Jörmungandr: A Plugin Runtime for Distributed Reinforcement Learning

research@strategynet.ai

August 2026

Abstract

A reinforcement-learning experiment combines an environment, data-collection policy, replay rule, learning objective, model representation, validation protocol, and artifact lifecycle. Treating one algorithm as the system makes those responsibilities difficult to compare or replace. Jörmungandr separates external actors from one versioned learner and resolves both the learning algorithm and rollout selector through plugins. The implemented fixed-discrete-action service supports C51, quantile-regression DQN, DQN, discrete conservative Q-learning, maximum-entropy soft Q-learning, behavior cloning, MARWIL, PPO, IMPALA, an IMPACT-style APPO profile, categorical SAC, and a compact vector-observation DreamerV3 profile. Training and validation stores remain isolated; legal masks, behavior probabilities, policy versions, checkpoints, TorchScript artifacts, TensorBoard series, and internal comparison metrics share one lifecycle. A second service profile transports variable-size typed entity sets and state-local semantic action candidates through central inference, prioritized transition replay, or policy-lag-aware joint PPO trajectories, plus reward-free weighted supervision over semantic factor choices and opt-in importance resampling of application-declared balance strata. Structured behavior-cloning checkpoints initialize the compatible PPO transformer directly while keeping new optimizer and rollout state. Each joint record carries one reward per environment turn, nested factor choices, and the exact sum of the conditional behavior log probabilities. Structured PPO reports episode-return spread, range, unique count, episode-length range, reward sparsity, and the direct oldest-residual GAE weight before optimization, which prevents a critic fitting one common outcome—or a terminal residual with negligible reach at early decisions—from being mistaken for a policy receiving useful credit. Version 1.7 additionally separates value-head learning from critic gradients through a pretrained shared policy backbone and can transactionally back off a PPO proposal until every selected behavior-action ratio in the complete batch lies inside declared bounds. A QUBO selector makes auditable yes/no decisions over prioritized transitions, contiguous rollout fragments, or a bounded search frontier by trading utility against redundancy. Quantile critics expose mean, lower-quantile, and lower-tail CVaR decisions for noisy outcomes. Volt can use frozen set or graph embeddings through the service today and can train its Torch Deep Sets or typed graph encoder through the in-process masked-policy interface; the current service transport has no graph-native payload. Rust is reserved for profiled replay, selection, collation, and numerical kernels while policy modules remain in Torch. A seeded synthetic comparison evaluates five compatible online transition learners on clean and sensor-noise cohorts; it demonstrates the comparison protocol within this cohort. It supplies no environment-general ranking. An independent three-seed CartPole diagnostic gives the built-in PPO and Stable-Baselines3 PPO equal 9,155-parameter networks and 49,152 interactions; both finish at the maximum held-out return in every run. This paper defines the implemented interfaces and their limits. A representation-parity control then gives the entity/candidate transformer the same CartPole seeds, PPO controls, and interaction budget; all three runs also finish at 500. A four-way masked Taxi control then records zero invalid choices for all masked policies while the otherwise identical unmasked PPO control accumulates 462,511. A generic exact-oracle constrained workbench then gives variable-factor joint PPO 12,288 terminal-only reward turns per run: random-start PPO finishes at 0.875 of oracle versus 0.469 for random legal play, while behavior cloning followed by PPO finishes at 0.923. These bounded diagnostics do not imply application-general performance.

Contents

1 Introduction

As we considered RL frameworks for research and production, we found that many libraries made it difficult to deploy without the overhead: drawing inferences required a specialized runtime. What we wanted was a clear separation between the agents generating experiences, the reinforcement learning algorithm, and most critically the policy. For our purposes, the ability to checkpoint and deploy policy parameters without the overhead of the entire framework is critical for production use and forms a core requirement for Jörmungandr. We also wanted to be able to compare different algorithms and policies without having to change the underlying system. This paper describes the design and implementation of a plugin-based distributed reinforcement learning system that meets these requirements.

Distributed reinforcement learning has two kinds of scale. More actors can produce more transitions, and a richer research programme can compare more objectives, policy models, replay rules, and risk criteria. The first kind is usually visible in the worker topology. The second is often hidden in a trainer whose storage, network, loss, checkpoint, and inference code all name one algorithm.

The original Jörmungandr service had this second shape. It provided a useful C51 path: actors published fixed-vector transitions, a central learner sampled prioritized replay, held-out items entered a separate validation store, and the model could be checkpointed and exported. C51 was nevertheless both an algorithm and an architectural assumption. Adding an on-policy learner would have required behavior probabilities and ordered trajectories; adding an offline learner would have changed the interpretation of replay; adding a world model would have introduced several optimizers and target components.

The present design keeps the service responsibilities and turns the algorithm into a plugin. A model record is authoritative for data identity, preprocessing, learner version, selector, metrics, and artifacts. An algorithm plugin implements networks, losses, optimizers, target updates, action selection, and serializable state. A selector plugin chooses training rows or rollout fragments. The application supplies the environment, reward, legal actions, and train–validation split.

This separation also clarifies robustness. Reward outliers, stochastic returns, observation corruption, Q-value overestimation, offline distribution shift, asynchronous policy lag, and learned-model error are different problems. C51, quantile regression, Huber loss, CQL, V-trace, maximum entropy, and world-model normalization address different subsets. No algorithm name removes the need to declare and test the disturbance.

> *The learner may change how experience is valued, but it must not change what the experience means, which actions were legal, which policy produced it, or whether it was reserved for validation.*

2 System scope

2.1 Actors and one learner authority

An actor runs the environment. It resets an episode, observes state, requests or computes an action, applies the transition, constructs reward, and publishes provenance. The service does not import the domain simulator in the primary distributed path. For model identifier m, one model record stores the replay stores, observation statistics, learner state, policy version, and artifacts.

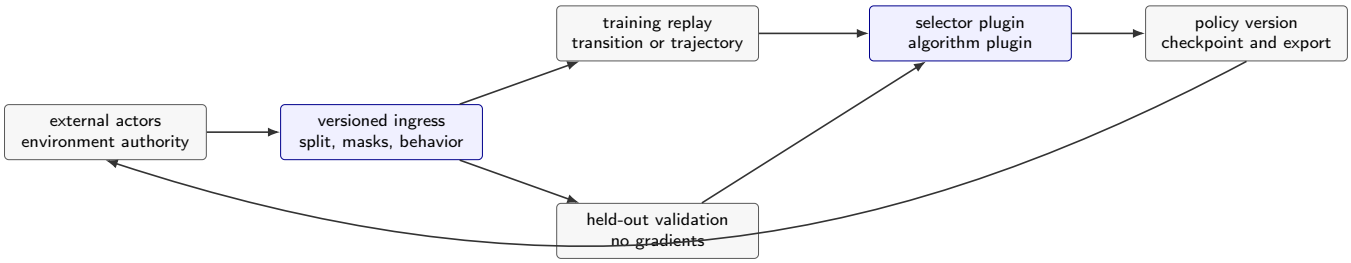

Figure 1: The domain remains outside the learner. Training and validation may arrive interleaved, but routing precedes storage and creates two state-changing stores.

Every public transition declares the split, actor, episode, timestep, and policy version. It contains current and next observations, chosen action, reward, and terminal state. Stochastic policy actors additionally retain the behavior log probability and value estimate. Boolean current and next legal masks align with the model's declared action vocabulary.

The replay buffer stores integer action indices. A domain action value can be supplied when it uniquely maps to one index. This is a bounded action interface: the compiler or application decides which slot means close, wait, resize, hedge, or exercise, and the mask decides whether that slot is admissible in the current state.

2.2 Split isolation

Training transitions update the running observation statistics and enter the training replay. Their priorities can change after an update. Validation transitions enter a separate bounded store. They use the training normalizer, but do not update it, change replay priority, execute optimizer steps, or select a checkpoint implicitly.

The split is assigned before an episode begins. Interleaving train and validation items in one network request changes neither role. For a pathwise financial experiment, assigning individual steps from the same path to both stores would still leak the path even though the service stores were separate; scenario-level assignment therefore remains an actor responsibility.

2.3 Trajectory construction

Trajectory objectives require more than a bag of transitions. The service groups physical replay rows by actor and episode, orders them by timestep, and splits a run at a missing timestep, terminal state, or configured rollout length. PPO, IMPALA, and APPO exclude fragments older than their configured behavior-policy lag; offline MARWIL deliberately retains historical demonstrations. This produces actual contiguous traces for returns, GAE, or V-trace; independently sampled batches are never retroactively sorted into trajectories.

Circular replay still means that an old prefix may have been overwritten. The surviving suffix starts a new fragment. Continuity resumes only from its first retained row.

3 Plugin and artifact model

3.1 Algorithm interface

An algorithm plugin has a semantic name and version, family, replay mode, backend, default export module, description, noise profile, and agent factory. The agent accepts arrays and plain metadata through the service interface and provides action selection, batched inference, training, held-out evaluation, and state serialization. Every update returns

$$U = (\ell, p_1, \ldots, p_B, \mathcal{M}),$$

where ℓ is the update loss, p_i is a replay-priority statistic aligned with sampled row i, and $\mathcal{M}$ is an algorithm-specific map of finite scalar metrics. The runtime need not know whether p_i came from a TD error, a value loss, or another residual.

Built-ins register by importing their modules. External packages use the Python entry-point group jormungandr.algorithms. The selector has an independent entry-point group, jormungandr.rollout_selectors. One broken optional package is isolated from built-ins during discovery.

3.2 Checkpoint lifecycle

The outer checkpoint remains `jormungandr.checkpoint.v1`. Its payload includes the resolved algorithm and plugin version, complete learner configuration, policy and target tensors, optimizer states, observation normalizer, feature order, action values, learner update, policy version, selector identity, and model metadata. Retaining the outer format preserves the original C51 lifecycle while allowing a different state dictionary inside it. New plugin checkpoints refuse silent restore through a different plugin name or version; a state-format change therefore requires an explicit migration or a compatible semantic plugin version.

