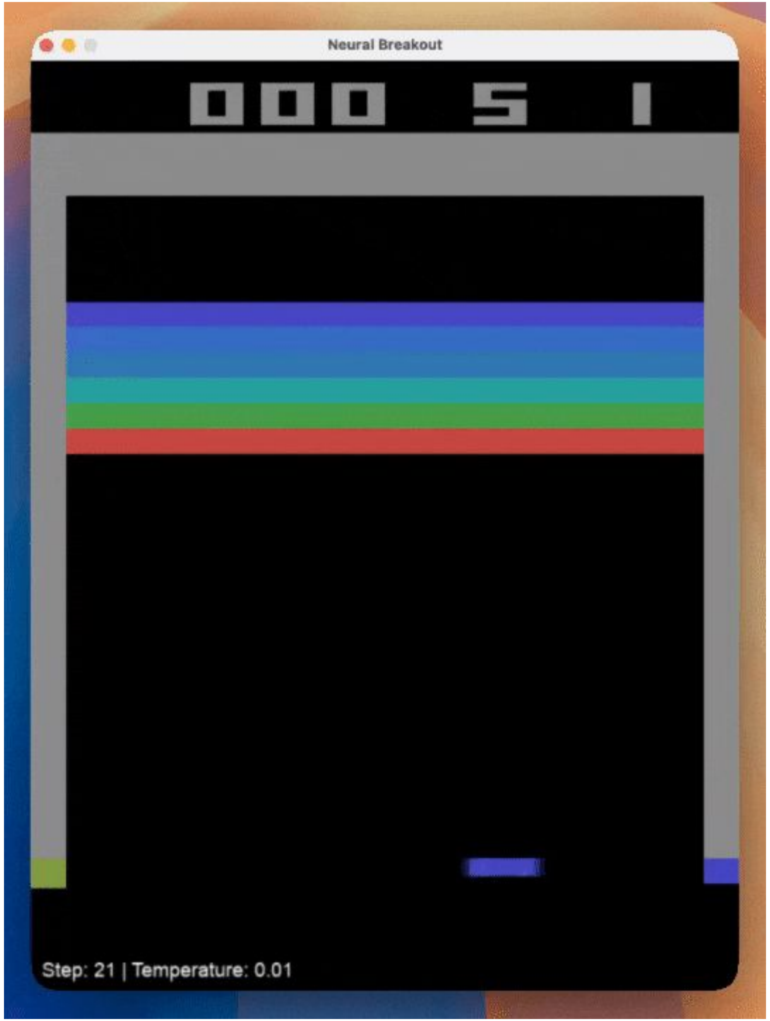
Neural Atari

How to build a playable fully-neural version of Atari Breakout

@paraschopra, founder of Lossfunk

Code: https://github.com/paraschopra/atari-pixels

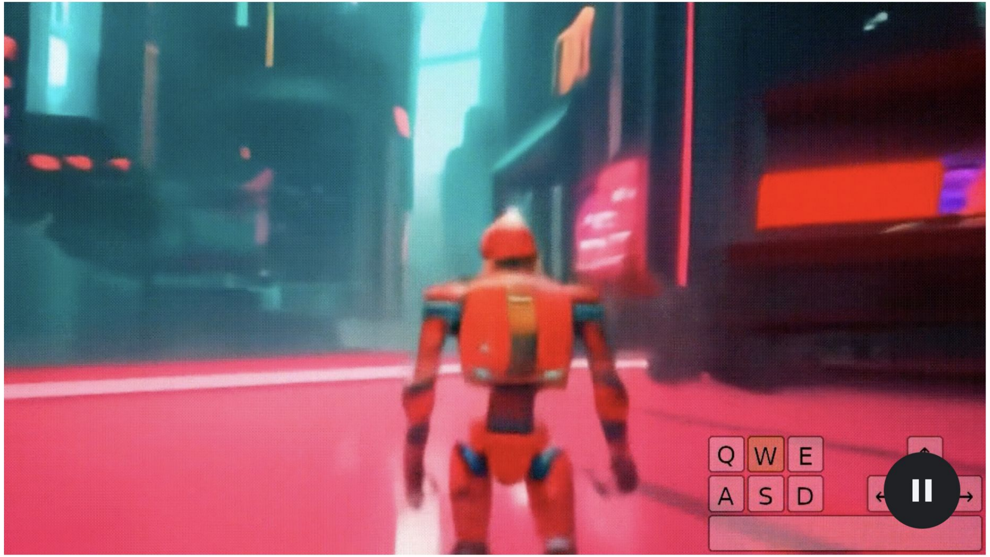
Demo



Inspiration

Generating unlimited diverse training environments for future general agents

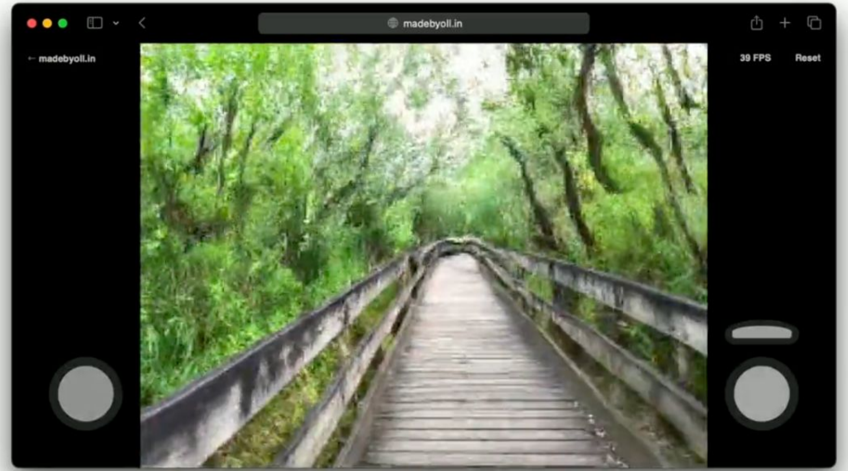
Today we introduce Genie 2, a foundation world model capable of generating an endless variety of action-controllable, playable 3D environments for training and evaluating embodied agents. Based on a single prompt image, it can be played by a human or AI agent using keyboard and mouse inputs.



World Emulation via Neural Network

POSTED 25 APRIL 2025

I turned a forest trail near my apartment into a playable neural world. You can explore that world in your web browser **by clicking right here**:



How cool is it to generate interactive experiences entirely from a neural networks

My plan

- Select a game: Atari Breakout
- Train an agent using Reinforcement Learning
  - An excuse to learn RL
- Generate videos of the agent playing the game
- Learn a world model
  - That takes in current frame + latent action to produce the next frame
- Map real actions (LEFT, RIGHT, etc.) to latent actions
- Deploy the world model as a playable game
  - Real actions to latent actions
  - Latent

Train an agent to play Atari Breakout

I used Q learning. Theory is pretty simple!

Exploration in any environment gives us these tuples
(state, action, next state, reward, done)

Your job is to learn a function that estimates cumulative future rewards for each possible action given a state.

