

Numerical Parity and Strategy Semantics for Independent Pine Script™ Runtimes

HOOX Research / PYNE Contributors
jango_blockchained
Independent Open Source Research
contact@hoox.sh

August 2026

Abstract

Independent reimplementations of financial domain-specific languages (DSLs) must answer a harder question than syntactic coverage: do computed series and backtest outcomes remain faithful under floating-point arithmetic, dual execution hosts, and order-fill semantics? We present a validation framework for PYNE, an independent open-source toolchain for the Pine Script™ language, covering numerical indicator accuracy, interpret↔compile plot parity, honest `request.security` / `na` policy, strategy broker semantics (entries, exits, commission, slippage, pyramiding, pending fills), alerts, and drawing resource limits. Across 85+ technical analysis indicators and 181 validated builtins, measured relative errors are on the order of 10^{-5} percent class claims (maximum reported 0.0022% on ADX under extreme volatility; mean $< 0.0005\%$; zero systematic bias; exact agreement for deterministic SMA/sum families) under IEEE 754 double arithmetic [IEEE Computer Society, 2019, Higham, 2002]. A dual-host harness compares `Runtime.run(..., mode="interpret")` against `mode="compile"` with nan-aware `allclose`, bucketizing residuals into OK, structural fill/hline-only differences, matched dual errors, and true value MISMATCH. Shared formula alignment for RSI (Wilder), ROC, WMA, cumulative sum, and highest/lowest bar indices reduces host drift to float noise ($\sim 10^{-11}$ absolute on aligned kernels). We formalize a security NA honesty policy (same-symbol simple OHLCV may passthrough; foreign/complex requests resolve to `na` rather than inventing series), sketch the order lifecycle including F2 pending-fill VWAP under pyramiding ≤ 0 , and report strategy golden cases for OCA, risk enforcement, and closed-trade accounting. We do *not* claim TradingView platform certification; residual mismatch tails and nested-EMA seed families remain compatibility work items.

Keywords. numerical validation; floating-point parity; trading strategy backtesting; order fill semantics; dual-runtime consistency; technical indicators.

Disclaimer. Pine Script™ and TradingView® are trademarks of TradingView, Inc. PYNE is an independent, unofficial implementation and is not affiliated with, authorized by, sponsored by, or endorsed by TradingView, Inc.

Contents

1	Introduction	4
2	Threat Model for Numerical Divergence	4
2.1	Formula drift	5
2.2	Seed and warm-up differences	5

2.3	NA and multi-asset handling	5
2.4	Floating-point non-associativity	5
3	Indicator Semantics and Reference Model	5
3.1	Series model	5
3.2	Wilder RSI	6
3.3	Deterministic families	6
3.4	Bollinger and envelope constructions	6
4	Validation Methodology	6
4.1	Campaign scope	6
4.2	Data regimes	6
4.3	Error metrics	7
4.4	Acceptance thresholds	7
4.5	Bias tests	7
4.6	Algorithm: single-indicator validation	7
5	Dual-Host Parity Framework	7
5.1	Motivation	7
5.2	Harness design	8
5.3	Residual taxonomy (buckets)	8
5.4	Shared formula alignment (2026)	9
5.5	Algorithm: dual-host compare	9
6	NA and request.security Honesty Policy	9
6.1	Design principle	9
6.2	Formal rules	10
7	Strategy Execution Model	10
7.1	Order lifecycle	11
7.2	Fill price with slippage	11
7.3	Commission models	11
7.4	Default quantity	11
7.5	Pyramiding and pending fills (F2 Round 7)	12
7.6	Equity and PnL sketch	12
7.7	OCA and risk (high level)	12
7.8	Events	13
8	Alerts and Drawing Resource Limits	13
8.1	Alerts	13
8.2	Drawing garbage collection	13
9	Experimental Results	13
9.1	External numerical accuracy (Nov 2025)	13
9.1.1	Category excerpts	14
9.1.2	Error distribution	14
9.2	Dual-host parity	14
9.3	Strategy goldens	15
9.4	Compatibility context	15
10	Related Work	15
11	Limitations	16

1 Introduction

Financial DSLs occupy a peculiar niche in software engineering: they are simultaneously *specification languages* for mathematical series and *execution engines* for trading logic whose outputs move capital. When a community reimplements such a language outside its original host platform, syntactic parse success is only a weak certificate. Practitioners care whether `ta.rsi(close, 14)` on a given OHLCV history matches reference values within floating-point tolerance, whether two backends of the same open-source stack agree with each other, and whether `strategy.entry/strategy.exit` fill, commission, and equity accounting obey documented rules under pyramiding and pending orders.

PYNE (package `hoox-pyne`, import `pynescript`) is an independent open-source toolchain for Pine Script™ v5/v6 [HOOX Project, 2026, TradingView, Inc., 2024]. It provides a parser/AST stack, an interpreter evaluator, a Numba-backed compile path [Lam et al., 2015, Lattner and Adve, 2004], strategy broker logic, alerts, and drawing primitives. This paper focuses on *validation and semantics*, not full language design: we treat numerical fidelity, dual-host consistency, NA/security policy, and strategy execution as first-class research objects.