Replay and validation contents are transient across checkpoints. A restored model resumes parameters, optimizers, versions, and preprocessing but begins with empty stores. This distinction prevents a model checkpoint from being mistaken for a complete distributed-job snapshot.

The exporter resolves the plugin's default Torch module and produces a TorchScript file, JSON manifest, and generated C++ specification. C51 manifests describe action-by-atom logits. QR-DQN manifests describe action-by-quantile values and retain the risk rule. Policy and scalar-Q profiles describe action-width outputs. Torch checkpoints use pickle-based loading and therefore require trusted local storage and handling.

4 Implemented algorithm profiles

The compatibility service uses a fixed discrete action vocabulary. Its profiles and the three implemented entity/candidate counterparts are listed in Table 1; the table does not extend results to unimplemented observation or action spaces.

Plugin	Data mode	Implemented profile
`c51`	off-policy	Categorical distributional Double-DQN on a fixed value support.
`qrdqn`	off-policy	Quantile-Huber Double QR-DQN with mean, lower-quantile, or CVaR action scoring.
`dqn`	off-policy	Double, dueling scalar Q-network with Huber TD loss.
`cql`	offline/off-policy	Discrete CQL(H) log-sum-exp penalty over the DQN objective.
`maxent`	off-policy	Discrete soft-Q Bellman backup and Boltzmann policy.
`bc`	demonstrations	Masked categorical behavior cloning.
`marwil`	offline trajectory	Moving-normalized, clipped exponential advantage weighting.
`ppo`	trajectory	GAE and clipped surrogate with multiple minibatch epochs.
`impala`	trajectory	V-trace actor–critic correction for behavior-policy lag.
`appo`	trajectory	IMPACT-style V-trace, target-policy clipping, and bounded circular replay.
`sac`	off-policy	Categorical SAC with twin critics, soft targets, and automatic temperature.
`dreamerv3`	model-based	Compact vector latent dynamics, symlog targets, imagination, and slow value target.
`structured_dqn`	semantic transition	Double-DQN over variable state-local candidates.
`structured_bc`	semantic supervision	Weighted reward-free likelihood over legal factor candidates.
`structured_ppo`	joint trajectory	PPO over the exact sum of sequential conditional factor log probabilities.

Table 1: Built-in algorithm profiles. The entries list service capabilities; benchmark parity with the cited references requires separate experiments.

4.1 Value distributions

C51 represents a return distribution with learned probabilities on fixed support atoms [3]. QR-DQN instead learns quantile locations $\theta_i(s,a)$ at fixed fractions τ_i [4]. For target samples y_j, its quantile-Huber loss is

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} |\tau_i - \mathbf{1}\{y_j - \theta_i < 0\}|\, L_\kappa(y_j - \theta_i).$$

Expected-value control averages the learned quantiles. A downside policy can instead use the estimated lower quantile or average the lowest fraction as an empirical lower-tail CVaR score. These statistics expose aleatoric return variation; they do not measure epistemic uncertainty and do not guarantee that the tails are calibrated outside the data distribution.

DQN supplies the scalar baseline and uses Double-DQN selection to reduce positive maximization bias [1, 2]. CQL adds a log-sum-exp conservative penalty that charges high values assigned to unobserved actions [12]. The implemented CQL profile is discrete; continuous CQL variants are outside this service action specification.

4.2 Policy, asynchronous, and offline objectives

PPO clips the behavior-to-current policy ratio and uses generalized advantage estimation [7, 6]. IMPALA applies V-trace corrections when actors and the learner are decoupled [9]. The APPO plugin follows the IMPACT combination of a target policy, circular replay, and truncated importance sampling; ordinary shuffled replay would violate the asynchronous PPO algorithm [10].

Clipping the surrogate does not directly bound a shared neural-network parameter step. Structured PPO 1.7 therefore offers two independent, default-off preservation controls. The value head always receives its full loss, while `value_backbone_gradient_scale`$\in [0,1]$ controls the critic gradient entering the shared actor representation. Optional minimum and maximum policy-ratio bounds audit every selected behavior action after a complete proposal. A violation restores policy, Adam, and random-number state and retries the identical proposal at a declared learning-rate backoff. The committed ratio range, attempts, backtracks, acceptance, and effective rate are reported separately from minibatch-average KL.

Behavior cloning ignores reward and provides the demonstration baseline. The structured profile predicts semantic candidate identity within one factor's current legal alternatives and accepts generic example weights and source or factor groups. Its learner schema excludes application-specific labels. MARWIL estimates trajectory returns and a learned value, normalizes the advantage scale with a moving statistic, and weights action likelihood by a clipped exponential advantage [13]. CQL, BC, and MARWIL address offline use in different ways; none licenses deployment outside the observed state and action support without evaluation.

4.3 Entropy and world models

The maximum-entropy objective augments discounted reward with policy entropy,

$$J_\alpha(\pi) = \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t \left(r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t)) \right) \right].$$

The `maxent` plugin realizes this objective through discrete soft-Q backups. The SAC plugin uses a categorical actor, twin critics, soft target updates, and optional automatic temperature [11]. Illegal actions have zero probability. Their log probabilities are replaced only inside expectations so the mathematical $0 \log 0 = 0$ limit does not become an IEEE NaN.

DreamerV3 learns a world model and improves an actor and critic through imagined trajectories [14]. The built-in profile retains vector encoding, latent action dynamics, reward and continuation heads, symlog scaling, imagination, lambda returns, and a slow value target. It does not reproduce the reference recurrent state-space model, categorical latent representation, pixel decoder, or benchmark recipe. Full DreamerV3 parity should be an external plugin or a separately validated profile.

5 Noise, disturbance, and risk

5.1 A disturbance taxonomy

Reward outliers change the TD target; observation noise changes the estimated state; stochastic transitions widen the return distribution; an offline dataset changes support; actor lag changes the behavior policy; and a world model introduces approximation bias. One robustness label cannot distinguish them.

The shared runtime offers training-only normalization, Huber losses in value profiles, gradient clipping, optional reward clipping, and opt-in Gaussian observation augmentation. Reward clipping and augmentation are checkpointed because they alter the objective. A clean validation store remains clean by default. Named perturbed evaluation cohorts should carry the intended noise law and severity.

Distributional critics are useful when outcome tails matter. Double targets and twin critics address value overestimation. CQL addresses offline action shift. V-trace and APPO clipping address policy lag. Dreamer transformations address numerical scale, while held-out multi-step prediction must measure world-model error. Selecting one of these profiles leaves ensembles, adversarial observation training, explicit robust-MDP optimization, and calibrated epistemic uncertainty unimplemented.

5.2 What maximum entropy proves

Eysenbach and Levine show that maximum-entropy RL maximizes a lower bound on a robust objective for characterized reward and transition disturbance sets [15]. The result supplies a principled hypothesis: an entropy regularizer may retain multiple viable behaviors when one route or reward perturbation changes. It does not show that MaxEnt RL is optimal for every sensor corruption, adversary, market regime, or reward misspecification. The experiment must therefore declare the temperature and disturbance family, and its robustness claim applies only to those settings.

For noisy Volt outcomes, a predeclared comparison begins with DQN or PPO, then compares QR-DQN under mean and fixed downside rules, and compares categorical SAC or soft Q at fixed temperatures. Seeds, path cohorts, chronological dates, observation perturbations, and reward perturbations are separate axes. Choosing a CVaR level after inspecting held-out outcomes would turn the held-out set into training data.

6 QUBO rollout and frontier allocation

6.1 The budgeted selection problem

The computational problem is deciding which paths merit an expensive evaluation before their scores are known. The service must make that decision using information that is already cheap. Let $\mathcal{B}_{t-1}$ be the beam retained at search depth $t-1$, and let

$$\mathcal{C}_t = \bigcup_{v \in \mathcal{B}_{t-1}} \text{child}(v) = \{c_1, \ldots, c_{n_t}\}$$

be the next candidate frontier. Candidate c_i has a cheap information set $\mathcal{I}_{t,i}$—for example a policy/value estimate, an admissible bound, a shallow rollout, or model disagreement—but an unknown expensive terminal value Y_i. A binary decision $x_i = 1$ retains the candidate. With a beam budget k_t, the ideal one-step search decision is

$$\arg\max_{x \in \{0,1\}^{n_t}} \quad \mathbb{E}\left[\max_{i : x_i = 1} Y_i \,\middle|\, \mathcal{I}_t \right] \qquad \text{subject to} \qquad \mathbf{1}^\mathsf{T} x = k_t. \tag{1}$$

The conditional joint distribution of the Y_i is generally unavailable. Moreover, selecting the individually largest point estimates can spend the entire beam on highly correlated siblings whose errors share the same cause. The implemented QUBO is therefore a tractable surrogate for (1): reward a cheap utility estimate while charging for pairwise redundancy. The recursion is

$$\mathcal{B}_t = \{c_i \in \mathcal{C}_t : x_{t,i} = 1\}, \qquad \mathcal{B}_0 = \{\text{root}\}, \tag{2}$$

and only $\mathcal{B}_D$ receives terminal evaluation at depth D. The same subset problem appears later in the learning pipeline when c_i is a replay transition or contiguous rollout fragment and Y_i denotes the learning value of evaluating that transition or fragment; terminal path value is a different quantity. Search pruning is potentially more valuable because it occurs before the expensive work.

Figure 2 makes the recursion explicit. The retained parents are first expanded, and one QUBO is formed over the complete frontier $\mathcal{C}_t$. Its single decision vector determines the next beam.

