Anton A. Nesterov · an@vski.sh · July 19, 2026

Active Perception I: A GRU DRQN for Discovery Under a Noisy Sensor

Reinforcement Learning · Deep Learning · POMDPs

The moment the value of the next action depends on what was just done, you need recurrence. That is the entire argument for a GRU Deep Recurrent Q-Network: a learned memory of the recent trajectory, trained with the standard DQN stabilisers.

Assets & Materials

Active Perception — Interactive simulation. A trained GRU DRQN collects value-graded gold on a 20×20 grid with a noisy, trajectory-coupled sensor.	https://blog.vski.ai/active-perception/
Source code and training setup for the Active Perception demo.	https://vski.sh/x/ap-demo/

A linear recommender can score *classical music* and *Eminem* both highly, and on the strength of those scores play them back to back. Both scores are correct in isolation. The sequence is wrong. The listener who likes Beethoven at 8 a.m. and Eminem at the gym does not want them in the same playlist, and the model that scored them independently has no way to know that — its value function sees one action at a time.

The moment the value of the next action depends on what was just done, independence stops being a useful approximation and starts being a bug. The fix is to let the value function *remember* what it just did. That is the entire argument for recurrence in a value-based reinforcement-learning agent.

When linear is enough, and when it is not

Three families of learned policy cover most real deployments, and the choice between them is structural.

Linear bandits (UCB, LinUCB, Thompson sampling) learn a linear trend *online* from very few samples. The right tool when each decision is cheap, the world is roughly stationary, and the value of action a in context x is well modelled by a single linear score $w \cdot \phi(x, a)$. Display ads, A/B-nested arms, short-horizon ranking.

Stateless linear planners extend the linear idea to multi-step problems where the agent has to *plan* but does not need to *remember* across many ticks. Convex, closed-form, non-divergent. The right shape for low-cost exploration: routing under demand clusters, coverage planning on a uniform field, session recommendation where each item is scored on its own merits.

Deep Q-Networks. When the value of the next action depends on the recent trajectory — satiation, fatigue, what was just sensed — the linear score is wrong in a structured way. The agent needs a function that represents *interactions between consecutive steps*, which means a nonlinear function approximator, which brings the deadly triad back (function approximation + bootstrapping + off-policy data), which means training has to happen off-device in a controlled environment. You do not train a DQN live in a robot.

The throughline is the **action-context interaction**. Linear is right when $Q(x, a)$ factors through x and a independently; recurrence is right when the value of a_t depends on the recent observation sequence $o_{t-k:t}$. The grid world this essay is about sits in the second regime: the value of moving east right now depends on whether the agent just swept a cluster to the west, whether its last eight readings agreed, whether it is facing a direction where its sensor sees five cells ahead or one.

Discovery with a noisy sensor

The setup is a partially observable MDP with three defining constraints.

The sensor is anisotropic and trajectory-coupled. This is the defining property. The agent sees only a small footprint centred on its current cell, and the shape of the footprint depends on the last move. Move east, and the sensor reaches roughly five cells into the eastern half-plane and one cell backwards; move north, and the same asymmetric fan rotates. The agent sees further along where it is going than to the sides. To look at a distant cell it has to *face* it, which means *commit* to a direction, which means *move*. Seeing is moving.

The sensor is noisy. Each reading is correct with probability $p = 0.85$; otherwise a uniformly-random wrong label. A single reading is weak evidence; the agent must combine many of them over many ticks to form a confident opinion about any one cell.

The agent is stateless across episodes. Within one episode it maintains a Bayesian *belief map* (per-cell posteriors over gold / stone / unknown, decaying with recency). That belief is its only memory inside an episode. Across episodes the belief is wiped; the agent starts over from a flat prior. What persists is the trained Q-function, and nothing else. The agent has to *generalise* across worlds it has never seen.