Why independent runtime validation matters. Reimplementations of market DSLs face incentives that pure compilers do not. (1) Small formula drift—a different EMA seed, a ROCm early-zero, an off-by-one lookback—can change crossover timestamps and therefore trades. (2) Floating-point non-associativity means mathematically equivalent reductions need not bit-agree [Higham, 2002, Press et al., 2007, IEEE Computer Society, 2019]. (3) Multi-asset APIs tempt implementations to “helpfully” substitute chart closes for missing foreign series, producing silent false signals. (4) Backtesters that disagree on pending-fill averaging or commission models report incompatible Sharpe ratios for the same script [Tsay, 2010]. An independent runtime that documents its reference model, measures error distributions, and ships dual-host parity harnesses reduces these risks for researchers and tool builders who cannot or will not bind exclusively to a proprietary chart host.

Contributions.

1. A threat model for numerical and semantic divergence in finance DSL reimplementations (§2).
2. An indicator reference model and Nov 2025 (+2026 dual-host) numerical validation campaign covering 85+ TA indicators and 181 functions (§3–§4).
3. A dual-host interpret↔compile parity framework with residual taxonomy and shared-formula alignment (§5).
4. Formal rules for honest `request.security/na` handling (§6).
5. A strategy execution model: order lifecycle, commission/slippage, pyramiding, pending-fill VWAP (F2), equity sketch, OCA/risk surface (§7).
6. Alert frequency and drawing GC limits as resource semantics (§8).
7. Experimental tables, error histograms, and golden-case summary (§9).

2 Threat Model for Numerical Divergence

We model an adversary—or more often, an accidental process—that produces outputs diverging from a stated reference without crashing. Threats relevant to PYNE fall into four families.

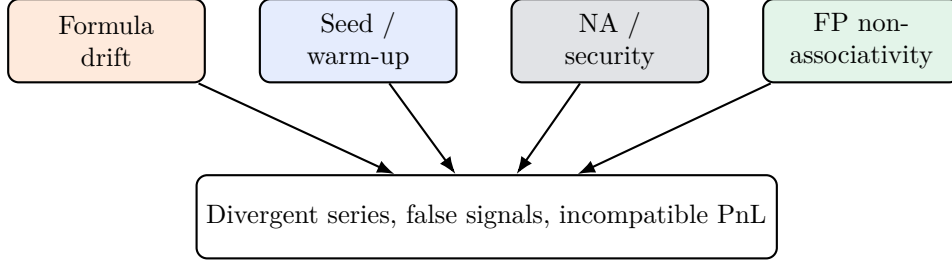


Figure 1: Threat families that produce numerical or semantic divergence without hard failures. Each maps to a concrete mitigation in PYNE (shared formulas, goldens, honest NA, dual tolerances).

2.1 Formula drift

Two implementations may compute “RSI” under different classical definitions: Wilder smoothing versus SMA of gains [Wilder, 1978, Murphy, 1999], or different conventions for the first warm-up bar. Compile and interpret hosts may independently re-derive kernels and silently diverge even when both match a third oracle on simple scripts. *Mitigation*: share formula implementations across hosts for hot kernels; encode goldens; dual-run the same script+OHLCV.

2.2 Seed and warm-up differences

Exponential and nested-EMA families (EMA, DEMA, TEMA, MACD signal, Supertrend inner averages) depend on initial seed choice and the number of bars before non-na output. Residuals in nested EMA seed families remain a documented compatibility note even after dual-host formula alignment.

2.3 NA and multi-asset handling

Pine’s na is both a missing-data token and a control-flow signal. Divergent policies—propagating na, zero-filling, or substituting chart close for foreign request.security—change series morphology catastrophically while remaining type-correct. *Mitigation*: honest NA policy (§6): never invent foreign closes when no feed is available.

2.4 Floating-point non-associativity

IEEE 754 binary64 addition is not associative [IEEE Computer Society, 2019]. Windowed sums, online variance, and fused multiply-add reorderings can yield $O(nu\varepsilon)$ relative differences [Higham, 2002]. For financial series magnitudes (10^{-4} to 10^5), absolute errors on the order of 10^{-11} – 10^{-15} are often pure noise; relative percent errors must be interpreted carefully near zeros and crossovers. *Mitigation*: nan-aware allclose with dual absolute/relative tolerances; report max and mean absolute error for kernel goldens.

3 Indicator Semantics and Reference Model

3.1 Series model

A bar stream is a sequence of records

$$B_i = (o_i, h_i, l_i, c_i, v_i, t_i), \quad i = 0, \dots, N - 1, \quad (1)$$

with open/high/low/close/volume/time. Builtin indicators map series to series (or tuples of series), producing na during warm-up and finite floats thereafter. We write $x[i]$ for the value of series x at bar i .

