From Offline Trajectories to Online Adaptation: A Multimodal JEPA Pretraining Study on Pokemon Red

Stefano Campese [1] **Alessandro Moschitti** [2]

Abstract

We study self-supervised pretraining for reinforcement learning in a long-horizon, partially-observable environment (Pokemon Red). A multimodal JEPA encoder fusing pixels and game RAM is pretrained on offline trajectories and used as a frozen feature extractor for downstream PPO. On 48 held-out starting states never seen during training, frozen JEPA outperforms generic vision pretraining (DINOv3), random initialization, and full-encoder fine-tuning by 2–8× in cumulative reward. Initializing JEPA from DINOv3 weights yields OOD performance comparable in mean to end-to-end DreamerV3 (4.48 ± 0.15 vs 4.19 ± 0.80), with $\sim 5\times$ lower cross-seed standard deviation, indicating substantially higher reproducibility under a simpler two-stage pipeline. Linear probes show fine-tuning catastrophically degrades the pretrained representation: event-progression MSE rises $3.7\times$ while map and party-HP probes lose 5–68%. A data ablation reveals that data *diversity* matters more than data *expertise*: trajectories collected by a biased-random policy from diverse starting states yield strictly better pretraining data than trajectories from a trained expert policy on the same start distribution.

1. Introduction

Pokemon Red is a long-horizon, partially-observable game in which an agent must navigate a 2D world, capture and train a party of creatures, and progress through a scripted storyline by defeating eight gym leaders and four Elite Four trainers. As an RL task it has three properties that make it challenging: (i) episodes span tens of thousands of frames, (ii) progress is gated by sparse, hand-coded rewards (badges, story flags, level gains), and (iii) the same screen can correspond to many different game states because critical information (party HP, event flags, item bag) is hidden in the emulator's RAM rather than visible on-screen. The community has adopted Pokemon Red as a benchmark for sample-efficient RL with sparse rewards (Whiddy, 2024; Suárez, 2024). End-to-end RL on Pokemon Red is expensive and high-variance: state-of-the-art world models like DreamerV3 (Hafner et al., 2023) require tens of GPU-hours per seed and exhibit large run-to-run variability. We ask whether the same task can be solved more efficiently and more reproducibly by separating representation learning from policy learning. In this paper we study how offline trajectories collected in Pokemon Red can be used to pretrain a representation that an online RL agent later consumes. Concretely: we pretrain a multimodal JEPA encoder (Assran et al., 2023; Bardes et al., 2024; LeCun, 2022) that fuses the screen pixels and a hand-engineered RAM feature vector on 5M frames of offline data; we then run PPO (Schulman et al., 2017) on top of this encoder for 5M environment steps; and we evaluate on 48 hand-crafted, human-validated starting states never seen during training. We compare against end-to-end DreamerV3 (Hafner et al., 2023) as a world-model reference, against raw DINOv3 features (Siméoni et al., 2025) as a generic vision pretraining reference, and against random-init and end-to-end fine-tuned variants as ablations. This setup sits in the broader space of offline-to-online learning, where two complementary strategies are common. The first is to fit a policy directly from offline data, e.g. via behavioral cloning (Pomerleau, 1991), conservative value regularization (Kumar et al., 2020), in-sample expectile learning (Kostrikov et al., 2022), or return-conditioned sequence modeling (Chen et al., 2021). The second, which we follow, is to use the offline data only to learn a representation, and then to plug an off-the-shelf online RL algorithm on top. Adopting the second strategy lets us isolate three practical questions that a practitioner faces independently of the policy class. **(Q1)** What self-supervised objective transfers from offline trajectories to online RL, and does multimodality (pixels + symbolic state) help? **(Q2)** How does the policy that collected the offline data shape the resulting representation, holding the data budget fixed? **(Q3)** Once the encoder is pretrained, should it be frozen or fine-tuned during online adaptation?

Findings. Each finding is supported by behavioral (down-

[1]University of Trento [2]Amazon AGI. Correspondence to: Stefano Campese <stefano.campese@unitn.it>.

Accepted at the ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning.

stream RL reward) and mechanistic (linear probe) evidence.

F1 (Q3). Freeze, do not fine-tune. Frozen multimodal JEPA dominates every other frozen-encoder choice and end-to-end fine-tuned JEPA by 2–8× on OOD reward; fine-tuning during PPO collapses performance below random init.

F2 (Q1). Combine SSL paradigms. JEPA initialized from DINOv3 and pretrained on game trajectories matches DreamerV3 in mean OOD reward (4.48 ± 0.15 vs 4.19 ± 0.80) with $\sim 5\times$ lower seed variance.

F3 (Q3, mechanistic). Fine-tuning destroys structure. Linear probes show fine-tuning raises event-progression MSE $3.7\times$ and loses 5–68% on map and party-HP probes; the downstream collapse mirrors a representational collapse and is itself consistent across seeds (σ=0.002 on OOD reward), i.e. structural rather than seed noise.

