DiffusionArcade: Diffusion-based framework for real-time simulation of Atari games

Damian Kopp (324944), Colin Smyth (376857), Adriana Orellana (376792), Tim Arni (274586)
CS-503 Project Progress Report 1

I. Milestone Progress

Introduction: Our project explores building a diffusion-based neural game engine for Pong, and potentially other Atari games, following [1]. We are training a reinforcement learning (RL) agent to generate human-like gameplay data, which is then used to train a diffusion model that predicts the next game frame conditioned on previous frames and the player's action. Our code is available at: https://github.com/timarni/DiffusionArcade

RL agent set-up: To collect gameplay data for training the diffusion model, we focus on implementing an effective RL agent playing Pong against a CPU opponent. We explored the PyGame Learning Environment (PLE) [2] and the Arcade Learning Environment (ALE) [3] to run simulations and train the agent. We experimented with three RL algorithms: Q-learning, Deep Q-Networks (DQNs), and Proximal Policy Optimization (PPO). Our goal is not to build the most efficient agent, but to generate diverse data suitable for training the diffusion model, so exploring different RL algorithms can help us increase the data diversity. We also tested different reward functions. (i) The *basic* policy rewards the agent for scoring a point (+1) or winning a game (+5), and penalizes it when the opponent scores (-1) or wins (-5). (ii) The *enhanced* policy additionally applies a penalty proportional to the distance between the agent's paddle and the ball whenever the opponent scores. (iii) The *enhanced_2* policy further rewards the agent (+1) each time it hits the ball. In addition, we experimented with a fully custom reward function designed to simulate human-like play. It rewards the agent for hitting the ball (+1), scoring (+10), and winning (+40), while applying a small penalty (-0.001 per timestep) for staying far from the center of the playing field. Since poorly specified incentives can lead to reward hacking, it is essential to explore alternative reward schemes and carefully audit the resulting behavior [4].

Training the RL agent: After setting up the simulation environments and writing the training and evaluation scripts, we train the RL agent between 500 - 50'000 episodes (i.e. games) using the three different RL algorithms. We monitor the training progress by recording the reward per episode and save the screen every 1000th step (one step being one frame change in the game, with a game having an average of approx. 15'000 steps) to create a mock version of the training dataset. To find working configurations, we experimented with different hyperparameters, namely the learning rate α, the ϵ-decay to balance exploration and exploitation, and the speed ratios of the agent, CPU and ball, as well as the different reward functions. As shown in Figure 1, our agent learns but more work needs to be done to train an agent that competes on par with the CPU player.

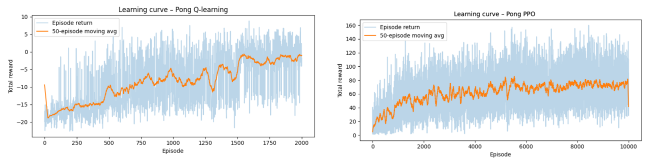

Figure 1. Training-reward curves for the Q-Learning agent in PLE (left) and the PPO agent in ALE (right) under their respective "human-like" reward settings.

Training the diffusion model: We trained our diffusion model using the HuggingFace Diffusers[1] library. We experimented with two schedulers: DDPM and DDIM. Different schedulers affect how noise is handled during sampling, so testing these options helps identify the most stable setup for training and generation. Moreover, we tried different beta schedulers, which control the amount of noise added at each timestep during training (forward diffusion) and inference (denoising) process. For this milestone, we selected the "squaredcos_cap_v2" cosine beta schedule, since it introduces noise more smoothly compared to the linear schedule.

We first trained a diffusion model using black-and-white Pong frames. These images were collected with the previously described reinforcement learning component. The training objective was learning to predict the noise added at different time steps. During inference, we generated random noise and provided it to the model, and it successfully generated new Pong images. However, between the two schedulers, DDPM produced better quality images, although it required more inference steps (1000 in total). In contrast, DDIM was faster, but the generated images were noisier, as shown in Figure 2.

The mock training dataset for the diffusion model consisted of 1592 Pong frames, with 80% used for training and 20% for validation. We used grayscale images of 128x128

[1]https://huggingface.co/docs/diffusers

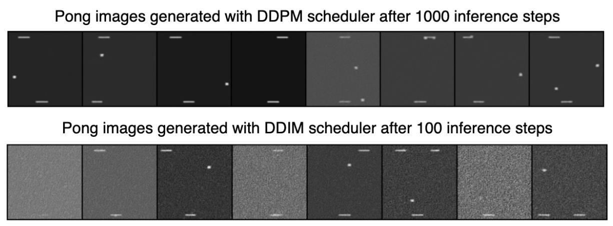
Pong images generated with DDPM scheduler after 1000 inference steps
Pong images generated with DDIM scheduler after 100 inference steps

Figure 2. Comparison of Pong frame generation with DDPM (1000 steps) vs. DDIM (100 steps)

pixels, a learning rate of 0.0004, and trained the model for 30 epochs. Regarding the UNet used, we keep the architecture provided by the Diffusers documentation. Finally, we also experimented generating latent representations using "stabilityai/sd-vae-ft-mse"[2] VAE model, which is designed to work with the Diffusers library. We generated latent representations for the Pong images, and used them to train a diffusion model with the same settings described before. The validation loss curves for all models trained during this milestone are shown in Figure 3.

