

Probability-First Range Pricing for Bullseye Markets

Black-Scholes bucket probabilities, centered grid geometry, grouped tails, and operational market controls

Revision date: 2026-09-10

This paper is the public pricing paper linked from the Bullseye information page. It supersedes the 2026-08-08 version and incorporates the current relayer pricing corrections, including the long-duration bucket-bias fix introduced in `06f4645`, the HOOD/SNDK regression coverage introduced in `0331dc7`, and the 2026-08-10 first-priced-grid immutability amendment.

The 2026-09-10 revision is a documentation-accuracy pass, not a model change. The kernel, the constants, and the geometry are unchanged and were re-verified line-for-line against the relayer at commit `4235882c0118`. This revision states the volatility clamp as the two-sided clamp the code actually applies and explains why its floor exists (section 4), records suite counts against the commit that produced them instead of as standing figures (section 22), and adds an explicit scope boundary naming the relayer subsystems this paper does not cover (section 23). Repository paths in the implementation map are also corrected: the relayer and frontend repositories were renamed from `prophet-\*` to `bullseye-\*` after the 2026-08-10 revision was published.

The original paper explained the central doctrine: Bullseye prices financial range tickets from a model instead of waiting for thin order-book liquidity to discover a price. That doctrine remains. The material update in this revision is more precise: the live model is no longer described as raw textbook `d2` interval mass used directly for every finite bucket. The deployed relayer now uses a Bullseye-specific Black-Scholes lognormal bucket kernel that starts from the Black-Scholes terminal distribution, centers the visible finite book on spot, caps total horizon volatility, log-width equalizes finite bucket masses, and then prices grouped tails from the same centered/capped distribution.

That update is not cosmetic. It fixes the failure mode where high volatility plus longer duration could make a low-priced asset look falsely bearish. CASHCAT made the problem visible, but the fix is asset-general. The same audit now covers CASHCAT, HOOD, SNDK, RIVN, ETH, BTC, and a low-price synthetic asset across multiple volatility levels and 1-day, 3-day, 7-day, 10-day, and 30-day model horizons.

Abstract

Bullseye is a probability-priced prediction market for financial ranges. A user does not need to wait for a seller to appear on the opposite side of an order book. The protocol posts executable outcome quotes. For terminal-price range markets, the quote begins from a Black-Scholes lognormal terminal distribution: the probability structure implied by spot, volatility, time to maturity, and the outcome boundaries.

The production pricing stack has four separate layers:

1. market-class geometry
2. Black-Scholes lognormal bucket kernel
3. product denomination
4. execution and lifecycle controls

Geometry decides which ranges exist. The kernel prices those ranges. The denomination layer converts model probabilities into executable three-decimal ticket quotes. Execution controls decide whether a user may still buy or match orders. Keeping those layers separate is the main safety property of the system. Once the first ticket-price grid has been posted, that grid is product state. It must be repriced in place even if there are no sales yet.

The current market-class structure is uniform at the product level:

- 1-day market: 23h wagering + 1h lock = 24h total horizon
- 3-day market: 60h wagering + 12h lock = 72h total horizon
- 7-day market: 6d wagering + 1d lock = 7d total horizon

- Each class:
- 9 finite buckets
 - 2 grouped tails
 - 11 total outcomes

The important distinction is that the bucket count is fixed, but the bucket width is not fixed. Wider assets and longer durations get wider clean buckets. Quieter assets get narrower buckets. Low-priced assets keep the finite book centered above zero and use grouped tails for the outside mass. The product keeps the same number of visible choices while the risk geometry adapts.

1. Executive Thesis

The thesis is probability first. A Bullseye ticket should cost what the model estimates the outcome is worth, not what a thin market happens to clear at. Traditional prediction markets often require liquidity before they are usable. If nobody is offering the other side, a user cannot enter. If the only offer is small, the user faces spread, slippage, and adverse selection. Bullseye inverts that structure. The market exists because the model can quote it.

The second thesis is partition first. A financial range market is not a loose set of independent guesses. It is a complete partition of terminal price. At maturity, exactly one outcome wins:

```
downside tail
or one finite bucket
or upside tail
```

The grouped tails are therefore part of the market, not decoration. They cover terminal prices below the finite grid and above the finite grid. Without them, the visible book would hide real probability outside the displayed ranges.