F4 (Q2). Diversity beats expertise. Trajectories from a biased-random policy on diverse starts yield strictly better pretraining data than trajectories from a trained expert policy on the same starts; adding the expert data *degrades* the representation.

Data. We release the 48-state OOD suite, and a 240-state curated benchmark (superset of the 48 states) on github[1].

2. Related Work

Offline-to-online RL. Offline policy learning fits a policy directly to the offline dataset, e.g. via conservative value regularization (Kumar et al., 2020), in-sample expectile learning (Kostrikov et al., 2022), or return-conditioned sequence modeling (Chen et al., 2021); offline-to-online finetuning then initializes online RL from this policy. Our work is complementary: we use the offline data only for *representation* learning, leaving the online algorithm (PPO) untouched. Findings F1 and F4 are therefore composable with any of the methods above.

Pretrained visual representations for control. The closest line of work to ours treats SSL pretraining as a frozen perception module for downstream policy learning. Nair et al. (2022) introduced R3M, a frozen Ego4D representation for manipulation; Majumdar et al. (2023) introduced VC-1, a large-scale MAE-pretrained "visual cortex" for embodied AI, and report, consistently with our F1, that frozen pretrained features outperform end-to-end training across a range of embodied tasks. Both target visual control with natural-image pretraining; our dual setting uses in-domain game trajectories with pixels + RAM.

JEPA family. Joint-Embedding Predictive Architectures predict in latent space rather than pixel space and have grown rapidly across modalities: I-JEPA for static images (Assran et al., 2023), V-JEPA for videos (Bardes et al., 2024), EB-JEPA as a unified energy-based library (Terver et al., 2026), and LeWorldModel as a stable end-to-end JEPA world model from raw pixels (Maes et al., 2026). The broader vision (LeCun, 2022) motivates predictive representations as a foundation for autonomous agents. EB-JEPA and LeWorldModel are the closest to ours: both study JEPA as a representation for control on visual benchmarks. We extend the line to a long-horizon, partially-observable game and add the data axis (F4).

Generalization in RL. Our OOD evaluation is a context-MDP study where the context is the starting state (Kirk et al., 2023); the ID/OOD gap we observe (Appendix E) is consistent with that survey's finding that ID parity masks OOD degradation.

Pokemon Red as an RL benchmark. Pokemon Red as an RL benchmark traces back to Whiddy (2024) and the `pokemonred-puffer` family (Suárez, 2024); we use the `r2dreamer` reimplementation of DreamerV3 (Hafner et al., 2023) as the end-to-end baseline, and DINOv2/v3 (Oquab et al., 2023; Siméoni et al., 2025) for the natural-image init variant.

3. Method

Our pipeline has three stages: (i) collect 5M frames of offline trajectories from 32 hand-crafted starting save-states using a biased-random behavior policy; (ii) pretrain a multimodal JEPA encoder on those frames; (iii) train PPO on top of the (frozen or fine-tuned) encoder for 5M environment steps.

Environment. All experiments are run inside the original Pokemon Red ROM, emulated via PyBoy, a Python Game Boy emulator. PyBoy exposes both the raw framebuffer and the full 8 kB main work RAM (WRAM) at every step, which we use to construct multimodal observations (below). The discrete action space has 7 elements (4-direction d-pad, A, B, START); the SELECT button is masked out as it is unused in normal gameplay.

Multimodal observations. We extract two modalities per step from PyBoy: the native Game Boy framebuffer $s_t \in \mathbb{R}^{144\times160\times3}$ (encoded by a ViT with 16×16 patches), and a 423-dim symbolic vector $r_t \in \mathbb{R}^{423}$ read from known WRAM addresses (processed by a small MLP). r_t is not raw bytes but a hand-engineered parse of game-mechanic state: party, position, story flags, battle state. The slice-by-slice composition is in Table 4.

We prefer this parsed feature vector to raw RAM bytes because it is semantically dense (no unused address space), cheap (a few hundred byte reads per step), and stable across all three regional release revisions of the cartridge: the WRAM offsets we use (`wBattleType`, `wPartyMons`, `wEventFlags`, etc.) are documented in the disassembled ROM (community, 2024).

Encoder fusion. The two streams are encoded independently and concatenated: $z_t^{\text{vis}} = \text{ViT}(s_t) \in \mathbb{R}^{384}$, $z_t^{\text{ram}} = \text{MLP}(r_t) \in \mathbb{R}^{128}$, $z_t = W_{\text{fuse}}[z_t^{\text{vis}}\,;\, z_t^{\text{ram}}] \in \mathbb{R}^{512}$, where

[1] https://github.com/sirCamp/jepa-pokemon-red

the RAM MLP is a 3-layer GELU network $(423 \to 256 \to 256 \to 128)$ and W_{fuse} is a single linear projection. The fused z_t is the latent on which the JEPA pretraining losses (described next) operate. Neither modality is sufficient alone: pixels miss off-screen RAM state (party HP, event flags), and RAM misses dialog text and menu cues. Linear probes (Table 2) confirm z_t recovers all RAM-derived targets near-perfectly while remaining usable as the sole input to the downstream policy/value heads. We also study a variant in which the visual ViT is initialized from DINOv3-ViT-S (Siméoni et al., 2025) pretrained on natural images (LVD-1689M).

