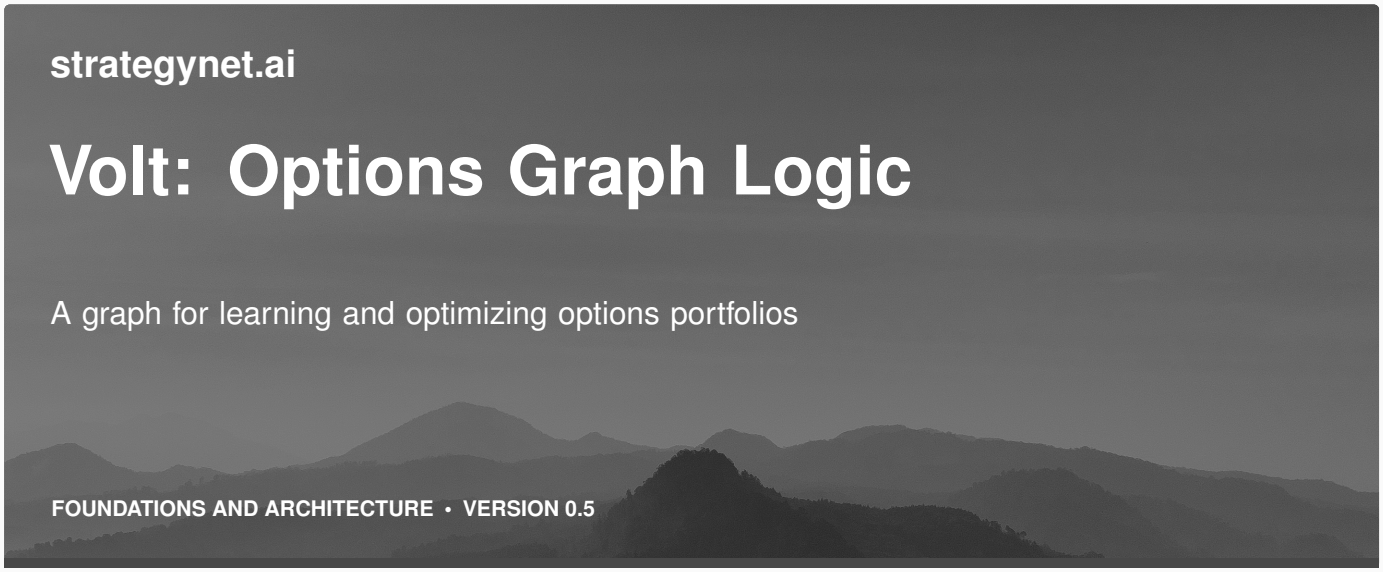


research@strategynet.ai

Open-source working paper • July 2026

ABSTRACT

Options portfolios are relational objects. Legs share underlyings, expirations, strikes, risk factors, exercise rights, settlement paths, and management choices. Flattening those relationships into a strategy name or an arbitrary row of leg features discards structure that learning, optimization, and search systems need.

Volt represents that structure as a deterministic options graph connected to exact contracts, signed positions, market evidence, lifecycle state, and simulation outcomes. The graph gives machine-learning models and optimizers a stable object on which to discover, compare, transform, and manage strategies without taking authority over their financial meaning. The public crate implements the semantic runtime, canonical identities, bounded candidate actions, scenario evidence, and an initial graph export. A future `cog` compiler remains an important efficiency layer for constructing, validating, and lowering these graphs; it is a supporting goal rather than Volt's central purpose.

Contents

PART I

Motivation

1 Why an options graph

Volt exists so that machine learning, AI, search, and numerical optimization can operate directly on options portfolios without first flattening away the relationships that make a strategy meaningful. An options graph makes contracts, signed legs, shared market objects, lifecycle rights, observations, and outcomes explicit. Its node ordering may change; the financial relationships do not.

A trader rarely begins with an option symbol. The starting point is usually a view or a constraint:

- "I expect the underlying to remain between 95 and 105 through this expiration."
- "Front expiries look rich relative to back expiries."
- "I want convex downside protection, but I can spend only 80 basis points."
- "I am willing to give up the upside above 110 to finance a put."

An order ticket realizes such an idea at a particular moment, through particular listed instruments, yet the position immediately acquires several other identities. It becomes a payoff over terminal spot, an exposure to implied volatility across strike and expiration, a consumer of margin and liquidity, a collection of exercise and assignment contingencies, and a state from which a management policy may act. Research adds another identity when the position is converted into features and labels for a model. Each view is legitimate, but their meanings must remain synchronized as the market moves and the portfolio evolves.

Many trading systems materialize these views in different places: the security master knows the listed contract, the risk service maintains Greeks, the payoff tool draws a terminal diagram, the order system records fills, and the research pipeline stores feature rows. The architecture appears modular until two copies disagree, at which point questions that ought to have mechanical answers require forensic work:

- Did the model rank the position the simulator actually traded?
- Did a "*calendar*" mean long the back month or long the front month?
- Did a payoff deduplicator erase an economically important early exercise or liquidity difference?
- Can a result be replayed using only information available at the decision time?

Volt gives those systems a shared semantic source. The public model records signed positions in exact listed option contracts; exercise schedules, settlement rules, observations, actions, emissions, cash, underlying units, and outcomes retain typed identities. Derived payoff and graph views link back to exact portfolio hashes instead of becoming independent copies. Account policy, collateral, futures, brokerage orders, and general portfolio margin are outside the current crate. They may later become inputs to a compiler or application layer without changing the meaning of the implemented option types.

Discovery therefore shapes the representation from the beginning. Legs that share an underlying, expiration, risk factor, or margin treatment interact through relationships that remain unchanged when the legs are displayed in a different order, so a graph provides a natural input to a relational learner while remaining useful to deterministic systems. Validation, canonicalization, scenario evaluation, and graph export already operate over the same portfolio meaning. Tensorization, account-aware masks, strategy macros, and a general compiler remain proposed layers. The current candidate grammar resolves bounded actions against an explicitly supplied listed universe, preserving every matching branch in stable contract order. The scenario layer applies declared fills and fees and delegates lifecycle actions to the deterministic runtime, returning outcomes with an auditable path back to the original contracts.

2 Related work and scope

The graph representation and the compiler serve different goals. The graph is Volt's interface to learning, optimization, simulation, and explanation. The future cog compiler can make that interface efficient and reliable by expanding concise source forms, checking them, and lowering them to runtime, graph, or tensor targets. Compilation matters because it preserves meaning across those targets; it is not the end product.

Machine-readable financial contracts have a substantial research lineage in programming languages and formal methods. RISLA, developed in an academic–industrial collaboration in the 1990s, described interest-rate products through their cash flows and compiled those descriptions into COBOL for use in a banking environment [1]. Peyton Jones, Eber, and Seward subsequently showed that a small collection of contract combinators could express a broad family of financial instruments while supporting several interpretations of the same description, including valuation and contract evolution [2]. Together these projects established the essential compiler idea: a financial product can be represented at a level above any particular pricing routine or operational implementation.

