Volt: Sequential Control of Option Portfolios

research@strategynet.ai

July 2026

Abstract

An entry portfolio fixes the initial option contracts and quantities; management determines the later closes, hedges, rolls, and exercises. Market moves, fills, assignment, expiration, and settlement also change the position and its available actions. Volt formulates management as sequential control over a versioned state/action graph. The compiler resolves semantic actions to listed contracts and supplies a structural legal mask. The runtime applies fills and lifecycle rules, updates cash and inventory, and journals each transition. A policy scores the remaining actions while a declared reward maps P&L, costs, drawdown, capital use, and constraint events to the training objective. This paper defines the control state, transition and reward contracts, the reinforcement-learning loop, chronological evaluation, and fixed-budget comparisons with rule-based management. QUBO enters as a later experiment for allocating rollout or replay batches; contract legality and portfolio accounting remain runtime responsibilities. The implementation includes a multi-expiry state/action graph, account-refined masks, a generated one-step comparison of set and graph action scorers, and a repeated hold/close experiment with Volt-reconciled trajectories. Sequential learning proceeds through entry selection, hold/close, hedge/resize, roll/replace, and full-control stages.

Contents

1 Introduction

Portfolio discovery chooses contracts and signed quantities at a decision time. Portfolio control manages the resulting state as option marks, underlying prices, cash, inventory, exercise opportunities, and executable actions change. Holding to expiry, closing at a profit target, rolling a tested short strike, and delta hedging are distinct policies applied to the same initial portfolio.

American options add holder exercise before expiry and short-side assignment events. Physical settlement may replace an option with underlying inventory and a cash posting. A policy therefore carries continuing positions and account consequences from one decision to the next.

2 Control state and transition

2.1 State

At decision time t, the policy input is

$$s_t = (G_t, A_t, M_t, p_t).$$

The graph G_t contains the current option positions, underlying and expiry relationships, cash, underlying inventory, market observations, and lifecycle state. The ordered set A_t contains resolved candidate actions. $M_t \in \{0,1\}^{|A_t|}$ is the aligned legal mask. Provenance p_t identifies the compiled program, runtime state, market observations, and assumptions used to construct the input.

The implemented `RuntimeStateActionGraphSample` supplies all distinct expiries, option lifecycle state, cash and delivered-underlying nodes, candidate-action nodes, typed action-to-state edges, rejection reasons, and the compiler mask. A declared research account adds limits for live positions, gross option quantity, minimum cash, underlying inventory, and option costs. Broker margin, liquidity, and eligibility can refine that mask before policy normalization.

2.2 Transition

One policy action and one subsequent market observation produce

$$(s_{t+1}, r_t, e_t) = \mathcal{T}_\omega(s_t, a_t, \xi_{t+1}),$$

where ω names the execution, exercise, assignment, settlement, and accounting rules. The observation ξ_{t+1} contains the next market state and external events. The reward record r_t is derived from the transition, and e_t contains immutable evidence.

The implemented runtime applies one action and one market observation to a versioned portfolio state. It validates the action, updates option lifecycle, cash, and underlying units, and returns a journal entry linked to the prior and next state hashes.

A short-put assignment illustrates the required continuity. Assignment can consume the option, debit strike cash, and create long underlying inventory. The next policy decision sees the delivered asset, remaining cash, open options, and a new action set.

3 Actions and legal masks

3.1 Action vocabulary

The implemented candidate grammar contains:

- `AddLeg` for a listed contract selected by explicit or semantic coordinates;
- `RemoveLeg` and `ResizeLeg` for a live position;
- `Wait`; and
- holder-authorized `Exercise`.

Runtime lifecycle actions are `Hold`, `Close`, `Exercise`, `Assign`, and `Expire`. Assignment and expiry enter as external or system events. A roll is represented by a multi-step close and addition plan; take-profit, stop, hedge, and roll instructions require a planner that emits those primitive actions.

