# NITROGEN: An Open Foundation Model for Generalist Gaming Agents

Supplementary Material

## 7. NitroGen model details

### 7.1. Training objective

Given a ground-truth action chunk $a \in \mathbb{R}^{16\times24}$, an observation $o \in \mathbb{R}^{256\times256}$, a flow-matching timestep $t \in [0, 1]$, and Gaussian noise $\epsilon \sim \mathcal{N}(0, \mathcal{I})$, we construct the noisy action as

$$a_t = (1 - t) \cdot \epsilon + t \cdot a$$

and define the conditional velocity field as

$$u^{\text{cond}}(x, t, a, \epsilon, o) = a - \epsilon.$$

The model is trained to predict the velocity field by minimizing the conditional flow-matching loss:

$$\mathcal{L}^{CFM}(\theta, \phi) = \mathbb{E}_{t,a,\epsilon}\left[\left\|\pi_\theta(a_t, \psi_\phi(o), t) - (a - \epsilon)\right\|^2\right], \tag{1}$$

where $\pi_\theta$ is the DiT and $\psi_\phi$ is the image encoder. Following [6, 7], we sample $t$ from a shifted beta distribution that prioritizes small timesteps.

### 7.2. Inference

At inference time, we initialize $a_0 \sim \mathcal{N}(0, \mathcal{I})$ and iteratively denoise for $k$ steps using Euler integration:

$$a_{t+1/k} = a_t + \frac{1}{k}\pi_\theta(a_t, \psi_\phi(o), t). \tag{2}$$

We use $k = 16$ denoising steps, as additional steps yield no measurable improvement.

## 8. Synchronous inference

As described in Section 2.2, we use a Gymnasium API that freezes the game while the model predicts the next action. Frequent pausing and resuming during gameplay could potentially affect the game's physics engine in unknown ways. To rule out this possibility, we record videos and actions (ground truth) of humans playing several games for five minutes each, focusing on parts of the game that are expected to be deterministic (e.g., no enemy behavior randomness). We then replay the same actions from the same initial position: (a) in real time without pausing, and (b) while pausing and resuming the game with random pause durations at high frequency. We find that replayed sequences begin visually diverging after one minute for games with continuous actions and after about three minutes for games with only discrete actions. This result is the same for both (a) and (b), confirming the correctness of our approach: the divergence is simply due to error accumulation.

## 9. Action labeling

### 9.1. Synthetic data generation

To train our action annotation model (Section 2.1), we generate synthetic gamepad overlay frames using four popular controller overlay tools: Open Joystick Display[5], Input Overlay[6], GamePad Viewer[7], and Mini Padder[8]. Across these tools, we curate a total of 438 different controller skins. Crucially, we generate temporally coherent videos in which joystick motions follow smooth trajectories rather than jumping to purely random positions. This allows us to train multi-frame segmentation models on sequences that closely resemble real gameplay footage. Figure 8 shows examples of raw generated overlays, prior to compositing them onto background gameplay video.

### 9.2. Action labeling quality

We measure the performance of our action extraction model. For the joystick we use the $R^2$ score and for the buttons we measure accuracy per frame. We achieve an average $R^2$ of 0.84 for joystick positions and an average button accuracy of 0.96 across the most popular controller families. Full results per controller families can be found in Figure 9.

## 10. Use of Large Language Models (LLMs)

We used large language models (LLMs) solely for polishing the writing and improving the readability of the paper. We did not use them for research ideation, literature search, or any other purpose.

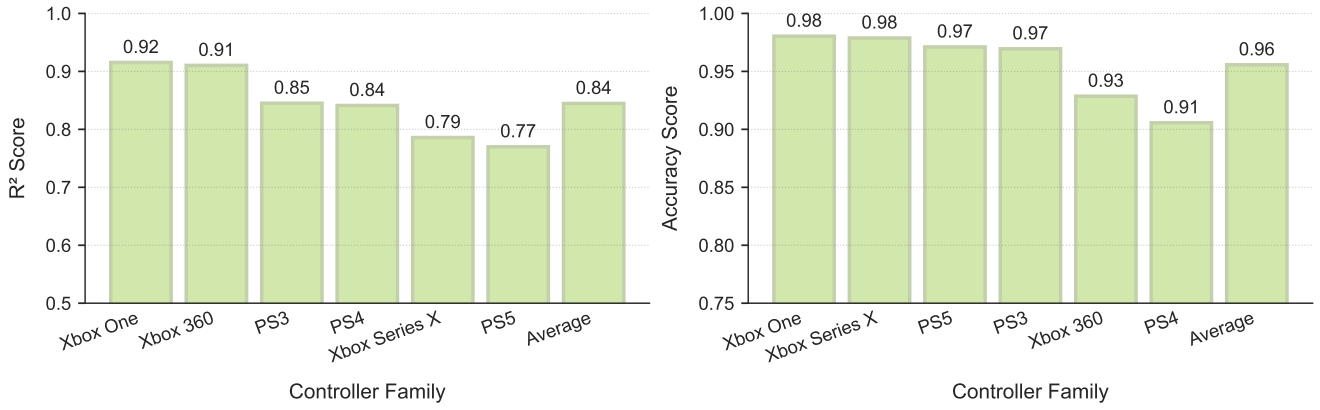
[5] https : / / github . com / AkikoKumagara / open - joystick-display

[6] https://github.com/univrsal/input-overlay

[7] https://beta.gamepadviewer.com/

[8] https://dinir.github.io/mini-padder/


Figure 8. **Synthetic controller overlays.** We programmatically generate gamepad overlay images using existing controller overlay software, producing paired RGB frames, joystick segmentation masks, and ground-truth button states. The figure illustrates the diversity of controller overlay representations across software and skins. For each pair, the left image shows the raw overlay, while the right image highlights joystick segmentation masks (purple) and pressed buttons (green). These synthetic data are used to train our action labelling model.




(a) Joystick $R^2$ performance

(b) Button frame accuracy

Figure 9. **Gamepad parsing performance for different controller families.** We verify the correctness of our action extraction pipeline by comparing performance across different controller families against ground-truth data. (a) shows joystick $\mathbf{R^2}$ correlation scores (averaged for both left and right joysticks) with an overall average of 0.84. (b) shows button frame accuracy with an overall average of 0.96.