Later research made the semantic guarantees considerably stronger. Bahr, Berthold, and Elsman developed a multi-party contract language with a precise cash-flow semantics, a causality type system, symbolic contract reduction, and Coq-verified analyses and transformations [3]. Annenkov and Elsman then presented a certified compilation path from templated contracts to a payoff intermediate language, with generated Haskell and Futhark code used in a data-parallel Monte Carlo engine [4]. That work is especially close to the compiler boundary developed here because it connects declarative contract meaning, verified lowering, stochastic evaluation, and GPGPU execution without allowing the numerical backend to redefine the contract.

Comparable representations have been used in production derivatives systems. Frankau, Spinellis, Nassuphis, and Burgard [5] describe a functional payout framework for exotic derivatives whose embedded language supported pricing, analysis, and lifecycle processing through several interpreters within a wider front-office system. LexiFi's contract-description language represents the commercial continuation of the algebraic approach [8], while ACTUS and the FINOS Common Domain Model pursue standardization of contract, product, trade, and lifecycle semantics [6, 7]. These efforts answer different institutional questions, yet they share the conviction that product meaning should be explicit enough to interpret, transform, and execute.

Volt begins from that foundation and turns toward portfolio search. Its implemented central object is an exact canonical collection of option positions. A terminal-payoff projection exposes one explicitly weaker equivalence, a graph sample exposes a deterministic learning view, and runtime state preserves lifecycle and ledger consequences. Account state, collateral, learned ranking, and general numerical lowerings are research extensions. Interoperability with broader contract standards may eventually be useful, but the immediate question is whether a compact public semantic core lets other systems explore strategy space without loosening the financial meaning of what they explore.

3 Strategy workflow

At a trader's level, the envisioned process begins with a market view and account constraints, expands them into admissible contracts and actions, evaluates the resulting portfolios, and returns a ranked set of alternatives with their risks and supporting evidence. The public crate currently covers the semantic middle of that path: exact contracts, bounded action resolution, canonical forms, deterministic simulation, journals, and graph samples. Figure 1 develops the larger research workflow, including layers not yet implemented in this repository.

For example, the phrase "*long the 100 call calendar*" conveys a recognizable view but leaves most of the executable position unresolved. Under the usual convention it calls for a short near-dated 100 call and a long far-dated 100 call, after which a future compiler would identify the listed contracts, quantities, multipliers, exercise styles, settlement rules, contemporaneous quotes, and account context. When the near expiration arrives, the short leg may expire, be closed, or assign an underlying position while the far option continues to carry time value and strike- and tenor-dependent volatility risk. A faithful representation must therefore preserve both the compact strategy pattern and the state that remains after its first lifecycle event.

3.1 Options profile

Listed U.S. options provide the initial proving ground because they combine composable payoffs with nontrivial market and lifecycle semantics. The same representation pattern can later include underlying positions, futures, financing, hedges, events, constraints, and other contracts without changing the meaning of the acronym. The near-term objective is more specific: represent, discover, and evaluate portfolios of listed U.S. options from simple primitives:

$$\ell = (u, r, K, T, q, m, \chi, \zeta),$$

where u is an underlying, $r \in \{\text{call}, \text{put}\}$, K is strike, T is expiration, q is signed quantity, m is the contract multiplier, χ is an exercise opportunity specification, and ζ is a settlement specification.

The larger research system would need to:

1. construct only well-typed and admissible contracts and actions;
2. canonicalize exact redundancies without erasing execution differences;
3. simulate American exercise, assignment, expiration, and portfolio continuation;
4. generate reproducible outcome and factor datasets;
5. train models that rank legal strategy transformations;
6. retain enough provenance to explain every result.

Named structures such as *verticals*, *calendars*, and *butterflies* are proposed as trader-facing macros whose expansions would be fully typed. Below that interface, the discovery system operates on legs, rights, state, and transformations, which lets it reproduce a familiar strategy, vary its implementation, or construct a valid portfolio with no conventional name. Part I develops this vocabulary through worked examples and identifies the relationships each graph must preserve. Part II gives those examples precise mathematical and executable semantics, and Part III distinguishes implemented candidate, simulator, and graph contracts from the proposed compiler and integration architecture. Separate papers develop graph-based discovery and sequential portfolio control against those interfaces.

3.2 Path outcomes

For any path, the outcome is determined jointly by:

$$\boxed{\begin{array}{c}\text{contract semantics} + \text{portfolio realization} \\ +\text{world/execution model} + \text{policy} = \text{path outcome.}\end{array}}$$

Contract semantics define rights, obligations, exercise opportunities, and settlement consequences. The portfolio realization identifies the listed instruments, signed quantities, fills, financing, and margin treatment. The world and execution model supplies option quotes, implied-volatility observations, dividends, borrow, transaction costs, fills, assignment events, and scenario probabilities. Finally, the policy selects admissible actions such as hold, hedge, close, roll, or exercise.

Keeping these objects separate prevents assumptions from migrating into the wrong layer. A leg definition establishes the events that its holder and writer may face, the realization records what was actually bought or sold, the world model supplies the prices and events that occurred, and the policy selects the trader's actions along that path. A backtest result is the joint product of all four, so it can be reproduced only when each contribution is identified and versioned.

4 Strategy macros

Named strategies can provide an efficient human interface to the graph. Terms such as "*iron condor*" and "*long call calendar*" carry a recognizable geometry and market view, allowing a trader to specify a useful starting point without enumerating every relationship by hand. The name expands into a small graph template, after which a future compiler would supply the signed quantities, contract identifiers, strike and expiration ordering, settlement conventions, and account-dependent constraints that desk shorthand leaves implicit.

Write $C(K,T)$ for a call and $P(K,T)$ for a put. A positive sign is long and a negative sign is short. Premium, fees, and financing are omitted from the compact leg expressions below; an executable realization adds them. In the diagrams, dark graphite nodes are portfolios, charcoal nodes are signed positions, and light gray nodes are shared economic objects such as underlyings and expirations. The diagrams intentionally omit quote, account, right, and provenance nodes when those would obscure the strategy's basic anatomy.

4.1 Single legs

Position	Leg expression	What the graph must make explicit
Long call	$+C(K,T)$	Bullish convex exposure; premium at risk; holder controls exercise; may create long stock and a cash payment under physical settlement.
Short call	$-C(K,T)$	Premium received; potentially uncapped upside loss; assignment is external to the holder's policy; may create short stock.
Long put	$+P(K,T)$	Downside convexity; premium at risk; exercise may sell or create a short position in the underlying, depending on account inventory.
Short put	$-P(K,T)$	Premium received in exchange for downside obligation; assignment may purchase stock at the strike and consume capital.