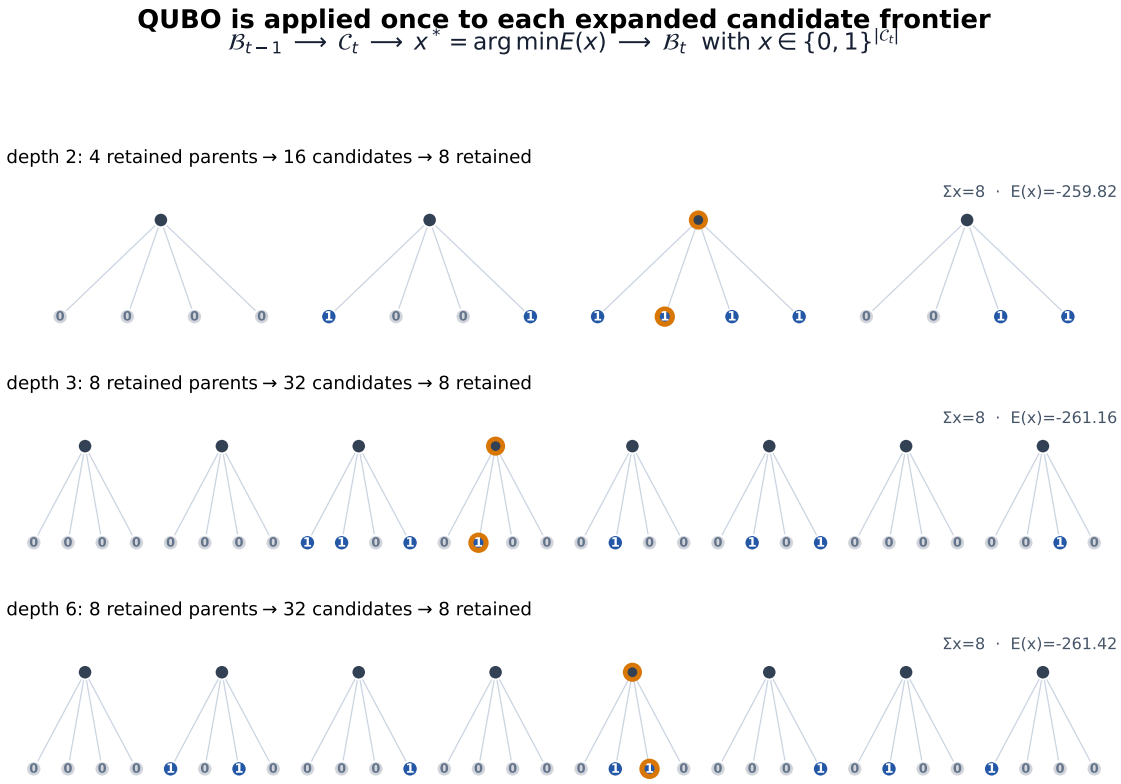

Figure 2: An actual width-eight trace for seed 20,260,802. Blue nodes have $x_i = 1$, grey nodes have $x_i = 0$, and a gold outline identifies the oracle-path prefix known only to evaluation. The trace explains the mechanism; the paired 500-tree experiment supplies aggregate evidence.

6.2 Utility and redundancy

Let $\widehat{v}_i$ be the available scalar utility. To prevent isolated outliers from determining the linear QUBO coefficient, the implementation uses the fifth and ninety-fifth percentiles $q_{.05}$ and $q_{.95}$:

$$u_i = \operatorname{clip}\left(\frac{\widehat{v}_i - q_{.05}}{q_{.95} - q_{.05}}, 0, 1\right). \tag{3}$$

If the percentile range is numerically zero, every candidate receives $u_i = \frac{1}{2}$. Non-finite values are replaced by the finite median before scaling. In replay mode, the default unscaled utility is $\widehat{v}_i = \operatorname{mean}_{j \in c_i} \log(1 + p_j)$ for replay priority p_j. Absolute reward can be added but is disabled by default because it can preferentially select lucky noisy outcomes.

Each candidate also has an embedding $z_i \in \mathbb{R}^d$. For coordinate r, define its median m_r, interquartile range a_r, and stabilized scale $\bar{a}_r = a_r$ when $a_r > 10^{-8}$ and $\bar{a}_r = 1$ otherwise. The robust distance and RBF similarity are

$$\begin{aligned} \widetilde{z}_{ir} &= \frac{z_{ir} - m_r}{\bar{a}_r}, \\ d_{ij}^2 &= \|\widetilde{z}_i - \widetilde{z}_j\|_2^2, \\ h &= \operatorname{median}\{d_{ij}^2 : d_{ij}^2 > 10^{-12}\}, \\ s_{ij} &= \exp\left(-\frac{d_{ij}^2}{2h}\right), \qquad s_{ii} = 0. \end{aligned} \tag{4}$$

The degenerate bandwidth is set to one. Transition embeddings contain the current state, state delta, action, reward, and terminal flag. Rollout embeddings summarize state and delta plus action, reward, terminal, and length. In the path experiment below, a one-hot branch-position embedding makes paths with long common prefixes similar.

6.3 Constrained subset objective and QUBO matrix

For n candidates and target cardinality k, the binary energy is

$$E(x) = -\lambda_u \sum_{i=1}^{n} u_i x_i + \lambda_s \sum_{i<j} s_{ij} x_i x_j + \lambda_k \left(\sum_{i=1}^{n} x_i - k\right)^2, \qquad x_i \in \{0,1\}. \tag{5}$$

The first term favors utility, the second charges for redundant pairs, and the third turns the cardinality constraint into an unconstrained binary penalty. Using $x_i^2 = x_i$ and a symmetric convention $x^\top Q x$, expansion of the final term gives

$$Q_{ii} = -\lambda_u u_i + \lambda_k(1 - 2k), \qquad Q_{ij} = \frac{\lambda_s s_{ij} + 2\lambda_k}{2} \quad (i \neq j), \tag{6}$$

such that $x^{\mathsf{T}}Qx = E(x) - \lambda_k k^2$. The omitted constant cannot change the minimizer. Equation (6) is the exported Q matrix consumed by an external annealer or binary optimizer. An unconstrained backend must validate that λ_k is strong enough for its coefficient scale. The built-in solver instead stays in the feasible set $\mathbf{1}^{\mathsf{T}}x = k$, so its cardinality is exact and the penalty is constant.

Within that feasible set, minimizing (5) is equivalent to maximizing

$$F(S) = \lambda_u \sum_{i \in S} u_i - \lambda_s \sum_{\substack{i<j \\ i,j \in S}} s_{ij}, \qquad |S| = k. \tag{7}$$

The classical reference solver greedily adds the candidate with largest marginal score

$$\Delta_i(S) = \lambda_u u_i - \lambda_s \sum_{j \in S} s_{ij}, \tag{8}$$

then applies deterministic one-for-one exchanges. Replacing selected item o by unselected item n changes (7) by

$$\Delta F_{o \to n} = \lambda_u (u_n - u_o) - \lambda_s \left(\sum_{j \in S \setminus \{o\}} s_{nj} - \sum_{j \in S \setminus \{o\}} s_{oj} \right). \tag{9}$$

Positive exchanges are accepted until no one-swap improvement remains or the configured pass limit is reached. The implementation uses a classical one-swap local heuristic, and the reported result covers only that solver. Q construction requires quadratic similarity storage; the candidate-pool factor bounds n in replay, and the beam bounds it during search. Candidate identities, decisions, energy, selected utility, redundancy, matrix bytes, and solve time remain auditable.

The formulation follows QUBO episode selection [18]; quantum-inspired priority mechanisms provide a related replay direction [19]. They motivate the formulation but do not establish quantum advantage. Deterministic selection after prioritized sampling also changes inclusion probabilities. Ordinary PER weights do not correct this second choice, so the implementation uses neutral learner weights and reports the absence of an importance correction.

6.4 Controlled branching experiment

The empirical question is whether the diversity term can preserve a good region under correlated heuristic error, and whether the saved evaluation work can repay solve overhead. Each of 500 paired trials constructs a complete four-way tree of depth six. A path is $p = (a_1, \ldots, a_\ell)$ with $a_t \in \{0, 1, 2, 3\}$. Independent edge increments at depth t satisfy

$$\xi_{p_{1:t}} \sim \mathcal{N}(0, 1/t), \qquad Y_p = \sum_{t=1}^{6} \xi_{p_{1:t}} \quad \text{for } |p| = 6. \tag{10}$$

Thus the standard deviation of an increment is $1/\sqrt{t}$. For every prefix p, the benchmark oracle computes

$$M_p = \max_{q:\, p \text{ is a prefix of } q} Y_q. \tag{11}$$

Search does not receive M_p directly. Its synthetic cheap estimate is

$$\widehat{v}_p = M_p + B_{a_1} + \varepsilon_p, \qquad B_a \sim \mathcal{N}(0, \sigma^2), \qquad \varepsilon_p \sim \mathcal{N}(0, (0.35\sigma)^2), \quad \sigma = 0.9. \tag{12}$$

B_{a_1} is shared by every path in a first-level region and therefore creates the correlated optimism that diversity is intended to hedge; ε_p is independent. The proxy is intentionally derived from the oracle plus controlled noise. It isolates subset selection under an oracle-derived noisy proxy; production value-model accuracy is outside this claim.

The tree contains $4^6 = 4{,}096$ terminal paths and $\sum_{t=1}^{6} 4^t = 5{,}460$ non-root candidates. Trial seeds are $20{,}260{,}802 + s$ for $s = 0, \ldots, 499$. Utility-only search retains the top k values of $\widehat{v}$; monotone robust scaling does not change that order. QUBO uses $\lambda_u = 1$, $\lambda_s = 0.12$, $\lambda_k = 4$, and four local-swap passes. Beam widths $k \in \{4, 6, 8, 10, 12, 16\}$ are evaluated on identical trees.