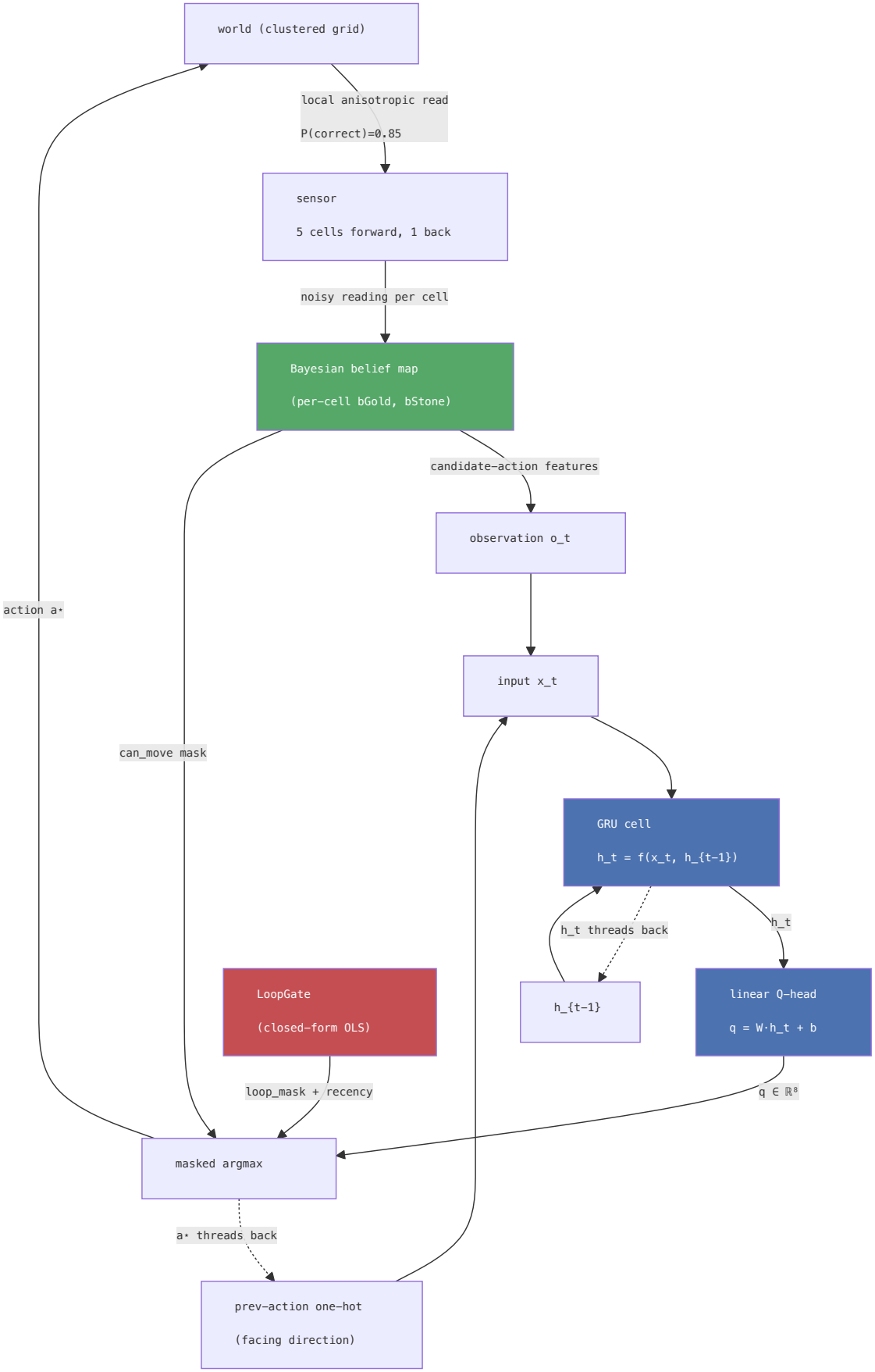
Two shortcuts are off the table by construction:

1. **Adding a sensor that detects clusters is cheating.** A cluster detector is the ground-truth side-channel the agent is supposed to be inferring from the sensor stream. Bolting one on turns the problem into supervised learning over a global view, which is a different problem.
2. **Adding a deterministic strategy is not ML.** A hand-coded rule that says "if you see three gold readings in a row, take a step in the same direction" *works* — and proves nothing. The question is whether the value function can *represent* that dependence, not whether a human can hand-write it.