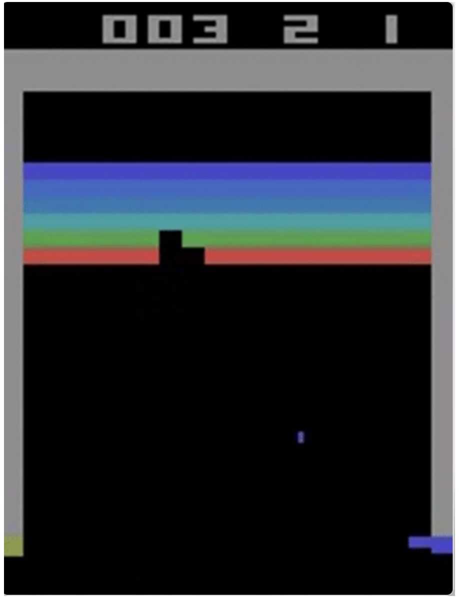
$$R(\boxed{\tau}) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots$$

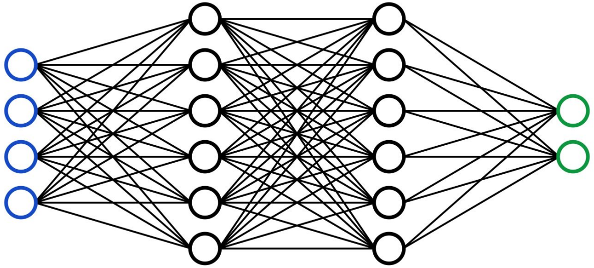
Return: cumulative reward

Gamma: discount rate

Trajectory (read Tau)
Sequence of states and actions

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$





CNN mostly

Q-values	
LEFT	40
RIGHT	10
NOOP	5
FIRE	20

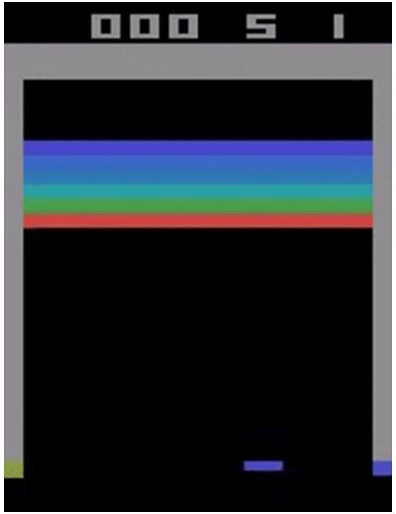
Future total reward
from the state

I used Double Q Learning

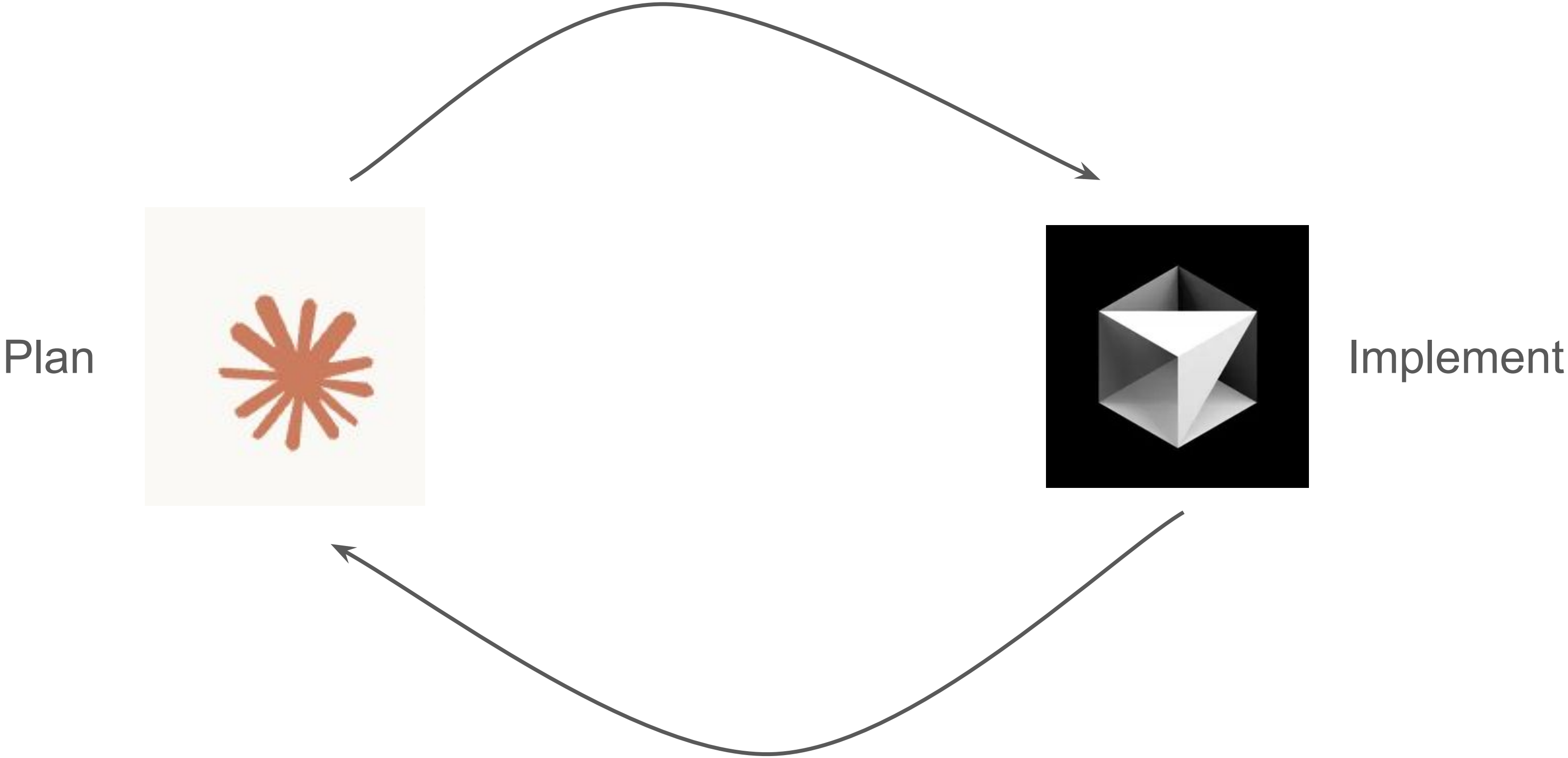
- You have two networks
  - Given (state, action, reward, next state)
  - **Policy network**: estimates Q value for a given state and the action taken
  - **Target network**: a lagging version of policy network that gives you target value to calculate loss against
    - Next action chosen = argmax(policy_network(next state))
    - Target Q value = Immediate reward + gamma * target_network(next action chosen)
- There is an exploration parameterized by epsilon
  - Epsilon probability -> random action (this decays over time)
  - Else -> take action with maximum Q value

This helped me train an agent that reached a score ~20

- It’s normal to get to scores 200 or more, but I just wanted to test water flowing through the pipes



My process: vibecode!