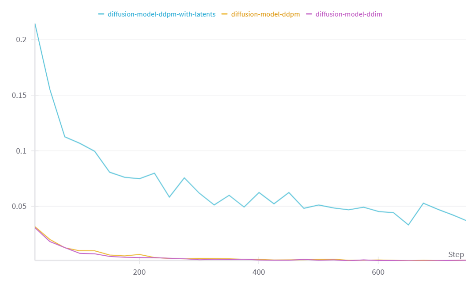

Figure 3. Validation loss curves over 30 epochs for diffusion-based Pong frame generators. Validation loss of DDPM and DDIM on grayscale images achieves the lowest loss, outperforming the latent-space DDPM on VAE representations.

II. Discussion & Next Steps

RL agent results: Initial results show that the agent is learning and that we are able to collect gameplay data. However, the training process of our RL agents is not yet optimal, and further work is needed to reach a level where they can consistently compete with the CPU player and achieve the expected performance [5]. This is crucial as the diffusion model relies on diverse and representative game states to ensure accuracy, requiring the generation of competitive gameplay scenarios.

Furthermore, given our ability to manipulate the reward function, further exploration of well-designed schemes that encourage human-like gameplay is essential. Our current approach, rewarding hits with the paddle and penalizing positions far from the center, promotes more human-like behavior. However, additional rewards for smooth movement and exploration of new states could further enhance agent diversity. Experimenting with different RL algorithms and reward functions will help us generate the diverse data needed for effectively training the diffusion model. To advance the RL component of the project, our next steps are:

- Refine and finalize the RL agents to ensure all three algorithms perform as intended.
- Explore additional human-like reward functions tailored to the Atari environment.
- Collect sufficient data for training the diffusion model and adapt the RL setup to its needs (e.g., balancing exploration vs. exploitation, using curriculum learning, etc.).

Diffusion model results: Concerning the second component of our project, we have successfully trained an initial diffusion model capable of generating new Pong frames. However, it faces two key limitations: it still does not generate high-quality images (some frames show two balls, three paddles, etc.) and the model does not maintain the game's continuity between frames. To address that, our next steps are:

- Train the diffusion model with a larger dataset to improve the quality and consistency of generated Pong frames.
- Explore latent representations: Assess whether fine-tuning the VAE encoder is needed after using a larger dataset.
- Action conditioning: Provide the model with a representation of the player's input at each time step (e.g. move the paddle up, down, or be quiet).
- Game context: Provide game context by concatenating the VAE's latent encodings of the previous frames and combining them with the action embedding.

Additionally, we will work on making the game playable by developing an appropriate interface. Our goal is to have a functional version of the diffusion-based neural game engine for Pong ready by the presentation deadline (21.05). After completing the Pong implementation, we will explore how to generalize our approach to other Atari games for the final submission (30.05).

III. Author Contribution Statement

T.A.: Implemented Q-learning and DQN in PLE; created training and evaluation scripts; adapted setup based on [5]. **D.K.:** Designed and integrated improved reward functions; hyperparameter tuning for RL algorithms; added a learning rate scheduler for more effective training. **C.S.:** Adapted ALE with Gymnasium API; implemented PPO based on [6]; developed custom reward wrappers. **A.O.:** Implemented the diffusion model using HuggingFace's Diffusers; integrated W&B tracking; created configs and documentation; explored VAE-based latent representation pipeline.

[2]https://huggingface.co/stabilityai/sd-vae-ft-mse

References

[1] D. Valevski, Y. Leviathan, M. Arar, and S. Fruchter, "Diffusion models are real-time game engines," 2024. [Online]. Available: https://arxiv.org/abs/2408.14837

[2] N. Tasfi, "Pygame learning environment," https://github.com/ntasfi/PyGame-Learning-Environment, 2016.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.

[4] S. Ibrahim, M. Mostafa, A. Jnadi, H. Salloum, and P. Osinenko, "Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications," 2024. [Online]. Available: https://arxiv.org/abs/2408.10215

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[6] A. Nanda, "Proximal policy optimization with pytorch and gymnasium," November 2024, accessed: 2025-05-11. [Online]. Available: https://www.datacamp.com/tutorial/proximal-policy-optimization