Pretraining objective. Following the JEPA family, the encoder is pretrained on offline trajectories under a sum of three losses (no pixel reconstruction): (i) an inverse-dynamics objective predicting the action from (z_t, z_{t+1}); (ii) a sigma-regularization term enforcing a non-collapsed embedding distribution; (iii) a latent-prediction loss training a small action-conditioned predictor $g_\phi(z_t, a_t) \approx z_{t+1}$. We use 5M frames of trajectory data drawn from 32 hand-crafted starting save-states and a biased-random behavior policy (*multistate* data, used by default unless otherwise stated).

Downstream RL. For each encoder variant we run PPO (Schulman et al., 2017) for 5M environment steps, sampling starting states uniformly from the same 32 saves used for pretraining. On top of the (frozen or fine-tuned) encoder we attach two shallow MLP heads: a policy head $\pi(a \mid z) = \text{softmax}(W_2 \tanh(W_1 z))$ with $W_1 \in \mathbb{R}^{256\times512}$, $W_2 \in \mathbb{R}^{7\times256}$, and an analogous value head with a scalar output. The policy/value heads are always trained; only the encoder is frozen or fine-tuned depending on configuration. We compare four *frozen-encoder* configurations (JEPA-from-scratch, JEPA-with-DINOv3-init, raw frozen DINOv3, random init) plus an *end-to-end fine-tuned* JEPA variant. As an end-to-end baseline we run DreamerV3 (Hafner et al., 2023) in the same env at the framework's default 64×64 resolution. The reward is the sum of 3 hand-coded shaping terms (event flags, gym badges, and tile exploration) read from the WRAM features; a precise breakdown is in Appendix J.

Evaluation states. We construct two evaluation sets, both as collections of starting save-states from which a fresh PPO rollout is initiated. *In-distribution* (ID) is a set of 11 starting states drawn from the same 32 saves used for both pretraining and PPO training; it measures performance on the training distribution. *Out-of-distribution* (OOD) is a set of 48 starting states never seen at any training stage, stratified across game progression (pre-Brock, mid-game, late-game, Elite Four). Each OOD state is reached by a human player via legitimate in-game progression and inspected for absence of glitches, soft-locks, and illegal party/item configurations which confound reward signals.

Experimental setup. We compare six conditions: four *frozen-encoder* configurations (JEPA-from-scratch, JEPA-with-DINOv3-init, raw frozen DINOv3, random init), one *end-to-end fine-tuned* JEPA, and *DreamerV3* trained end-to-end as a world-model reference (Section 3, "Downstream RL"). Each condition runs three seeds (42, 43, 44). We roll out for T=2048 steps per evaluation state (Pokemon Red has no terminal state; one step = one button press + 24 emulator ticks) and average across seeds and states. We report (i) mean cumulative shaped reward (Appendix J; higher reward = further story progression in the same step budget) and (ii) unique $(\texttt{map_id}, \texttt{x}, \texttt{y})$ tiles visited (COORDS, exploration coverage). Headline numbers use the OOD set; ID is in Appendix E.

4. Results

4.1. Frozen-encoder family comparison

Table 1 reports OOD reward and exploration on the 48 OOD states defined in Section 3 (a bar-chart visualization is in Appendix H). Frozen JEPA dominates every other frozen-encoder choice by a large margin on both metrics. Notably, fine-tuning the pretrained JEPA encoder during PPO *collapses* OOD performance below the random-init baseline (0.21 vs 0.42 reward); fine-tuning is not a free win.

Table 1. OOD evaluation on 48 starting states, 3 seeds. Reward is mean ± cross-seed std summed over T=2048 steps. Coords is the number of unique $(\texttt{map_id}, \texttt{x}, \texttt{y})$ tiles visited per episode (mean only); per-seed numbers in the appendix.