I vibecoded what was SOTA in 2013

arXiv
https://arxiv.org › cs

[1312.5602] Playing Atari with Deep Reinforcement Learning

by V Mnih · 2013 · Cited by 17542 — We present the first **deep learning model** to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning.

Caution: I wasted 10 days chasing a subtle bug

My agent was getting stuck in a local optima.

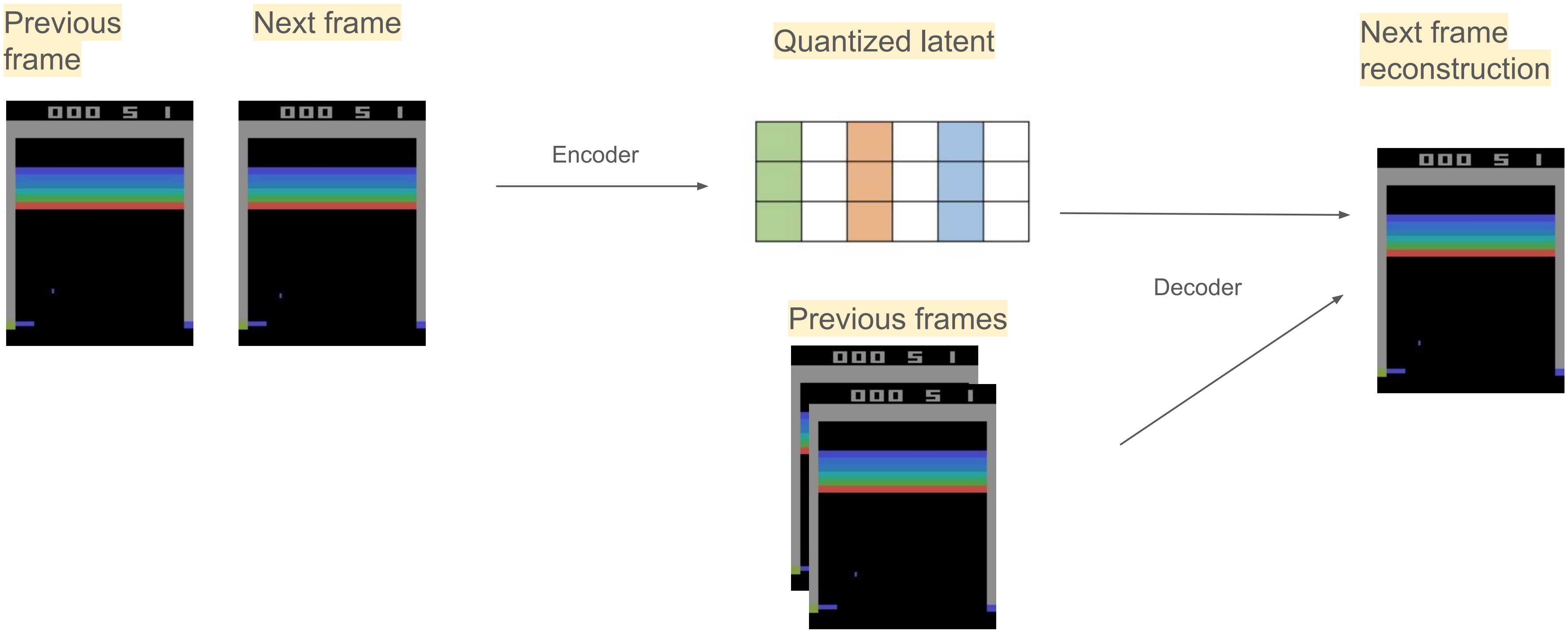
It went LEFT, scored a point, and then did nothing.

I went mad trying to debug, ended up learning a lot about RL.

Finally realized that LLM generated code was normalizing data twice (divide by 255), once while passing frames and once in forward pass.

But it was fixed and I had lots of lots of videos of Atari Breakout!

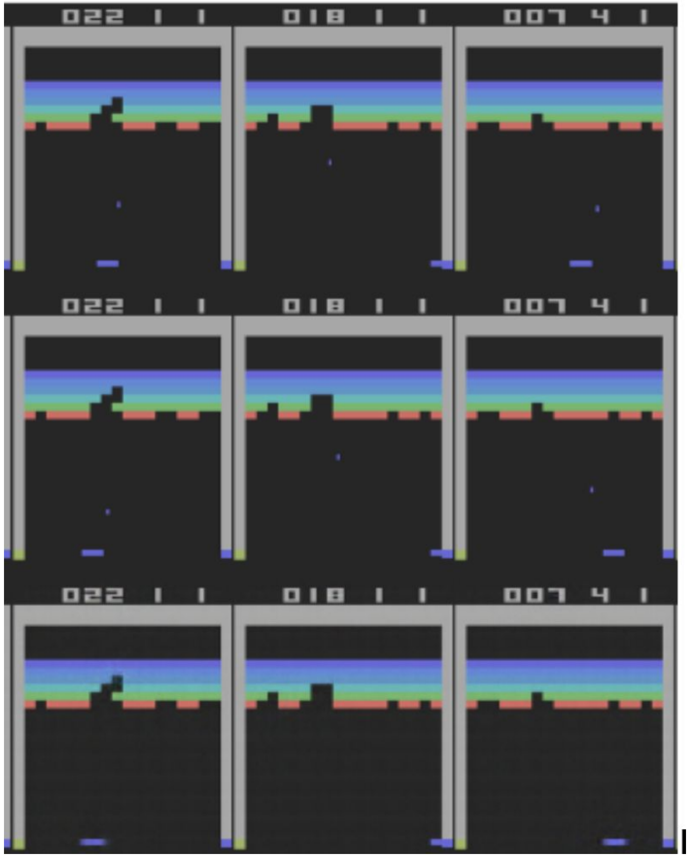
World model for dynamics of Atari Breakout



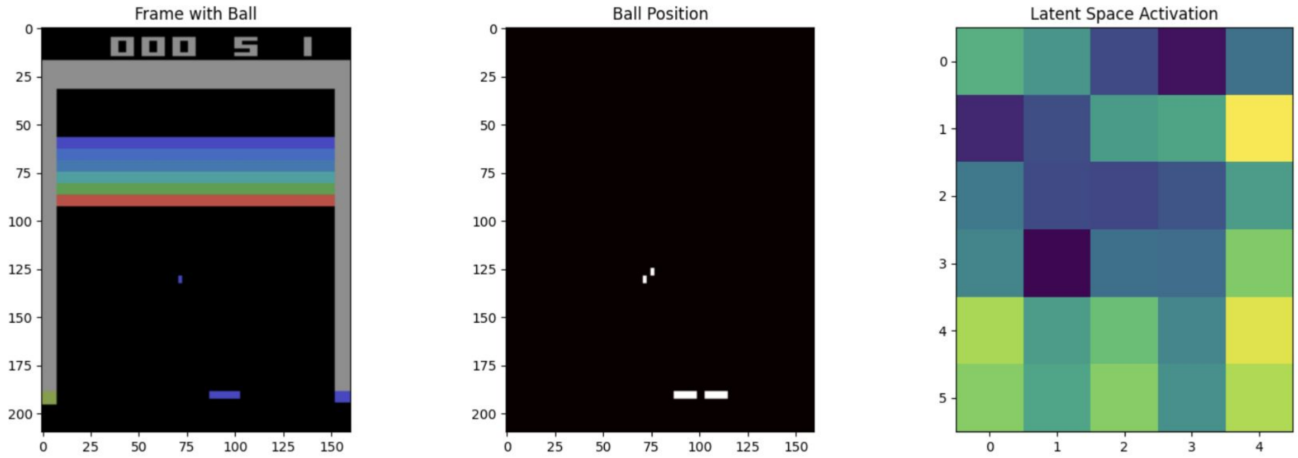
First attempt: ball disappeared

Frames were getting reconstructed, but the ball was missing.