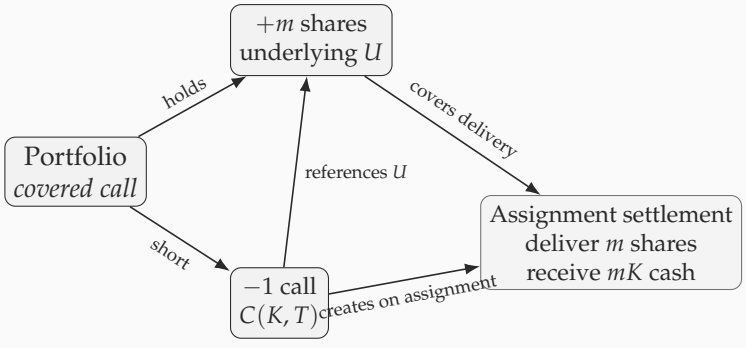
Long and short positions refer to the same listed contract while occupying different places in its lifecycle: the holder owns an exercise decision, whereas the writer faces assignment generated outside its policy. Signed quantity can therefore remain an attribute of the position node, with exercise authority, assignment exposure, and settlement consequences carried by typed relationships that remain visible to validation and simulation.

4.2 *Covered calls*

Coverage arises at the portfolio and account level, where the settlement obligation created by a short option can be linked to resources capable of satisfying it. The option retains its ordinary contract definition; the graph records the additional relationship to shares, cash, or eligible collateral, together with the period during which those resources remain available. For one physically settled equity call with multiplier m, the familiar covered-call position is

$$w_{\text{cc}} = mU - C(K,T), \qquad \Pi_T = mS_T - m(S_T - K)^+ = m\min(S_T, K),$$

before entry premium, dividends, financing, and costs. The long shares remain a distinct position and satisfy the call's deliverable if assignment occurs, at which point the runtime consumes the assigned option, delivers m shares, and receives mK cash. During the life of the trade those shares continue to carry downside exposure, dividends, financing, and stock-specific events, while the short call exchanges appreciation above the strike for its premium and volatility exposure.



Other forms of coverage follow by changing the obligation and the typed resource edge. A cash-secured short put links its possible assignment debit, commonly mK, to reserved cash or eligible collateral, after which assignment converts the option into shares and a cash posting. Determining whether the relationship remains valid requires the compiler to track the coverage ratio, haircut, currency, availability interval, and any competing pledge of the same resource. A delta hedge belongs to a different relationship because it reduces economic sensitivity without guaranteeing the asset or cash required at settlement.

Cash-settled index options make the distinction concrete. A short index call may owe $m(I_T - K)^+$ in cash and therefore links to funding capacity or collateral; no shares of the index are deliverable. An ETF, future, or basket can offset part of the market exposure, subject to basis and tracking risk, so the graph connects it through an *economic-hedge* edge. Separate edge types for covered delivery, funded settlement, collateral eligibility, and economic hedging let risk and simulation reason about each relationship on its own terms.

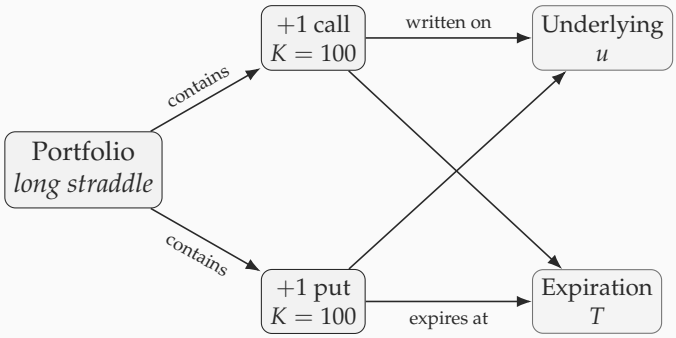
4.3 Multi-leg structures

Strategy	Graph macro	Trader interpretation
Long straddle	$+C(K,T) + P(K,T)$	Long movement around one strike. Initially long gamma and vega and usually short theta; direction must be managed or intentionally retained.
Short straddle	$-C(K,T) - P(K,T)$	Short movement and time value, with large risk outside the body and two independent assignment paths.
Long strangle	$+P(K_1,T) + C(K_2,T),\ K_1 < K_2$	A wider, usually cheaper long-volatility structure whose movement must first cross one of two strikes.
Short strangle	$-P(K_1,T) - C(K_2,T),\ K_1 < K_2$	A short-volatility range position with substantial but bounded loss as the underlying approaches zero, uncapped upside risk through the call, and assignment exposure on both legs.
Call butterfly	$+C(K_1,T) - 2C(K_2,T) + C(K_3,T)$	A bounded, body-centered exposure, usually with $K_2 - K_1 = K_3 - K_2$.
Iron condor	$+P(K_1,T) - P(K_2,T) - C(K_3,T) + C(K_4,T)$	A defined-risk short-range position with ordered strikes $K_1 < K_2 < K_3 < K_4$.
Long call calendar	$-C(K,T_N) + C(K,T_F),\ T_N < T_F$	Short near-dated and long far-dated time value at one strike, creating a term-structure and path-dependent exposure.
Long put calendar	$-P(K,T_N) + P(K,T_F),\ T_N < T_F$	The put analogue, with additional practical sensitivity to downside skew, dividends, and early exercise.

Desk usage of “long” and “short” can become ambiguous for *calendars* and iron structures because the adjective may refer to premium, volatility, the body, or the wings. The graph resolves the convention through signed quantities, ordered strikes, expiration order, and cash-flow direction, so the expanded structure remains stable when terminology varies.

4.4 *Long straddle*

Consider one long 100-strike call and one long 100-strike put sharing an underlying and expiration. A leg table records the two positions, while the graph additionally expresses their common market objects and preserves the distinct exercise right attached to each contract.



Ignoring premium, the expiration payoff is

$$(S_T - 100)^+ + (100 - S_T)^+ = |S_T - 100|.$$

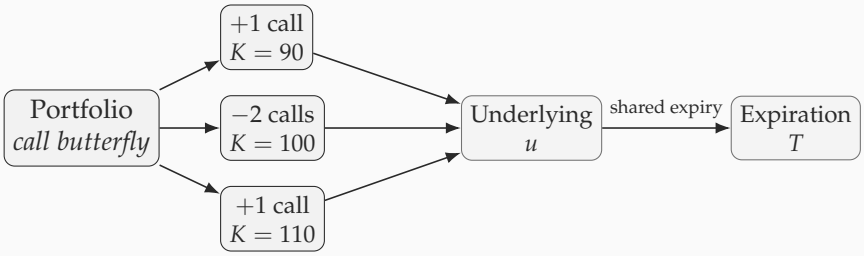
The compact V-shaped payoff captures the expiration geometry and provides a useful canonical projection. Entry implied volatility and spread, interim delta hedges, discrete gamma capture, and American exercise decisions enter through the market, policy, and lifecycle portions of the graph, allowing two realizations with the same terminal shape to produce different path outcomes.