Encoder	OOD reward	OOD coords
JEPA frozen (DINO-init)	$\mathbf{4.48 \pm 0.15}$	10.6
JEPA frozen (scratch)	1.63 ± 0.37	**23.7**
DINOv3 frozen	0.81 ± 0.36	4.1
Random init	0.42 ± 0.04	5.7
JEPA fine-tune (end-to-end)	0.21 ± 0.00	2.9
DreamerV3 (end-to-end WM)	4.19 ± 0.80	22.3

4.2. DINO-init JEPA matches DreamerV3 on OOD

The most striking comparison in Table 1 is JEPA-with-DINOv3-init (frozen) against DreamerV3. We frame this comparison primarily as a *reproducibility* result: JEPA's cross-seed standard deviation is $\sim \mathbf{5}\times$ smaller than DreamerV3's (0.15 vs 0.80), with all three JEPA seeds tightly clustered in $[4.32, 4.61]$ versus DreamerV3's $[3.48, 5.06]$ (per-seed numbers in the appendix). Reproducibility is a claim about variance and is robust at n=3. Mean OOD reward is comparable in expectation (4.48 vs 4.19, CIs overlap), and we report it as a secondary observation rather than a parity claim, since n=3 is too small to rule out a residual mean difference in either direction. Combined with $\sim 4\times$ cheaper downstream compute (Appendix I), the simpler two-stage pipeline offers a more reproducible and computationally cheaper route to performance comparable in expectation to end-to-end world-model RL.

A trade-off is visible in coverage: DreamerV3 and frozen

JEPA-from-scratch explore $\sim 22-24$ unique tiles on average, whereas DINO-init JEPA explores only 10.6; it exploits more reliably and explores less. We discuss this trade-off in Appendix F.

4.3. Fine-tuning destroys the representation

The collapse of fine-tuned JEPA on OOD reward (Table 1) is mirrored at the representation level. We train linear probes on features extracted by three encoders. The first is the original JEPA pretraining checkpoint. The second is the encoder embedded in the *frozen* PPO agent (which should be bit-identical to the first, as a sanity check). The third is the encoder embedded in the *fine-tuned* PPO agent. All three encoders are evaluated on a held-out probe slice using accuracy for categorical targets and MSE for regression targets (Table 2; Figure 1 in the appendix shows the same data as preservation ratios).

Table 2. Linear probe accuracy / MSE before and after PPO. Frozen PPO preserves the JEPA features exactly (sanity check). Fine-tuned PPO drives event-MSE up $3.7\times$ and map accuracy down 5.7 pp.

Probe target	JEPA	Frozen PPO	FT PPO
map_id (acc)	0.913	0.919	**0.856**
battle_state (acc)	0.996	0.996	0.990
badge_count (acc)	1.000	1.000	0.994
event_count (MSE ↓)	24.3	24.3	**89.3**
party_hp (MSE ↓)	0.041	0.041	**0.069**

The frozen-PPO encoder is bit-identical to the JEPA pretraining checkpoint, and its probe accuracies match the JEPA pretraining checkpoint exactly, confirming that freezing is implemented correctly. The fine-tuned encoder, in contrast, loses 5.7 percentage points on map identification, 68% relative MSE on party-HP regression, and an order of magnitude on event progression. We interpret this as catastrophic interference between the dense SSL gradient that shaped the encoder and the sparse, advantage-weighted PPO gradient that overwrites it. Notably, the downstream collapse is itself remarkably consistent across seeds (cross-seed standard deviation of 0.002 on OOD reward), suggesting the failure is *structural* rather than a high-variance seed effect. A caveat: probes capture the encoder, so we cannot fully rule out that the policy/value heads contribute to the collapse; partial-freeze ablations are left to follow-up work.

4.4. Data diversity beats data expertise

We pretrain three additional JEPA encoders, varying only the *trajectory data source*, and evaluate them downstream under the same frozen-PPO protocol (Table 3). Both data sources start from the same 32 save states; they differ only in the policy used to roll out trajectories. The *Multistate* source uses the biased-random policy of Section 3; the *Puffer* source uses a publicly-released PufferLib-trained policy rolled out from the same 32 saves (see Appendix D for provenance).

Table 3. Data ablation. JEPA pretrained on biased-random (multistate) trajectories outperforms JEPA pretrained on expert (PufferLib-trained) trajectories on the OOD set, even though both are 5M frames from the same starting states.

JEPA pretraining data	OOD reward	OOD coords
Multistate-only (random)	**2.48**	**18.1**
Multistate + Puffer	1.63	23.7
Puffer-only (expert)	1.82	7.8

The *worst* pretraining source is the trained expert policy alone; adding it to the diverse-random data *degrades* representation quality. We trace this effect to behavioral concentration of the trained policy: from canonical starting states the expert policy spends 50K frames visiting only ~ 90 unique tiles, never enters combat, and concentrates 60–75% of its actions on movement (vs the much more uniform action distribution of the random behavior policy that produced the multistate data; full numbers in Appendix C). We interpret this as a general pattern for SSL-for-RL: *diverse* state-action coverage drives representation quality; reward-optimal trajectories are not a substitute.

We note that a confound applies to the *Puffer-only* row: the checkpoint we used to roll out trajectories carries no environment configuration metadata, and we could not reproduce its expected observation byte layout in any of five public PufferLib-family codebases. Whether the degradation we observe is intrinsic to expert-policy data or amplified by environment-format mismatch is not fully separable from this evidence; we discuss this as a methodological note in Appendix D, and we argue there that the qualitative claim (diverse-random $>$ trained-expert + diverse-random $>$ trained-expert alone) is robust to the confound. A breakdown of the in-distribution vs out-of-distribution generalization gap for each encoder is reported in Appendix E.