- Top frame is initial frame
- Middle frame is next frame
- Bottom frame is reconstructed frame given initial frame + predicted latent action



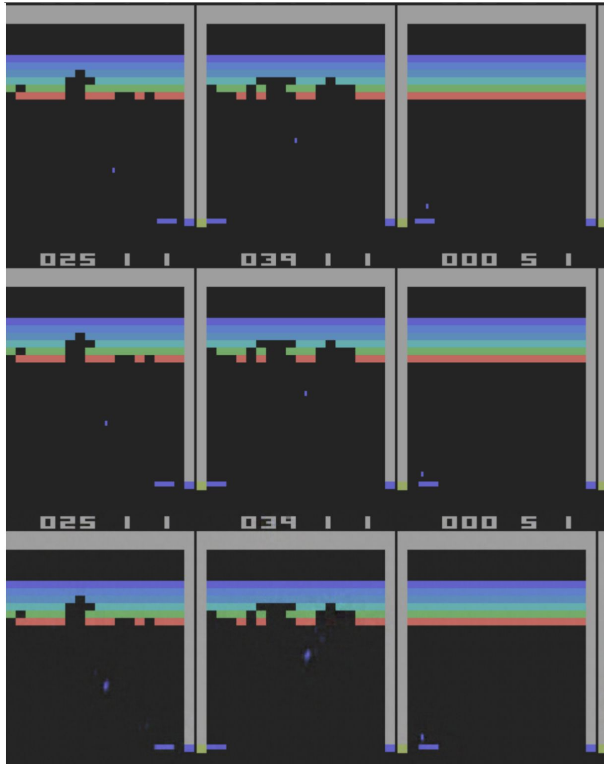
Latent space showed it’s capturing change! (notice blue)

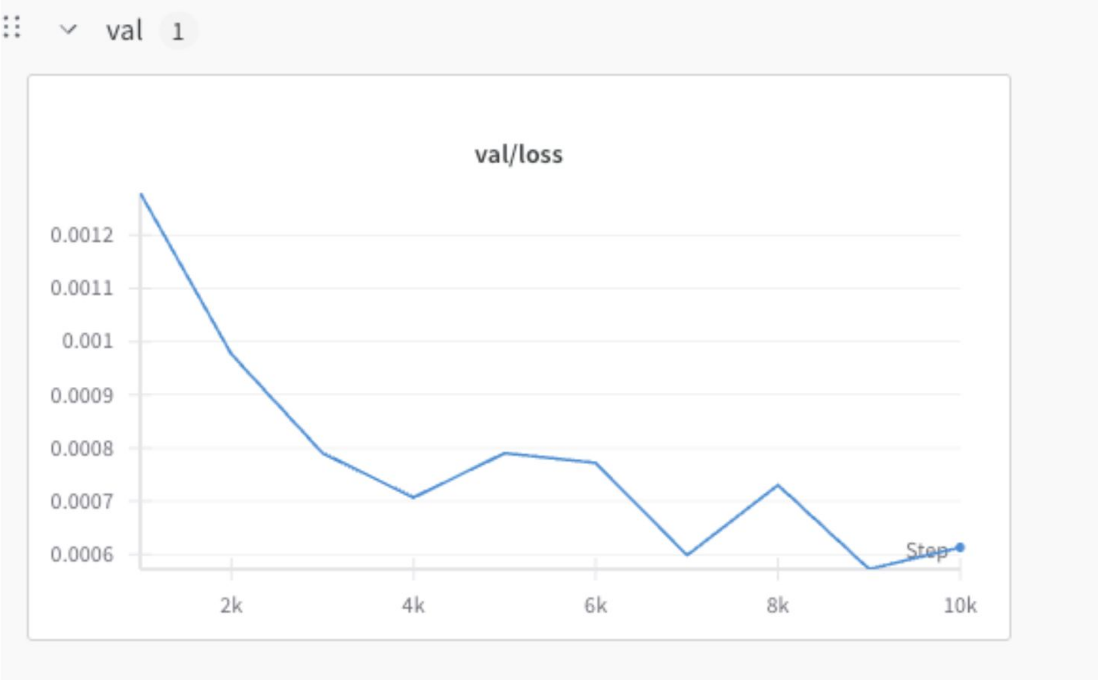


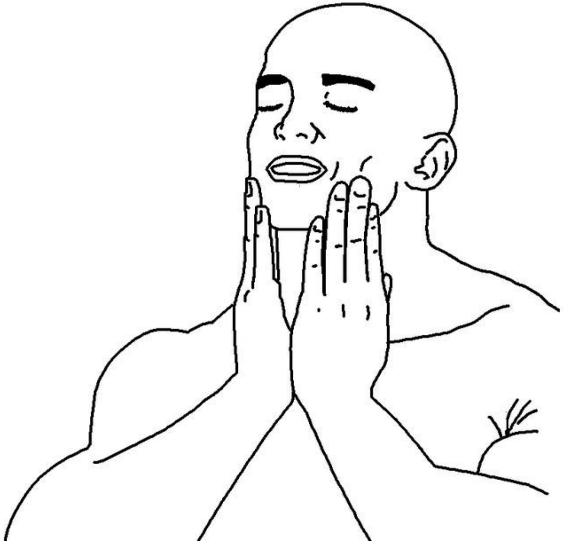
Claude me to told add motion loss and it worked!