For terminal beam $\mathcal{B}_6$, oracle value $Y^* = M_\emptyset$, and selected value $\widehat{Y} = \max_{p \in \mathcal{B}_6} Y_p$, the primary metrics are

$$R = Y^* - \widehat{Y}, \qquad H = \mathbf{1}\{R \le 10^{-12}\}, \qquad G = \sum_{t=1}^{6} |\mathcal{C}_t|, \qquad L = |\mathcal{B}_6|. \tag{13}$$

R is simple terminal regret, H is exact-optimum recovery, G counts cheap candidate generations, and L counts terminal evaluations. Mean-regret bars use normal 95% intervals; recovery bars use 95% Wilson intervals. Because selectors see the same tree in each trial, paired regret improvement is $D_s = R_{s,\text{utility}} - R_{s,\text{QUBO}}$ with interval

$$\overline{D} \;\pm\; 1.96\, s_D/\sqrt{500}. \tag{14}$$

The run used one Python 3.11.15 process with NumPy 1.26 on an AMD Ryzen Threadripper PRO 7985WX. Timing is machine-specific; oracle construction is evaluation instrumentation and is excluded from search accounting.

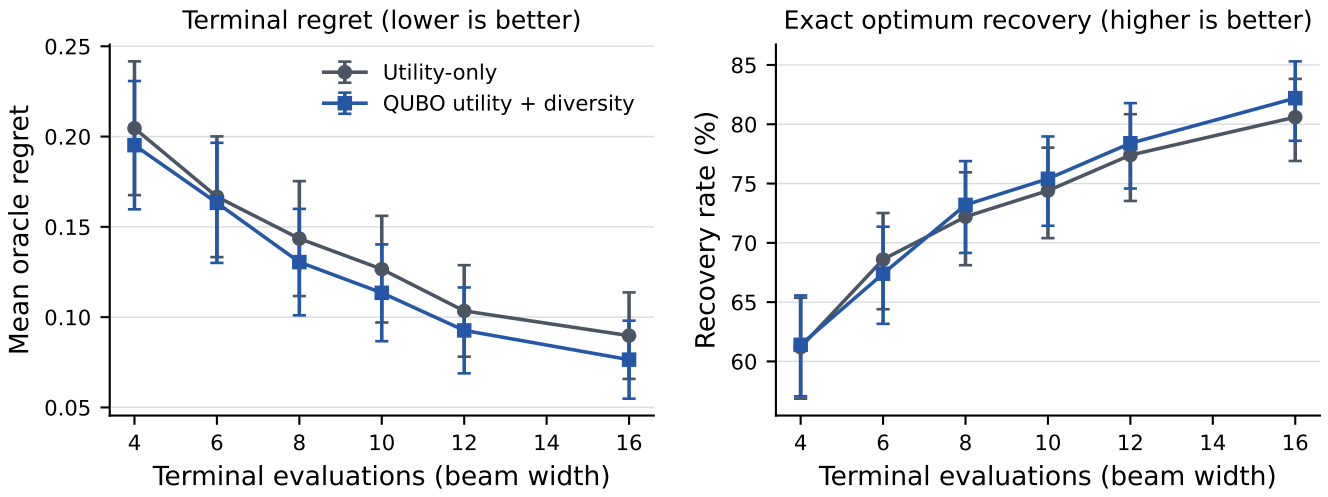

Figure 3: Paired frontier-search results over 500 trees. QUBO has lower mean regret at every tested budget, but confidence intervals overlap at several widths; exact recovery is also slightly worse at width six. Bars are the intervals defined after Equation (13).

Table 2: The two beam budgets used for the break-even comparison.

Selector	k	Recovery	Mean R	P95 R	G	Selector ms
Utility	8	72.2%	0.1435	1.0047	148	0.09
QUBO	8	73.2%	0.1305	0.8060	148	2.87
Utility	10	74.4%	0.1266	0.7706	180	0.11
QUBO	10	75.4%	0.1135	0.7230	180	3.57

At width eight, QUBO reduced mean regret from 0.1435 to 0.1305 and increased exact recovery from 72.2% to 73.2%. The paired mean improvement was $\overline{D} = 0.0130$ with 95% interval $[0.0022, 0.0239]$. At width sixteen the paired interval also excluded zero; at widths four, six, ten, and twelve it did not. At width six, QUBO reduced mean regret slightly but exact recovery fell from 68.6% to 67.4%. Figure 3 therefore supports a bounded result: diversity improved average terminal quality in this noise model, but did not dominate every metric at every budget.

6.5 Compute break-even and interpretation

Let c_g be the average cost of generating and cheaply scoring one candidate, c_e the average terminal-evaluation cost, and $T_{s,A}$ selector time for method A. Its accounted search cost is

$$T_A = T_{s,A} + c_g G_A + c_e L_A. \tag{15}$$

QUBO width eight is close in aggregate quality to utility width ten. QUBO uses $G_Q = 148$ candidates and $L_Q = 8$ terminal evaluations; utility width ten uses $G_U = 180$ and $L_U = 10$. QUBO is faster whenever

$$32c_g + 2c_e > T_{s,Q} - T_{s,U} \approx 2.75 \text{ ms}. \tag{16}$$

If candidate generation is treated as free, Equation (16) requires $c_e > 1.38$ ms per avoided terminal evaluation. Any nonzero c_g lowers that threshold. Conversely, if $\widehat{v}_i$ itself requires the full rollout, the optimization is too late to recover rollout cost.

This experiment demonstrates a plausible classical QUBO integration benefit under one synthetic error model and one coefficient set. Exact solvers, quantum execution, learning performance, and Volt require separate evidence. The experiment does not yet test coefficient sensitivity, learned heuristic calibration, graph transpositions, path-dependent market regimes, or end-to-end policy improvement. The next gate is a predeclared paired experiment on immutable Volt path cohorts using a production-available cheap estimate and Equation (15) for total wall time.

7 Policy representation and Volt

7.1 Fixed embeddings and end-to-end encoders

Volt constructs the option portfolio, state/action graph, action identity, legal mask, financial transition, and reward evidence. Jörmungandr trains and serves a statistical choice among the admissible action slots. Its permitted operations exclude creating financial terms, bypassing a mask, or applying a fill.

Deep Sets is the permutation-invariant set baseline [16]. A typed graph network adds relation-specific message passing when graph structure earns its extra compute [17]. A frozen encoder can produce a fixed-size vector consumed by any service plugin today. This is the simplest path for entry selection, hold/close, and bounded hedge/resize stages.

The library also supplies an encoder-independent masked actor–critic loss. Any Torch model that emits one policy-logit row and one value per state can use it. A graph trajectory buffer retains external state references, masks, chosen slots, rewards, behavior probabilities, values, terminal flags, and policy versions, then computes GAE targets. A Volt in-process research driver resolves references to Deep Sets or GNN batches and backpropagates through the encoder.

The compatibility service stores fixed NumPy observations. The implemented v2 profile instead transports a global feature vector, a variable typed-entity matrix, state-local candidate descriptors and identifiers, and a legal mask. It pads only during model collation, serves one centrally versioned policy, and supports three explicit data modes. Structured Double-DQN records one semantic candidate in prioritized transition replay. Structured PPO records a complete joint trajectory: one observation, centralized value, scalar reward, and next observation per environment turn, with factor choices nested inside each turn as part of the single reward-bearing sample.

For joint sampling, `policy/score` returns candidate-aligned logits and one value. The actor applies environment-owned masks in factor order and records the conditional log probability actually used at each choice. Their sum is the joint behavior log probability used by PPO. A later constraint update may restrict the current or future factors but cannot rewrite an earlier factor's distribution; the service rejects this post-hoc projection as mislabeled on-policy data. It validates complete episode ordering, semantic identity, split, policy lag, and duplicate provenance before queueing a trajectory. Training and validation queues stay separate.

Long structured episodes need not repeat every intermediate observation on the wire. The compact sequence schema sends an ordered chain of (N+1) observations beside (N) factor/action/reward records; the decoder reconstructs the same immutable step sequence and applies the same semantic, ordering, split, duplicate, and policy-lag checks. The standard client transparently gzip-compresses requests above a configurable byte threshold, and the HTTP handler rejects unsupported or malformed content encodings. Transport size changes while PPO semantics remain fixed. The current wire schema supports complete episodes; correctly bootstrapped partial fragments and durable graph artifact references are absent.

Structured behavior cloning uses a third, explicitly reward-free mode. For label i, the application supplies state s_i, the current legal candidate identities C_i for one factor, semantic target $a_i \in C_i$, and positive sample weight w_i. The generic learner minimizes

$$\mathcal{L}_{\mathrm{BC}} = \frac{\sum_i w_i[-\log \pi_\theta(a_i \mid s_i, C_i)]}{\sum_i w_i}.$$

The reporting target class and the training balance stratum are independent. An application may therefore retain a coarse metric label while declaring a finer conditional choice-set or selected-prefix stratum. Jormungandr can derive mean-one weights proportional to $\mathrm{count}(g_i)^{-\alpha}$ for $0 \le \alpha \le 1$. Its historical sampler draws records uniformly and applies w_i in the loss. The opt-in importance sampler instead draws i with probability $w_i / \sum_j w_j$ and supplies unit loss weights. The expected batch mean is the same normalized objective above, without multiplying by w_i twice, but rare high-weight strata are less often absent from finite batches. Group semantics remain application-owned.

Training and validation labels occupy separate bounded buffers. Validation calculates accuracy, negative log likelihood, entropy, calibration error, and source/factor-group summaries without gradients. The target is keyed by candidate ID, so a candidate permutation does not relabel the example. The BC and PPO profiles share the entity/candidate transformer schema. A service-owned initialization operation therefore copies a schema-compatible BC policy into a new PPO learner while resetting optimizer, update/version, and trajectory state; the value head receives no BC objective and remains an untrained starting critic. Gradient scaling lets that head fit returns without allowing its initial error to overwrite the copied actor backbone; the historical fully shared update remains the default.