4.5 *Butterfly* and *iron condor*

Take a symmetric 90–100–110 *call butterfly*:

$$+C(90, T) - 2C(100, T) + C(110, T).$$

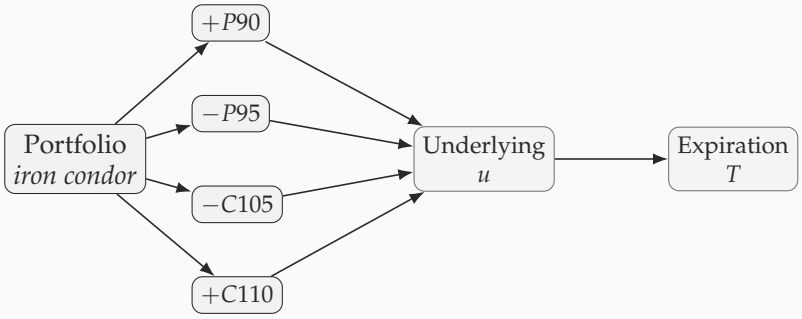
Ignoring premium, it pays zero at or below 90 and at or above 110, rising linearly to 10 at the body strike before falling symmetrically to zero. Three leg nodes describe the structure, while the quantity of -2 at the body produces a gross contract count of four.



Now take a 90–95–105–110 *iron condor*:

$$+P(90, T) - P(95, T) - C(105, T) + C(110, T).$$

If its entry credit is c, expiration profit before fees remains c between 95 and 105, declines linearly through either five-point wing, and reaches a maximum loss of $5 - c$ beyond the long strikes. The graph makes the two defined-risk verticals visible as parts of one position sharing an account, underlying, and expiration:



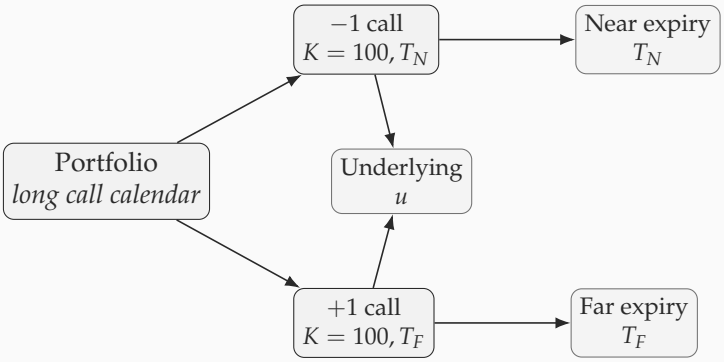
Both structures produce bounded expiration payoffs organized around ordered strikes, which may place them near one another under a chosen scenario metric. Their implementations retain materially different economics: the condor is commonly entered for a credit, spans puts and calls, uses four distinct contracts, and exposes two short legs to assignment, whereas the *call butterfly* concentrates two short contracts at one strike. Behavioral clustering can use the similarity to allocate search effort, while execution and lifecycle identity continue to distinguish the portfolios.

4.6 *Calendar*

A *long 100 call calendar* is conventionally

$$-C(100, T_N) + C(100, T_F), \qquad T_N < T_F.$$

At T_N, the near call expires or is assigned while the far call still has time value and remains sensitive to spot, volatility, rates, and dividends, which makes the state at the first expiration an input to the continuing portfolio. Its economics therefore require a path through at least two expirations and cannot be summarized by the same-expiration terminal-payoff quotient used for a *vertical* or *butterfly*.



In trading terms, the *calendar* expresses a view on relative time value and on changes in implied volatility across its two expirations. A sufficiently large move away from the strike can weaken the desired decay relationship, while roll, pin, assignment, and term-structure risks determine how the trade behaves around the near expiration. A useful backtest must process that event, carry the far leg and any assigned inventory into the next state, and value the continuing portfolio under the market information then available.

The examples establish a progression that matters for the remainder of the paper. A *straddle* shows why shared underlyings and expirations belong in the representation; a *butterfly* and an *iron condor* show how canonical payoff structure can coexist with distinct execution and assignment paths; and a *calendar* shows why a portfolio that crosses expirations cannot be reduced to a single terminal diagram. Covered positions add inventory, collateral, and settlement resources to the same picture. The strategy names remain valuable because they compress a large amount of desk convention into a phrase, while the expanded graphs provide the precision required by a future compiler and a learning system.

A trader may therefore enter through a named macro, inspect its resolved contracts and obligations, and compare variations generated from the underlying grammar. Search may arrive at the same graph through a sequence of primitive actions, or it may discover a valid portfolio for which no standard name exists. In both cases the expanded object carries the economic and executable meaning; the familiar vocabulary supplies useful entry points without limiting the set of portfolios the system can express.

PART II

Language and Runtime

5 Portfolio algebra

5.1 Portfolio module

Let $\mathcal{L}$ be a finite set of listed contracts available at a decision time. A portfolio is a sparse integer vector

$$w \in \mathcal{P} = \mathbb{Z}^{|\mathcal{L}|}.$$

Component w_i gives the signed number of contracts in leg i, and ordinary addition combines books or applies a trade to an existing position, so the unconstrained space forms an abelian group. Trading rules select a state-dependent feasible subset

$$\mathcal{F}(x) \subseteq \mathcal{P}$$

through the account state x, the instrument universe, position limits, margin, and available liquidity. Long-only restrictions, asymmetric margin rules, and finite capital can turn this feasible region into a monoid or constrained cone even though the ambient position algebra remains a group.

Trader interpretation. The notation formalizes the familiar act of building a book by adding and subtracting positions: buying contributes $+1$, selling contributes -1, and a closing order offsets some existing quantity. The free mathematical book describes every integer combination that the algebra can express, after which $\mathcal{F}(x)$ imposes the practical requirements that the contract be listed, liquid, permitted, and affordable in the current account.

5.2 Payoff quotient

For a fixed underlying and expiration, define the terminal payoff map

$$\Phi_T : \mathcal{P} \longrightarrow \mathcal{H}_T,$$

where $\mathcal{H}_T$ is a space of payoff functions of terminal spot. Two portfolios are terminal-payoff equivalent if

$$w_1 \sim_T w_2 \quad \Longleftrightarrow \quad \Phi_T(w_1) = \Phi_T(w_2).$$

The distinct payoff search space is the quotient

$$\mathcal{P} / \ker(\Phi_T).$$

The quotient removes ordering, duplicated legs, offsetting quantities, and other exact terminal-payoff redundancies, thereby giving search a smaller set of payoff shapes to explore. Its scope remains precise: American rights, assignment exposure, liquidity, margin, funding, taxes, and the path of mark-to-market values live outside this terminal projection and can distinguish two portfolios that occupy the same quotient class.

Suppose, for example, that an account buys one call and later sells the identical contract. The open-position vector and terminal payoff both reduce to zero, which is the correct canonical representation of the remaining holding, while the execution journal retains two fills, their possibly different prices, and both sets of fees. Canonicalization can therefore remove redundancy from the object being compared without erasing the evidence needed to reconstruct how the account reached it.

5.3 Canonical hinge form

For European vanilla options on one underlying and expiration, every static portfolio can be written

$$\Pi(S_T) = a + bS_T + \sum_{i=1}^{n} c_i (S_T - K_i)^+, \qquad K_1 < \cdots < K_n.$$

Here a is cash, b is linear underlying exposure, and the c_i are changes in payoff slope at ordered strike knots.

The representation follows from

$$(K - S)^+ = K - S + (S - K)^+.$$

Thus a put with signed share-equivalent weight q contributes

$$\Delta a = qK, \qquad \Delta b = -q, \qquad \Delta c_K = q,$$

while a call contributes only $\Delta c_K = q$.