Semantic actions resolve at the decision clock. A request for a call in a tenor and delta bucket uses the contract universe and observations visible at that time. The resolved action records the listed contract, quantity, branch identifier, source rule, and artifact hash.

The control environment separates financial authority from statistical choice. The compiler and application determine which actions are admissible. The runtime applies contract, execution, exercise, assignment, and settlement rules. The policy assigns scores or probabilities to the admissible actions. The reward definition and path cohort determine the behavior encouraged by training.

The first experiment uses a small action grammar and bounded path set for which policy trees can be evaluated exhaustively. Rule-based management, tree search, and a standard actor–critic algorithm establish the baseline. Rollout allocation and QUBO follow after state transitions, rewards, and accounting agree on those bounded environments.

3.2 Mask composition

Let M_t^{compiler} contain contract, portfolio, authority, and lifecycle legality. Let M_t^{account} contain capital, margin, liquidity, and application policy. The policy receives

$$M_t(a) = M_t^{\text{compiler}}(a) M_t^{\text{account}}(a).$$

The composed mask changes after a fill, exercise, assignment, expiration, market halt, or account-state change. Each excluded action retains a reason from the component that rejected it.

The learner observes the mask and scores admissible choices. Deterministic legality remains reproducible when the model or training algorithm changes.

4 Rewards and horizons

4.1 Outcome record

One transition can produce the financial vector

$$r_t = \begin{pmatrix} \Delta\text{equity}_t,\ \text{cost}_t,\ \Delta\text{drawdown}_t, \\ \text{capital}_t,\ \text{turnover}_t,\ \text{constraint}_t \end{pmatrix}.$$

The current scenario evaluator supplies fills, costs, journals, marked pathwise P&L, leg attribution, and terminal P&L. Drawdown, capital, turnover, and constraint-event measures require named account and risk components.

A scalar training reward may be

$$\begin{aligned} R_t = \Delta\text{equity}_t &- \lambda_c \text{cost}_t - \lambda_d \Delta\text{drawdown}_t \\ &- \lambda_k \text{capital}_t - \lambda_v \text{violation}_t, \end{aligned}$$

with coefficients fixed before test evaluation. The vector record remains available for economic reporting even when the learner uses a scalar.

4.2 Episode and learning horizons

An episode can end at portfolio liquidation, the final contract expiry, a risk breach, or the end of the available path. This financial termination rule is separate from the computational return horizon.

For an n-step actor–critic update,

$$\widehat{R}_t^{(n)} = \sum_{j=0}^{n-1} \gamma^j R_{t+j} + \gamma^n \widehat{V}(s_{t+n}).$$

The bootstrap state retains every open option, right, cash balance, and underlying position. Truncating an update therefore changes credit assignment while leaving the economic episode intact.

4.3 Declared hold/close objectives

Training, checkpoint selection, and a performance claim use related but different quantities. For the generated hold/close stage, the learning return is the undiscounted sum

$$G_i(\pi) = \sum_t R_{i,t}, \qquad \gamma = 1.$$

It contains the policy-dependent marked-equity changes after the first decision and the closing cost. Entry-to-first-decision P&L and the opening cost are already realized at the first state. They do not change the optimal continuation action, but they are included in validation and test P&L.

Let $P_i(\pi)$ be terminal net P&L on validation episode i, including opening and closing costs, and let $D_i(\pi)$ be maximum mark-to-market drawdown. For n validation episodes, set $k = \lceil 0.10n \rceil$ and define the signed lower-tail mean

$$T_{0.10}(\pi) = \frac{1}{k} \sum_{j=1}^{k} P_{(j)}(\pi),$$

where $P_{(1)} \le \cdots \le P_{(n)}$. The checkpoint score declared before the next run is

$$S_{\text{val}}(\pi) = \overline{P}(\pi) + 0.25\, T_{0.10}(\pi) - 0.10\, \overline{D}(\pi). \tag{1}$$

The lower-tail statistic is signed, so a negative tail reduces the score. All three terms are measured in dollars per one-contract episode. The 0.25 coefficient retains the lower-tail convention used by the preceding static discovery experiment; the 0.10 coefficient adds a smaller penalty for the path taken to the terminal result. No coefficient grid is fit to the validation or test cohort. The candidate set contains the one-shot initialization and the actor–critic checkpoints evaluated every five epochs. Test observations do not select a checkpoint, coefficient, epoch, encoder, or random seed.