The third thesis is geometry before quote. Black-Scholes can price any interval it is handed, but it cannot make a bad interval set look good. If a high-vol asset is forced into narrow equal-dollar buckets for a long market, the model will put too much visible weight into tails or into distorted low-side buckets. That is not solved by hand-tuning a row. It is solved by constructing a coherent grid before pricing.

The fourth thesis is neutral visible launch shape. A new market should launch with the spot/current range in the center of the finite grid, with prices rising from both wings into the current region and then falling away on the other side. Users read this as a distribution. If the book instead starts at zero or drifts far above the current asset price, the product looks broken even if some formula inside it is mathematically defensible.

The fifth thesis is operational minimalism on-chain. The contract should enforce durable rules: no new exposure after close, no buys/matches while suspended, no grid mutation after creation/first pricing, and maturity-based resolution. The contract should not compute Black-Scholes, estimate volatility, or require heartbeat transactions that can drain gas on quiet markets. Those concerns belong in the relayer and monitoring stack.

2. The Range Ticket Primitive

A Bullseye market has:

```
underlying asset
open timestamp
close timestamp
maturity timestamp
finite bucket grid
downside grouped tail
upside grouped tail
collateral token
posted ticket prices
```

The open-to-close period is the wagering window. During that window, users may buy primary tickets or match valid secondary orders, provided purchases are not suspended. The close-to-maturity period is the lock window. During the lock, users cannot add exposure, but the asset continues moving until the settlement observation.

A finite bucket is a contiguous interval:

```
$80 to $82
$82 to $84
$0.10 to $0.12
```

Finite buckets are adjacent and ordered. The downside tail covers everything below the first finite lower bound. The upside tail covers everything at or above the finite end. Together, finite buckets plus two tails cover the whole terminal price axis.

A ticket is a claim on exactly one outcome. If that outcome wins, winning tickets share the winning pool according to the settlement contract. This is not a fixed '\$1' redemption product. The quote is a modeled chance of winning the range; the

payout is the user's share of the winning prize pool.

3. The Pricing Stack

Bullseye pricing has four layers.

Layer one is market geometry. It decides how many finite buckets exist, how wide they are, where the first finite bucket begins, and where the two grouped tails attach.

Layer two is the Black-Scholes lognormal bucket kernel. It prices the exact finite ranges chosen by geometry and prices the two tails from the same distribution.

Layer three is product denomination. It applies the explicit product margin, the `\$1` cap, the `\$0.001` executable floor, three-decimal quantization, and sub-cent compression.

Layer four is execution. It checks lifecycle state, closeTimestamp, purchase suspension, fixed-grid immutability, and whether the relayer is allowed to push the price array that will become on-chain truth.

The layers must not be collapsed. If the distribution shape is wrong, do not hide that by changing frontend labels. If the grid is wrong, do not hand-edit probabilities. If the relayer has no gas, do not call the model broken. Each failure has one owner.

4. Current Model: Centered Black-Scholes Bucket Kernel

The deployed pricing model starts from the Black-Scholes lognormal CDF, but the finite bucket quote is not the old raw textbook d^2 interval probability used directly. The current kernel is:

```
v_raw = sigma * sqrt(T)
v      = min(max(v_raw, 0.01), 0.45)
drift = 0
d(K)   = (ln(S / K) + drift) / v
mass_i = Phi(d(K_low_i)) - Phi(d(K_high_i))
score_i = mass_i / ln(K_high_i / K_low_i)
finite_quote_i = sum(mass_i) * score_i / sum(score_i)
```

Where:

```
S      = current spot/reference price
sigma  = annualized volatility from prepared hourly returns
T      = time to maturity in years, with a one-hour minimum while T > 0
v      = total horizon log-volatility, floored at 0.01 and capped at 0.45
Phi    = standard normal CDF
```

The kernel keeps the Black-Scholes terminal distribution as the foundation, but adds three product-neutrality controls:

1. center the visible finite distribution on spot
2. clamp total horizon volatility to [0.01, 0.45]
3. log-width equalize finite bucket masses

The centered distribution removes the hidden bearish median shift created by the textbook $-\sigma^2 / 2$ drift term when volatility and duration become large. In pure option pricing, that term belongs to the risk-neutral terminal stock distribution. In this product, using it directly inside a visible prediction-market bucket book made high-vol longer-duration markets look like crash forecasts even when spot was unchanged.