7.2 Quantile tree baselines

CatBoost and LightGBM are useful after a Volt encoder has been frozen. CatBoost supports a multi-quantile objective in one CPU model; LightGBM supports a quantile objective with an alpha parameter and normally uses one model per level. Both provide fast tree controls over numeric embeddings [20, 21, 22, 23]. Neither backpropagates through a Deep Sets or GNN encoder. Torch is therefore the default for end-to-end policy and critic training. Its distribution package supports categorical and reparameterized sampling; QR-DQN itself requires only the differentiable quantile-Huber tensor loss.

8 Metrics and model comparison

Every plugin reports common loss and priority statistics plus named internal metrics such as entropy, value loss, Q mean, CQL penalty, temperature, importance ratio, quantile spread, or model loss. TensorBoard stores them under algorithm-qualified paths. The selector has a separate namespace. A runtime comparison endpoint aligns the latest rows for all model identifiers, including externally logged held-out performance measures, and a per-model endpoint retains recent history and the latest selection audit.

The v2 structured profile exposes the full observability schema, including the latest scalar. Its metrics endpoint retains learner update history, accepts externally computed held-out measures at an explicit step, and writes learner and evaluation series to TensorBoard. Environment-specific runners can therefore keep an append-only curve whose every evaluation row names the exact checkpoint, while the public service remains unaware of the business case.

Objective losses from CQL, PPO, SAC, and Dreamer use distinct performance scales. Model selection uses predeclared held-out environment measures at equal interaction or data budgets: return, lower-tail outcome, constraint events, turnover, drawdown, stability across seeds, and inference/learner cost as the domain requires. Validation loss diagnoses its training objective; economic policy choice requires the held-out environment measures.

8.1 Controlled online transition comparison

The first convergence experiment restricts its cohort to plugins that consume the same online one-step transition stream: DQN, C51, QR-DQN, maximum-entropy Soft-Q, and categorical SAC. This condition establishes stream compatibility. Expected-winner selection is a separate empirical question. PPO, APPO, and IMPALA require behavior-policy trajectories; BC, MARWIL, and CQL require a declared offline dataset; and DreamerV3 needs a sequence and model-compute budget. Putting those objectives on this curve would change both their input data and their update budget.

Each included algorithm uses a two-layer width-64 network, learning rate 3×10^{-4}, discount 0.97, prioritized replay, batch size 32, and its native exploration rule. Five independently initialized runs receive 64 fixed-schedule episodes of the synthetic OU spread environment, each of length 32. Replay warms for 128 transitions, after which one update follows every new transition, giving 2,048 interactions and 1,921 updates per run. Every four episodes the deterministic policy is evaluated without learning on the same 16 held-out path seeds. A second cohort adds independent $\mathcal{N}(0, 0.15^2)$ noise to the spread and spread-change sensors while leaving position and remaining time exact. Intervals are $\overline{R} \pm 1.96 s_R/\sqrt{5}$ across the five run-level held-out means.

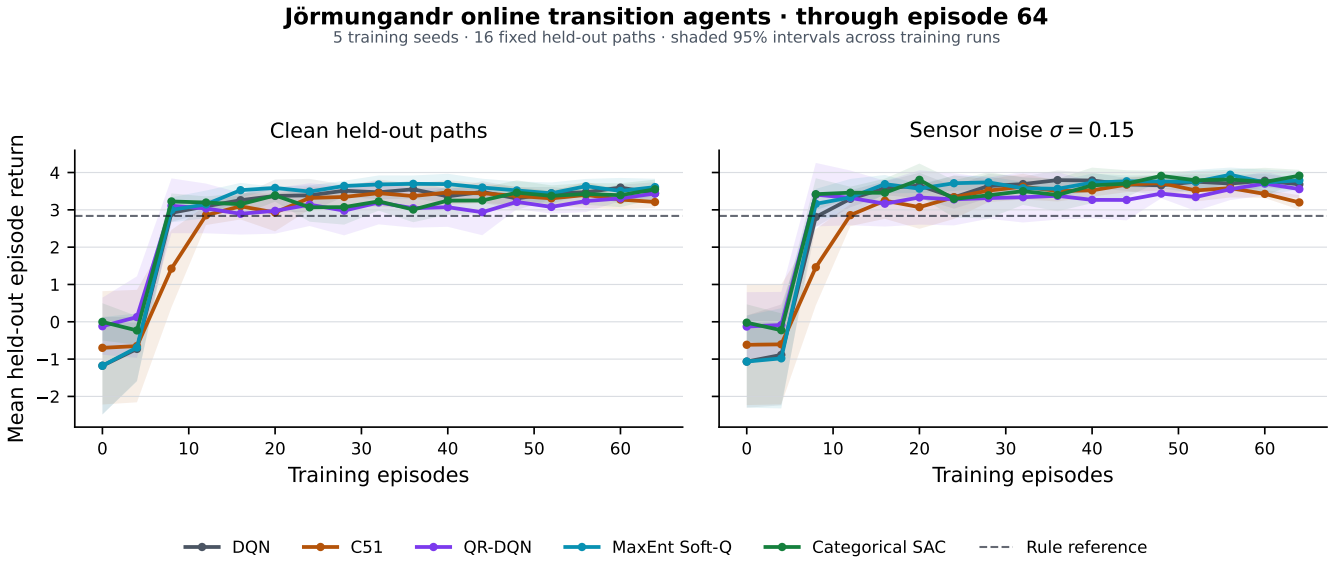

Figure 4: Held-out convergence for compatible online transition learners. The dashed rule reference returns 2.8367 on the fixed paths. Shading is the normal 95% interval across five independent training runs; it quantifies training-seed variation for this cohort. Environment-general uncertainty remains unmeasured.

Table 3: Final mean held-out episode return with normal 95% intervals.

Algorithm	Clean	Sensor noise $\sigma = 0.15$
DQN	3.4413 [3.1439, 3.7387]	3.6837 [3.5181, 3.8493]
C51	3.2101 [3.0546, 3.3655]	3.1983 [3.0047, 3.3919]
QR-DQN	3.4382 [3.1950, 3.6814]	3.5558 [3.3141, 3.7974]
MaxEnt Soft-Q	3.6087 [3.3839, 3.8334]	3.7893 [3.5476, 4.0310]
Categorical SAC	3.5508 [3.2913, 3.8104]	3.9155 [3.7581, 4.0728]

Most agents crossed the reference point estimate at the episode-eight checkpoint; C51 improved more slowly. MaxEnt Soft-Q has the largest final clean point estimate and SAC the largest perturbed point estimate, but their intervals overlap several alternatives. Perturbation also improves some point estimates in this threshold-like environment. The result therefore demonstrates a functioning comparison protocol and exposes behavioral differences; it does not establish a general winner or a robustness guarantee. The committed animation reveals convergence checkpoints, and a separate same-path playback shows final median-run actions and accumulated reward. The single playback path supplies an explanation only and adds no evidence.

8.2 Independent PPO reference diagnostic

The trajectory cohort begins with CartPole-v1 because a familiar environment separates PPO mechanics from application representation. The reference is Stable-Baselines3 2.9.0 [8] in an isolated CPU environment with Gymnasium 1.3.0 and Torch 2.8.0. Both participants use separate policy and value MLPs with two width-64 hidden layers—exactly 9,155 trainable parameters in each implementation—and share learning rate 3×10^{-4}, discount 0.99, GAE $\lambda = 0.95$, clip ratio 0.2, rollout length 1,024, minibatch size 64, ten epochs, zero entropy bonus, value coefficient 0.5, and gradient norm 0.5. Three initialization seeds receive 49,152 environment steps. Every 4,096 steps, deterministic policies are measured on the same twenty held-out reset seeds; 475 is the declared solved threshold.

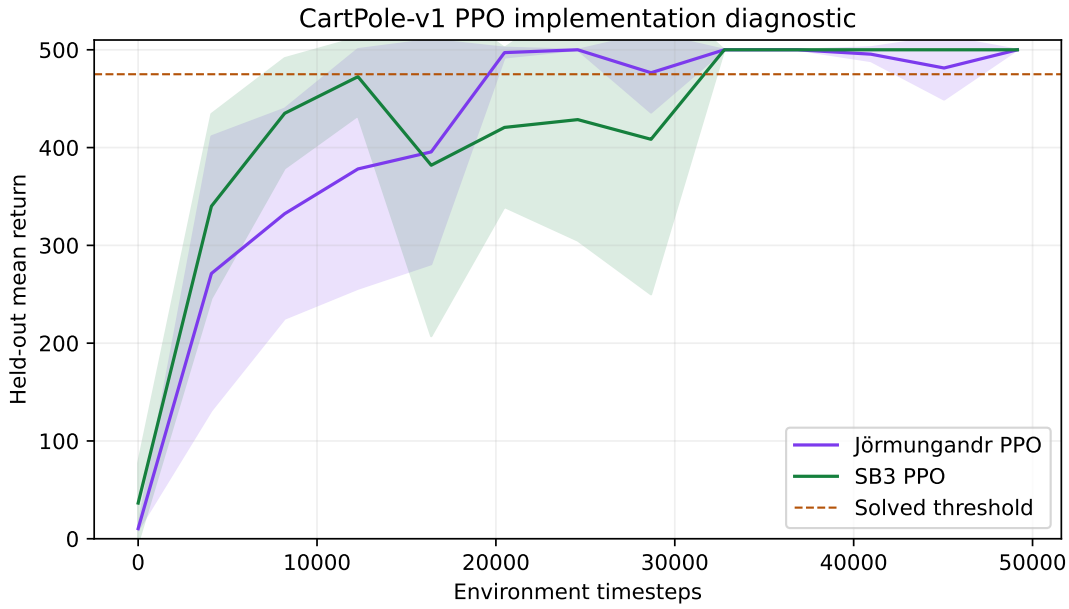

Figure 5: Matched-capacity CartPole PPO diagnostic. Lines are the mean of three independent training seeds and shading is one standard deviation of the three held-out run means. The curve verifies CartPole wiring across the compared implementations; library-wide performance requires a broader benchmark.