This is the regime that defeats a stateless linear planner. The value of moving east right now depends on whether the last three eastward readings agreed, whether the belief at the destination is high-confident or speculative, whether the agent has just been sweeping a cluster and should *keep sweeping* rather than turn. All of that is sequential structure. None of it is in the current observation. It is in the recent observation *sequence* — which is what recurrence is for.

The architecture

The agent is built around one question: *what does the value function need to see, and what does it need to remember?* Everything else is plumbing.



Four subsystems. The divisions are load-bearing.

The belief map (green) is the agent's statistical summary of the world it has seen so far. Per-cell posteriors over gold and stone, updated bilaterally from each noisy

reading (a gold reading pulls `bGold` up and `bStone` down by symmetry), decaying with recency. The belief map is *not* the truth — it is the agent's current best statistical opinion, and it can be wrong. It is the legitimate replacement for the global view the agent is not allowed to have.

The recurrent core (blue) is the only learned model on the inference path. A Gated Recurrent Unit whose hidden state h_t threads across ticks, followed by a linear Q-head that projects h_t to one Q-value per action. The hidden state is the agent's *learned* memory of the recent trajectory; it is what lets the Q-function tell whether a high-belief reading just now is a fresh discovery or the seventh in a row.

The masked argmax (white) is where the Q-values meet the world. The raw argmax of q is constrained by a `can_move` mask (structural physics: in-bounds, not certainly stone). The gate adds a second mask on top when it fires. The result is an action that is both *learned* (the Q-function chose it) and *safe* (the masks ruled out the physically impossible and the currently pathological).

The LoopGate (red) is the one piece of *non-learned* logic on the inference path, and it deserves its own colour because it does something the trained network cannot. It diagnoses an oscillation basin from the agent's own trajectory — closed-form, no learned weights — and masks the directions that are looping. The trained GRU handles discovery; the gate handles the one failure mode that survives training.

Formal details

The Q-function and its update

The action-value function is

$$Q_\theta(s_t, a_t) \;=\; W_{hq}\, h_t + b_q, \qquad h_t \;=\; \mathrm{GRU}(x_t, h_{t-1};\, \theta_{\mathrm{gru}}).$$

The input vector x_t is the concatenation of two things the value function cannot do without:

- **The previous action.** Because the sensor footprint is trajectory-coupled, the agent has to know which direction it is currently facing to interpret the current reading. The previous action *is* the facing direction.
- **Candidate-action features.** For each candidate move, the belief values at the destination cell (`bGold`, `bStone`), a structural in-bounds flag, and a constant bias. The same features a linear planner would need — they are the legitimate per-action summary of the belief map. The recurrence does not replace them; it sits on top.

The training target is the **Double DQN** Bellman update:

$$y_t \;=\; r_{t+1} \;+\; \gamma \cdot Q_{\theta^-}\Big(s_{t+1},\; \arg\max_{a'} Q_\theta(s_{t+1}, a')\Big)\,,$$

with θ the live network and θ^- a frozen target network. The action is *picked* by the live network and *evaluated* by the target network. Decoupling selection from evaluation removes the systematic overestimation bias of vanilla DQN — the bias that lets a single optimistically-misestimated state propagate its bad value across the policy. The loss is the squared TD error. This is the standard DQN recipe; it is the price of admission for any nonlinear value function trained off-policy.