The selected policy is compared with close-first and hold-to-horizon rules on the untouched test episodes. A control-performance claim requires a positive lower endpoint of a paired 95% interval for the difference in (1) from the strongest declared non-clairvoyant rule. Episode resampling is paired across policies and recomputes the tail statistic inside each sample. Empirical studies replace independent-episode resampling with blocks defined before the test by decision date and overlapping outcome horizon. Failure of this gate is reported as such; mean P&L alone does not establish the claim.

5 Policy and learning loop

5.1 Policy model

A set or graph encoder maps the current portfolio state to h_t. Each candidate action has embedding $e_t(a)$. A shared policy head computes

$$\ell_\theta(s_t, a) = f_\theta(h_t, e_t(a), h_t \odot e_t(a)).$$

Masked action probabilities are

$$\pi_\theta(a \mid s_t) = \frac{M_t(a) \exp\{\ell_\theta(s_t, a)\}}{\sum_{a' \in A_t} M_t(a') \exp\{\ell_\theta(s_t, a')\}}.$$

A value head estimates expected discounted return or a return distribution.

Offline portfolio training supplies parameter-matched Deep Sets and typed graph encoders. Deep Sets is the default control encoder because it is smaller and materially faster in the present benchmark. A configuration switch selects the graph encoder under the same state, action, mask, policy head, value head, reward, and evaluation protocol. Each matching supervised checkpoint can initialize its encoder; transfer is compared with control training from an independent initialization.

5.2 Deep Sets and the typed graph encoder

Both encoders receive the same typed nodes, action slots, and legal mask. They differ in the information used to combine those nodes. The runtime Deep Sets encoder applies one shared map independently,

$$z_v = \phi_\theta(x_v), \qquad h_t = \frac{1}{|V_t^{\text{state}}|} \sum_{v \in V_t^{\text{state}}} z_v,$$

and gathers z_a for each action node. Reordering the serialized nodes does not change the state summary. The static portfolio model in the discovery paper uses the canonical sum form; the runtime policy uses a masked mean because graph size also varies with the compiler action set. Node type, contract, account, and market attributes must therefore carry the information needed by the shared map [1].

The typed graph encoder first projects every node and then performs relation-specific message passing. In schematic form,

$$m_v^{(\ell)} = \sum_{(u,v,r) \in E_t} W_r^{(\ell)} h_u^{(\ell)}, \qquad h_v^{(\ell+1)} = F_\ell\left(h_v^{(\ell)}, m_v^{(\ell)}\right).$$

An option position can consequently receive different information from its underlying, expiry, portfolio, and account neighbours; an action representation can depend on the particular position to which it applies. The final state representation joins the updated portfolio node, account node, and a pooled state summary [2].

Deep Sets is a permutation-invariant set model: it consumes neither E_t nor relation types. Aggregation through a virtual global node provides a useful analogy, although its global pooling supplies no local or relation-specific message passing. Both models are permutation invariant at readout, and both can score a variable number of legal actions. The compiler constrains the contract and action universe and the runtime mask removes illegal actions before either model is trained; this is separate from the encoder choice.

The present hold/close state has one live option and two meaningful actions. Most relations are recoverable from node attributes, so typed message passing adds computation without adding much identifiable signal. Multi-expiry portfolios, stock and cash inventory, position-specific rolls, assignment consequences, and account constraints create neighbourhoods that the set model must infer from pooled attributes. Those stages provide the relevant test of the graph inductive bias. Until a paired held-out comparison improves the declared control objective, Deep Sets remains the default and the graph encoder remains an ablation.

