NitroGen: An Open Foundation Model for Generalist Gaming Agents

Loïc Magne[1 *], **Anas Awadalla**[1 2 *], **Guanzhi Wang**[1 3 * †]
Yinzhen Xu[1], Joshua Belofsky[4], Fengyuan Hu[1], Joohwan Kim[1]
Ludwig Schmidt[2], Georgia Gkioxari[3], Jan Kautz[1]
Yisong Yue[3 †], Yejin Choi[1 2 †], Yuke Zhu[1 5 †], Linxi "Jim" Fan[1 †]
[1] NVIDIA, [2] Stanford, [3] Caltech, [4] UChicago, [5] UT Austin
[*] Co-lead, [†] Co-advise

https://nitrogen.minedojo.org

Abstract:
We introduce NitroGen, a vision-action foundation model for generalist gaming agents that is trained on 40,000 hours of gameplay videos across more than 1,000 games. We incorporate three key ingredients: 1) an internet-scale video-action dataset constructed by automatically extracting player actions from publicly available gameplay videos, 2) a multi-game benchmark environment that can measure cross-game generalization, and 3) a unified vision-action model trained with large-scale behavior cloning. NitroGen exhibits strong competence across diverse domains, including combat encounters in 3D action games, high-precision control in 2D platformers, and exploration in procedurally generated worlds. It transfers effectively to unseen games, achieving up to 52% relative improvement in task success rates over models trained from scratch. We release the dataset, evaluation suite, and model weights to advance research on generalist embodied agents.

1. Introduction

Building generally capable embodied agents that can operate in unknown environments has long been considered a holy grail of AI research. While computer vision and large language models (LLMs) have achieved this generalization through large-scale pre-training on internet data [????], comparable progress in embodied AI has been impeded by the lack of large, diverse, and labeled action datasets. Video games present an ideal domain for advancing embodied AI since they offer visually rich interactive environments and tasks that span a wide range of complexities and temporal horizons. However, prior approaches face substantial limitations. **LLM–based methods** exploit either (1) hand-crafted programmatic APIs to expose internal game states and control agents [???] or (2) complicated perception modules for textual information extraction and object detection [?]. They enable complex task-solving but require complicated domain-specific design and tuning. **Reinforcement learning** has achieved superhuman performance in individual games such as StarCraft II and Dota 2, but these agents are narrow, costly to train, and depend on specialized simulators rarely available for arbitrary games [?????]. **Behavior-cloning approaches** based on pixel observations have relied on expensive-to-collect demonstrations, constraining training to only a few game titles due to prohibitive data collection costs [??]. To date, there has been little progress on developing open-source frameworks that can support the training and evaluation of generalist gaming agents, further hindering progress in this direction.

To address these limitations, we introduce NitroGen, an open foundation model for video game environments trained on 40,000 hours of publicly available internet videos covering more than 1,000 games. We make three major contributions (Figure 1):

1. Internet-scale dataset of action-labeled videos. We propose to use a new source of data from publicly available videos where content creators overlay their input commands in real time. We train an annotation model to extract frame-level actions with high accuracy, removing