The GRU forward pass

The cell is the standard **Gated Recurrent Unit**:

$$\begin{aligned}
z_t &= \sigma(W_{xz}\, x_t + W_{hz}\, h_{t-1} + b_z) && \text{(update gate)}\\
r_t &= \sigma(W_{xr}\, x_t + W_{hr}\, h_{t-1} + b_r) && \text{(reset gate)}\\
\tilde{h}_t &= \tanh(W_{xh}\, x_t + W_{hh}\,(r_t \odot h_{t-1}) + b_h) && \text{(candidate)}\\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t && \text{(state)}
\end{aligned}$$

The update gate z_t interpolates between the previous state and the candidate; the reset gate r_t controls how much of the previous state is allowed to influence the candidate. The hidden state h_t is what the linear Q-head reads. The whole point of the cell is the interpolation on the last line: the state is a *learned* weighted average of

"what I had" and "what I just saw", with the weights themselves learned from the TD signal.

Why GRU and not LSTM. Train the three cells side by side — Elman, GRU, LSTM — on identical TD/BPTT plumbing and the result is the textbook one. LSTM encodes cluster structure most sharply: a linear probe on h_t reaches the highest AUC for "is the agent in a cluster?", and the AUC grows monotonically Elman → GRU → LSTM. But the richer the cell the more the deadly triad bites — a more expressive value function is more prone to divergence under bootstrapping, not less — and the policy gap between the three is small once stabilisers are in place. GRU is the right compromise: nine tensors instead of thirteen, identical policy quality once replay + target + clipping are on. LSTM would be the choice if representation quality were the binding constraint; here, stabilisation is.

BPTT with truncation, and episode replay

Training a recurrent value function is not standard supervised learning. The gradient of the TD loss at step t flows back through $h_{t-1}, h_{t-2}, \ldots$, accumulating into the recurrent tensors over the whole episode. Unrolling that to the start of the episode is expensive and unstable; truncating it to the last K steps is **truncated backpropagation through time**, the DRQN training recipe. The hidden state at the truncation boundary is detached — gradients do not flow past it. Shorter truncation is more biased (the network cannot learn dependencies longer than K); longer truncation is more expensive and more prone to exploding gradients. Gradient clipping on the total norm is the third of the three standard DQN stabilisers, non-negotiable with a recurrent value function.

The replay buffer stores **full episodes**, not one-step transitions. This is structural for DRQN: to backprop through K steps you need K consecutive observations, which a transition buffer does not have.

Potential-based reward shaping

The bare task reward — gold collected minus stone cost minus bump penalty — is sparse. On a clustered map the gold is a small target and the agent has to discover it before any learning signal appears. The fix is a **potential-based shaping bonus** applied at every step:

$$r'_t \;=\; r_t \;+\; \beta \cdot \left(\gamma\, \Phi(s_{t+1}) - \Phi(s_t)\right), \qquad \Phi(s) \;=\; \text{coverage}(s),$$

with $\Phi(s)$ the fraction of cells the belief map is confident about. The bonus rewards the agent for *expanding its certain knowledge of the world* — exactly the active-perception sub-goal.

The reason this is not cheating is the theorem: **potential-based shaping is policy-invariant** (Ng, Harada & Russell 1999). Adding a bonus of the form $\gamma\Phi(s') - \Phi(s)$ at every step changes the optimal value function by a known additive constant

(Φ at the start state) but **does not change the optimal policy**. The shaping bonus makes learning faster; it does not move the optimum.

The temptation to use a non-potential bonus — "free reward for entering unknown cells", "small bonus for any new reading" — should be resisted. Those formulations are not policy-invariant; they create a degenerate optimum where the agent oscillates along the belief frontier forever, racking up shaping bonus without ever collecting gold. The potential-based form is the only one that ships safely.

Why production learned policies carry masks

Current design is simplified and the trained GRU has one visible failure mode I decided to use to demonstrate *the masks*.