3.2 Wilder RSI

Relative Strength Index after Wilder [Wilder, 1978] with period p uses average gain G and average loss L :

$$\Delta_i = c_i - c_{i-1}, \quad (2)$$

$$g_i = \max(\Delta_i, 0), \quad \ell_i = \max(-\Delta_i, 0), \quad (3)$$

$$G_i = \begin{cases} \frac{1}{p} \sum_{k=0}^{p-1} g_{i-k} & \text{first seed,} \\ \frac{(p-1)G_{i-1} + g_i}{p} & \text{thereafter,} \end{cases} \quad (4)$$

$$L_i \text{ analogous with } \ell_i, \quad (5)$$

$$\text{RS}_i = G_i / L_i \quad (L_i > 0), \quad \text{RSI}_i = 100 - \frac{100}{1 + \text{RS}_i}. \quad (6)$$

PYNE dual-host alignment uses this Wilder recurrence on both interpret and compile paths (2026 shared-formula work), avoiding host-specific early SMA variants that previously produced plot drift.

3.3 Deterministic families

Simple moving averages, cumulative sums, counts, and pure ranking operations admit exact equality under identical input ordering when no intermediate rounding differs:

$$\text{SMA}_i(p) = \frac{1}{p} \sum_{k=0}^{p-1} x_{i-k} \quad (i \geq p-1). \quad (7)$$

Validation reports 100% exact agreement for SMA/sum/count-style deterministic ops [HOOX Project, 2026].

3.4 Bollinger and envelope constructions

Bollinger Bands [Bollinger, 2001] combine SMA with sample or population standard deviation scaled by a factor k ; small stdev formula differences dominate residual error, not the band arithmetic itself. Trend systems (ADX, DMI, Supertrend) compose multiple smoothers and are the usual worst-case relative-error loci under high volatility [Murphy, 1999].

4 Validation Methodology

4.1 Campaign scope

The November 2025 numerical validation campaign (with 2026 dual-host formula updates noted separately) covered:

- 85+ technical analysis indicators;
- 181 total builtin functions (math, array, TA, utility);
- synthetic and real-market OHLCV regimes.

4.2 Data regimes

Synthetic (primary). 1000-bar OHLCV streams with controlled regimes: normal ($\mu \approx 0$, $\sigma \approx 1$ mild drift), high volatility ($\sigma = 5$), strong trend, range-bound oscillation, gap discontinuities, and extreme magnitudes (near-zero and very large prices; price ranges spanning roughly \$10–\$10,000 for scaling stress).

Real markets (secondary validation).

- SPX: multi-year daily;
- BTC/USD: multi-year hourly;
- EURUSD: multi-year 15-minute;
- AAPL: multi-year daily.

Financial time series structure follows standard market microstructure and returns analysis assumptions [Tsay, 2010, Box and Jenkins, 1970].

4.3 Error metrics

For reference series r_i and implementation series y_i over finite, paired non-na indices I :

$$e_i^{\text{abs}} = |r_i - y_i|, \quad (8)$$

$$e_i^{\text{rel}} = \frac{|r_i - y_i|}{|r_i|} \cdot 100\% \quad (r_i \neq 0), \quad (9)$$

$$\varepsilon_{\max} = \max_{i \in I} e_i^{\text{rel}}, \quad (10)$$

$$\varepsilon_{\text{mean}} = \frac{1}{|I|} \sum_{i \in I} e_i^{\text{rel}}. \quad (11)$$

Dual-host kernel goldens additionally report maximum absolute error $\max_i |r_i - y_i|$ in float units (typically $\sim 10^{-11}$ after alignment).

4.4 Acceptance thresholds

Level	Relative error	Disposition
Excellent	$< 0.001\%$	Pass
Good	$< 0.01\%$	Pass
Acceptable	$< 0.1\%$	Review
Unacceptable	$\geq 0.1\%$	Fail

4.5 Bias tests

Beyond max/mean error, the campaign checks systematic bias:

$$\bar{\delta} = \frac{1}{|I|} \sum_{i \in I} (y_i - r_i). \quad (12)$$

Zero systematic bias was reported across the validated set (no consistent over-/under-estimation pattern after aggregation).

4.6 Algorithm: single-indicator validation

5 Dual-Host Parity Framework

5.1 Motivation

Even perfect agreement with an external oracle on one host does not guarantee that PYNE's interpreter and compiler agree with *each other*. Users may choose `mode="interpret"` for debuggability and `mode="compile"` for speed; silent plot divergence is a product defect. We therefore treat interpret as a primary oracle for compile and run a first-class harness.