Table 4: First solved checkpoint and final held-out return by training seed.

Implementation	Run 0	Run 1	Run 2	Final means
Jörmungandr PPO	20,480	20,480	20,480	500 / 500 / 500
Stable-Baselines3 PPO	8,192	24,576	12,288	500 / 500 / 500

The reference often crosses the threshold earlier, while both curves are non-monotone at intermediate checkpoints. The result establishes that the generic Jörmungandr PPO policy, value bootstrap, GAE ordering, behavior-policy metadata, clipping, and optimizer path can solve this control problem at the same capacity and budget as an independent implementation. In all final Jörmungandr updates, both behavior-log-probability and behavior-value fallback rates are zero. It does not validate an unrelated reward, joint-action trajectory, variable representation, or multi-agent protocol; those remain separate application gates.

8.3 Entity/candidate representation parity

The S0 control changes only the CartPole representation and compatible policy encoder. Each four-coordinate vector becomes four typed entities with stable identifiers; left and right become two state-local candidates. The structured policy is a two-layer, width-32 entity/candidate transformer with 20,674 parameters. Environment, reward, three initialization seeds, held-out reset seeds, PPO controls, evaluation cadence, and 49,152-turn budget are identical to the flat Jörmungandr cohort above. The predeclared gate requires at least 90% of the flat final median and no more than twice its median time to 475.

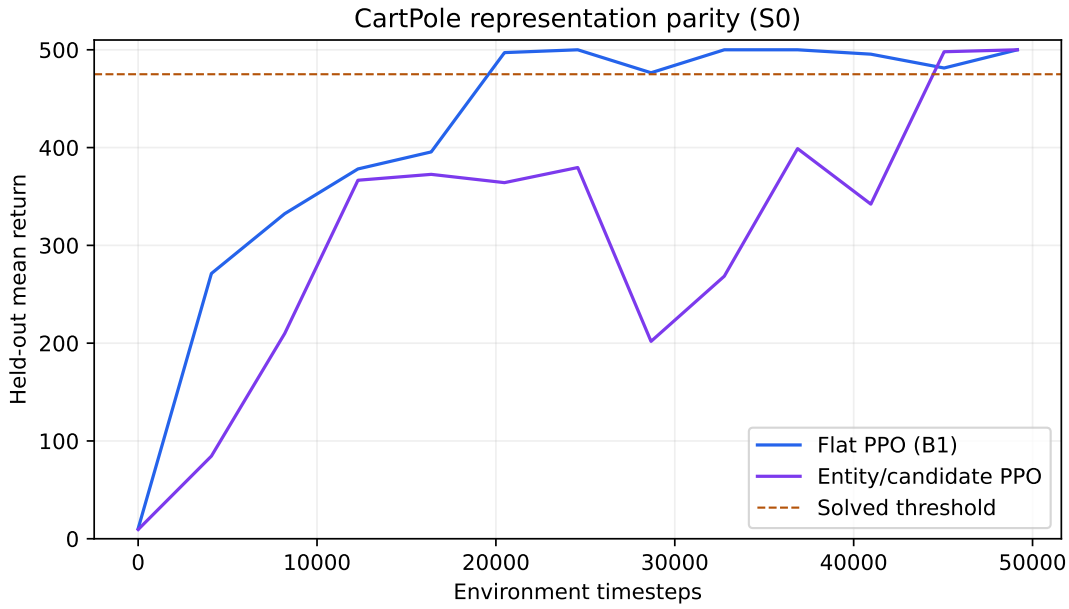

Figure 6: Flat versus entity/candidate CartPole PPO. Both lines are means of the same three training seeds evaluated on the same twenty held-out resets. This isolates representation and collation from application dynamics.

Table 5: Representation-parity first solved checkpoint and final held-out return.

Representation	Run 0	Run 1	Run 2	Final means
Flat MLP	20,480	20,480	20,480	500 / 500 / 500
Entity/candidate transformer	8,192	32,768	12,288	500 / 500 / 500

S0 passes: both final medians are 500 and the structured median first-solved checkpoint is 12,288 versus 20,480 flat. Permuting entity and candidate rows while retaining IDs changes semantic logits by at most 1.08×10^{-6} and values by at most 7.63×10^{-6} in the recorded runs, below the 10^{-5} tolerance. Wire round trip, trajectory storage, checkpoint restore, and the versioned structured inference bundle preserve the selected semantic candidate. This rules out a gross variable-representation failure; it does not show that a particular application's features, reward, or joint constraints are sufficient.

8.4 Structured supervision diagnostic

The BC0 functional gate uses eight weighted labels from two factor families and four independent candidate permutations. The structured learner reaches 100% accuracy on the held-out copy of this tiny corpus. Candidate-row permutation preserves semantic logits within 10^{-5}, and validation evaluation leaves all parameters unchanged. The wire record contains no reward, requires every supplied factor alternative to be currently legal, and reports overall and source/factor-group accuracy, negative log likelihood, entropy, calibration error, and gradient norm.

A central-service test interleaves the eight training and eight validation labels, retains them in separate bounded buffers, and updates only from the training split. Its checkpoint initializes a frozen structured PPO model with identical candidate logits, a new optimizer, and update and policy-version counters at zero. BC0 therefore validates reward-free semantic imitation and the BC-to-PPO handoff; it does not establish that a demonstration source is optimal or that imitation improves terminal return.

Teacher provenance is checked separately from learnability. A non-executing Python source audit records source hashes, large literal tables, opaque embedded payloads, and output lookups indexed by an observation-derived turn. It reports a mechanism rather than declaring every schedule invalid. A second diagnostic pairs supervision from distinct episodes by timestep and factor, requires the exact numeric model input to differ, and measures whether the semantic target changes. A third summary accepts only input and decision fingerprints from application-owned counterfactual interventions. Thus Jörmungandr can expose trace-compatible teachers without importing a simulator or pretending that an unchanged action proves state independence.

Two exact, domain-neutral "calculation friends" provide a narrower boundary than asking a policy to approximate deterministic combinatorics. The first solves a caller-declared bipartite assignment exactly. It can maximize assignment cardinality and then declared utility, or maximize total declared utility while treating an unassigned worker and task as zero utility; the latter omits zero- and negative-improvement optional work. Caller-declared positive task capacities default to one and permit several workers to share one logical task without copying numbered slots; worker capacity remains one. The second applies Dijkstra's algorithm to caller-declared positive-cost directed edges and returns a node path plus semantic action identifiers. Assignment ties use a recorded seed and route ties use input edge order. Jörmungandr does not invent tasks, feasibility edges, costs, traversability, or action meanings; those remain application inputs. Unassigned and unreachable results are explicit and serializable. These solvers can therefore sit below a learned task or target choice without silently replacing an already sampled policy action.

The structured transformer can optionally apply candidate-to-entity cross-attention after the entity encoder. A candidate first incorporates its declared entity pointers and then queries the masked encoded global/entity set. Thus a worker-relative candidate need not recover every worker–target relation from one pooled global token. The same option is accepted by structured BC, PPO, and DQN; zero layers remains the checkpoint-compatible default.

A streaming metrics accumulator applies the exact selected-prefix correction before reporting raw and weighted accuracy, negative log likelihood, and entropy by source, factor, and semantic target. A seeded SHA-256 selector also produces input-order-independent, capped diagnostic subsets with an exact receipt. In a six-worker synthetic routing control with unseen validation worlds and matched minibatches, pooled context reaches 98.4375% and one candidate-attention block reaches 100%. The 1.5625-point difference verifies the mechanism; it is not a broad sample-efficiency claim.

8.5 Conditional supervision-sampling diagnostic

`ConditionalSupervisionSampling-v0` removes policy capacity and domain semantics from the minibatch question. A synthetic corpus has 1,000 records in one common balance stratum and ten records in each of two rare strata. Square-root inverse-frequency balancing gives mean-one weights 0.85 and 8.5; both arms therefore target the same exact weighted diagnostic objective, 0.2916667.

Across 3,000 deterministic batches of size 64, uniform draws followed by weighted loss contain 1.2617 rare records on average and contain none in 26.07% of batches. Their objective estimate has mean absolute error 0.150488. Weight-proportional draws with unit post-draw loss weights contain 10.6707 rare records on average, never omit them, and reduce mean absolute error to 0.056728; absolute bias from the exact objective is 0.000639. All predeclared conditions pass. This validates the generic buffer estimator, while application balance-group definitions and downstream learning gains require their own validation.

8.6 Prefix-conditioned preference diagnostic

The optional structured prefix head separates learned preference conditioning from hard feasibility. One central inference emits a base logit ℓ_j and key/value vectors (k_j, v_j) for each candidate. Before factor i, the conditional logit for current candidate j is

$$\ell_{j|p_{<i}} = \ell_j + \frac{k_j^\top \sum_{a \in p_{<i}} v_a}{\sqrt{d_p}}.$$

The environment-owned callback still supplies the legal mask. The actor and PPO recompute the same expression from the recorded prefix, so conditioning does not require repeated service inference or an approximate behavior probability. Setting $d_p = 0$ recovers the previous prefix-independent model.

The isolated `PrefixPreferenceGate-v0` contains two balanced labels with an identical observation, identical current candidates, and identical legal mask. The correct quantity is small after a `wait` prefix and large after a `produce` prefix. Across five fixed seeds, the prefix-independent arm stays at accuracy 0.5 and median NLL 0.693147, its binary information limit. A width-16 model with $d_p = 4$ reaches accuracy one on every seed and median NLL 0.009255. Because this case has no reward or environment transition, it attributes the difference specifically to learned preference conditioning. Masking, PPO credit, and horizon length are held constant. This narrow capability test covers only the stated joint-action problem size.