The distributional curvature is

$$\frac{\mathrm{d}^2 \Pi}{\mathrm{d} S_T^2} = \sum_i c_i \delta_{K_i}.$$

The ordered strike-weight sequence is therefore a sparse and canonical description of terminal convexity.

The named examples from Part I become small coefficient vectors:

Same-expiry strategy	Cash a	Linear b	Nonzero hinge weights c_K
Long 100 call	0	0	$c_{100} = 1$
Long 100 put	100	-1	$c_{100} = 1$
Long 100 straddle	100	-1	$c_{100} = 2$
90–100–110 call butterfly	0	0	$c_{90} = 1,\ c_{100} = -2,\ c_{110} = 1$
90–95–105–110 iron condor	-5	0	$c_{90} = 1,\ c_{95} = -1,\ c_{105} = -1,\ c_{110} = 1$

The coefficient table shows how canonicalization saves learning and search capacity. Different leg orderings, duplicated entries, and offsetting routes collapse to the same $(a, b, \{K : c_K\})$ representation, encoding permutation invariance directly instead of asking the learner to infer it repeatedly from examples.

Trader interpretation. The hinge weights are changes in the slope of the payoff diagram: a positive weight bends the line upward at its strike, while a negative weight bends it downward. The *butterfly*'s $+1, -2, +1$ sequence produces the familiar tent; the *iron condor*'s $+1, -1, -1, +1$ sequence produces a flat center and bounded wings. These compact coefficient patterns are both mathematically canonical and recognizable to a trader.

5.4 Portfolio and payoff identity

The implementation keeps:

1. an *economic projection*, such as the canonical terminal payoff;
2. an *execution realization*, containing the actual contracts.

This separation allows search to discover an attractive economic behavior and then optimize the contracts and orders used to implement it. In trader terms, the first identity answers "what exposure do I want?", while the second asks "which realization gives me that exposure on acceptable terms?" A narrow, liquid *butterfly* and a wider, less liquid structure may sit close together on one scenario grid even as their spreads, fill probabilities, assignment paths, and margin requirements diverge. Preserving both identities lets the system exploit the similarity and still choose between executable realizations.

5.5 Transaction costs

Adding a and then $-a$ returns the open-position vector to zero, giving the position algebra a symmetry that disappears once the trades are executed. If $\mathcal{C}_t(a)$ denotes the cash consequence of trading a at time t, a market with spread, fees, or impact generally satisfies

$$a + (-a) = 0 \qquad \text{but generally} \qquad \mathcal{C}_{t_0}(a) + \mathcal{C}_{t_1}(-a) < 0.$$

For an immediate round trip of q contracts at displayed bid b, ask a, multiplier m, and total fees f, the simplest loss is

$$-|q|m(a - b) - f.$$

The realized cost may be larger once market impact, partial fills, legging risk, financing, and hedge turnover are included.

Canonicalization must consequently preserve the execution history alongside the current holding. Two routes may reach zero or the same hinge vector while producing different cash paths: one might fill a four-leg structure as a package, another might leg into it after buying a temporary hedge, and a third might exhaust displayed size and move through the book. Assignment can add further stock or cash transactions whose quantities and timing are absent from the entry algebra.

Transaction costs thus alter the shape of the search problem rather than subtracting a universal constant from every result. They can reverse rankings, break apparent long/short symmetry, make a frequently rebalanced hedge inferior to a coarser policy, and place payoff-equivalent realizations in different capital or liquidity budgets. Holdings remain available for exact semantic deduplication, while dominance tests and learning labels operate on net outcomes produced by an explicit execution model.

6 Lifecycle semantics

6.1 American exercise

Let $\mathcal{T}_\chi$ be the set of stopping times permitted by an exercise opportunity specification χ. Ignoring portfolio coupling, a holder's value has the form

$$V_0 = \sup_{\tau \in \mathcal{T}_\chi} \mathbb{E}[D_{0,\tau} h(S_\tau)] .$$

The familiar styles are restrictions of $\mathcal{T}_\chi$:

$$\begin{aligned} \text{European:} \quad & \mathcal{T}_\chi = \{T\}, \\ \text{Bermudan:} \quad & \mathcal{T}_\chi = \{t_1, \ldots, t_n = T\}, \\ \text{American:} \quad & \mathcal{T}_\chi = \{\tau : 0 \le \tau \le T\}. \end{aligned}$$

Numerical simulation realizes the continuous American opportunity set on an explicit event or time grid, producing a Bermudan approximation whose spacing, calendar, and event alignment become part of the experiment manifest. This choice affects both value and policy because an exercise opportunity omitted from the grid cannot be recovered later by the learner.

Exercise is represented as a consumable right:

```
ExerciseRight {
    holder,
    valid_time_predicate,
    exercise_guard,
    settlement_consequence,
    quantity_remaining,
}
```

Authority separates the two sides of the same listed contract. The long holder controls exercise within $\mathcal{T}_\chi$, while the short writer controls orders to hold, resize, or close and receives assignment as an external event generated by another holder or the clearing process. For a short put, that event may consume the option, debit strike cash, and create long stock in the account. Encoding authority in the action type prevents a learning agent from being offered "assign my short put" as a voluntary decision, while guarded state transitions make the stopping-time definition operational and keep the resulting cash, assets, and rights explicit.

6.2 State

At event t, state is decomposed as

$$x_t = (x_t^C, x_t^P, x_t^M, x_t^A, x_t^R),$$

where the superscripts denote contract, portfolio, market, account, and protocol/rule state. The initial implementation uses only the components required by the reference tests, while preserving this boundary.

6.3 Transitions

The primitive operation is

$$\begin{aligned} \phi(x_t, a_t, \omega_{t+1}) &\Longrightarrow (x_{t+1}, e_{t+1}), \\ x_{t+1} &= T(x_t, a_t, \omega_{t+1}), \\ e_{t+1} &= F(x_t, a_t, \omega_{t+1}). \end{aligned}$$

An observation ω has an effective market time, source/provenance, and the values required by the fired rule. An action a has an authority: holder, counterparty, system, or policy.

The transition emits typed economic consequences so later components can apply accounting and risk rules to their actual meaning. The first release distinguishes:

- cash postings;
- underlying-asset postings;
- consumed rights.

Lifecycle changes are represented in the next `PortfolioState`, not as a fourth emission type. Later releases must distinguish creation of an obligation, settlement instruction, completed transfer, failure, reversal, and correction.

As a concrete example, suppose an account is short one physically settled 100-strike put with multiplier 100. A full assignment observation is checked against the contract, the event authority, and the remaining position; once accepted, the transition consumes the short option and emits $+100$ shares with a $-\$10{,}000$ cash posting. Those shares enter the next portfolio state and continue to affect risk, margin, financing, and every subsequent policy decision after the option has terminated.

6.4 Physical settlement

For $n > 0$ long call contracts with multiplier m, physical exercise emits

$$\Delta Q_u = nm, \qquad \Delta C = -nmK.$$

For a long put,

$$\Delta Q_u = -nm, \qquad \Delta C = nmK.$$

Assignment of a short position emits the opposite postings, whereas cash settlement emits signed intrinsic value and leaves the underlying inventory unchanged. In either case the event transforms a portfolio that continues beyond the terminating option, potentially with new stock and cash balances that later actions must manage.