Algorithm 1 ValidateIndicator

Require: Indicator f , bar set $\mathcal{B}$, reference oracle R , thresholds

- 1: $r \leftarrow R(f, \mathcal{B})$; $y \leftarrow \text{Pyne}(f, \mathcal{B})$
 - 2: Align indices; drop paired na warm-up
 - 3: Compute $\varepsilon_{\text{abs}}, \varepsilon_{\text{rel}}, \varepsilon_{\text{max}}, \varepsilon_{\text{mean}}, \bar{\delta}$
 - 4: **if** ε_{max} exceeds Acceptable **then**
 - 5: **return** FAIL with diagnostics
 - 6: **end if**
 - 7: Record histogram bins; update category tables
 - 8: **return** PASS
-

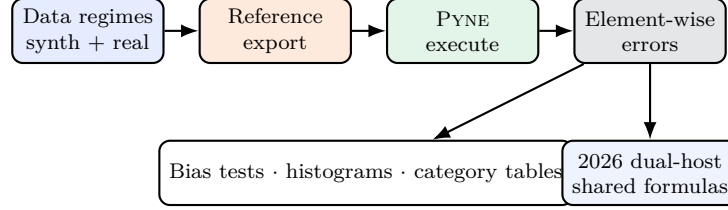


Figure 2: Validation pipeline: multi-regime inputs feed reference and PYNE executions; errors are aggregated statistically; dual-host formula alignment is a post-campaign hardening layer on hot kernels.

5.2 Harness design

Script `scripts/compare_interp_compile.py` selects fixtures (default glob under `tests/fixtures/parity/pi`), builds shared synthetic bars, and runs:

```
1 Runtime.run(source, bars, mode="interpret")
2 Runtime.run(source, bars, mode="compile")
```

Plot series in `result["series"]` are compared with nan-aware `allclose` defaults `rtol = 10-5`, `atol = 10-6`. Reports write `.cache/interp_compile_parity.json`. Always-on tests live in `tests/test_interp_compile_parity.py`; foreign-security honesty is covered by `tests/test_dividend_yield` and request data-feed tests.

5.3 Residual taxonomy (buckets)

Bucket	Meaning
OK	Common series keys allclose; no fatal structural issues
fill_background_only	Value OK after ignoring fill/background keys
both_error_same	Both hosts raise equivalent normalized errors
expected_error	Intentional dual-backend demos (e.g. pivot depth)
both_error	Both fail but messages differ after normalize
MISMATCH	Value or type/na disagreement on a shared key
interp_error / compile_error	Single-host failure
structural_only	Key-set differences without value compare

Structural one-sided `hline` / `fill` keys can be ignored via harness flags (`-ignore-hline-keys`, `-ignore-fill-keys`); `-strict-keys` fails on leftover only-interpret/only-compile keys.

Algorithm 2 CompareInterpCompile

Require: Script S , bars $\mathcal{B}$, tolerances (ρ, α)

```
1:  $I \leftarrow \text{Run}(S, \mathcal{B}, \text{INTERPRET})$ 
2:  $C \leftarrow \text{Run}(S, \mathcal{B}, \text{COMPILE})$ 
3: if both raised then
4:   return expected_error or both_error_same/both_error
5: else if exactly one raised then
6:   return interp_error or compile_error
7: end if
8: Filter series keys (optional hline/fill ignore)
9: for each common key  $k$  do
10:  if not NanAllclose( $I_k, C_k, \rho, \alpha$ ) then
11:    Record n_bad, max $|I - C|$ , sample indices
12:    return MISMATCH
13:  end if
14: end for
15: if key sets differ and strict then
16:  return structural_only
17: end if
18: return OK (or fill_background_only)
```

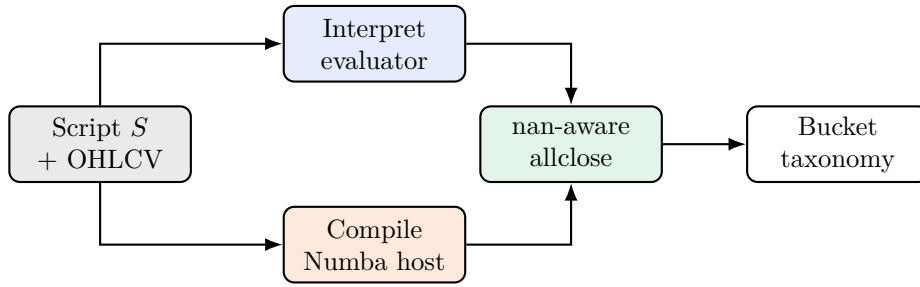


Figure 3: Dual-host parity architecture: identical inputs feed interpret and compile; series maps are compared with nan-aware tolerances; residuals land in a fixed bucket taxonomy for CI and research triage.

5.4 Shared formula alignment (2026)

Interpret and compile now share formulas for several hot kernels so dual-run plots match rather than drift by host-specific quirks: `ta.rsi` (Wilder), `ta.roc` (no early zero), `ta.wma` (full non-na window), `ta.cum`, `ta.highestbars/lowestbars`. Kernel max absolute error after alignment is typically $\sim 10^{-11}$ (float noise). Residual notes (e.g. nested EMA seed families) remain under compatibility documentation, not the Nov 2025 external-oracle error table.