The total-vol cap prevents launch-regime realized volatility from making 3-day and 7-day books collapse into extreme tails. It is a horizon cap, not a static sigma multiplier. Normal volatility/time products remain inside the clamp and therefore use their actual measured total volatility.

The clamp is two-sided, and the floor is not cosmetic. An under-sampled or artificially quiet return history can drive total volatility toward zero, and a lognormal CDF with near-zero volatility collapses into a point mass at spot: every off-centre bucket then quotes at the executable floor regardless of where spot actually sits, producing a book that looks broken rather than confident. `BS_MIN_TOTAL_LOG_VOLATILITY = 0.01` guards that degenerate case only. It sits far below any volatility this venue has observed on a real asset, so it never binds in normal operation; it exists so that a data outage degrades into a wide book rather than a nonsensical one.

One implementation note for reviewers reading the source directly. The clamp is applied by `_effective_total_volatility()` in `src/models/black_scholes.py`, which enforces both ends. `src/relayer/market_geometry.py` defines a same-named local helper that currently applies only the 0.45 cap, because grid derivation already rejects a degenerate grid through its

`TARGET\_FINITE\_MASS` assertion rather than through the volatility floor. The quote kernel described in this section is the two-sided one.

The log-width equalization is the key fix for equal-dollar buckets. A bucket from `\$0.02` to `\$0.04` is a 100 percent relative interval. A bucket from `\$0.12` to `\$0.14` is a much smaller relative interval. Raw lognormal CDF mass therefore rewards the low bucket's width in log space. The relayer now converts finite CDF masses into comparable log-moneyness density scores and then renormalizes those scores back to the same total finite mass. That removes the fake advantage of low equal-dollar buckets while preserving the finite-grid mass.

5. Why Raw d2 Was Not Enough

The previous public paper correctly said that Black-Scholes can compute terminal range probabilities. It was incomplete because it implied the raw textbook interval mass could be used directly for every visible finite bucket.

That direct approach failed under the combination of:

```
low asset price
equal-dollar bucket widths
high annualized volatility
longer market duration
textbook  $-\sigma^2 / 2 \log \text{drift}$ 
```

For CASHCAT-like inputs, the short-term book looked acceptable because $\sigma^2 * T$ was still small. The 3-day and 7-day books became biased because the total horizon variance grew. The old model could assign more visible probability to a hard downside retrace bucket than to a moderate upside bucket, even when the asset was trading near the center of the displayed grid.

This was not solved by abandoning Black-Scholes. It was solved by using the Black-Scholes lognormal distribution in a way that is appropriate for a fixed-count, equal-dollar, user-facing prediction-market book.

6. Complete Partition and Tails

The finite grid is not the whole state space. A lognormal terminal distribution has support from zero to infinity. Bullseye therefore includes two grouped tails:

```
downside tail = P(S_T < finite_start)
upside tail   = P(S_T >= finite_end_exclusive)
```

The current relayer computes tails from the same centered and total-vol-capped kernel used for finite bucket CDF mass:

```
d_start = (ln(S / finite_start) + drift) / v
d_end   = (ln(S / finite_end_exclusive) + drift) / v

downside_probability = 1 - Phi(d_start)
upside_probability   = Phi(d_end)
```

The audit invariant is:

```
finite_grid_mass_before_equalization
+ downside_probability
+ upside_probability
= 1.0
```

After finite log-width equalization, the finite rows still sum to the same finite mass. Therefore the displayed model probabilities and the grouped tails remain a complete partition of the centered/capped model distribution.

The visible ticket price may differ from raw mathematical probability for tiny outcomes because the product denomination layer applies a `\$0.001` executable floor and sub-cent compression. That is a product rule, not hidden probability math.