5. Conclusion

Multimodal *frozen* JEPA encoders dominate every other frozen-encoder choice and end-to-end fine-tuning on OOD starting states by 2–8$\times$. Initializing JEPA from DINOv3 yields OOD reward comparable in mean to end-to-end DreamerV3 with $\sim 5\times$ lower seed variance, a reproducibility result under a substantially simpler two-stage pipeline (frozen encoder + PPO). Probe analysis explains the fine-tuning failure mode, and a data ablation shows behavioral diversity beats reward optimality for SSL pretraining data. The practitioner takeaways are direct: freeze the encoder downstream, combine natural-image SSL init with domain SSL, prefer diverse over expert offline data, and version observation dtype and env constants with any released RL checkpoint. Limitations and follow-up directions are discussed in Appendix F.

Impact Statement

This paper studies self-supervised pretraining for reinforcement learning on a single, well-studied benchmark (Pokemon Red); the contribution is methodological and does not raise application-specific ethical concerns. We highlight two practical effects we anticipate as positive: first, the reduction in downstream compute we report ($\sim 4\times$ relative to end-to-end world-model RL, with the one-shot SSL pretraining cost amortized across multiple downstream tasks) contributes to lowering the energy footprint of long-horizon RL research; second, the 240-state human-validated benchmark we release improves reproducibility in a sub-area where evaluation protocols are currently ad hoc and cross-paper comparisons are difficult. We also note one neutral but worth-stating point: as with most offline-data-based methods, the quality and biases of the offline dataset propagate into the learned representation (our F4 result is a concrete instance of this), and practitioners applying similar pipelines should audit the data-collection policy as carefully as the architecture.

References

Assran, M., Duval, Q., Misra, I., Bojanowski, P., Vincent, P., Rabbat, M., LeCun, Y., and Ballas, N. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Bardes, A., Garrido, Q., Ponce, J., Chen, X., Rabbat, M., LeCun, Y., Assran, M., and Ballas, N. Revisiting feature prediction for learning visual representations from video. *arXiv preprint arXiv:2404.08471*, 2024.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

community, T. p. pret/pokered: A disassembly of Pokémon Red/Blue. GitHub repository, https://github.com/pret/pokered, 2024. Accessed 2026.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Kirk, R., Zhang, A., Grefenstette, E., and Rocktäschel, T. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76: 201–264, 2023.

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations (ICLR)*, 2022.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

LeCun, Y. A path towards autonomous machine intelligence, 2022. OpenReview preprint, https://openreview.net/forum?id=BZ5a1r-kVsf.

Maes, L., Le Lidec, Q., Scieur, D., LeCun, Y., and Balestriero, R. LeWorldModel: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026.

Majumdar, A., Yadav, K., Arnaud, S., Ma, Y. J., Chen, C., Silwal, S., Jain, A., Berges, V.-P., Wu, T., Vakil, J., Abbeel, P., Malik, J., Batra, D., Lin, Y., Maksymets, O., Rajeswaran, A., and Meier, F. Where are we in the search for an artificial visual cortex for embodied intelligence? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Nair, S., Rajeswaran, A., Kumar, V., Finn, C., and Gupta, A. R3M: A universal visual representation for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2022.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Howes, R., Huang, P.-Y., Xu, H., Sharma, V., Li, S.-W., Galuba, W., Rabbat, M., Assran, M., Ballas, N., Synnaeve, G., Misra, I., Jégou, H., Mairal, J., Labatut, P., Joulin, A., and Bojanowski, P. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research (TMLR)*, 2023.

Pomerleau, D. A. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Siméoni, O., Vo, H. V., Seitzer, M., Baldassarre, F., Oquab, M., Jose, C., Khalidov, V., Szafraniec, M., Yi, S., Ramamonjisoa, M., Massa, F., Haziza, D., Wehrstedt, L., Wang, J., Darcet, T., Moutakanni, T., Sentana, L., Roberts, C., Vedaldi, A., Tolan, J., Brandt, J., Couprie, C., Mairal, J., Jégou, H., Labatut, P., and Bojanowski, P. DINOv3. *arXiv preprint arXiv:2508.10104*, 2025.

Suárez, J. PufferLib: Making reinforcement learning libraries and environments play nice, 2024. GitHub repository, https://github.com/PufferAI/PufferLib.

Terver, B., Balestriero, R., Dervishi, M., Fan, D., Garrido, Q., Nagarajan, T., Sinha, K., Zhang, W., Rabbat, M., LeCun, Y., and Bar, A. A lightweight library for energy-based joint-embedding predictive architectures. *arXiv preprint arXiv:2602.03604*, 2026.

Whiddy, P. Pokemon red experiments, 2024. GitHub repository, https://github.com/PWhiddy/PokemonRedExperiments.