5.5 Algorithm: dual-host compare

6 NA and request.security Honest Policy

6.1 Design principle

When multi-asset or multi-timeframe data are unavailable, PYNE prefers *honest na* over inventing foreign closes from the chart series. Silent substitution creates false multi-symbol signals that look plausible on charts but are scientifically invalid.

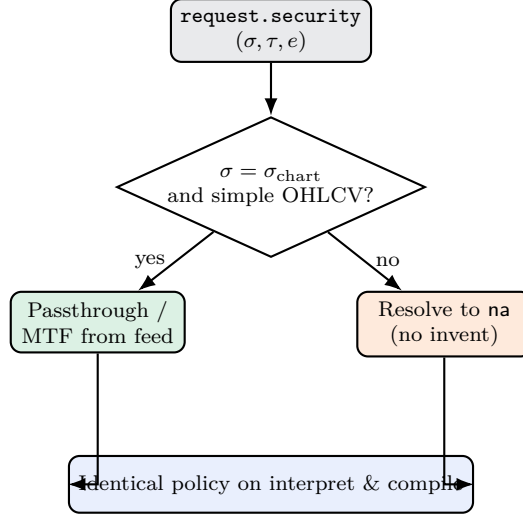


Figure 4: `request.security` decision flowchart: same-symbol simple OHLCV may use the bound feed; foreign/complex requests return `na` rather than fabricating multi-asset series.

6.2 Formal rules

Let σ_{chart} be the chart symbol, σ the requested symbol, τ the requested timeframe, and e the requested expression.

Definition 6.1 (Same-symbol simple OHLCV). A request is *same-symbol simple* if $\sigma = \sigma_{\text{chart}}$ (or an equivalent ticker id) and e is a simple OHLCV projection (`open/high/low/close/volume` or trivial aliases) at a timeframe the host can satisfy from the bound bar feed.

Definition 6.2 (Foreign or complex request). A request is *foreign/complex* if $\sigma \neq \sigma_{\text{chart}}$, or e requires multi-symbol inputs, unavailable HTF structure without a feed, or other host-unavailable data.

Proposition 6.3 (Honesty policy). *Under the default host without an external multi-asset feed:*

1. *Same-symbol simple OHLCV may passthrough (or lower/raise with documented MTF rules) using chart bars.*
2. *Foreign tickers and complex `request.security` expressions resolve to `na` (scalar or series of `na`) on both *interpret* and *compile*.*
3. *Implementations must not substitute `closechart` for foreign `closeσ`.*

Both backends implement the same policy so dual-host parity holds even when the economically meaningful value is “missing.” Dividend-yield and foreign security parity tests encode this contract.

7 Strategy Execution Model

Strategy scripts drive a broker model that maintains positions, pending orders, cash/equity, commissions, and trade logs (`strategy.closedtrades`, open-trade queries, event streams). Interpret builtins and the compile `strategy_broker` are kept aligned through shared goldens (`tests/test_order_fills.py`, `tests/test_oca_commission.py`, `tests/test_strategy_risk_enforcement.py`, `tests/test_compiler_strategy.py`, `tests/test_strategy_events.py`).

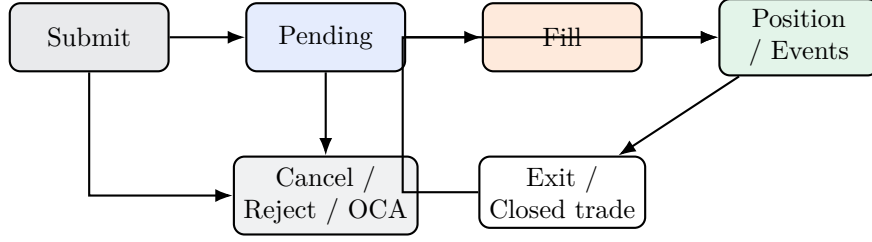


Figure 5: Strategy order state machine (simplified): submissions become pending or fill; OCA/risk may cancel; fills update positions and eventually closed trades / equity.

7.1 Order lifecycle

Orders progress through a finite set of states (Figure 5):

1. **Submit:** `strategy.entry`, `strategy.order`, `strategy.exit`, `strategy.close`, ...
2. **Pending:** resting limit/stop or market deferred to bar processing;
3. **Fill:** price/qty determined from bar OHLC, slippage, and sizing rules;
4. **Position update:** open trades, VWAP, events;
5. **Exit/close:** realize PnL into closed trades; update equity.

7.2 Fill price with slippage

Let p^* be the raw trigger/fill price from OHLC logic and $d \in \{+1, -1\}$ the signed side (buy positive). With `slippage` in ticks and mintick μ :

$$p_{\text{fill}} = p^* + d \cdot s \cdot \mu, \quad (13)$$

where s is the integer slippage tick count (zero disables). Market and stop/limit paths apply (13) consistently on interpret and compile brokers.