8.7 Constrained joint-action learning diagnostic

`ConstrainedWorkbench-v0` is a Gymnasium-compatible generic task with no application concepts. On each of three turns, two to four typed workers choose a state-local job or PASS. A job may be used once; jobs consume shared capacity and members of one conflict group cannot coexist. Utility accumulates internally, but learner reward is zero until termination and is then divided by the exact enumerated episode optimum. The oracle return is therefore one by construction. Changing worker count changes only nested factor count; the three trajectory records and the single nonzero reward stay fixed.

All learned arms use the same 12,770-parameter width-32, one-layer transformer. Behavior cloning receives 256 exact-oracle training episodes and 500 updates. Random-start and BC-initialized PPO each receive 12,288 environment turns in each of three training runs. Deterministic evaluation uses the same 64 held-out episode seeds. Figure 7 and Table 6 report the recorded cohort.

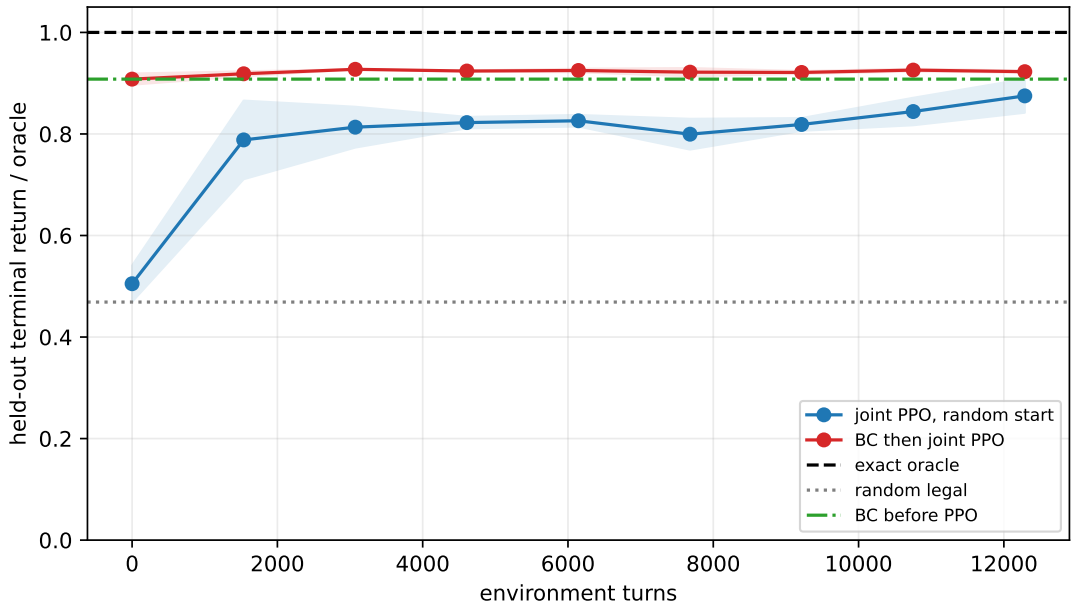

Figure 7: Terminal return relative to the exact workbench oracle. Lines are means of three training runs; shading is one between-run standard deviation. Random legal and BC-only policies are fixed held-out references.

Table 6: Constrained-workbench held-out terminal return.

Policy	Before PPO	At 12,288 turns
Exact oracle	1.0000	1.0000
Random legal	0.4689	0.4689
Structured BC	0.9080	—
Joint PPO, random initialization	0.5050	0.8750
Structured BC then joint PPO	0.9080	0.9227

BC accuracy is 0.8228 and 0.8159 for the two worker kinds, versus respective action-frequency baselines of 0.4875 and 0.4755. All three tiny deterministic corpora reach 100%. Final PPO improves over paired random legal episodes by 0.4061 on average. A hierarchical paired bootstrap resampling training runs and common evaluation seeds gives a 95% interval $[0.3581, 0.4533]$. BC-to-PPO median return is 0.9337 of oracle, above the predeclared 0.80 gate, and no evaluated arm emits an infeasible action. J1 passes. The task is small enough for complete enumeration and isolates the structured formulation. Larger combinatorial environments require separate evidence.

8.8 Masked-action representation diagnostic

Taxi-v4 isolates legal-mask behavior on 500 discrete states and six actions. The neural policies receive a one-hot state and two separate width-64 MLPs, giving 72,903 trainable parameters to both Jörmungandr PPO and SB3-contrib MaskablePPO. Four participants receive 131,072 interactions over three seeds: masked Jörmungandr PPO, the same Jörmungandr PPO with masking deliberately disabled, SB3-contrib MaskablePPO, and a 3,000-entry masked tabular Q-learning reference. Evaluation uses fifty fixed reset seeds every 16,384 steps. The primary gate requires zero environment-invalid actions for every masked participant and masked Jörmungandr success-area-under-curve no worse than its unmasked twin.

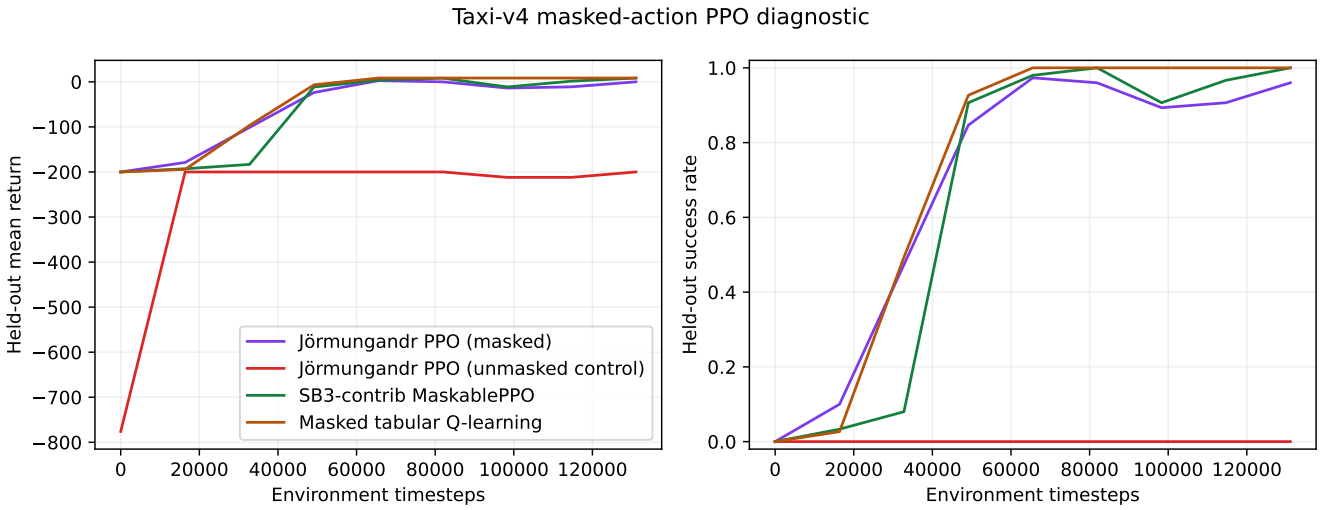

Figure 8: Taxi-v4 return and task-completion learning curves. Every masked participant uses the environment-supplied mask; the red control alone may select masked actions. Curves are means across three training seeds.

Table 7: Final Taxi-v4 held-out results by seed. Each cell is success rate / mean return.

Implementation	Run 0	Run 1	Run 2
Jörmungandr PPO, masked	.96 / 0.18	1.00 / 8.20	.92 / −8.34
Jörmungandr PPO, unmasked	.00 / −200	.00 / −200	.00 / −200
SB3-contrib MaskablePPO	1.00 / 8.44	1.00 / 8.24	1.00 / 8.44
Masked tabular Q-learning	1.00 / 8.44	1.00 / 8.44	1.00 / 8.44

All masked participants record zero invalid choices in both training and every evaluation. The unmasked control accumulates 462,511 invalid choices over the same interaction and evaluation schedule and never completes a held-out task. Masked Jörmungandr success AUC is 0.7042 versus zero unmasked, so the declared B2 mask gate passes. Its final seed variance remains material: the independent masked references finish at 100% on all seeds, whereas Jörmungandr finishes at 96%, 100%, and 92%. The masked policy had reached 97.3% mean success at 65,536 steps before later regression. This supports hard constraint handling and also motivates checkpoint selection and stability diagnostics. Complex-application efficacy requires separate evidence.

9 Python, Torch, and a future Rust path

Python is appropriate for orchestration and research iteration, while Torch provides modules, automatic differentiation, device execution, and model export. The first native migration should follow measured profile evidence. The likely CPU candidates are replay indexing, contiguous-fragment assembly, QUBO construction and local search, graph collation, preprocessing, and return kernels such as GAE or V-trace.

A PyO3 extension can preserve the plugin interface by accepting contiguous arrays and returning priorities, metrics, selections, or tensors. DLPack or a documented NumPy ownership rule avoids copies. Keeping policy modules in Torch avoids a second optimizer and checkpoint ecosystem. A native backend must retain semantic plugin version, tensor names or an explicit migration, declared dtypes and shapes, deterministic seeds, and parity tests against the Python reference.

Moving the whole algorithm to Rust before measurement would couple artifact handling more tightly to autograd. Moving a quadratic selector or replay scan after it dominates a profile is a smaller and testable change. The current plugin metadata already records `python-torch`; a future backend can declare `rust-pyo3-torch` without changing actor messages.