Jormungandr's C51 core accepts a point-in-time legal mask for action selection and for the next-state bootstrap. Its complete service path still exposes a fixed action vocabulary and does not yet carry masks through experience records. Proximal Policy Optimization with generalized advantage estimation supplies the later actor–critic comparison once its rollout lifecycle is connected end to end [3, 4]. Tabular and tree-based control on small state spaces exposes transition and reward errors before either neural policy is interpreted.

5.3 Training loop

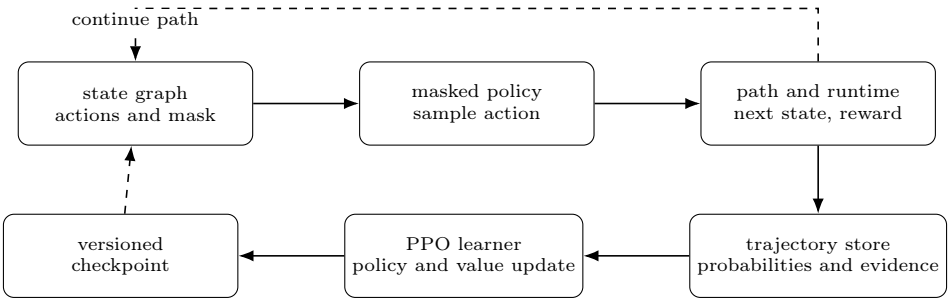

Figure 1: Proposed reinforcement-learning loop. Historical, generated, or forecast-conditioned paths supply market observations; the runtime supplies financial transitions; the learner updates from versioned trajectories.

Figure 1 makes the feedback loop explicit. Collection workers use one frozen checkpoint for an epoch, run independent path environments, and write immutable trajectory fragments. A deterministic merge orders fragments by stable identifiers before optimization.

Each trajectory step stores state and action identifiers, legal mask, action probability, value estimate, reward components, next-state identifier, termination status, path identifier, runtime journal, and policy version. These fields support replay, off-policy diagnostics, and reproduction of a reported update.

6 Rollout and replay allocation

6.1 Allocation problem

An action tree branches through portfolio decisions, market scenarios, execution outcomes, exercise, and assignment. A fixed simulation allowance may be assigned uniformly, through tree-search rules such as UCT [5], or through an allocator using value uncertainty, tail relevance, novelty, path probability, and estimated simulation cost.

The first comparison uses random allocation, stratified coverage, UCT, greedy utility, and greedy utility with diversity. Exhaustive small trees measure the actions and losses omitted by each allocator.

6.2 QUBO experiment

Let $x_i \in \{0,1\}$ indicate additional computation for frontier item i. For an equal-cost batch of size k,

$$\min_{x\in\{0,1\}^n} -\sum_i u_i x_i + \lambda \sum_{i<j} s_{ij} x_i x_j + \rho \left(\sum_i x_i - k \right)^2 .$$

The acquisition value u_i combines declared signals such as uncertainty, tail relevance, path probability, and estimated improvement. Similarity s_{ij} penalizes a batch concentrated in one region of state or scenario space. Variable simulation costs require a weighted budget formulation.

A recent preprint applies QUBO to Monte Carlo reinforcement-learning episode selection using return and state-coverage terms in a finite GridWorld [7]. Option portfolio control supplies a separate test: American lifecycle events, path-dependent costs, heterogeneous action sets, and variable rollout lengths.

Frontier allocation occurs before outcomes are observed and may reduce simulation cost. Replay allocation occurs after collection and reduces learner cost; prioritized experience replay is its baseline [6]. Selection probability, score components, solver configuration, matrix construction time, solver time, and solution quality are charged to the allocator.

Random or stratified capacity is reserved for exercise, assignment, and tail paths. The quadratic matrix grows as $O(n^2)$, so large frontiers require sparse neighborhoods, clustering, or staged allocation before QUBO construction.