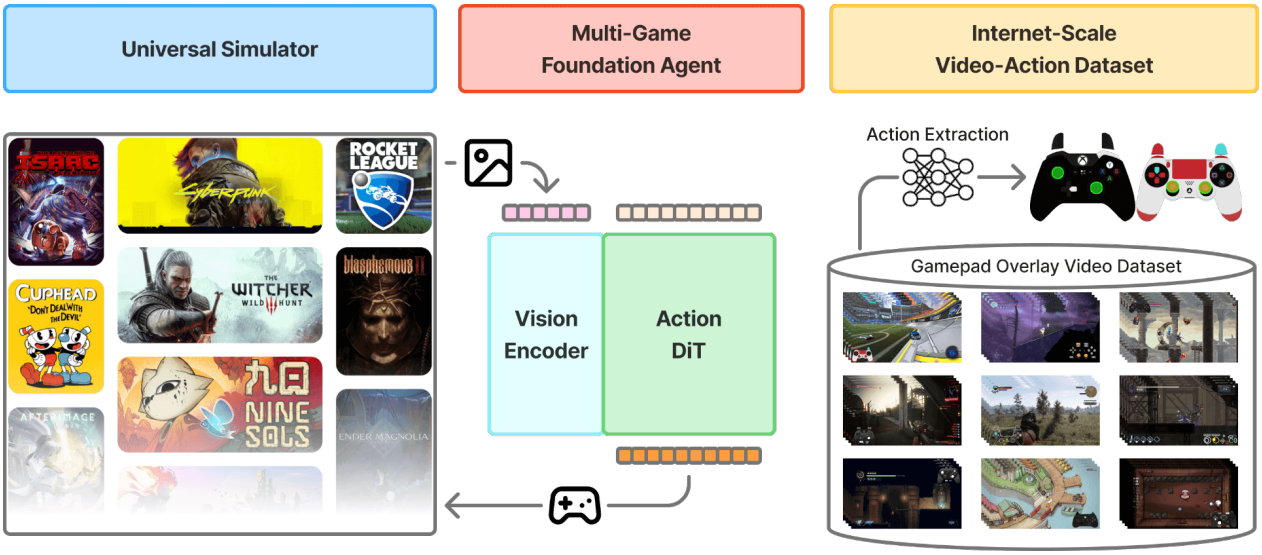


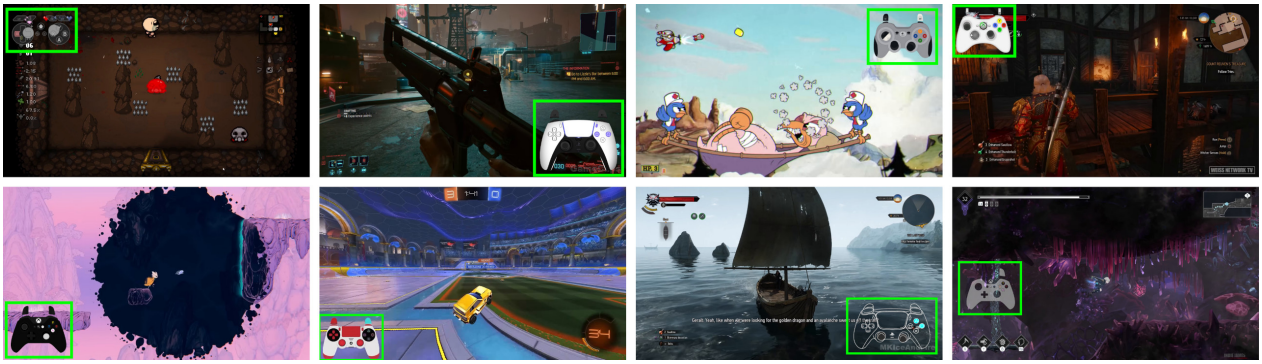
Figure 1: **NitroGen overview.** NitroGen consists of three main components: (1) **Multi-game foundation agent (center)** - a generalist vision-action model that takes in game observations and generates gamepad actions, enabling zero-shot gameplay across multiple titles and serving as a foundation for fine-tuning on new games; (2) **Universal simulator (left)** - an environment wrapper that allows any commercial game to be controlled through a Gymnasium API; and (3) **Internet-scale dataset (right)** - the largest and most diverse open-source gaming dataset curated from 40,000 hours of publicly available gaming videos, spanning more than 1,000 games with extracted action labels.

the need for costly manual data collection and capturing a wide spectrum of real player behaviors. Using this approach, we curate a dataset of 40,000 hours of video spanning more than 1,000 games, providing diverse demonstrations for large-scale training.