TLDR: add diff of next (predicted) frame and actual previous frame as an additional loss term

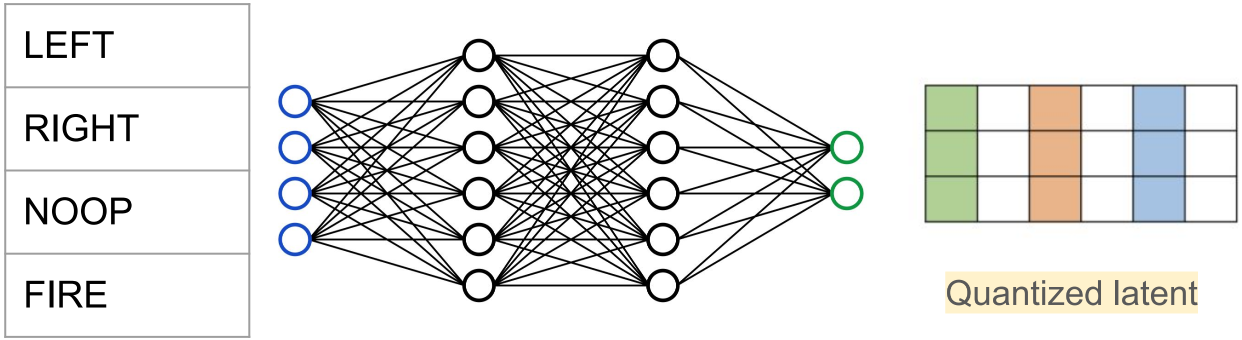
```python
# In the training loop, replace:
rec_loss = F.mse_loss(recon, frame_tp1)

# With:
frame_diff = torch.abs(frame_tp1 - frame_t)
motion_weight = 1.0 + 10.0 * (frame_diff.sum(dim=1, keepdim=True) > 0.05).float()
rec_loss = (motion_weight * (recon - frame_tp1)**2).mean()
```



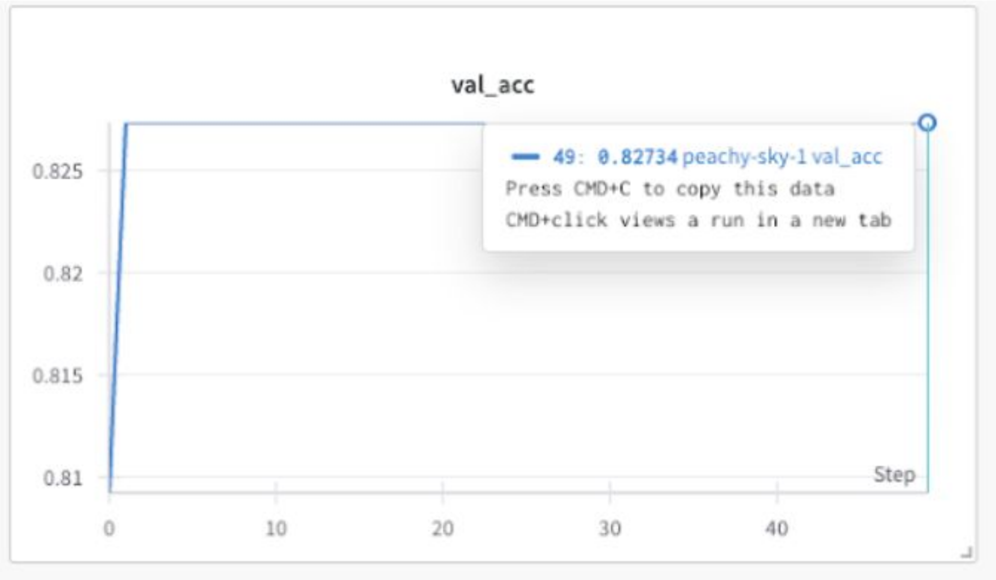

val 1
val/loss
0.0012
0.0011
0.001
0.0009
0.0008
0.0007
0.0006
2k
4k
6k
8k
10k
Step



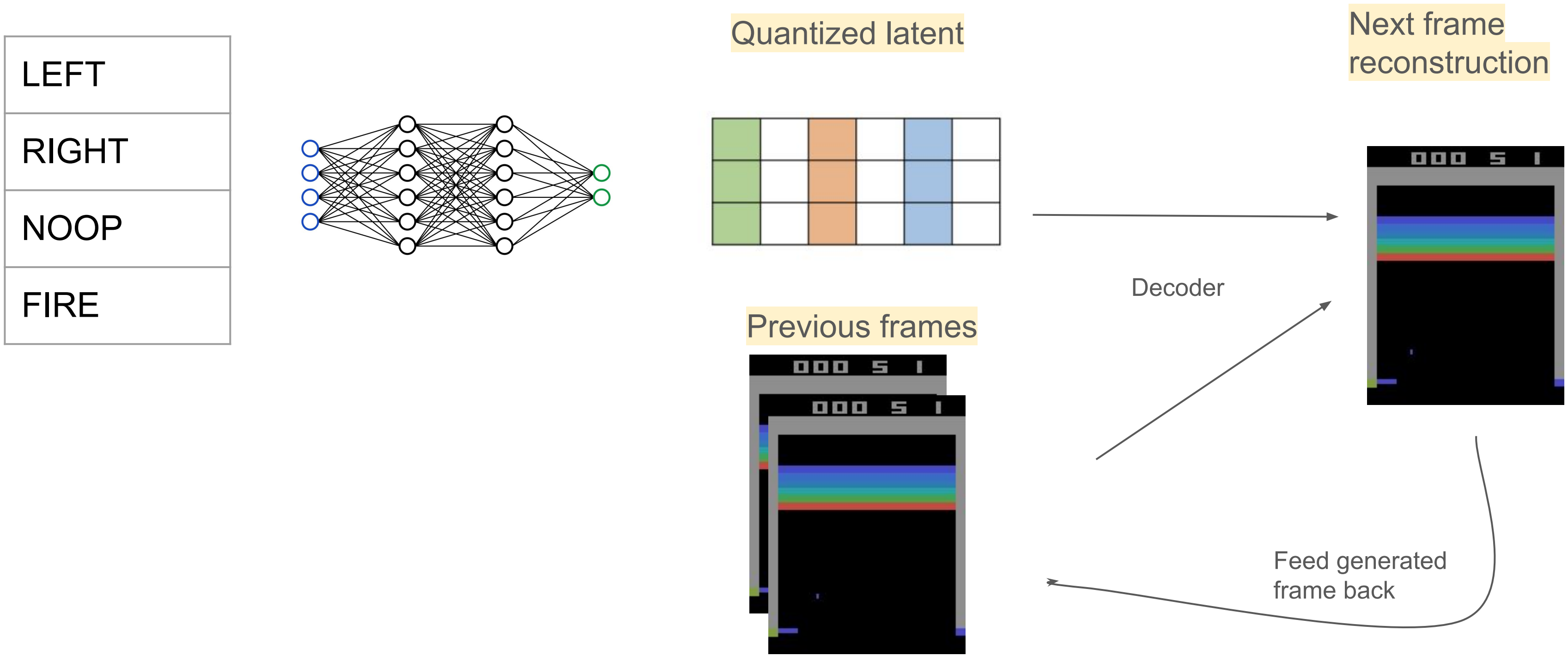
Action to latent model



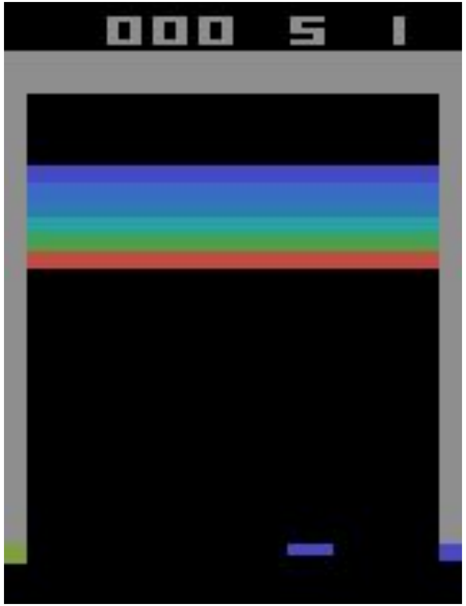
Got 83% accuracy for real to latent prediction model