7 Empirical design

7.1 Initial environment

The initial environment contains American-style options on one underlying, a bounded position size, a small action grammar, fixed decision cadence, and declared historical or generated path cohorts. The mark or pricing adapter, fill convention, fees, exercise policy, assignment process, account constraints, reward coefficients, and termination rule are shared by every policy.

Small instances retain exhaustive action trees. Development then increases one dimension at a time: decision horizon, number of contracts, action count, scenario count, execution uncertainty, and path length.

7.2 Management baselines

Economically interpretable policies include hold to expiry, fixed take-profit and stop rules, clock-based hedging, delta-threshold hedging, and a declared roll convention. Learning comparisons include tabular control, tree search, PPO, and the same policy with alternative set or graph encoders.

Measurements include:

- net and gross P&L, drawdown, tail loss, turnover, and capital use;
- exercise, assignment, settlement, and constraint-event coverage;
- stability under alternative costs, spreads, path sources, and assignment rules;
- value calibration and bootstrap error by remaining horizon;
- simulator calls, expanded nodes, environment steps, learner updates, and elapsed time; and
- allocation overhead and selected-state diversity.

Equal simulator calls measure statistical efficiency. Equal elapsed-time budgets include batching, environment, and allocation overhead.

7.3 Time split and policy evidence

Training, validation, and test periods follow the decision clock. Overlapping outcome horizons are assigned to one split or separated by a purge interval. Reward coefficients, stopping rules, exploration share, checkpoint selection, and rollout budget are fixed using training and validation data.

The test report retains gross and net results, path-level outcomes, policy and environment seeds, failures, legal-mask violations, exercise and assignment counts, and comparisons with the rule-based policies. A path cohort gives results under its recorded historical observations or generated model.

8 Current implementation and development order

Volt currently supplies exact option contracts, European/Bermudan/American exercise schedules, portfolio netting, semantic hashes, immutable runtime state, exercise and assignment transitions, cash and underlying emissions, versioned historical or generated market

observations, ordered scenario paths, scenario fills and costs, pathwise P&L, compiler action sets, graph samples, and structural legal masks. The runtime graph represents multiple expiries, completed option lifecycles, cash, delivered underlying, account limits, exact action nodes, and the composed point-in-time legal mask.

A static research driver now supplies generated cohorts, American CRR marks, fixed exercise and assignment decisions, invariant portfolio tensors, and evidence packets. Its fixed-horizon module compares unhedged, scheduled delta-hedged, and delta-band policies with reconciled P&L attribution, turnover, costs, drawdown, and exercise events. These rules establish the first management baselines. Deep Sets and a parameter-matched typed graph network consume the same runtime samples. A configurable policy model maps either encoder to aligned masked policy logits, per-action values, and a state value. The selected supervised representation initializes the matching encoder. A repeated hold/close experiment now supplies the first temporal policy and value updates. A batch scenario interface sends its held-out action schedules back through the Volt simulator and runtime journal.

8.1 Generated runtime-graph benchmark

The first richer-state experiment evaluates supervised action ranking. Policy returns require a separate temporal experiment. Each of three seeds generates 120 states from twelve American contracts: calls and puts at three strikes and two expiries. A state contains live and completed option lifecycles, cash, underlying inventory, account limits, and approximately 68 compiler action slots, of which 13.5 are legal on average. Every legal action uses 48 common price paths. American exercise is checked after seven days and at expiry; a short position is assigned when the paired holder decision exercises.

The target is the action's path utility minus the mean utility of the other legal actions in that state. Training, validation, and test splits occur by state. Table 1 reports the mean and population standard deviation across seeds.