6.5 Replay

Every accepted transition produces a journal entry containing:

```
schema_version
sequence
prior_state_hash
next_state_hash
action
observation
typed_emissions
rule_id
entry_hash
```

Hashes are version-pinned:

$$H = \mathrm{SHA256}(\text{schema namespace} \| 0\text{x}00 \| \text{canonical JSON}).$$

The hash identifies canonical content under a particular schema version, so a semantic schema change is expected to produce a new value. Replay requires the stronger and more useful property that the same versioned definitions and input journal reproduce the same sequence of states and emissions.

7 Equivalence

The system must state which equivalence relation is being used:

1. **Definition equivalence**: identical canonical contract IR.
2. **Dynamic path equivalence**: identical state transitions and economic consequences for every admissible input path.
3. **Terminal-payoff equivalence**: identical payoff at a selected horizon.
4. **Model-relative value equivalence**: equal value under a named world model and valuation functional.
5. **Approximate behavioral similarity**: nearby outcome vectors across a specified scenario distribution and metric.

Within a supported finite or symbolic fragment, proven instances of the first three relations can justify exact deduplication. Model-relative equality and approximate behavioral similarity instead guide clustering, search allocation, and implementation choice while preserving each portfolio's semantic identity.

PART III

Learning and Integration

8 System cycle

The options graph is the exchange boundary between authoritative financial processing and statistical estimation. It exposes portfolio structure to models while the runtime remains responsible for what contracts, actions, and outcomes mean. Solid arrows carry versioned inputs and artifacts forward; dashed graphite arrows return selected candidates or actions to the runtime for new evidence. The public release implements the typed portfolio, bounded grammar, simulation, journal, and graph-sample pieces. Account constraints, complete legal masks, learned components, and the feedback orchestrator in this figure remain application or research work.

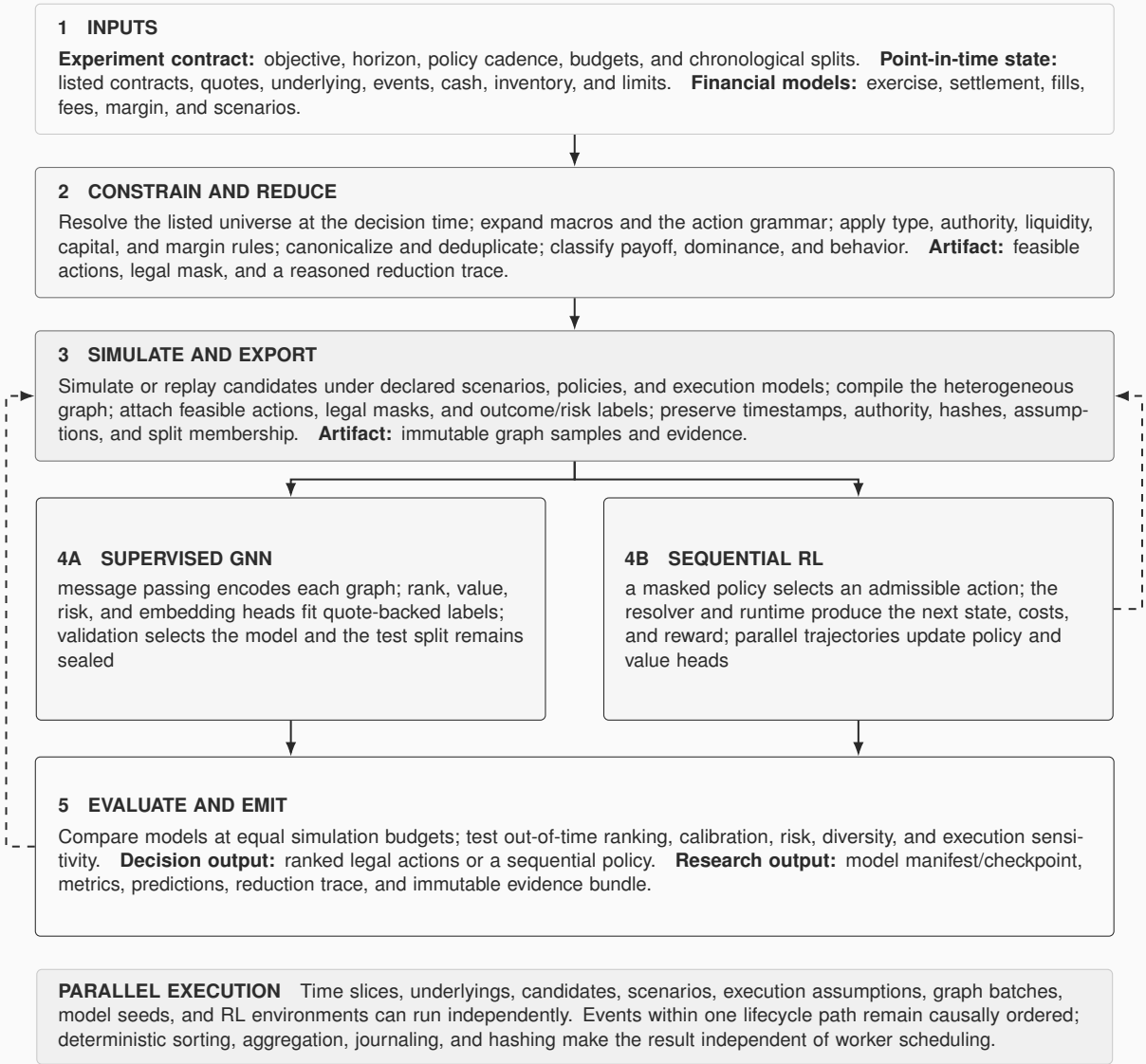


Figure 1. The proposed Volt preparation, learning, evaluation, and feedback cycle. Solid semantic components exist in the public crate; learned ranking, sequential control, and full orchestration are not release claims.

In the proposed application boundary, a compiler would emit a graph G_t, an admissible action set A_t, and its legal mask M_t. Simulation or replay may add outcome labels y_t and an evidence record p_t, producing

$$d_t = (G_t, A_t, M_t, y_t, p_t).$$

The current `PortfolioGraphSample` covers the graph component, and `ScenarioSimulator` can produce deterministic outcomes. It does not yet combine those artifacts with an account-aware action mask in one dataset record. The graph schema may evolve with an experiment, but exact contract definitions, portfolio state, action authority, and journals remain governed by the versioned Volt runtime.

Graph-based ranking and sequential control consume this boundary in different ways. A discovery model scores portfolios or one-step transformations from a fixed decision state; a control policy chooses repeatedly as fills, market events, exercise, assignment, and settlement alter the state. Both remain clients of the same resolver and simulator. Companion discovery and control papers own their model architectures, training losses, rollout schedules, and empirical comparisons.

Historical slices, candidates, scenario paths, model seeds, and independent environments could be distributed across workers. Event order within one portfolio path remains causal, after which stable identifiers, deterministic sorting, and hashing assemble the partial results. Parallelism may change elapsed time without changing the artifact that a future compiler identifies.