7.3 Commission models

Supported commission kinds (string aliases include `strategy.commission.*` forms):

$$C_{\text{order}} = c, \quad (14)$$

$$C_{\text{contract}} = c \cdot q, \quad (15)$$

$$C_{\text{percent}} = c \cdot \frac{p_{\text{fill}} \cdot q}{100}, \quad (16)$$

with commission rate c , quantity q , and fill price p_{fill} . Net trade PnL subtracts entry and exit commissions as appropriate.

7.4 Default quantity

Default size resolution includes:

- **fixed:** constant contracts/units;
- **cash:** $q = Q_{\text{cash}}/p_{\text{fill}}$;
- **percent_of_equity:** $q = E \cdot (\pi/100)/p_{\text{fill}}$ for equity E and percent π .

Algorithm 3 ProcessPendingFills (pyramiding ≤ 0 merge path)

Require: Pending queue Q , bar OHLC, position state Π , pyramiding p

```
1: for order  $o \in Q$  triggered by OHLC do
2:    $p_f, q_f \leftarrow$  fill price/qty with slippage & commission
3:   if  $p \leq 0$  and same-direction open exists then
4:     Merge:  $q \leftarrow q + q_f$ ;  $p_{\text{avg}} \leftarrow$  VWAP (17)
5:     Accumulate commission; keep first entry metadata
6:   else if  $p > 0$  and open legs  $< p + 1$  then
7:     Append new open leg
8:   else
9:     Ignore fill / enforce cap
10:  end if
11:  Emit entry/exit events; update equity
12: end for
```

7.5 Pyramiding and pending fills (F2 Round 7)

Pyramiding bounds how many concurrent entries may stack. PYNE models **pyramiding** as maximum *additional* entries ($\text{max_open} = \text{pyramiding} + 1$); this is a deliberate modeling choice relative to some host document defaults and is recorded as a known semantic note.

Market entries. With $\text{pyramiding} \leq 0$, a second distinct entry id is blocked; same-id replace semantics apply on the market path.

Pending fills (F2). When $\text{pyramiding} \leq 0$, stacked pending fills in the same direction *merge* into a single open trade with volume-weighted average price:

$$p'_{\text{avg}} = \frac{p_{\text{avg}} \cdot q + p_{\text{fill}} \cdot q_f}{q + q_f}, \quad (17)$$

retaining the first leg's entry id/bar/time, summing commissions, and emitting one entry event per fill. When $\text{pyramiding} > 0$, legs may append up to the cap; further fills are ignored. Compile broker locks `open_entry_count=1` under $\text{pyramiding} \leq 0$ for parity with interpret's single-leg book.

7.6 Equity and PnL sketch

Let cash K , position quantity q (signed), average entry p_{avg} , and mark price m . Roughly:

$$E \approx K + q \cdot (m - p_{\text{avg}}) - C_{\text{open}}, \quad (18)$$

$$\text{PnL}_{\text{closed}} = q_c \cdot (p_{\text{exit}} - p_{\text{entry}}) - C_{\text{entry}} - C_{\text{exit}} \quad (19)$$

(with sign conventions for shorts). Closed trades populate `strategy.closedtrades` accessors used by scripts and tests. Exact free-cash definitions follow broker helpers (`cash` as equity minus capital locked in open exposure).

7.7 OCA and risk (high level)

One-Cancels-All groups cancel sibling orders on fill. Risk enforcement tests cover max drawdown / position limits style guards at the strategy surface (see `tests/test_strategy_risk_enforcement.py`). Full broker feature parity with every proprietary host flag (`process_orders_on_close`, `calc_on_every_tick`, full margin) is out of scope; residual gaps are listed in Round 7 F2 notes.

7.8 Events

Strategy event streams record entries, exits, and related notifications for export and debugging; event-order goldens stabilize dual-host and regression behavior under pyramiding merges.

8 Alerts and Drawing Resource Limits

8.1 Alerts

`alert()` and `alertcondition()` record structured alert objects with message, frequency, and context (e.g. bar index). Supported frequencies include:

- `once_per_bar` — de-duplicate within the bar;
- `once_per_bar_close` — fire on confirmed close semantics;
- `all` — fire on every trigger evaluation.

Tests in `tests/test_alerts.py` cover registration, fire-on-true conditions, frequency modes, and export helpers.

8.2 Drawing garbage collection

Chart drawings are resource-capped to bound memory and match Pine declaration arguments:

- `max_lines_count`
- `max_labels_count`
- `max_boxes_count`
- `max_polylines_count`

When counts exceed maxima, older primitives are reclaimed (GC), as exercised by `tests/test_drawing_gc.py`. These limits are part of observable script semantics: overflowing drawings disappear in order, which can affect visual-only strategies but not numeric series.