7. Product Quote Layer

The current quote constants are:

```
PROTOCOL_PRICE_MARGIN = 1.0
MAX_PROBABILITY_PRICE = 1.0
MIN_TICKET_PRICE      = 0.001
TAIL_COMPRESSION_UPPER_QUOTE = 0.01
TAIL_COMPRESSION_BAND_TOP   = 0.009
```

```
TAIL_COMPRESSION_LOWER_ANCHOR = 1e-10
```

The old hidden ``15 / 10 = 1.5`` multiplier is not part of the final path. The current margin is explicit and set to ``1.0``.

For each finite bucket and tail:

```
fair_model_quote = model_probability
product_quote = fair_model_quote * PROTOCOL_PRICE_MARGIN
final_quote = denominate(product_quote)
```

Denomination means:

```
quote >= 0.01      -> pass through before final rounding
quote <= 1e-10     -> $0.001
1e-10 < quote < 0.01 -> log-compress into $0.001 to $0.009
then round to 3 decimals
then clamp to [$0.001, $1.000]
```

This is why a far tail can show ``$0.001`` even when its model probability is far below 0.1 percent. The ticket is executable; the model probability is lower. The relayer keeps metadata fields such as ``raw_probability``, ``fair_model_quote``, ``product_quote``, and ``final_ticket_price`` so audits can separate model truth from product denomination.

8. Market Classes

The current active classes are:

```
duration index 0:
  product name: 1-day
  wagering window: 23 hours
  final lock: 1 hour
  model horizon: 24 hours
  finite buckets: 9
  grouped tails: 2
  total outcomes: 11
```

```
duration index 1:
  product name: 3-day
  wagering window: 60 hours
  final lock: 12 hours
  model horizon: 72 hours
  finite buckets: 9
  grouped tails: 2
  total outcomes: 11
```

```
duration index 2:
  product name: 7-day
  wagering window: 6 days
  final lock: 1 day
  model horizon: 7 days
  finite buckets: 9
  grouped tails: 2
  total outcomes: 11
```

All active new markets now use 9 finite buckets plus 2 tails. Legacy markets with older 21-row grids can still be serviced if they already exist, but new market creation derives from the active 9-bucket authority.

The reason for the uniform count is product clarity. Users should not need to learn a different row count every time duration changes. The uncertainty of a longer or more volatile market is expressed through wider buckets and larger tails, not through a sprawling table of extra rows.

9. Bucket Count Authority

The relayer now uses one active finite-count authority:

```
ACTIVE_FINITE_BUCKET_COUNT = 9
MAX_PRICES_TO_PUSH = ACTIVE_FINITE_BUCKET_COUNT
ONE_DAY_FINITE_BUCKET_COUNT = ACTIVE_FINITE_BUCKET_COUNT
STANDARD_FINITE_BUCKET_COUNT = ACTIVE_FINITE_BUCKET_COUNT

LOTTERY_FINITE_BUCKET_COUNTS = {
  0: ONE_DAY_FINITE_BUCKET_COUNT,
  1: STANDARD_FINITE_BUCKET_COUNT,
```

```

    2: STANDARD_FINITE_BUCKET_COUNT,
}

```

That matters because earlier architecture allowed the answer to "how many buckets?" to exist in too many places. Conflicting constants, permissive sets, contract assumptions, and first-push behavior made it possible for one part of the system to believe it had created 9 buckets while another part accepted or expected a different count.

The current rule is:

```

new active market = 9 finite buckets + 2 grouped tails
already priced market = preserve exact on-chain grid until close
read-back cap = large enough to read every existing finite row

```

`MAX\_FINITE\_BUCKETS\_READ = 1000` is deliberately separate from `MAX\_PRICES\_TO\_PUSH = 9`. One is a producer count for new markets; the other is a safe read ceiling for immutable legacy grids.

10. Bucket Width and Clean Ticks

The count is fixed, but width is derived. The geometry engine uses:

```

spot price
annualized sigma
duration class
model horizon
target finite mass
clean tick rounding
spot-centering cap

```

The target finite mass is:

```

TARGET_FINITE_MASS = 0.93

```

The raw width is derived from the centered/capped lognormal quantiles needed to cover the target finite mass around spot. The relayer then rounds to a clean tick ladder:

```

1, 2, 5 times 10^n

```

Examples:

```

raw $1.63 -> clean $2
raw $2.01 -> clean $5
raw $0.031 -> clean $0.05

```

Clean upward rounding is conservative because it widens the finite region. The only approved downward rounding is the spot-centering cap used when an otherwise wider clean width would push the visible low side below zero and move spot away from the middle of the book.