10 Current implementation and limits

The public implementation registers fixed-vector and structured profiles for C51, QR-DQN, DQN, CQL, maximum-entropy soft Q, BC, MARWIL, PPO, IMPALA, APPO, categorical SAC, and the compact DreamerV3 profile. The HTTP service lists plugins, applies legal masks at inference and backup, stores behavior metadata, assembles contiguous trajectories, filters excessive policy lag for online trajectory learners, selects ordinary or QUBO batches, logs algorithm-qualified metrics, checkpoints arbitrary plugin state, restores it, and exports each plugin's declared Torch module. The v2 structured service additionally provides central entity/candidate inference, prioritized transition replay, complete joint-trajectory ingress, policy-lag checks, exact conditional-log-probability PPO, reward-free weighted structured supervision, conditional balance groups, importance-resampled supervision batches, direct BC-to-PPO policy initialization, structured checkpointing

and inference bundles, recent learner history, external held-out metrics, source/time-dependence/counterfactual teacher-provenance diagnostics, and TensorBoard persistence. Complete joint trajectories may use either the original step-array wire or a deduplicated observation-chain sequence with transparent gzip request compression. Each structured-PPO update also records raw episode-return mean, standard deviation, minimum, maximum, unique-value count, episode-length range, nonzero-reward fraction, $gamma\lambda$, and the oldest-residual weight $(\gamma\lambda)^{N-1}$ before GAE. This is a diagnostic distinction: high explained variance may describe a critic that predicts one common loss and its distance to termination, while the policy still has almost no evidence for ranking action sequences.

The frozen `DelayedTerminalCredit-v0` gate checks this exact mechanism on balanced $-1/+1$ terminal rewards over 719 structured steps with zero behavior values. Jormungandr and Stable-Baselines3 2.9.0 each compute both $\lambda = 0.98$ and $\lambda = 1$. Every raw target is checked against $z(\gamma\lambda)^d$; maximum disagreement between implementations is 4.7684×10^{-7}, within the declared 2×10^{-6} tolerance. After the same batch normalization used by PPO, the positive opening advantage is 2.6763×10^{-6} in the first arm and one in the episodic arm. This selects $\lambda = 1$ for a terminal-only child experiment as an estimator design decision. No learning or variance conclusion follows.

The independent `StructuredPPOSafety-v0` gate tests policy preservation over five seeds. In a value-only joint update, the historical shared critic moves a policy-backbone parameter by about 0.0200. Zero critic-backbone scale leaves every non-value parameter bit-identical while the value head still moves by about 0.0200. A second deliberately oversized actor proposal leaves the frozen ratio interval $[0.5, 2]$ in all unguarded seeds, reaching 2.061×10^{-9}. Transactional backoff accepts the third common-random proposal in every seed at effective rate 0.01; all committed ratios lie in $[0.670, 1.518]$ or a tighter interval. These results establish isolation and containment. Return improvement requires a separate environment experiment.

The joint sampler can now be autoregressive in both legality and learned preference. Optional low-rank candidate key/value heads condition later factor logits on the selected prefix while retaining one central inference per turn. Exact masks remain environment inputs. The additive prefix summary conditions choices within one turn; temporal memory across turns requires recurrent state.

The recorded functional suite contains 169 passing cases. It includes direct finite updates and state round trips for every built-in, C51 projection and mask checks, graph-reference GAE, validation isolation, HTTP ingress, masked SAC entropy, QR-DQN tail output, QUBO cardinality and fragment audit, checkpoint restore, frontier accounting and local-swap checks, replayable visualization audits, structured metrics history and TensorBoard persistence, exact joint sampling and PPO updates, multiprocess structured-trajectory ingress, compact sequence round trip and compressed HTTP ingress, policy-lag/duplicate/order rejection, structured export, supervised split isolation, semantic candidate permutation, conditional balance compatibility, importance-sampling objective checks, BC-to-PPO initialization, the seeded online comparisons, critic-backbone isolation, transactional joint-ratio rollback and backoff, the structured PPO safety gate, and artifact manifests. The exact-calculation cases additionally verify cardinality-first assignment, optional total-utility assignment, declared utility and seeded ties, positive task capacities, positive-cost shortest routes, caller-ordered route ties, multiple targets, source-as-target, and unreachable certificates. A separate threaded matrix gave every built-in eight mask-aware transitions; each completed an update, checkpoint, TorchScript export, restore, and masked inference.

These are lifecycle tests. They do not reproduce Atari, continuous-control, Minecraft, offline-RL, or Volt trading benchmarks. The built-in CQL and SAC profiles are discrete. The Dreamer profile is smaller than the reference. Replay data expire on restart. Structured PPO currently consumes complete episodes; bootstrapped partial fragments remain unsupported. The constrained workbench is a small exact-oracle benchmark and does not establish scaling to a large action system. Application actors still supply dynamic feasibility callbacks and execute actions. Multi-learner synchronization, authentication, binary high-throughput transport, exact QUBO solving, native kernels, and a Volt-scale empirical QUBO advantage are absent from the release.

11 Conclusion

An RL library should preserve experiment identity while allowing the learning rule to change. Jörmungandr keeps actors, splits, masks, replay provenance, versions, validation, metrics, and artifacts stable, and makes algorithms and rollout allocation replaceable plugins. Distributional, offline, asynchronous, maximum-entropy, and model-based profiles can therefore be compared through one lifecycle without calling the system a C51 model.

Quantile regression supplies a direct return-risk representation for noisy outcomes; maximum entropy supplies a bounded robustness hypothesis; CQL and V-trace address different shifts. QUBO supplies an auditable allocation experiment whose binary unit is a rollout fragment or frontier node. Volt retains financial and graph authority while Torch set or graph models supply the policy. Rust can later accelerate measured CPU kernels behind the same interfaces. The remaining test is empirical: under predeclared disturbances, paths, budgets, and controls, do these results survive valid offline, trajectory, model-based, and Volt-scale cohorts, and does added complexity repay its compute cost?

References

[1] V. Mnih et al., *Human-level control through deep reinforcement learning*, Nature 518, pp. 529–533, 2015, https://doi.org/10.1038/nature14236.

[2] H. van Hasselt, A. Guez, and D. Silver, *Deep Reinforcement Learning with Double Q-learning*, AAAI Conference on Artificial Intelligence, 2016, https://arxiv.org/abs/1509.06461.

[3] M. G. Bellemare, W. Dabney, and R. Munos, *A Distributional Perspective on Reinforcement Learning*, International Conference on Machine Learning, 2017, https://arxiv.org/abs/1707.06887.

[4] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, *Distributional Reinforcement Learning with Quantile Regression*, AAAI Conference on Artificial Intelligence, 2018, https://arxiv.org/abs/1710.10044.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, *Prioritized Experience Replay*, International Conference on Learning Representations, 2016, https://arxiv.org/abs/1511.05952.

[6] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, *High-Dimensional Continuous Control Using Generalized Advantage Estimation*, International Conference on Learning Representations, 2016, https://arxiv.org/abs/1506.02438.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, 2017, https://arxiv.org/abs/1707.06347.

[8] J. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, *Stable-Baselines3: Reliable Reinforcement Learning Implementations*, Journal of Machine Learning Research, 22(268):1–8, 2021, https://jmlr.org/papers/v22/20-1364.html.

[9] L. Espeholt et al., *IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures*, International Conference on Machine Learning, 2018, https://arxiv.org/abs/1802.01561.

[10] M. Luo, J. Yao, R. Liaw, E. Liang, and I. Stoica, *IMPACT: Importance Weighted Asynchronous Architectures with Clipped Target Networks*, International Conference on Learning Representations, 2020, https://arxiv.org/abs/1912.00167.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, International Conference on Machine Learning, 2018, https://arxiv.org/abs/1801.01290.

[12] A. Kumar, A. Zhou, G. Tucker, and S. Levine, *Conservative Q-Learning for Offline Reinforcement Learning*, Advances in Neural Information Processing Systems, 2020, https://arxiv.org/abs/2006.04779.

[13] Q. Wang, J. Xiong, L. Han, P. Sun, H. Liu, and T. Zhang, *Exponentially Weighted Imitation Learning for Batched Historical Data*, Advances in Neural Information Processing Systems, 2018, https://proceedings.neurips.cc/paper/2018/hash/4aec1b3435c52abbdf8334ea0e7141e0-Abstract.html.

[14] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, *Mastering Diverse Domains through World Models*, 2023, https://arxiv.org/abs/2301.04104.

[15] B. Eysenbach and S. Levine, *Maximum Entropy RL (Provably) Solves Some Robust RL Problems*, International Conference on Learning Representations, 2022, https://arxiv.org/abs/2103.06257.

[16] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. Smola, *Deep Sets*, Advances in Neural Information Processing Systems, 2017, https://arxiv.org/abs/1703.06114.

[17] P. W. Battaglia et al., *Relational Inductive Biases, Deep Learning, and Graph Networks*, 2018, https://arxiv.org/abs/1806.01261.

[18] H. Salloum, A. Jnadi, Y. Kholodov, and A. Gasnikov, *Quantum-Inspired Episode Selection for Monte Carlo Reinforcement Learning via QUBO Optimization*, 2026, https://arxiv.org/abs/2601.17570.

[19] C. Y. Chen, H. H. Hsu, and H. T. Lin, *Quantum-Inspired Experience Replay*, 2021, https://arxiv.org/abs/2101.02034.

[20] L. Prokhorenkova, G. Gusev, A. Vorobev, A. Dorogush, and A. Gulin, *CatBoost: Unbiased Boosting with Categorical Features*, Advances in Neural Information Processing Systems, 2018, https://arxiv.org/abs/1706.09516.

[21] G. Ke et al., *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*, Advances in Neural Information Processing Systems, 2017, https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

[22] CatBoost, *Regression objectives*, https://catboost.ai/docs/en/concepts/loss-functions-regression.

[23] LightGBM, *Parameters*, https://lightgbm.readthedocs.io/en/stable/Parameters.html.