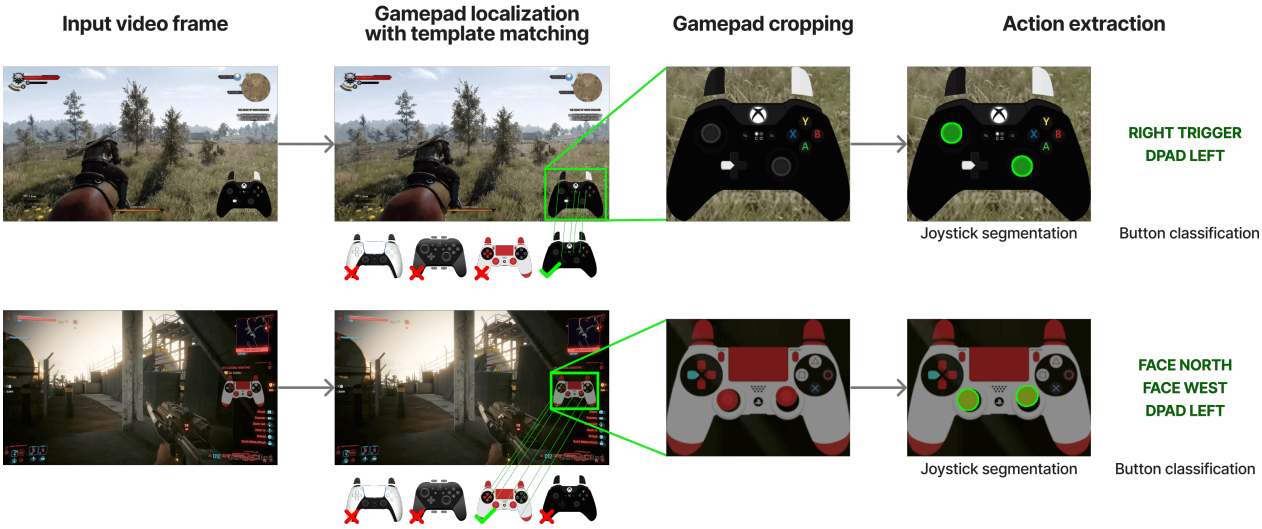
2. Multi-task multi-game evaluation suite. To assess generalization in realistic settings, we design benchmark environments that comprise 30 tasks of varied complexity from 10 commercial games, covering diverse challenges such as combat, navigation, decision-making, platforming, exploration, and puzzle-solving. This benchmark reflects the demands of modern game environments, where agents must learn to adapt across heterogeneous mechanics and objectives. We provide a universal Gymnasium API [**?**] for our evaluation suite that allows users to wrap any game to test diverse agent capabilities. This API is what we refer to as the **universal simulator** in Figure 1.

3. Large-scale behavior-cloning pre-training. To demonstrate the feasibility and benefits of internet-scale pre-training, we train a vision-action transformer model on our dataset. We demonstrate strong results on our benchmark suite, validating our end-to-end pipeline and showing that it is possible to train a strong multi-game policy using only noisy internet data. We show the benefits of behavior-cloning pre-training by post-training our base model on games not seen during training. The model fine-tuned from the pre-trained NitroGen weights shows up to 52% relative improvement in success rates over the model trained from scratch, given a fixed data and compute budget.

We open-source the NitroGen dataset, simulator, and pre-trained weights. We envision NitroGen as a foundational resource that will enable the research community to accelerate progress toward building more generalist embodied agents, fostering new algorithms, model architectures, and applications in this emerging area.



(a) Examples of gamepad overlay videos.

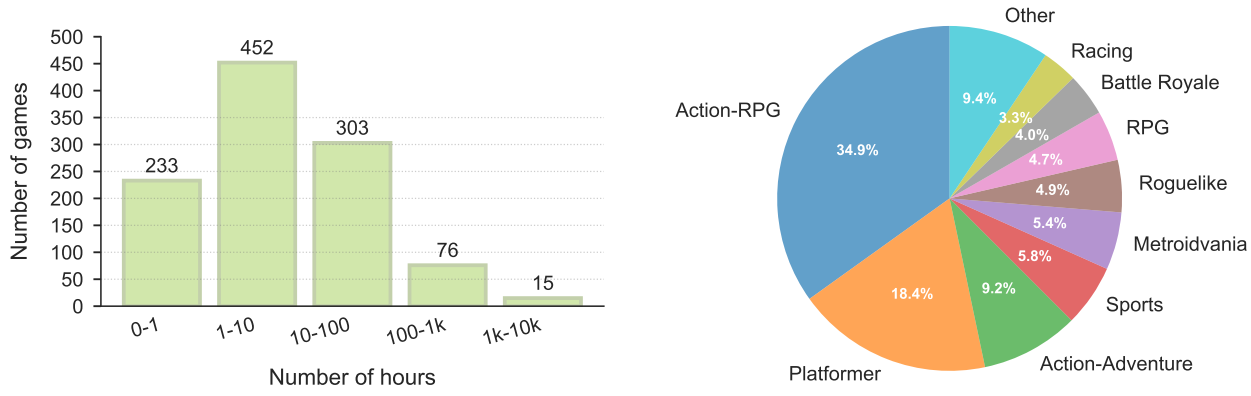


(b) Action extraction pipeline.

Figure 2: **Video-action dataset pipeline overview.** We extract actions from on-screen displays which show the gamepad actions of the player in real-time; called "input overlays". **(a) Dataset curation.** We collect publicly available videos displaying a "gamepad overlay". The diversity of these overlays presents significant challenges, as gamepads vary widely across content creators in controller types (e.g., Xbox, PlayStation, or others), transparency levels, and visual artifacts introduced by video compression. **(b) Action extraction.** For each collected video, we localize the gamepad by sampling 25 frames and running **keypoint matching** against a curated set of templates using SIFT and XFeat features. We use the template-matching results to localize and crop the gamepad region from each video. A **hybrid classification–segmentation network** is then trained to predict joystick positions and button states from the cropped controller images, enabling accurate reconstruction of player inputs.