Model	Parameters	Rank ρ	Best action	Regret ($)	States/s
Deep Sets	3,977	0.433 ± 0.180	0.403 ± 0.071	103.68 ± 42.73	50,223
Typed graph	5,391	0.450 ± 0.048	0.403 ± 0.104	101.30 ± 24.92	7,085

Table 1: Held-out within-state action ranking and single-thread CPU inference. Best action is the fraction of test states in which the predicted choice attains the evaluated maximum, including ties between economically equivalent slots. Regret is evaluated utility lost by the predicted choice.

The graph model has a small average advantage in full-ranking correlation and regret; both models recover a maximal action in the same fraction of test states. Seed dispersion is larger than the mean quality differences. Deep Sets has 7.1 times the state throughput. The comparison therefore supplies no general performance claim for the GNN.

With the Rust binary already compiled, JSON serialization, process startup, action compilation, account masking, and graph construction together process 53.4 states per second. Tensor encoding processes 3,267 states per second, and the path evaluator processes 8,724 path–action evaluations per second. Deep Sets and the typed graph score approximately 658,868 and 92,724 legal actions per second. Their mean training times are 1.07 and 6.98 seconds. Measurements use one PyTorch CPU thread on the recorded Linux host and exclude Rust compilation.

8.2 Generated hold/close control

The first temporal experiment is an optimal-stopping problem. Each of 180 generated episodes opens one long American call or put on day zero. The policy may hold or close on days 5, 10, 15, 20, 25, and 30; every position is closed before its day-35 expiry. The restricted action set excludes voluntary exercise and contains no short position subject to assignment. These are stage-specific controls. The American contract definition remains unchanged.

Latent physical drift and volatility generate the underlying path. The policy observes the point-in-time graph, trailing return, and realized volatility. Latent labels remain hidden from the policy. American lattice values supply generated proxy marks. Episode assignment occurs before graph construction, with 117 training, 27 validation, and 36 test paths. One-shot training uses realized full-path action targets; 80 actor–critic epochs then use ordered graph-reference trajectories and generalized-advantage targets. Every test schedule is rerun by the Volt scenario simulator. Its opening and closing fills, journal, and terminal P&L must reconcile with the learner reward records.

Policy	Mean P&L	Median	ES 10%	Drawdown	Close day	Score
Deep Sets, selected	−10.80	−6.88	−332.21	71.22	5.14	−100.98
Deep Sets, final RL	134.32	34.86	−504.42	236.17	19.31	−15.40
Deep Sets, one-shot	149.62	8.19	−508.03	230.59	17.50	−0.44
Typed graph, selected	−12.29	−6.88	−332.21	71.22	5.00	−102.46
Typed graph, final RL	112.28	−8.89	−569.71	292.90	26.39	−59.43
Close on day 5	−12.29	−6.88	−332.21	71.22	5.00	−102.46
Hold to day 30	109.67	−11.24	−589.37	309.05	30.00	−68.58
Clairvoyant bound	335.17	191.40	−147.57	117.45	17.64	286.54

Table 2: Single development-seed generated results in dollars. ES 10% is the mean of the worst ten percent of terminal outcomes; Score is (1). The seed was observed before (1) was fixed and supplies no prospective performance test. The clairvoyant row uses the complete future path and is included only as an upper bound.

The development seed originally selected checkpoints by validation mean P&L. After (1) was fixed, the same seed still selected an almost immediate close; two other development seeds changed checkpoint. Those reruns verify that the declared score controls the implementation. They cannot test the score because their outcomes contributed to its design.

Three new Deep Sets seeds use the same 117/27/36 split sizes and update budget. Neither their validation paths nor their 108 pooled test paths were inspected when (1) was fixed. Table 3 reports this prospective generated cohort.

Policy	Mean P&L	Median	ES 10%	Drawdown	Close day	Score
Validation-selected	−0.21	−4.19	−596.29	157.60	10.97	−165.05
Final RL iterate	10.81	−2.87	−587.71	162.24	12.87	−152.34
One-shot	62.62	−20.24	−626.20	245.67	17.36	−118.49
Close on day 5	27.68	0.62	−302.23	54.04	5.00	−53.28
Hold to day 30	15.83	−53.42	−780.56	352.61	30.00	−214.57
Clairvoyant bound	264.49	71.96	−223.87	112.56	14.77	197.26

Table 3: Prospective three-seed Deep Sets results over 108 held-out generated episodes. Final RL and one-shot rows are diagnostics; neither was selected by the locked validation rule.