11. Spot Centering and the Zero Boundary

For a new unsold grid, the relayer places the finite rows around the bucket that contains spot:

```

center_bucket_start = floor(spot / bucket_width) * bucket_width
first = center_bucket_start - (finite_count // 2) * bucket_width
first = max(bucket_width, first)

```

The first finite lower bound must be positive. The downside tail covers everything below it. Nothing should start at zero as a finite bucket; zero is covered by the downside tail.

For low-priced assets, the relayer also applies:

```

centered_width_cap = round_down_to_clean_tick(spot / (downside_buckets + 1))
final_width = min(clean_target_width, centered_width_cap)

```

This prevents a near-zero asset from launching with spot stuck on the left edge of the finite book. If the centering cap is active, the finite grid may capture less than the 93 percent target mass. That is intentional: the remaining mass belongs in the tails. A readable centered finite grid plus honest tails is better than a distorted finite grid that pretends to be precise.

12. Worked Example: HOOD

Consider a HOOD market near `\$82`.

For a one-day class, geometry can select a `\$2` clean width under moderate volatility. The finite grid can look like:

```
$74 - $76
$76 - $78
$78 - $80
$80 - $82
$82 - $84    <- spot/current bucket
$84 - $86
$86 - $88
$88 - $90
$90 - $92
```

The tails are:

```
downside tail: below $74
upside tail: $92 and above
```

If volatility or duration rises, the finite count remains 9 but the bucket width can expand to `\$5`, `\$10`, or another clean width. The grid stays centered around the current region, and the model prices the exact final clean ranges.

The permanent regression sweep now includes HOOD with configured `\$3` buckets across 1-day, 3-day, and 7-day horizons and sigma levels from `0.25` to `10.0`. In all cases, the model peak remains in the spot bucket and the finite probabilities slope upward into spot and downward away from spot.

13. Worked Example: SNDK

SNDK is a volatile equity-style asset priced from Hyperliquid `xyz:SNDK`. Configured legacy hints show approximately `\$45` bucket granularity near a spot around `\$1828.9`, and active creation supports 3-day and 7-day classes.

A configured-width audit uses:

```
spot = 1828.9
bucket width = 45
finite grid = 1620 to 2025
spot bucket index = 4 of 0..8
```

The stress audit covered sigma values:

```
0.5, 1.0, 2.0, 3.0, 6.0, 8.0, 10.0
```

and durations:

```
3-day class
7-day class
```

Every tested SNDK case kept the peak at the spot bucket, preserved monotone shape on both sides, and preserved the probability partition with tails.

This matters because SNDK is not CASHCAT. It has a much higher nominal price, larger bucket size, and different source route. Passing both CASHCAT and SNDK shows the fix is not a one-token patch. It is a geometry/kernel fix.

14. Worked Example: CASHCAT

CASHCAT exposed the original distortion. At a spot around `\$0.1043` and a two-cent finite width, the centered finite grid can be:

```
$0.02 - $0.04
$0.04 - $0.06
$0.06 - $0.08
$0.08 - $0.10
$0.10 - $0.12    <- spot/current bucket
$0.12 - $0.14
$0.14 - $0.16
$0.16 - $0.18
$0.18 - $0.20
```

The old direct formula could make low-side retrace buckets look too likely as duration and volatility increased. A numerical stress audit showed the old formula's peak falling to the lowest bucket for long/high-vol cases.

The current kernel fixes that:

```
CASHCAT stress spot=0.1043, step=0.02, annualized sigma ~= 7.4876

24h: new peak bucket = 4
72h: new peak bucket = 4
168h: new peak bucket = 4
240h: new peak bucket = 4
720h: new peak bucket = 4
```

For the 7-day stress case, the corrected model kept the current bucket as the peak and made the moderate-upside bucket materially more probable than the hard downside retrace bucket:

```
$0.02 - $0.04: 0.043960
$0.10 - $0.12: 0.167786
$0.12 - $0.14: 0.150662
```

That is the intuitive distribution shape the user should see at launch: both wings lower, current region highest, and no fake bearish shelf.