2. Approach

NitroGen consists of three novel components: (1) an internet-scale video dataset with action labels, (2) a multi-game benchmark with a Gymnasium environment wrapper, and (3) a vision–action model pre-trained through large-scale behavior cloning. In this section, we provide details of each component.



(a) Distribution of hours per game

(b) Genre distribution by total hours

Figure 3: **Distribution of the NitroGen dataset across games and genres.** After filtering, the NitroGen dataset contains 40,000 hours of gameplay videos spanning more than 1,000 games. (a) Hours per game shows broad coverage, with 846 games having over one hour of data, 91 games with over 100 hours, and 15 games exceeding 1,000 hours each. (b) Genre distribution reveals Action-RPG games are most common (34.9% of total hours), followed by Platformer (18.4%) and Action-Adventure (9.2%) games, with the remainder distributed across seven genres.

2.1. Internet-scale multi-game video-action dataset

Annotation challenge. A central challenge in training policies from internet videos is recovering the corresponding actions, since most gameplay recordings typically do not include the player's inputs. We address this limitation by using a novel source of publicly available videos in which such labels can be recovered. These videos feature *input overlay* software that displays a real-time visualization of the player's actions, typically as a 2D image of a gamepad in a corner of the screen with pressed buttons highlighted (Figure 2a).

Dataset curation. Although input overlays appear in only a fraction of online gameplay videos, they occur frequently enough to enable the construction of a large-scale dataset. We collect 71,000 hours of raw video containing gamepad overlays. While input overlay software was originally used primarily within the speedrunning community, its use has since expanded to many action games and among both expert and casual players. To avoid over-representation of any single title, we use a combination of keyword-based searches and curation guided by content diversity, ensuring coverage across games, genres, and skill levels. This approach balances casual and competitive play styles while maintaining broad genre representation. Figure 3 shows the distribution of gameplay hours by title and genre. The dataset covers more than 1,000 unique games, making it the largest labeled video–action dataset for video games to date. It contains 38,739 videos from 818 different content creators, with an average video duration of 1 hour and 50 minutes.

Action extraction. We extract player inputs from gameplay videos through a three-stage pipeline: (1) template matching to locate and crop the gamepad overlay, (2) gamepad action parsing using a fine-tuned segmentation model, and (3) quality filtering to ensure accurate and meaningful data.

Stage 1: Template matching. To locate gamepad overlays within gameplay videos, we apply template matching using a curated set of approximately 300 common controller templates. For each video, we sample 25 frames and perform feature matching with SIFT [**?**] and XFeat [**?**] against

all curated templates. We estimate an affine transformation from the paired keypoints and require at least 20 inliers for a match to be considered valid. We then extract the region with the highest matching score, which defines the gamepad location for subsequent processing. Figure 2b shows examples of successful match.

Stage 2: Gamepad action parsing. We parse controller states using a fine-tuned SegFormer [**?**] segmentation model that processes pairs of consecutive frames. The model takes two consecutive frames as input (concatenated along the spatial dimension) to capture short-term temporal dynamics. It outputs a segmentation mask to localize joystick positions on a discrete 11×11 grid, and binary button states (Figure 2b). Empirically, we find that estimating joystick positions via segmentation masks significantly outperforms direct regression of joystick coordinates.

We train the annotation model using synthetic data generated by sampling frames from the NitroGen training set and programmatically overlaying controller templates using the Open Joystick Display[1], Input Overlay[2], and GamePad Viewer[3] software. For each template, we produce multiple frames with random button states and joystick positions, yielding 8M labeled frames. To simulate real-world visual artifacts, we vary overlay opacity, controller size, and video compression, generating multiple variants per frame. We train the action parsing SegFormer model using the AdamW optimizer [**??**] with a learning rate of 0.0001, linear learning rate decay, weight decay of 0.1, and a batch size of 256.

At inference, we compute precise joystick positions by detecting contours for each joystick over the entire video. To estimate the center position of each joystick, we average positions from all frames where the joystick is classified as centered in the 11×11 discrete grid. We then normalize the positions to the range $[-1.0, 1.0]$ using the 99th percentile of absolute x and y values over the video to reduce the influence of outliers.