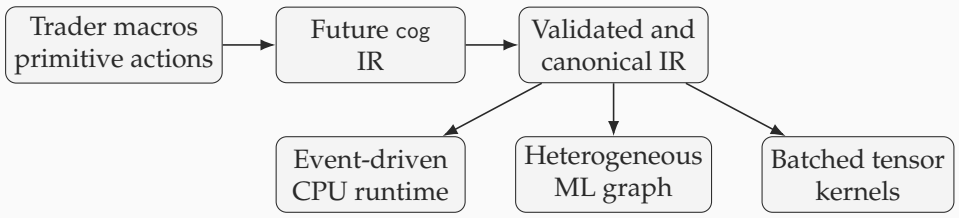
9 Compiler as an efficiency layer

The compiler is an important supporting layer, not Volt's organizing thesis. Its job is to construct the graph and its executable views efficiently while preserving their declared meaning. The name `cog` is reserved for this future Volt compiler. Its source may eventually be a named strategy macro, an existing portfolio snapshot, or a sequence of primitive actions, resolved into an intermediate representation containing typed contracts, positions, relationships, account context, and lifecycle state.

The present crate does not claim that general compiler. It implements one narrow pass, `CandidateGrammar::compile_action`, which turns a typed candidate action into zero or more exact listed-contract branches. It also provides separate validated lowerings to `TerminalPayoffProjection`, `PortfolioState`, and `PortfolioGraphSample`. The complete pass sequence below is a roadmap for coordinating those pieces with future macro, constraint, and tensor layers.

The principal passes are:

1. **expand (planned):** turn a macro such as `IronCondor(90,95,105,110)` into signed call and put legs;
2. **type-check**: verify right, multiplier, exercise authority, settlement, strike ordering, and expiration relationships;
3. **resolve**: select contracts that were actually listed and observable at the decision time;
4. **constrain (planned):** apply liquidity, position, capital, and account rules;
5. **canonicalize**: merge exact offsets and compute economic projections without discarding execution identity;
6. **lower (partly implemented):** emit the representation required by a simulator, graph learner, optimizer, API, or numerical kernel;
7. **attach evidence (partly implemented):** retain semantic hashes, source times, assumptions, and reduction reasons.



Code generation is one possible final pass. The nearer-term value of `cog` would be a single diagnostic and artifact boundary over semantic operations that Volt already exposes separately. Until that language, diagnostics model, artifact format, and execution contract exist, the short name remains reserved rather than attached to one example program.

9.1 GPU lowerings

Some workloads a future compiler could produce are naturally parallel:

- evaluate many portfolios over the same spot/volatility scenario grid;
- price or compute sensitivities for large homogeneous option batches;
- aggregate leg values into portfolios with segmented reductions;
- perform message passing over batches of similarly typed graphs;
- score a large set of already validated candidate actions.

These can be lowered to structure-of-arrays buffers, sparse adjacency, masks, and dense or ragged tensors. Stable node and edge type identifiers could let the compiler batch many small strategy graphs without confusing their portfolio boundaries.

Contract validation, corporate actions, early exercise, external assignment, corrections, and journal construction have a more irregular, control-heavy shape and may remain on the CPU while numerical kernels operate on compiled batches. The eventual boundary should be chosen empirically by measuring end-to-end throughput, transfer cost, numerical reproducibility, and operational complexity. Volt's versioned semantic objects make such an experiment possible, but no GPU lowering is implemented in the current public repository.

10 Candidate reduction

The action grammar presents the agent with semantic coordinates that remain stable as listed series appear and expire:

$$a = (\text{operator}, \text{right}, \text{tenor bucket}, \text{moneyness/delta bucket}, \Delta q).$$

A point-in-time resolver converts that request into a contract present in the listed universe supplied by the caller. Volt validates the clock, reference spot, and externally supplied delta coordinates; it does not certify the caller's data completeness or account eligibility. This separation keeps transient symbol selection in a deterministic semantic pass while allowing a policy to reason in coordinates such as tenor, moneyness, right, and quantity.

The public crate implements the first four layers below; the remaining layers belong to callers or future compiler work:

1. typed grammar and authority rules;
2. point-in-time instrument availability;
3. exact branch resolution in stable contract order;
4. exact canonicalization;
5. *external:* liquidity, capital, margin, and account constraints;
6. *external:* dominance, clustering, and learned ranking.

The implemented operators are `AddLeg`, `RemoveLeg`, `ResizeLeg`, `Wait`, and holder-authorized `Exercise`. Close, roll, hedge, move-strike, move-expiration, and multi-leg macros are not current grammar variants. In trader terms, a caller can request "add one put in this tenor and moneyness bucket"; Volt returns every matching exact contract branch, and caller policy decides which branches remain feasible.

10.1 Public synthetic trace

The checked-in `zap` example makes two semantic reductions concrete without relying on private market data:

Experiment	Raw realizations	Exact portfolios	Payoff classes
Two ordered actions	144	73	73
Six quantities in $[-2, 2]$	15,625	15,625	13,321

Exact canonicalization removes 49.31% of the ordered action representations. In the bounded quantity experiment, terminal projection merges 14.75% of exact portfolios into an existing payoff class. Those classes are projections, not claims of dynamic American-option equivalence. The companion `graph` example enumerates all $3^6 = 729$ portfolios for quantities in $\{-1, 0, 1\}$ and emits deterministic JSON Lines graph fixtures. These are functional synthetic results, not performance or investment evidence.

10.2 Runnable risk monitor

The checked-in `risk` example constructs a cash-settled European *iron condor* as four exact `OptionContract` values, passes their signed positions through `CanonicalPortfolio`, and derives the terminal hinge projection. It then advances a correlated spot and mean-reverting log-volatility path. An example-local Black–Scholes adapter produces educational marks and Greeks; it is not part of Volt's price authority.

The top-like display reports leg marks and P&L, portfolio delta, gamma, vega, and theta, expiration P&L across spot shocks, and a spot–volatility scenario grid. The separation is visible in the running program: Volt owns contract identity, canonicalization, and terminal payoff, while the example owns its synthetic market path and teaching pricer.

11 Simulation

11.1 Managed paths

A terminal payoff chart shows the contract cash flow at a selected expiration and terminal spot. A trader's backtest asks more:

1. At what executable prices could the legs have been opened?
2. How were options still open marked as spot, time, and volatility moved?
3. Did the account hedge, take profit, stop out, close, roll, exercise, or receive assignment?
4. Which cash, stock, fees, financing, and margin consequences followed?
5. What information was actually available at every decision?

A *iron condor* held to expiration, the same structure closed after earning half its original credit, and the same initial position rolled when a short strike is breached represent three policies over one contract realization. They share the entry evidence and initial graph, then diverge through their actions, transaction costs, lifecycle events, and continuing positions; a backtest must preserve that common origin while producing distinct transition paths.