15. Repricing and Grid Immutability

After the first successful ticket-price publication, the on-chain grid is authoritative. The relayer must not move the user's settlement ranges. This is true even before the first sale. A visible priced market has already told users what the outcomes mean, so repricing can update quotes on the fixed grid, but it cannot change:

```
first finite lower bound
finite bucket count
bucket width
tail boundaries implied by the finite grid
```

Only a never-priced market may build its first grid around the latest spot. Once finite bucket rows exist in the subgraph read-back, those rows become the authority for all later pushes:

```
if finite bucket rows exist:
    fixed_grid_start = firstBucketLowerBound
    fixed_grid_count = len(finite bucket rows)
    movement outside finite range is not a recenter trigger
else:
    derive the initial centered launch grid from spot, volatility, duration, and clean tick rules
```

This amendment is required because "no sales" is not the same as "no product state." CASHCAT made the distinction visible. A prior unsold market could show a downside tail as "below \$0.02" on one push and later show the same tail sentinel as "below \$0.06" after recentering. The frontend then compared the same tail key across two different economic definitions and displayed a large positive 24-hour move in the lowest range while the underlying asset had rallied. That was not a Black-Scholes probability signal. It was mutable outcome geometry.

For fixed-grid repricing, bucket starts are generated by integer index/count:

```
start + i * bucketSize for i in range(count)
```

The relayer must not use floating endpoint iteration such as:

```
np.arange(start, start + count * bucketSize, bucketSize)
```

because fractional bucket sizes can include an endpoint and accidentally create one extra row. That would violate the product invariant that a priced market's grid cannot change.

For priced markets, the relayer also suppresses movement-triggered recentering. If spot leaves the finite grid, the grouped tails already cover the outside state space. Repricing because of that movement would remain true forever and could send a transaction on every scheduler wake. The correct response is cadence repricing on the immutable grid, plus operational monitoring if the market spends too much time in a tail.

16. Time Variables

Bullseye has two important clocks:

```
time_to_close = closeTimestamp - now
time_to_maturity = maturityTimestamp - now
```

They are not interchangeable.

Pricing math uses time to maturity because maturity is when the terminal price is observed:

```
T_model = time_to_maturity
```

Trading cadence and actionability use time to close because close is when new exposure stops:

```
tradable window = now < closeTimestamp
```

For one-day markets, this distinction is essential. A one-day market has a final 1-hour lock. Users cannot add exposure during that hour, but the asset still resolves at maturity. The model horizon includes the lock; the trading permission does not.

17. One-Day Expiry

The one-day class is not a shortened 3-day market. It has its own lifecycle:

```
open
23 hours wagering
close
1 hour lock
maturity
resolution
```

The current product rule is:

```
no primary purchase at or after closeTimestamp
no secondary match at or after closeTimestamp
no buys or matches while purchases are suspended
resolution remains maturity-based
```

One-day markets use the same visible outcome structure as the longer classes:

```
9 finite buckets
2 grouped tails
11 total outcomes
```

For launch, one-day V5 markets use:

```
maxQuoteAge = 0
```

This does not mean stale quotes are acceptable. It means the contract does not enforce quote-age heartbeat rules. Freshness is owned by the relayer and monitoring stack, because strict on-chain heartbeat checks can force gas-heavy updates on quiet assets or abruptly block otherwise healthy markets.

18. Contract and Relayer Boundary

The contract should remain small and durable. It enforces:

```
no primary purchase after closeTimestamp
no exchange match after closeTimestamp
no buys or matches while purchases are suspended
market grid cannot be changed after creation/first pricing
V5 market creation atomically sets the starting quote/grid
resolution is maturity-based
```

The relayer owns:

```
asset approval
source routing
price history fetch
return preparation
volatility estimation
market-class selection
bucket width derivation
spot-centered grid placement
finite-mass admission
finite bucket pricing
tail pricing
quote denomination
```

- price pushing
- gas monitoring
- source health monitoring
- deployment/version alignment

This boundary is deliberate. The contract should not compute Black-Scholes, volatility, target mass, clean ticks, or source freshness. Those are operational and model concerns. They are easier to monitor, upgrade, and halt in the relayer than in immutable contract code.