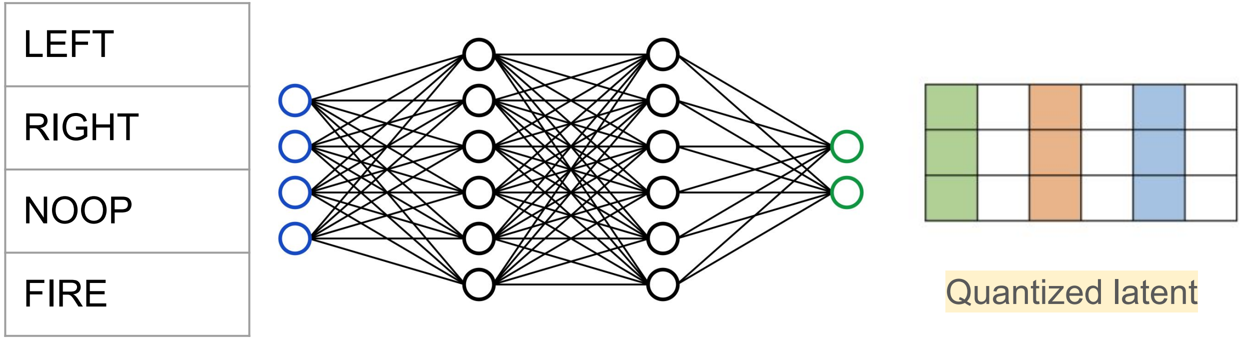
First attempt at neural game via learned world model



Disappearing act! The generated game descended into randomness



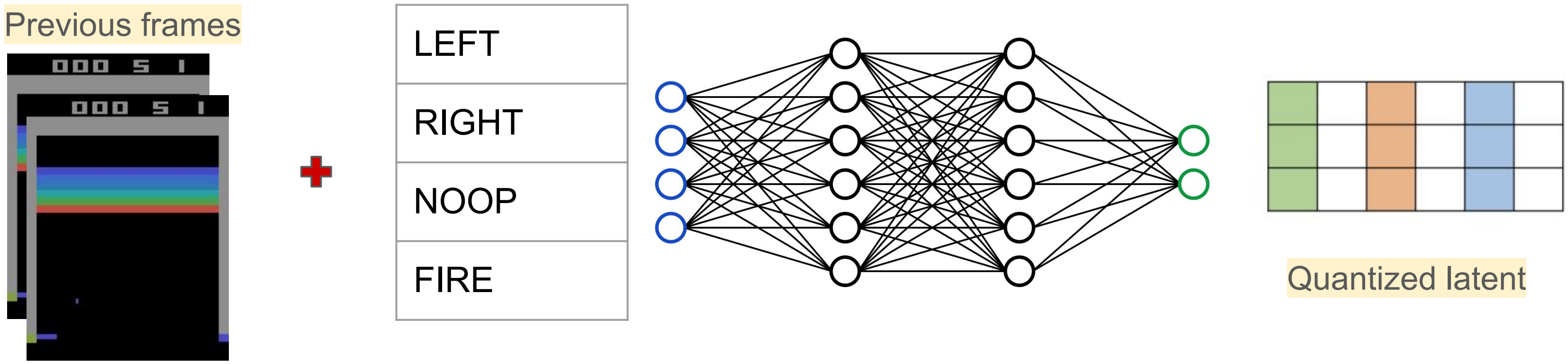
Notice anything odd here?



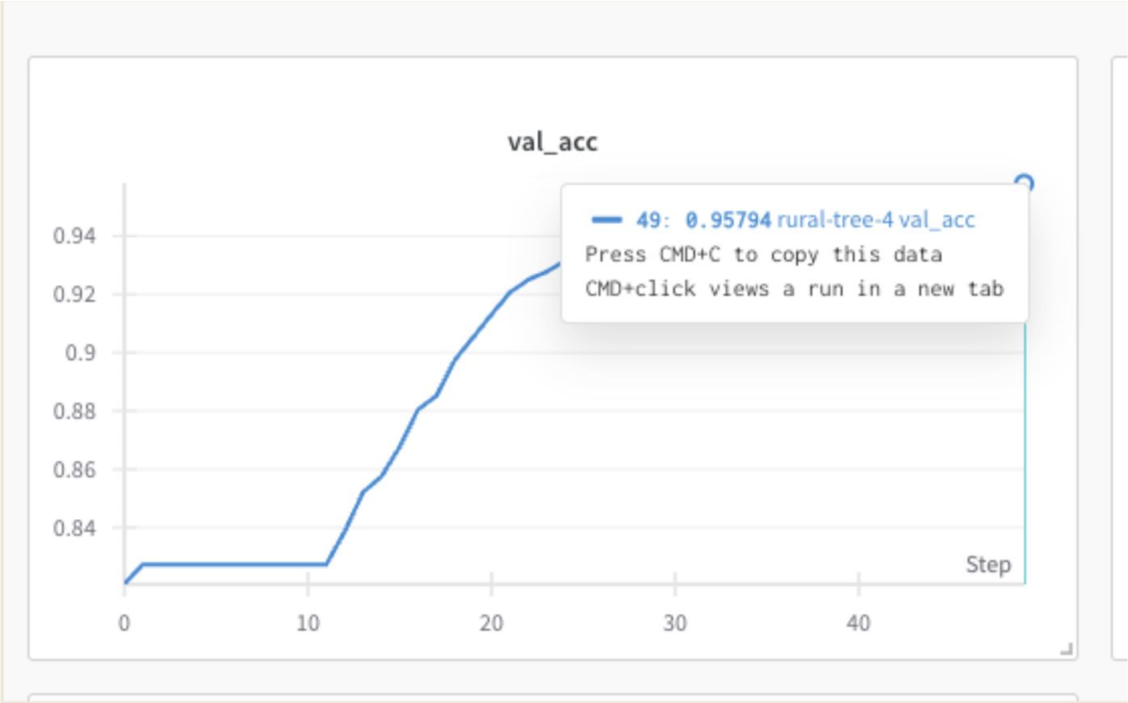
The same action could lead to different latents depending on the state

Mommy, it’s a one to many mapping!