After enough episodes, the agent settles into a four-by-four or four-by-six box, bouncing off a wall, a stone, or a *believed* stone — a cell the shadow map wrongly flagged as stone on a noisy reading and never re-queried — for twenty, forty, sometimes seventy ticks. Telemetry across thirty baseline episodes shows **about 29% of all ticks are spent inside such an oscillation basin**.

This is the failure mode the trained network does not fix on its own. Retrain longer, reshape the reward, swap GRU for LSTM — the basin survives all of it. The belief map still has a wrong cell flagged as stone; the Q-function still scores the looping directions as marginally best because the alternatives look worse under the wrong belief; the agent still oscillates.

The fix is not to retrain. The fix is to mask.

Why masks, not retraining

Two arguments, both of which matter for production.

The engineering argument. A deployed learned policy is a frozen artifact. It was trained off-device, validated against an acceptance bar, shipped. Retraining it inside the deployment is the deadly triad in the most dangerous possible place: an online loop on real inputs, no held-out validation, a policy that is allowed to act before it has converged. A divergent value function on a recommender shows a user a bad item; a divergent value function on a robot drives it into a wall. The responsible default for any learned policy that controls hardware is *frozen at deploy time, masks layered on top*. The mask is a tiny piece of classical logic — closed-form, no learned weights, no divergence risk — that catches failure modes the trained policy still has, without touching the policy itself.

The mathematical argument. The failure mode is *localised*: the trained policy is correct on most states and wrong on a specific class. For localised failure modes, the cheapest fix is a per-state action mask — detect the bad class, forbid the action that is wrong there — not a retrained policy. Retraining risks regressions on states the policy was already handling correctly; a mask cannot, by construction, change behaviour outside the states where it fires.

The same argument is the case for action masks in safety-critical RL generally. A robot's collision-avoidance layer is a mask (forbid trajectories that intersect obstacles). A recommender's content-policy filter is a mask (forbid actions that serve disallowed items). A trading system's risk limit is a mask (forbid actions that breach position limits). All of them are classical logic layered on top of a learned policy, and all of them share the same justification: *the learned policy is responsi-*

ble for quality; the mask is responsible for safety; conflating them is how learned systems fail dangerously.

The mask recipe

The mask on this agent has three parts, in this order. Each is a principle, not a tuning knob.

Detector (closed-form, regime-faithful). Keep a rolling ten-tick window of the agent's own trajectory $(r_t, c_t, \text{reward}_t)$. Fit two simple linear regressions — row against tick, column against tick — by closed-form one-dimensional OLS:

$$\sigma_r \;=\; \frac{\operatorname{cov}(r,\, t)}{\operatorname{var}(t)}, \qquad \text{speed} \;=\; \|(\sigma_r,\, \sigma_c)\|.$$

The speed is the agent's net displacement per tick over the window. Combine with the gold collected over the window:

$$\text{isLoop} \;=\; (\text{speed} < \tau_s) \;\wedge\; (\text{gold}_W < \tau_g).$$

The gold term is the discriminator: a slow drift through a cluster the agent is actively sweeping is *not* a loop — gold is being collected. A slow drift through a basin the agent already emptied *is* a loop. The two look identical from speed alone; the gold term separates them. The detector consumes only the agent's own trajectory — no truth queries, no oracle. Closed-form OLS, no learned weights, no divergence.

Persistent hard mask. When the detector fires, the directions that keep appearing in the window go on the mask. The mask **stays on until the agent has demonstrably escaped** — several consecutive ticks of recovered speed, or any gold event. The persistence is load-bearing: a fixed cooldown (mask for k ticks, then drop) lets the agent escape one basin, the mask expires, and the wrong belief that pulled it into the basin in the first place pulls it straight back. Persistence breaks that cycle. The mask drops only on evidence of escape, not on a timer.