A. Composition of the RAM feature vector

Table 4. Composition of the 423-dim RAM feature vector r_t. "Slice" is the chunk of the vector; "Source addr" is the WRAM base from the disassembled ROM (community, 2024); "Norm." is the normalization scheme applied before feeding the MLP. All slices are concatenated in the order shown. The 8-dim badges slice is the gym-badge bitmask read from a single WRAM byte; the 320-dim event-flag slice is the contiguous 40-byte event-progression bitset (`0xD747` onwards), each bit encoding whether a specific scripted event has been triggered (Oak's gift received, Mt. Moon fossil chosen, Snorlax woken, S.S. Anne departed, etc.).

Slice	Dim	Source addr	Norm.
party.count	1	`wPartyCount`	$/6$
party.species (6 slots)	6	`wPartyMons[i].species`	$/190$
party.hp_ratio (6 slots)	6	`wPartyMons[i].(cur,max)_hp`	cur$/$ max
party.levels (6 slots)	6	`wPartyMons[i].level`	$/100$
party.status (6 slots)	6	`wPartyMons[i].status`	bitfield $\rightarrow [0,1]$
party.moves (6 slots $\times$ 4)	24	`wPartyMons[i].moves[k]`	$/165$
position.x, y	2	`wXCoord`, `wYCoord`	$/255$
position.map_id	1	`wCurMap`	$/247$
position.direction	1	`wPlayerDir`	code$/12$, codes $\in \{0,4,8,12\}$
badges	8	`wObtainedBadges`	per-bit $\{0,1\}$
battle.in_battle	1	`wIsInBattle`	$/2$
money	1	`wPlayerMoney` (BCD)	$/999{,}999$
event_flags	320	`wEventFlags` (40 bytes)	per-bit $\{0,1\}$
bag.item_ids (20 slots)	20	`wBagItems[i].id`	$/255$
bag.quantities (20 slots)	20	`wBagItems[i].qty`	$/99$
total	**423**		

B. Per-seed numbers

Table 5. Held-out OOD reward per seed (3 seeds per row). Per-seed values are rounded to two decimals for display; the mean and cross-seed std reported in Table 1 are computed from the unrounded values, which can lead to small visual mismatches between tables (e.g. $\mathrm{mean}(0.20, 0.20, 0.21) = 0.21$ vs 0.20 depending on the underlying precision).

Encoder	seed 42	seed 43	seed 44
JEPA frozen (DINO-init)	4.50	4.32	4.61
JEPA frozen (scratch)	1.61	2.00	1.27
DINOv3 frozen	0.91	0.41	1.12
Random init	0.47	0.40	0.39
JEPA fine-tune	0.20	0.20	0.21
DreamerV3	4.03	3.48	5.06

C. Behavioral diagnostic from canonical states

From a single canonical starting state (`init.state`, the start of the game) over 50,000 frames we record the action distribution and exploration metrics of the biased-random behavior policy and the PufferLib-trained policy:

The same pattern (no battles, single-direction action concentration, zero menu interaction) reproduces from a second canonical state (`Bulbasaur.state`), ruling out state-dependent collapse and suggesting an environment-format mismatch (see Appendix D).

D. Methodological note: observation-format provenance for released RL checkpoints

The *Puffer-only* data source in Section 4.4 is collected by rolling out a publicly-released PufferLib checkpoint (`2spt2yun`, HuggingFace Hub) in our environment. Establishing that the rollout is faithful to the policy's training distribution turned out to be non-trivial, and we report the resulting protocol because we believe it generalizes beyond this paper.

The checkpoint expects an observation of shape $(128, 40, 4) = 20{,}480$ bytes. None of the five public PufferLib-family codebases we surveyed (the `pokemonred-puffer` fork, both v1 and v2 environments of Whiddy (2024), the wrapped variant via Suárez (2024), and two adjacent forks) exposes an observation byte layout that matches this shape; the closest reconstruction misses the target by 1,664 bytes. The released checkpoint carries neither its training environment configuration nor its byte-level observation dtype.

Table 6. Behavioral metrics from `init.state`, 50K frames.

Metric	Random	Puffer
unique maps	5	2
unique (map, x, y)	324	91
battle frames	61	0
battle entries	1	0
cumulative reward	53.91	4.82
left action share	15%	**63%**
A-button share	25%	6%
start-button share	5%	**0%**

We therefore validate the rollout behaviorally rather than byte-exactly. From two canonical starting states (`init.state`, `Bulbasaur.state`) we observe that, in our environment, the policy spends 50K frames visiting ~ 90 unique tiles, never enters combat, and concentrates 60–75% of its actions on a single direction (Appendix C). This is not the behavior expected of a trained Pokemon Red policy on a canonical start, and it is consistent across both starts, ruling out a state-dependent failure and strongly suggesting an environment-format mismatch.