Stage 3: Quality filtering. The final stage applies targeted filtering strategies to ensure high-quality data. During training, we observe that using the raw 71,000 hours of data leads to the model over-predicting the null action as noted in VPT **?**. To avoid that, we discard segments based on action density: we only keep chunks where at least 50% of the timesteps have non-zero button or joystick actions, resulting in 55% of the data being kept. For all gameplay videos, we mask the on-screen controller to prevent models from exploiting it as a shortcut.

2.2. Evaluation suite

Universal simulator for any game title. Many research environments provide a Gymnasium API [**?**] that enables programmatic control of the simulation. To bring this capability to commercial video games, which typically lack such an interface, we develop a universal simulator that can wrap any game title with a Gymnasium API for model development. The library intercepts the game engine's system clock to control simulation time, enabling frame-by-frame interaction without modifying game code. This approach works with any title that uses the system clock for physics and interactions, which is a common practice in game development. We leave real-time or asynchronous deployment to future work. Frequent pausing and resuming during gameplay could potentially affect the game's physics engine in unknown ways, we verify that this process does not alter games' physics and behaviors (see Appendix B.1.).

Unified observation and action space. Using this simulator, we introduce a multi-game, multi-task benchmark with a shared interface across all titles. Observations are single RGB frames.

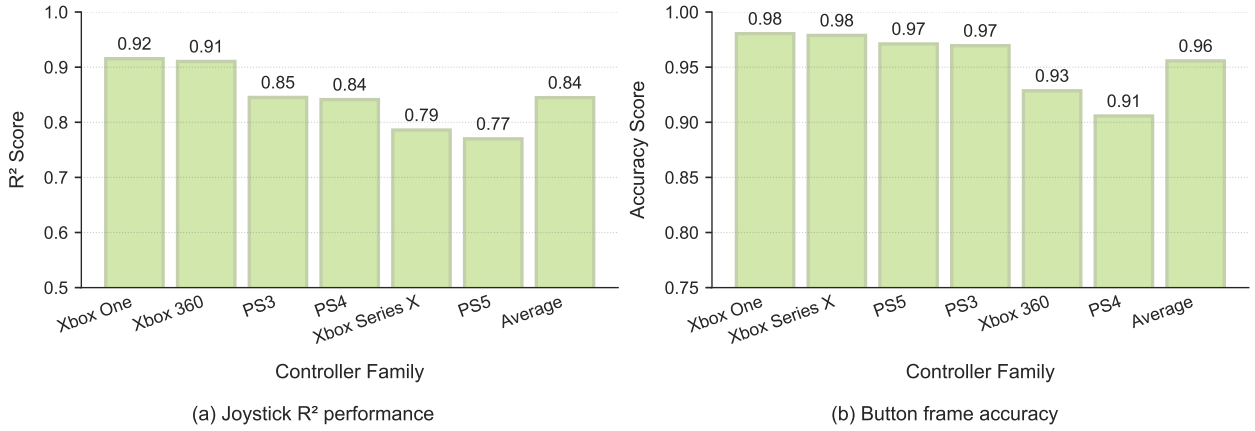
[1] `https://github.com/AkikoKumagara/open-joystick-display`
[2] `https://github.com/univrsal/input-overlay`
[3] `https://beta.gamepadviewer.com/`

Figure 4: **In-game rollouts.** We show NitroGen performing tasks in diverse 2D and 3D environments. These tasks can take from a few seconds to a few minutes to perform. Some of them include memorization, while others are performed in procedurally generated worlds and require the model to adapt.

Actions consist of a standardized 16-dimensional binary vector for gamepad buttons (4 d-pad buttons, 4 face buttons, 2 shoulders, 2 triggers, 2 joystick thumb buttons, start, back) plus a 4-dimensional continuous vector for joystick positions. Unlike prior work that defines game or task-specific action spaces [**??**], this unified layout facilitates direct policy transfer across diverse games.

Diverse evaluation tasks. The evaluation suite serves as a universal evaluation framework for multi-game policies, covering 10 games across diverse visual styles and genres with 30 tasks total. The suite includes five 2D games and five 3D games, each testing different skill combinations. The 2D games include three side-scrollers and two top-down roguelikes with procedurally generated levels. The 3D games consist of two open-world games, two combat-focused action-RPGs, and one sports game.