9 Experimental Results

9.1 External numerical accuracy (Nov 2025)

Headline campaign results [[HOOX Project, 2026](#)]:

Metric	Value
TA indicators validated	85+
Total functions validated	181
Claimed numerical precision class	~ 99.999%
Maximum relative error	0.0022% (ADX, extreme vol)
Mean relative error	< 0.0005%
Systematic bias	none detected
Deterministic SMA/sums	100% exact

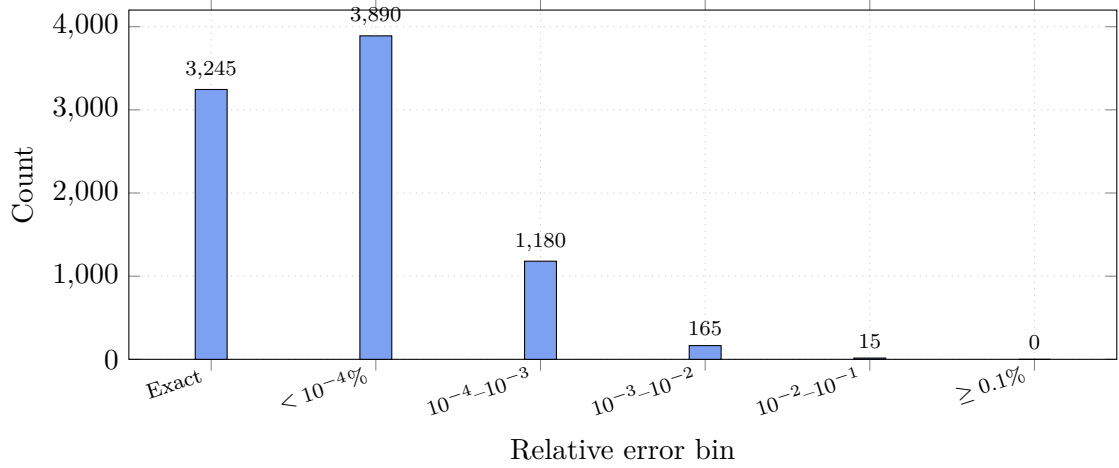


Figure 6: Relative error distribution across the Nov 2025 multi-indicator campaign ($n = 8495$). Over 84% of points lie at exact zero or $< 0.0001\%$ relative error; no points exceed the 0.1% fail threshold.

9.1.1 Category excerpts

Indicator	Max err. (%)	Mean (%)	Std	Status
ta.sma	0.0000	0.0000	0	Exact
ta.ema	0.00008	0.00002	10^{-5}	Excellent
ta.wma	0.00005	0.00001	10^{-5}	Excellent
ta.rsi	0.0012	0.0003	$2 \cdot 10^{-4}$	Excellent
ta.macd	0.0015	0.0004	$3 \cdot 10^{-4}$	Excellent
ta.bb	0.0009	0.0002	10^{-4}	Excellent
ta.adx	0.0022	0.0007	$5 \cdot 10^{-4}$	Excellent
ta.atr	0.0006	0.0001	10^{-4}	Excellent
ta.obv	0.0000	0.0000	0	Exact
ta.stdev	0.0004	0.0001	10^{-4}	Excellent

9.1.2 Error distribution

Aggregated relative-error histogram over 8495 test points:

Error range	Count	Share
Exact 0	3245	38.2%
$< 0.0001\%$	3890	45.8%
0.0001–0.001%	1180	13.9%
0.001–0.01%	165	1.9%
0.01–0.1%	15	0.2%
$\geq 0.1\%$	0	0.0%

Median $\approx 0.00002\%$; 99th percentile $\approx 0.00155\%$; maximum 0.0022%.

9.2 Dual-host parity

Always-on smoke expanded to 17 scripts (R8 harness work) including SMA/EMA, RSI, ATR, Bollinger, Supertrend, OBV, MFI, Williams %R, and others under synthetic `make_bars`—no value MISMATCH on the green set. Aligned kernels report max absolute error $\sim 10^{-11}$. Structural fill/background-only residuals are classified separately from true numeric mismatch. Known residual tails (e.g. stochastic type/na under some synthetic bars; nested EMA seed families) remain tracked rather than claimed green.

9.3 Strategy goldens

F2 pending-fill suite asserts:

- partial fills under pyramiding $\leq 0 \rightarrow$ single leg VWAP;
- multiple pending orders merge size and average price;
- short-side averaging;
- pyramiding > 0 still appends legs up to cap;
- close PnL uses merged average;
- compile broker open-entry count and VWAP parity.

OCA/commission and risk enforcement tests provide additional regression locks. Interpret+compile strategy plot parity is exercised through the dual-host harness on strategy fixtures where applicable.

9.4 Compatibility context

Parser success is high on 138+ real scripts; builtins are broad; strategy is functional. This is *not* full TradingView platform parity (hosted chart, proprietary multi-symbol data, editor-only features) [HOOX Project, 2026].