The public `ScenarioSimulator` makes execution assumptions explicit. `DeclaredMark` uses the declared mark as a synthetic or proxy fill; `CrossAtTouch` buys at the ask and sells at the bid and fails closed when the required side is absent. The included tests establish deterministic accounting and sensitivity to missing quote evidence. They do not include a historical ranking study or support an economic-performance claim.

11.2 Information boundary

Every decision is evaluated under an explicit filtration. At observation time t, the feature builder may use only values published and available by that time, and each dataset row records the timing and version information needed to enforce that rule:

- event time and ingestion/as-of time;
- source and revision identifier;
- instrument universe visible at the decision time;
- quote/fill convention;
- exercise and assignment convention;
- corporate-action and dividend version.

11.3 Execution

The simulator carries theoretical valuation and executable cash consequences as separate fields, enabling diagnostic comparisons without allowing a mark to become an implicit fill. Its minimum execution contract covers bid–ask treatment, fees, multiplier, physical or cash settlement, American exercise, short-assignment scenarios, and corporate actions; margin, borrow, market impact, and probabilistic fills enter as independently versioned models whose effects can be isolated.

For a quoted market, three useful but distinct calculations are:

Proxy mark
: A convenient research valuation that may come from an aggregate close or model, used for comparison and diagnostics without an execution claim.

Quote midpoint
: A contemporaneous reference between bid and ask that measures location within the quoted market while carrying no promise of a fill.

Cross at touch
: Buy at the observed ask and sell at the observed bid, bounded by displayed size and the rest of the fill model. This convention charges the displayed spread and provides a conservative oracle relative to midpoint; observed executions require separate fill evidence.

11.4 Outcomes

The learning target is a vector that preserves both reward and the risks or resources through which it was obtained:

$$y = (\text{net P\&L}, \text{drawdown}, \text{CVaR}, \text{turnover}, \\ \text{capital use}, \text{exercise loss}, \text{scenario vector}).$$

Search objectives are declared separately and may reduce this vector to a scalar utility or retain several dimensions in a Pareto comparison. Because the outcome schema sits downstream of the contract IR, researchers can change the objective without changing the financial semantics of the candidate.

12 Evidence and limits

12.1 Explanation

The journal establishes which definitions, guards, transitions, rewrites, and emissions produced a state. Financial attribution then reconciles P&L and risk to legs, scenarios, actions, and costs. When a learned score enters the decision, its diagnostics—feature and relation sensitivity, ablations, and rank stability—remain attached as a separate record describing the model's behavior. The simulator journal continues to own the accounting explanation.

These evidence objects may be rendered as prose, tables, or interactive views. Every quantitative statement retains a link to its source object, so the presentation can change without blurring the difference between a model explanation and a financial reconciliation.

12.2 Limits

Canonicalization removes duplicated representations while preserving the economic dimensions introduced by multi-asset path dependence, stochastic volatility, transaction costs, assignment behavior, and portfolio margin. Those features create real state and prevent a finite canonicalizer from offering general program equivalence or a universally minimal Markov representation. Each verification claim therefore names the symbolic or finite fragment in which it holds and carries its assumptions into the evidence record.

The grammar, dataset, scenario model, and objective also bound what discovery can find. A learned search procedure is particularly effective at locating regularities in its evaluation environment, including accidental leakage, optimistic fill assumptions, and simulator defects, so novel output must face adversarial scenarios, explicit costs, capacity analysis, and out-of-time evaluation before its apparent utility is interpreted economically.

12.3 Implementation

Public package
: The Cargo package is `volt-options`; consumers import the Rust library as `volt`.

Contracts and identities
: `OptionContract`, `Position`, `CanonicalPortfolio`, and `TerminalPayoffProjection` implement exact and projected identities with versioned semantic hashes.

Runtime and scenarios
: `VoltRuntime` is the public alias for the deterministic lifecycle runtime. `ScenarioSimulator` records fill and cost models, journals, pathwise P&L, leg attribution, and result hashes.

Search and graph views
: `CandidateGrammar` compiles a bounded action vocabulary; `PortfolioGraphSample` exports portfolio, underlying, expiry, and option-position nodes with exact and payoff hashes. The `zap` and `graph` examples provide reproducible synthetic fixtures, while `risk` provides a live educational strategy monitor.

Outside this repository
: Brokerage connectivity, live vendor adapters, account and margin policy, strategy macros, the `cog` compiler, tensor or GPU lowerings, training loops, learned checkpoints, and trading authority are not part of the public release.

13 Conclusion

Volt makes the options graph the common object shared by deterministic financial processing and statistical systems:

$$\text{typed contracts and portfolio state} \longrightarrow \text{options graph} \longrightarrow \text{learning, search, and optimization} \longrightarrow \text{evaluated actions and runtime evidence}$$

Typed semantics remove invalid states and duplicated constructions before expensive evaluation. Simulation supplies outcomes under declared market and execution assumptions, after which a discovery or control system may allocate its evaluation budget without acquiring authority over contract meaning. Provenance on definitions, observations, transitions, and labels keeps the result replayable as models and search methods change. A future cog compiler can expand concise strategy descriptions and lower validated graphs to runtime or tensor targets more efficiently. That is a major engineering sub-goal, but the graph remains the representation that makes the resulting learning and optimization possible.

For a trader, the machinery should resolve into a coherent workflow: state a view, inspect the contracts and obligations through which it will be expressed, compare feasible alternatives, simulate an explicit management policy, and trace the resulting P&L and risk back to market evidence. The machine sees the same strategy as typed, canonical, graph-shaped, batchable data available to several optimization backends. Volt provides the shared language between those perspectives, allowing a discovered structure to move from search to simulation and explanation without being reinterpreted at every boundary.

References

[1] B. R. T. Arnold, A. van Deursen, and M. Res, *An Algebraic Specification of a Language for Describing Financial Products*, ICSE-17 Workshop on Formal Methods Application in Software Engineering, pp. 6–13, 1995.

[2] S. Peyton Jones, J.-M. Eber, and J. Seward, *Composing Contracts: An Adventure in Financial Engineering*, ICFP, 2000.

[3] P. Bahr, J. Berthold, and M. Elsman, *Certified Symbolic Management of Financial Multi-party Contracts*, ICFP, pp. 315–327, 2015, https://doi.org/10.1145/2784731.2784747.

[4] D. Annenkov and M. Elsman, *Certified Compilation of Financial Contracts*, PPDP, 2018, https://doi.org/10.1145/3236950.3236955.

[5] S. Frankau, D. Spinellis, N. Nassuphis, and C. Burgard, *Commercial Uses: Going Functional on Exotic Trades*, Journal of Functional Programming, 19(1), pp. 27–45, 2009, https://doi.org/10.1017/S0956796808007016.

[6] ACTUS Financial Research Foundation, *Algorithmic Contract Types Unified Standards: Fundamentals*, https://www.actusfrf.org/methodology.

[7] FINOS, *Common Domain Model: Event Model*, https://cdm.finos.org/docs/event-model/.

[8] LexiFi, *LexiFi's Contract Description Language*, https://www.lexifi.com/blog/structured-thoughts/contract-description-language/.

[9] The Options Clearing Corporation, *Characteristics and Risks of Standardized Options*, June 2024.