The selected policy's score is 111.76 dollars below close-first. A paired 10,000-sample episode bootstrap gives a 95% interval of $[-199.19, -24.79]$ dollars for its improvement over the stronger rule. The claim gate fails, and all three per-seed gates also fail. The experiment therefore establishes a working temporal learner and records adverse generated control performance. It provides no evidence of a trading advantage.

The Deep Sets policy has 8,123 parameters, trains in 8.49 seconds, and scores 292,789 states per second on the recorded single-thread CPU run. The typed graph has 13,443 parameters, trains in 18.62 seconds, and scores 52,359 states per second. Deep Sets is 5.6 times faster at inference and remains the default.

8.3 Staged learner integration

Offline supervised training precedes sequential learning. Historical states and generated paths produce candidate-level outcomes; the compiler supplies the legal portfolio universe; and the selected encoder checkpoint initializes the control representation. Deep Sets is selected unless an experiment explicitly requests another encoder. The incremental stages are:

1. *Entry selection.* One decision chooses among a small compiled entry set. This contextual-bandit check verifies checkpoint transfer and masking without temporal credit assignment.
2. *Hold or close.* A fixed action vocabulary adds repeated decisions, transaction costs, early termination, and marked-equity rewards. This stage is now implemented for generated long American positions, with Volt-reconciled test trajectories and rule policies.
3. *Hedge and resize.* Fixed delta buckets and unit quantity changes add inventory, financing, turnover, and drawdown to the state and reward. The existing C51 service can represent these bounded actions.
4. *Roll and replace.* The compiler emits state-dependent contract actions, and the runtime graph supplies aligned descriptors and masks. The Jormungandr service and replay contracts must transport them.
5. *Full portfolio control.* Multi-expiry positions, exercise, assignment, delivered underlying, account constraints, and dynamic actions now enter one state/action graph. The remaining experiment adds repeated transitions and compares C51, PPO, transfer, and rule-based policies under chronological path splits.

Every stage fixes its action grammar, reward components, termination rule, path split, and accounting checks before adding the next source of complexity. QUBO follows the conventional rollout and replay baselines at the final two stages.

The learner adapter uses Jormungandr's masked actor–critic objective and graph-reference trajectory records. Durable graph replay and the public service schema remain future work. Volt retains the financial transition and legal-action authority; Jormungandr receives versioned observations, masks, actions, rewards, and next states after the environment passes deterministic replay and rule-policy checks.

9 Conclusion

Sequential option management is defined by continuing state, admissible actions, financial transitions, and a declared reward. The Volt compiler and runtime supply contract and lifecycle authority; the policy chooses among the actions that remain; the trajectory connects each choice to its fills, cash, inventory, and outcome.

The bounded hold/close environment now provides the first replayable temporal test with rule controls and a one-shot initialization. Its single-seed result motivated a precommitted downside-aware checkpoint score. A prospective three-seed generated test then rejected the learned policy under that score. A separate three-date static event study also recovers no final-date optimum, and direct enumeration remains faster after shared option-leg work is compiled. Empirical control therefore requires more chronological dates and actions whose future costs cannot be reduced to the common leg basis. Dynamic action scoring and PPO follow that gate. QUBO is reserved for a fixed-budget allocation comparison after conventional rollout and replay methods have established the baseline.

References

[1] 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.

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

[3] 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.

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

[5] L. Kocsis and C. Szepesvári, *Bandit Based Monte-Carlo Planning*, European Conference on Machine Learning, 2006, https://doi.org/10.1007/11871842_29.

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

[7] 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.