Implication for the data ablation. The Puffer-only row in Table 3 should therefore be read as a lower bound on what an expert-policy data source can yield: any environment-format mismatch between the released checkpoint and our environment can only *narrow* the behavioral diversity of the resulting rollouts (a faithful rollout would visit at least as many states as the mismatched one, in expectation). The paper's qualitative claim (diverse-random $\succ$ trained-expert + diverse-random $\succ$ trained-expert alone) is robust to this caveat: even under the optimistic assumption that the Puffer rollouts in our environment underestimate their faithful diversity, the *addition* of these rollouts on top of the multistate set still *degrades* the downstream representation, which cannot be explained by the format mismatch.

Recommendation. For published RL checkpoints we recommend bundling: (i) the exact observation dtype and byte layout, (ii) the environment configuration hash or commit pin, and (iii) at least one canonical rollout sequence (state, action, reward triples) sufficient to verify byte-level reproducibility on a fresh install. Absent these, downstream reuse is forced into behavioral validation, which is weaker and noisier.

E. In-distribution vs out-of-distribution gap

Comparing the in-distribution evaluation (11 training-matched states, 3 seeds) to the OOD set quantifies how each method generalizes from training to unseen starts. Frozen JEPA-from-scratch achieves 7.79 ± 1.65 ID and 1.63 ± 0.37 OOD ($4.78\times$ drop); DreamerV3 achieves 8.58 ± 2.32 ID and 4.19 ± 0.80 OOD ($2.05\times$ drop). The ID parity is therefore masked by an OOD generalization gap that the JEPA scratch encoder does *not* fully close. This observation motivates the DINO-init variant of Section 4.2, whose full ID-OOD behavior we leave to follow-up work since it requires per-method in-distribution re-evaluation.

F. Limitations

Single environment. All experiments are on Pokemon Red. Generality of the four findings to other long-horizon RL environments is plausible but unverified within this short paper.

Resolution asymmetry vs DreamerV3. DreamerV3 is run at its standard 64×64 resolution; our JEPA agents at the native 144×160 Game Boy resolution. We did not re-train DreamerV3 at matched resolution. This asymmetry should, if anything, favor JEPA via richer pixel input, so the mean-OOD parity we report holds despite a handicap to the baseline running in the opposite direction.

Reward shaping. PPO is trained against the three-term shaped reward defined in Section 3 (events, badges, tile exploration). The reward is deliberately sparse on the badge term, at most one $+5.0$ pulse per gym leader cleared, and dense on the exploration term but very small ($+0.01$). DreamerV3 receives the identical scalar reward but also fits a learned latent reward predictor that may exploit pretrained features more effectively than the fixed shaped reward does. Whether a richer reward signal (party HP, level progression, catch counts) would close the OOD-reward gap for frozen JEPA is an open question that we leave to future work.

Three seeds. We report three seeds per condition. The two claims that carry scientific weight in this paper, the F2 reproducibility result for DINO-init JEPA (σ=0.15) and the F3 fine-tuning collapse (σ=0.002 on OOD reward), are unusually consistent across seeds, so additional seeds are unlikely to change their direction. The higher-variance rows (DINOv3-frozen, JEPA-scratch) bound intermediate baselines and are not load-bearing for any headline claim.

G. Probe quality as preservation ratio

The same probe data shown in Table 2 is plotted in Figure 1 below as a preservation ratio relative to the JEPA pretraining checkpoint, making the per-target collapse of fine-tuning visually immediate.

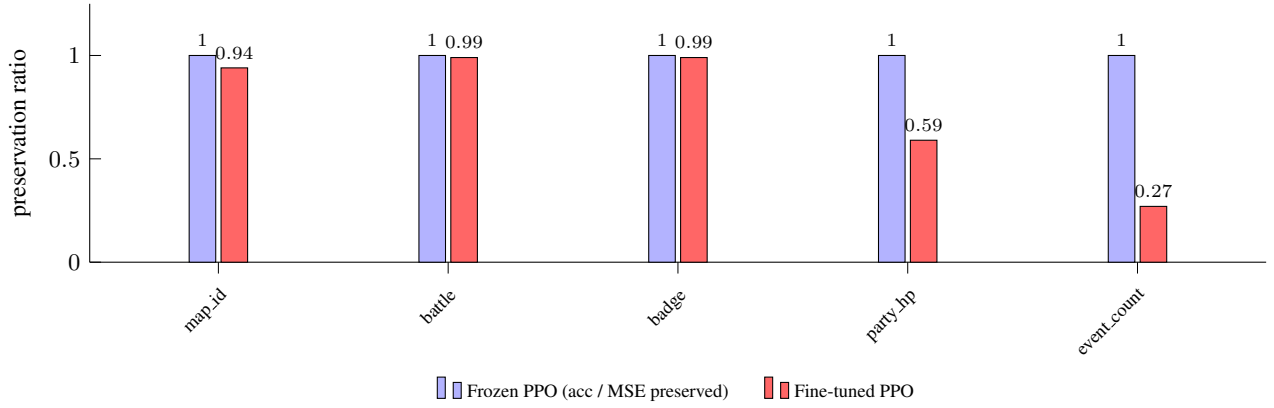

Figure 1. Linear probe quality after PPO, expressed as a preservation ratio relative to the JEPA pretraining checkpoint (FT acc/JEPA acc for accuracy targets; JEPA MSE/FT MSE for regression targets). A value of 1.0 (dashed line) means perfect preservation. Frozen PPO is identical by construction; fine-tuning degrades event-count prediction to 27% and party-HP to 59% of the pretrained encoder's quality.