Tasks are distributed across three categories: 11 combat tasks (boss fights, enemy encounters), 10 navigation tasks (reaching specific locations, traversing environments), and 9 game-specific tasks (unique mechanics particular to individual games). Each task has clearly defined start and goal states, with attempts typically lasting a few minutes, though human players may require several hours



(a) Joystick R^2 performance

(b) Button frame accuracy

Figure 5: **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 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.

of repeated attempts to succeed. We select tasks where the initial visual state provides sufficient context to elicit correct behavior, leaving language-conditioned specifications to future work. Success rates are measured through human evaluation.

2.3. NitroGen foundation model

Architecture. Building on recent advances in generative modeling and robotics, NitroGen employs flow matching [**?**] to generate chunks of future actions conditioned on visual observations. The architecture is adapted from GR00T N1 [**?**] with the language and state encoders removed, and a single action head. RGB inputs at 256×256 resolution are encoded using a SigLIP 2 vision transformer [**?**], producing 256 image tokens per frame. Actions are generated with a diffusion transformer (DiT) [**?**] that outputs multiple actions per forward pass. Noisy action chunks are first encoded by an MLP into one action token per timestep, then processed through several DiT blocks consisting of alternating self-attention and cross-attention layers. Cross-attention conditions action generation on the encoded frame tokens. The final action tokens are decoded into continuous action vectors using an MLP applied independently across the time dimension. Full mathematical details are provided in Appendix A.

Design choices. Although the model can condition on multiple frames, we find no benefit from using more than one past frame, even with increased temporal gaps. This is likely because the initial state of these action games already provides sufficient context to elicit the appropriate behavior. We instead use a single context frame and generate 16-action chunks, which improves temporal consistency compared to single-action generation.

Training and inference. We train NitroGen using the standard conditional flow-matching objective [**??**], applied to 16-action chunks, with one 256×256 frame of context. Inference follows the corresponding denoising process with $k = 16$ steps.

During training, we apply the following image augmentations: random brightness, contrast, saturation and hue, random rotation between -5 and 5 degrees, and random crops. We train all models using AdamW [**??**] optimizer with a weight decay of 0.001. We use a warmup-stable-decay (WSD) schedule [**?**], which allows us to train for longer without a fixed training budget, with a

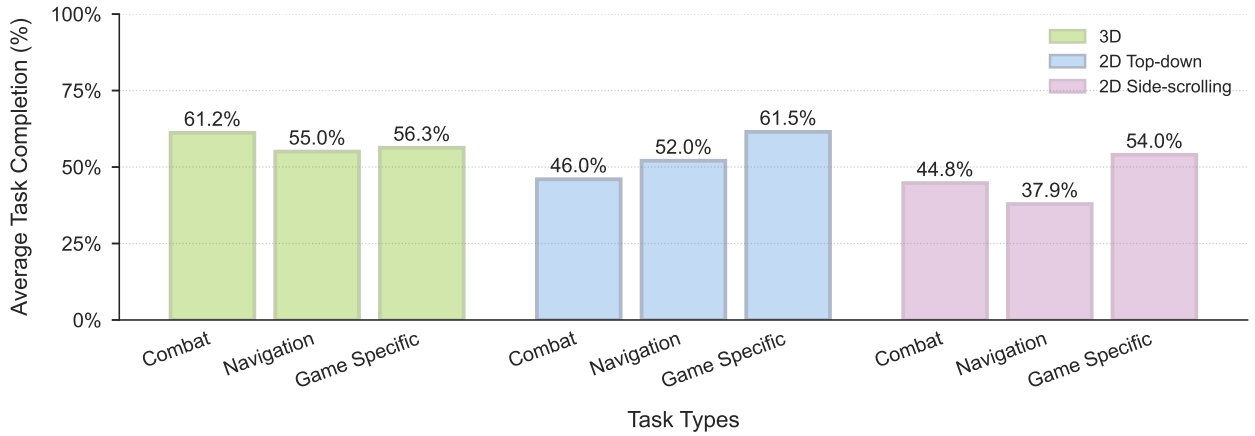


Figure 6: **NitroGen 500M pre-training results across different games**. We evaluate NitroGen after behavior-cloning pre-training. The model is not fine-tuned for specific games. For each game, we measure the average task completion rate on 3 tasks with 5 rollouts per task. Despite being trained on a very noisy internet dataset, NitroGen is able to perform non-trivial tasks over games with different visual styles (3D, 2D top-down, 2D side-scrolling) and genres (platformer, action-RPG, roguelike, etc.).

constant learning rate phase of 0.0001. Following **?**, we maintain an exponential moving average (EMA) of model weights during training with a decay of 0.9999. All our results are obtained with the EMA weights, which we find consistently outperform the non-EMA weights.