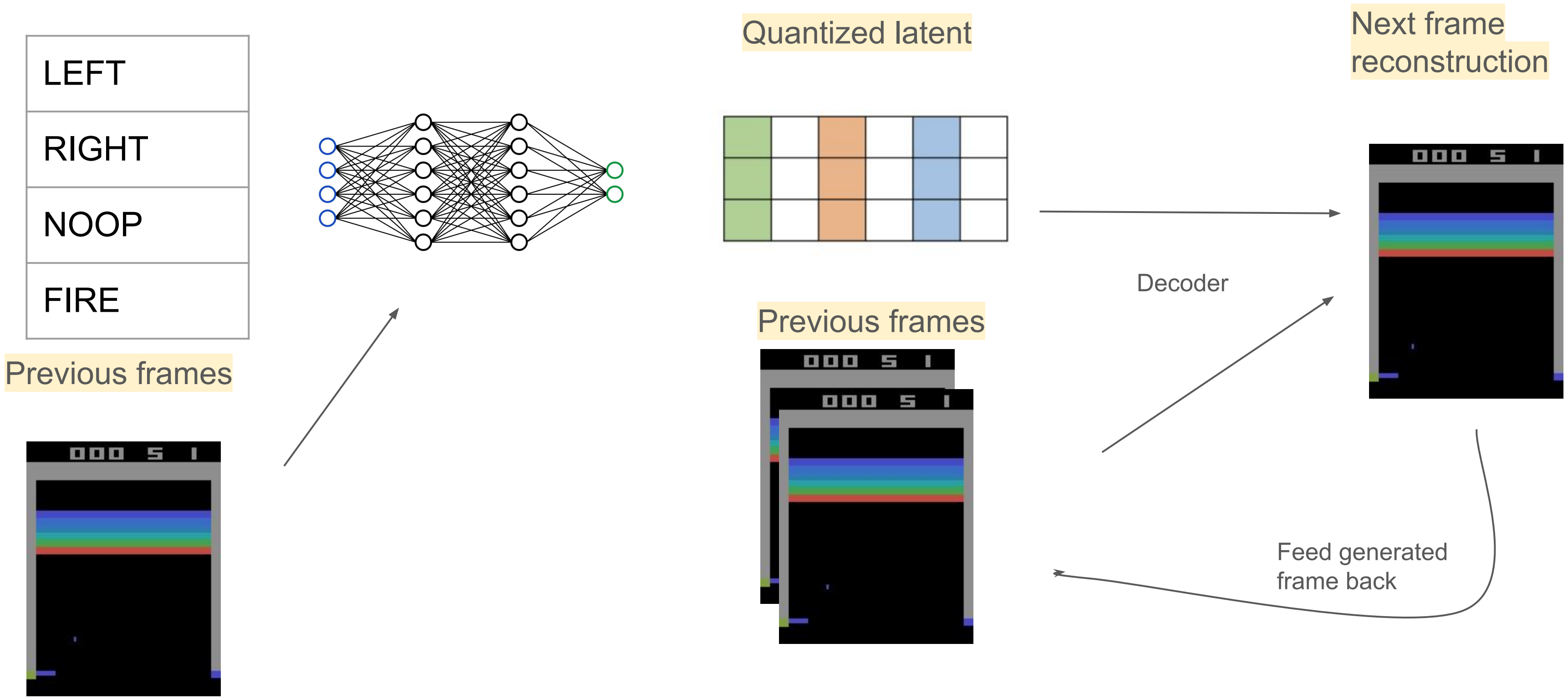
V2 of action to latent model



Accuracy improved to 95%!

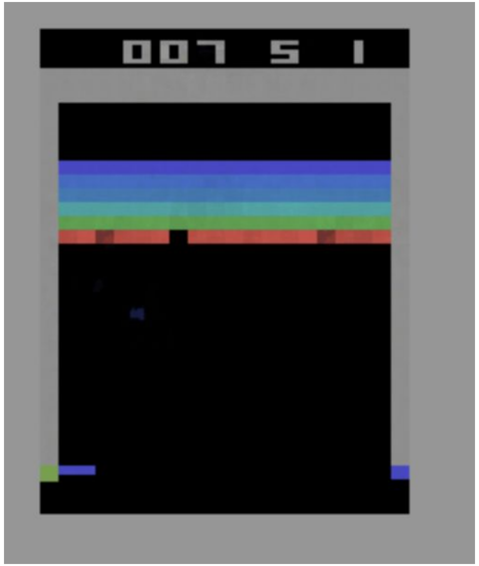


First attempt at neural game via learned world model



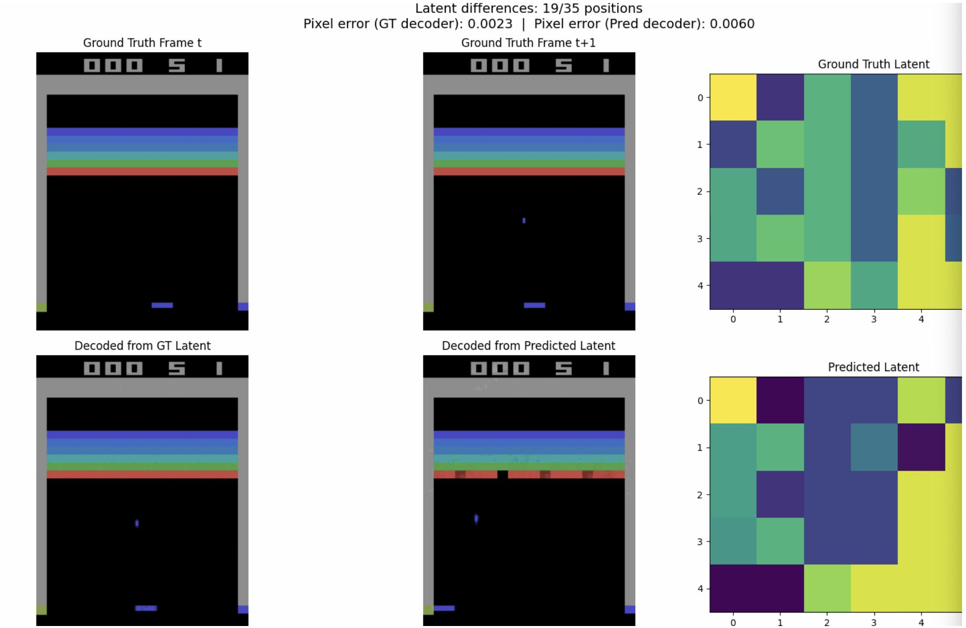
It should work now, right?

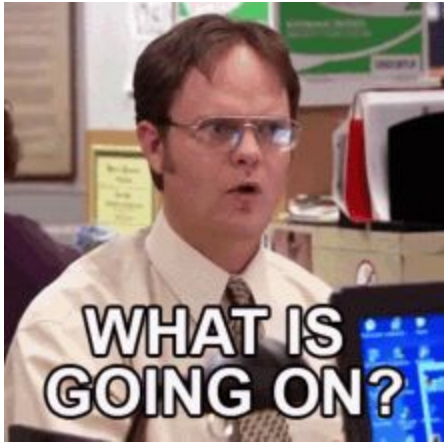
Nope :(



Paddle went to left, score increases from 0 to 7, and it stays there!