Recency tie-break, not argmax-of-rest. While the mask is active, action selection is *not* `argmax` over the unmasked actions. It is **least-recently-used** among unmasked destinations: pick the action whose destination cell was visited least recently. This is the single decision that separates a mask that works from a mask that does not. The naive choice — `argmax` over unmasked Q-values — fails because the Q-values of the opposite directions were learned against the same wrong belief that caused the loop. Masking the looping direction just makes the agent pick the opposite one and oscillate in a new pattern. The recency tie-break cannot oscillate by construction: a two-cycle visits two cells; the recency tie-break sends the agent to a third.

The general recipe

Anywhere a deployed learned policy has a localised failure mode — wrong on a specific class of states, right elsewhere:

1. **Diagnose.** Measure the failure mode on telemetry. Get a quantitative rate (here: 29% of ticks in basins).
2. **Detect.** Find a closed-form signal that separates failure states from sensible-exploration states. Both look superficially similar; the discriminator is the load-bearing feature.
3. **Mask.** Forbid the action that causes the failure, persistently, until you have evidence the agent has escaped the failure state.
4. **Re-pick.** Among unmasked actions, pick the one the policy has *least recently* chosen — never `argmax` over the same wrong Q-values that caused the failure.

The whole mask is classical logic, closed-form, no learned weights. It cannot diverge. It cannot overfit. It cannot silently regress on states the trained policy was handling well, because it does not touch the trained policy — it forbids actions on a state class the policy was demonstrably mishandling.

Conclusion

The narrow claim: a value function that scores each action independently is the right tool when the value of the next action does not depend on what was just done, and the wrong tool when it does. The classical-then-Eminem example is the whole argument in two items.

In this regime the responsible deployment pattern is *train off-device with the standard DQN stabilisers, freeze at deploy time, layer classical masks on top for the failure modes the trained policy still has*. The learned value function handles discovery; the mask handles the one pathology the value function cannot fix on its own.

The architecture that follows from those two claims is small enough to state plainly. **One recurrent value function** — GRU cell, linear Q-head, trained with Double DQN targets, episode replay, target-network sync, gradient clipping, truncated BPTT.

The pattern generalises. The grid is the cleanest place to see it work, but the pattern is not about the grid.

References

1. **Hausknecht, M. & Stone, P.** *Deep Recurrent Q-Learning for Partially Observable MDPs*. (arXiv .06527, 2015) — the DRQN paper. The case for adding recurrence to DQN in partially observable environments, the truncated-BPTT training recipe, and the empirical demonstration that recurrence recovers performance lost to partial observability.
2. **Mnih, V. et al.** *Human-level control through deep reinforcement learning*. (Nature 518, 2015) — the DQN paper. The three stabilisers — function approximation, target network, experience replay — that the deadly triad makes structurally necessary.
3. **van Hasselt, H., Guez, A. & Silver, D.** *Deep Reinforcement Learning with Double Q-Learning*. (AAAI 2016) — Double DQN. Decouples action selection from action evaluation to remove the overestimation bias of vanilla DQN.
4. **Cho, K. et al.** *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. (EMNLP 2014) — the GRU cell.
5. **Hochreiter, S. & Schmidhuber, J.** *Long Short-Term Memory*. (Neural Computation 1997) — the LSTM cell; the comparison point for "richer cell, sharper representation, sharper deadly triad."
6. **Sutton, R. S. & Barto, A. G.** *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018 — §11.3, the deadly triad. Full PDF.
7. **Ng, A. Y., Harada, D. & Russell, S.** *Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping*. (ICML 1999) — the theorem that potential-based shaping is policy-invariant.
8. **Spaan, M. T. J. & de Vries, T. J.** *Perception and Action in POMDPs*. (AAAI 2008) — the formal framing of active perception as a POMDP in which sensing is itself an action choice.
9. **Executable companion.** Active Perception demo — the trained GRU DRQN next to a random-policy comparator and the closed-form action mask.