3. Experiments

Performance of the gamepad action extraction model. To evaluate our action extraction pipeline, we construct a benchmark dataset by recording gameplay from six video games using OBS[4], with randomized opacity, gamepad size, and gamepad type to mimic real-world conditions. We record ground-truth controller inputs at each frame and compare them with the extracted actions. We measure joystick accuracy with the R^2 score and button accuracy per frame. As shown in Figure 5, 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.

NitroGen demonstrates strong capabilities across a wide range of games. We train a single model on the entire dataset from Section 2.1. Without further fine-tuning, NitroGen achieves non-trivial success rates across many games and tasks. Figure 6 summarizes the main results. We observe that NitroGen performs well both on tasks that can be memorized and on tasks that require zero-shot generalization. For example, some games feature fixed layouts that the model may have partially encountered during training, while others employ procedural generation that ensures each playthrough is unique. We do not find significant differences in performance between these two categories, suggesting that NitroGen can both leverage memorization and adapt to unseen scenarios.

This result validates that it is possible to train a robust policy using only noisy internet-scale data. The dataset includes several sources of noise that could hinder training: **(a) actions are not strictly ground truth**, since input overlay software introduces small delays, and parsing adds further inaccuracies; **(b) video frames often contain creator-specific artifacts** such

[4]Open Broadcaster Software: `https://obsproject.com/`; Input recording tool: `https://github.com/loicmagne/input-rec`

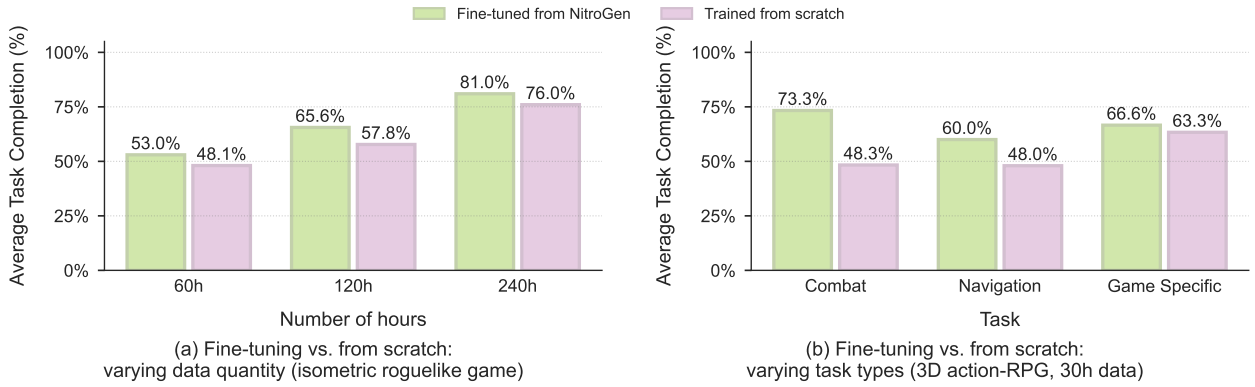


Figure 7: **Post-training experiments: NitroGen pre-training improves downstream agents in unseen environments.** We pre-train NitroGen on the dataset described in Section 2.1, holding out one game. We then fine-tune the pre-trained checkpoint on the held-out game and compare the results with a model trained from scratch using the same architecture, data and compute budget. (a) When varying data quantity, task-completion rate scales with dataset size, and fine-tuning achieves on average a 10% relative improvement in task-completion rate. (b) When varying task type in the low-data regime (30h), fine-tuning achieves up to 52% relative improvement in task-completion rate.

as livestream chats, subscribe prompts, or progress trackers; and **(c) controller configurations vary across players**, differences in sensitivity settings or custom button mappings can change the semantic meaning of the same input. Despite these challenges, Figure 6 shows that large-scale pre-training yields a robust multi-game policy.

NitroGen pre-training improves downstream fine-tuning on unseen environments. We evaluate transfer learning by pre-training NitroGen on the full dataset except for a held-out game, then fine-tuning on this game with a limited amount of data. We compare this fine-tuned model with an identical architecture trained from scratch using the same data and compute budget. Results are shown in Figure 7. We study two representative games with different visual styles and genres: an isometric roguelike and a 3D action-RPG.

The effectiveness of pre-training varies by game type and task category. Across different data quantities, fine-tuning achieves an average relative improvement of 10% on the isometric roguelike, whereas the 3D action-RPG shows a 25% average relative improvement. This difference likely stems from better representation of 3D action-RPGs in the training distribution, while the isometric roguelike has gameplay mechanics and visual style that are less common in the training data.