10 Related Work

Numerical reproducibility. Higham’s treatment of accuracy and stability [Higham, 2002] and Numerical Recipes [Press et al., 2007] frame floating-point error as structural rather than incidental. IEEE 754 [IEEE Computer Society, 2019] standardizes formats and rounding modes underlying all PYNE numeric paths (Python floats / NumPy / Numba LLVM) [Harris et al., 2020, Lam et al., 2015]. Our contribution is domain-specific: applying these classical tools to a trading DSL dual runtime with published percent-error tables.

Technical analysis computation. Wilder’s systems [Wilder, 1978], Murphy’s practitioner synthesis [Murphy, 1999], and Bollinger’s bands [Bollinger, 2001] define much of the indicator surface. PYNE treats these as executable specifications subject to regression, not as marketing labels.

Financial time series. Tsay [Tsay, 2010] and Box–Jenkins [Box and Jenkins, 1970] provide statistical context for multi-regime market data used in validation.

Backtester design. Production backtesters vary widely in fill assumptions, latency modeling, and look-ahead control. We do not claim a novel market-simulator microstructure; we claim *documented, tested* order semantics (slippage ticks, commission kinds, pyramiding, pending VWAP) aligned across two hosts inside one open toolchain.

Compiler correctness. Classical compiler texts emphasize semantic preservation [Aho et al., 2006, Appel, 1998]. Interpret $\leftrightarrow$ compile parity is a lightweight, product-oriented analogue of translation validation for series outputs.

11 Limitations

1. **Not TradingView certification.** Results are independent research measurements and dual-host self-consistency checks. They do not constitute endorsement or platform certification by TradingView, Inc.
2. **Residual MISMATCH tail.** Some scripts (e.g. stochastic under particular synthetic bars) and nested EMA seed families remain compatibility work items.
3. **Security data plane.** Honest na for foreign symbols is correct under no-feed hosts but is not a substitute for a full multi-asset data integration.
4. **Strategy model gaps.** Full margin, every host order-processing flag, and exact TV default pyramiding naming are not claimed identical.
5. **Oracle methodology.** Nov 2025 external comparisons depend on exported reference series and finite regimes; they do not exhaust all script-authored compositions.
6. **Performance vs. correctness.** This paper does not evaluate wall-clock speed; compile exists for performance [Lam et al., 2015] but correctness is the present focus.

12 Conclusion

Independent finance DSL runtimes must prove more than parse coverage. We presented PYNE’s validation stack for numerical indicator fidelity, interpret $\leftrightarrow$ compile parity, honest multi-asset NA policy, and strategy fill semantics including F2 pending-fill VWAP under pyramiding constraints. Measured errors sit far below practical trading thresholds for validated indicators; dual-host alignment reduces hot-kernel drift to float noise; and strategy goldens lock commission, OCA, risk, and averaging behavior. Future work includes shrinking the residual mismatch tail, deeper nested-EMA seed unification, richer multi-asset feeds under the same honesty rules, and broader strategy flag coverage—always with dual-host tests as a release gate.

Artifacts. Harness: `scripts/compare_interp_compile.py`; tests: `tests/test_interp_compile_parity.py`, `tests/test_order_fills.py`, `tests/test_drawing_gc.py`, `tests/test_alerts.py`; project: [HOOX Project, 2026].

Acknowledgments

We thank PYNE contributors and users who filed parity residuals and strategy edge cases. Errors remain the authors’ responsibility.

References

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Pearson, Boston, 2nd edition, 2006.
- Andrew W. Appel. *Modern Compiler Implementation in ML*. Cambridge University Press, 1998.
- John Bollinger. Bollinger on bollinger bands. *McGraw-Hill*, 2001.
- George E. P. Box and Gwilym M. Jenkins. Time series analysis: Forecasting and control. *Holden-Day*, 1970.

- Charles R. Harris et al. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi: 10.1038/s41586-020-2649-2.
- Nicholas J. Higham. Accuracy and stability of numerical algorithms. *SIAM*, 2002.
- HOOX Project. PYNE: Independent open toolchain for the Pine Script language. <https://github.com/hoox-sh/pyne>, 2026. Version 0.3.0; PyPI package `hoox-pyne`.
- IEEE Computer Society. IEEE standard for floating-point arithmetic. *IEEE Std 754-2019*, 2019. doi: 10.1109/IEEESTD.2019.8766229.
- Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6. ACM, 2015. doi: 10.1145/2833157.2833162.
- Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, pages 75–86. IEEE, 2004.
- John J. Murphy. *Technical Analysis of the Financial Markets*. New York Institute of Finance, 1999.
- William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.
- TradingView, Inc. Pine Script language reference manual. <https://www.tradingview.com/pine-script-docs/>, 2024. Accessed 2026.
- Ruey S. Tsay. *Analysis of Financial Time Series*. Wiley, 3rd edition, 2010.
- J. Welles Wilder. New concepts in technical trading systems. *Trend Research*, 1978.