H. Held-out OOD reward bar chart

Figure 2 below visualizes the OOD reward reported in Table 1, with cross-seed error bars and per-seed values in the caption.

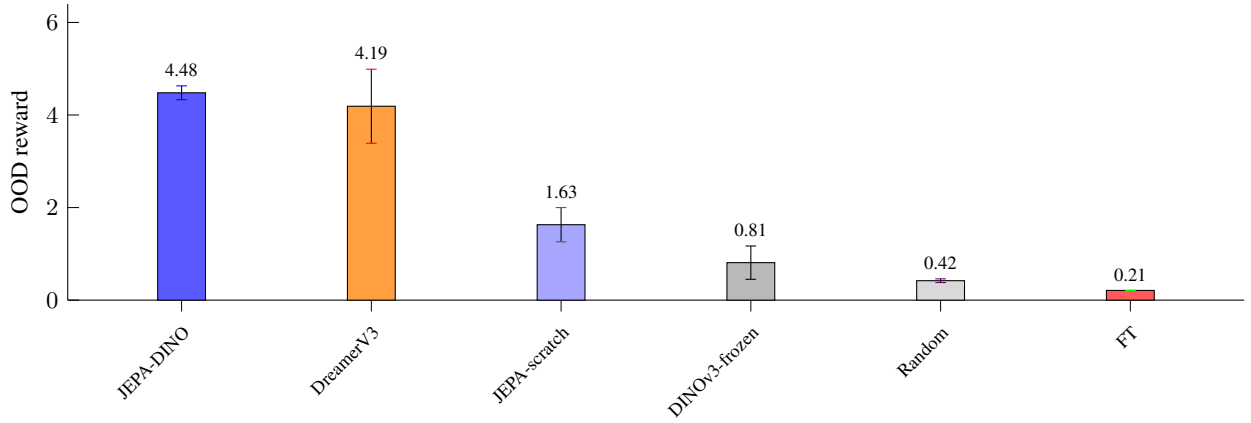

Figure 2. Held-out OOD reward (mean ± cross-seed std, 3 seeds) by encoder. JEPA-DINO seeds: $\{4.50, 4.32, 4.61\}$, range 0.29; DreamerV3 seeds: $\{4.03, 3.48, 5.06\}$, range 1.58. Frozen JEPA with DINOv3 initialization (*JEPA-DINO*) matches end-to-end DreamerV3 in mean while being substantially more reproducible. Fine-tuning the JEPA encoder during PPO (*FT*) collapses below random initialization.

I. Compute budget

JEPA pretraining takes $\sim$ 4h on 8× RTX 5090 ($\sim$ 32 GPU-h, one-shot and amortizable across downstream tasks). PPO downstream takes $\sim$ 6h on a single RTX 5090 (5M env steps); DreamerV3 takes $\sim$ 24h on the same single GPU for the same budget. Frozen-encoder PPO is therefore $\sim 4\times$ cheaper at downstream time than end-to-end DreamerV3, and the SSL pretraining cost is amortized after $\sim$ 2 downstream tasks.

J. Reward shaping breakdown

We use a per-step shaped reward $r_t = r_t^{\text{event}} + r_t^{\text{badge}} + r_t^{\text{coord}}$ summed from three terms, all derived from the WRAM features of Table 4:

- *event flags*: $r_t^{\text{event}} = 4.0 \cdot (\text{events}_t - \text{events}_{t-1})$, where events_t is the popcount of the 320-bit `wEventFlags` bitset. Each newly-set bit corresponds to a scripted progression event (item received, NPC dialog completed, cutscene cleared); the term is positive on every new bit and effectively zero in normal play once a region is exhausted.

- *gym badges*: $r_t^{\text{badge}} = 5.0 \cdot (\text{badges}_t - \text{badges}_{t-1})$, the popcount delta of `wObtainedBadges`. At most $+5.0$ per gym leader cleared.
- *tile exploration*: $r_t^{\text{coord}} = 0.01$ when the triple $(\texttt{wCurMap}, \texttt{wXCoord}, \texttt{wYCoord})$ is visited for the first time in the episode, 0 otherwise.

The episode return is $R = \sum_{t=0}^{T} r_t$ with T=2048; the "OOD reward" column of Table 1 is R averaged across the 48 OOD starting states and 3 random seeds. The same shaped reward is consumed by both PPO and the DreamerV3 baseline.