Debugging latents: error in full pipeline is 50% (19/35), while isolated action to latent error is 5%




WHAT IS
GOING ON?

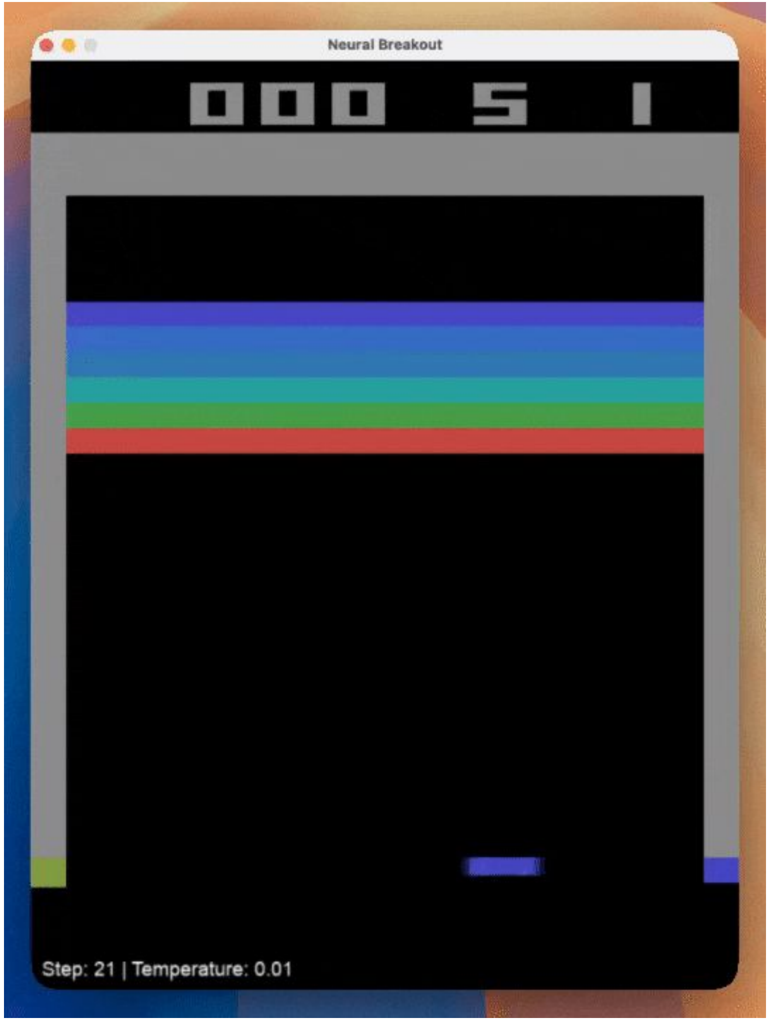
After many days of debugging!

The frames were ordered RGB at one place, but RBG at other place
(Python PIL reorders it!

“Dammit, Claude”

But also *“Thanks Claude”*


drumroll

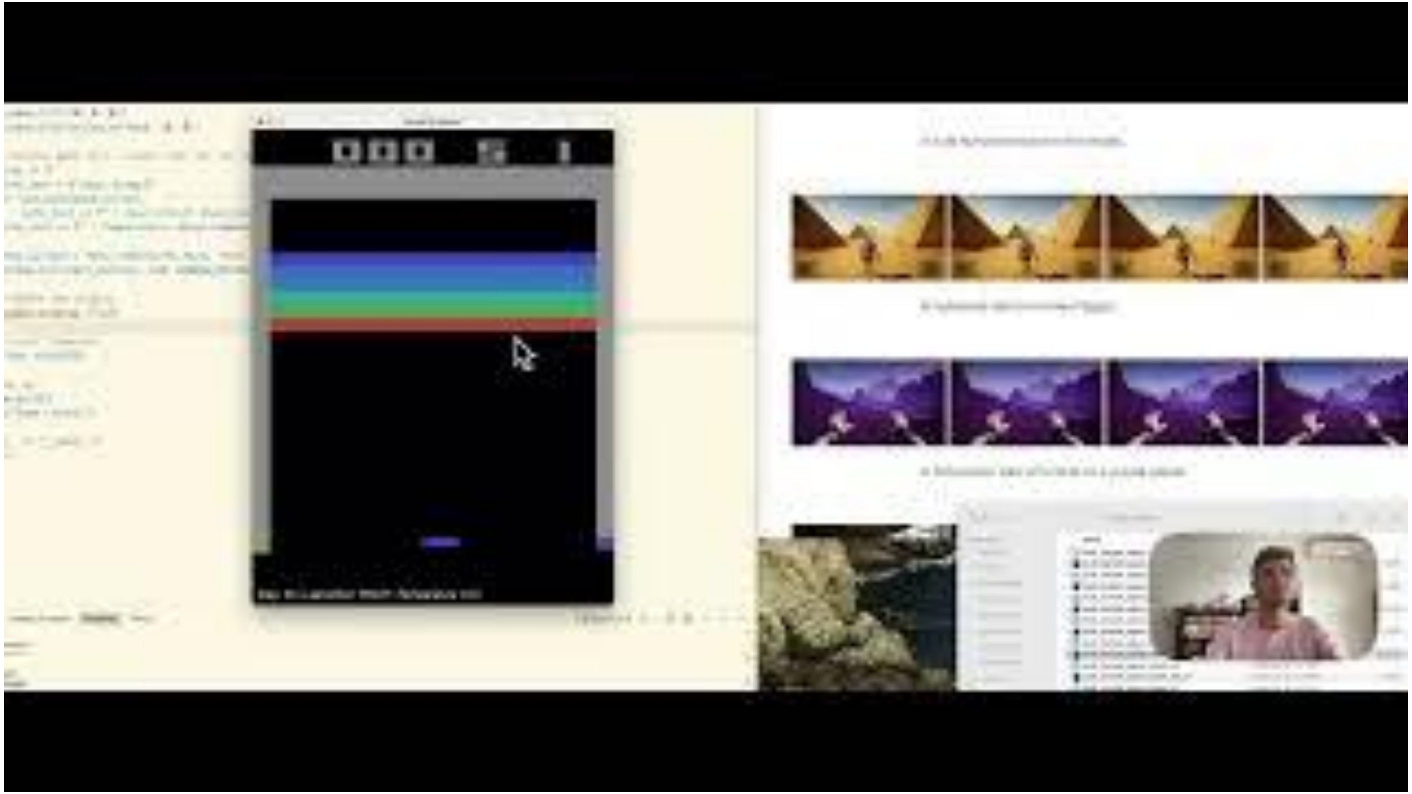


Actions at the bottom -> mine

Notice score increase (0->1) and life lost (5->4)

The entire game (including score and life tracking) is generated in pixels via a neural network

A walkthrough of the entire thing..



Questions?