Furthermore, pre-training benefits are not uniform across task types. On the 3D action-RPG, generic tasks such as combat (52% relative improvement) and navigation (25% relative improvement) benefit substantially from pre-training, while game-specific tasks show only marginal gains (5% relative improvement). This suggests that NitroGen effectively learns transferable skills for common gameplay patterns, but game-specific mechanics still require targeted training on the new environment.

4. Limitations and future work

Design limitations. NitroGen is limited to being a fast-reacting system-1 sensory model. It cannot plan over long horizons or follow language instructions; the model only reacts to the short context it sees. We develop NitroGen aiming for it to serve as a foundation for future generalist agent development, where post-training for language-following and reinforcement learning can be applied

to enhance planning capabilities and improve success rates.

Dataset bias. While diverse, our data collection method still restricts the types of games included in our dataset. The data distribution of the NitroGen dataset is biased toward action games (Figure 3), and games that are typically played with a gamepad. Keyboard-only games or those that involve complex manipulation are less represented in the dataset. This bias may limit the agent's ability to generalize to genres like strategy or simulation games that rely more on planning and keyboard input.

5. Related works

Gaming agents. Video games have long been testbeds for AI, with approaches generally following three directions. Reinforcement learning achieved landmark successes from Atari with DQN [**??**] to AlphaGo [**?**], AlphaStar [**?**], and OpenAI Five [**?**], but these rely on engineered rewards, hand-crafted features, and specialized simulators. More recent vision-based methods like Dreamer 3 [**?**] still require dedicated simulators and environment-specific training. A second line leverages large language models for high-level reasoning with structured APIs, as in Voyager [**?**] and Cradle [**?**], but these depend on hand-crafted interfaces. A third category learns directly from pixels or states via behavior cloning, including MineRL [**?**], VPT [**?**], SIMA [**?**], GATO [**?**], Dreamer 4 **?**, Lumine **?**, and Farhang et al. **?**, but they all rely on datasets bootstrapped from human demonstrations or RL-generated data. NitroGen advances this third direction by scaling behavior cloning to internet-scale, enabling training across hundreds of games without costly collection. Game-TARS **?** is a concurrent work that also train a multi-game agent. They combine contractor data and multi-modal reasoning data to train on a total of 20,000 hours.

Embodied foundation models. Foundation models for embodied AI generally adopt either hierarchical reasoning or end-to-end learning. Hierarchical methods pair pre-trained LLMs or VLMs with low-level policies [**?????**] treating the foundation models as black-boxes. Vision-Language-Action (VLA) models [**???????**] instead train policies end-to-end on embodied data, though generalizing across tasks and embodiments remains challenging. NitroGen differs by discarding language conditioning and focusing purely on scalable vision-action mapping using diverse gameplay data.

Large-scale action datasets. Progress in vision and NLP has been driven by large labeled datasets, but embodied AI lags behind due to the difficulty of collecting action-labeled data and defining standardized action spaces. Gaming datasets like MineRL [**?**] provide limited coverage, while MineDojo [**?**] scales video data without action labels. VPT [**?**] annotates 70,000 hours via inverse dynamics but is limited to Minecraft. Other work seeks to infer latent actions from videos [**????**], though scalability is unclear. In robotics, teleoperation has produced datasets such as Roboturk [**???**], ALOHA [**?**], TeleMoMa [**?**], Open X-Embodiment [**?**], and AgiBot World [**?**], but these are costly, limited in scale, and lack organic diversity. NitroGen introduces a scalable alternative by leveraging input overlay software, which naturally provides action labels in publicly available gameplay videos.

6. Conclusion

In this work, we introduce NitroGen, an approach to scale up foundation pre-training for video game agents and demonstrate how internet pre-training can yield a generalist policy. We leverage a new source of publicly available data to build an internet-scale video-action dataset, and empirically demonstrate its effectiveness by successfully training a multi-game policy. NitroGen shows positive

signs of generalization in fine-tuning experiments. By lowering the barrier to train agents on new environments, NitroGen serves as a starting point to develop more powerful and general-purpose agents.

A. NitroGen model details

A.1. Training objective

Given a ground-truth action chunk $a \in \mathbb{R}^{16\times 24}$, an observation $o \in \mathbb{R}^{256\times 256}$, 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:

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

where π_θ is the DiT and ψ_ϕ is the image encoder. Following **??**, we sample t from a shifted beta distribution that prioritizes small timesteps.

A.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.

B. Evaluation

B.1. 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.