19. Operational Monitoring

Because quote freshness is relayer-owned for launch, monitoring is not optional. The production operator must track:

- relayer process health
- scheduler health
- latest successful quote push by market
- source-data failures
- source quote age
- subgraph indexing status
- gas balance and runway
- transaction failures
- nonce/replacement errors
- skipped market reasons
- purchases suspension status
- frontend deployment version
- settlement progress

The latest production deployment is online, but recent logs showed ticket and token pushes failing when the relayer wallet lacked enough gas. That is not a model failure. It is an operational failure. The model can compute the right price and the service can be healthy while on-chain writes fail because the wallet cannot pay for them. Monitoring must distinguish those states.

Suspension is the emergency brake. It should block new exposure without blocking claims, refunds, or resolution. If source data, gas, indexing, or version alignment is unhealthy, operations can suspend purchases while fixing the underlying condition.

20. Volatility Inputs and Future Smile Work

The current production fallback uses one realized/historical volatility value per market:

```
sigma = std(hourly returns) * sqrt(365 * 24)
```

The relayer caches volatility history briefly because hourly candles do not advance every minute. This reduces upstream load without serving stale data as a silent fallback.

Single-sigma Black-Scholes is the approved deterministic baseline, but it is not the final institutional volatility surface. Historical option-smile review showed that a single volatility can be close near the money while underpricing wings versus real option markets.

The approved future path is a mini-smile:

- center buckets: near-the-money volatility
- downside wings: downside wing volatility
- upside wings: upside wing volatility
- interpolation: deterministic by log-moneyness or delta

This does not abandon Black-Scholes. It changes the volatility input by bucket while keeping the range-pricing kernel and partition structure.

Static global sigma multipliers are rejected. A global multiplier can improve one snapshot while damaging another. Smile inputs must be structured, logged, and reviewable.

21. Asset Eligibility

The model can price many assets, but that does not mean every asset should be listed automatically. A market is eligible only if it passes:

- approved token registry

supported duration class
price-source routing
positive finite spot
positive finite annualized volatility
sufficient price history
clean grid derivation
first push compatibility
subgraph read-back compatibility
frontend display compatibility
operational gas/source monitoring

Highly volatile assets can be supported if the width and tails honestly express the distribution. They should not be forced into a narrow static bucket size. Quiet assets can be supported if their quotes still move enough to create a meaningful book and the relayer does not need gas-heavy heartbeat churn.

22. Verification Evidence

The current relayer implementation has been verified with unit tests, integration-style relayer tests, and numerical stress audits.

Key implementation commits:

```
06f4645 fix(relayer): neutralize long-duration bucket bias
0331dc7 test(relayer): cover volatile equity bucket shape
7afb96a fix(relayer): bump underpriced nonce replacements
```

Focused pricing and geometry verification after the correction:

```
tests/test_black_scholes_pricing.py
tests/test_market_geometry.py
```

22 passed

Re-verified on 2026-09-10 against relayer commit `4235882c0118`: still 22 passed.

Full relayer suite after adding HOOD/SNDK coverage:

460 passed, 2 skipped

Suite counts move as coverage is added, so they are recorded against the commit that produced them rather than as a standing figure. The count above was taken at the 2026-08-10 revision. Re-run on 2026-09-10 against relayer commit `4235882c0118`, the same suite reports:

656 passed, 2 skipped

The growth is added coverage, not changed pricing behaviour: every assertion in this paper still holds, and the focused pricing and geometry files above return the same 22 passed at both commits.

The permanent pricing sweep covers:

CASHCAT	spot 0.1043	width 0.02
LOW1	spot 1.13	width 0.05
RIVN	spot 18.8	width 0.5
HOOD	spot 82.0	width 3.0
SNDK	spot 1828.9	width 45.0
ETH	spot 3500.0	width 50.0
BTC	spot 65000.0	width 1000.0

with annualized sigma values:

0.25, 0.75, 1.5, 3.0, 6.0, 8.0, 10.0

and horizons:

1 day, 3 days, 7 days, 10 days, 30 days

For each case, the test asserts:

```
peak finite probability is the spot bucket
left side is monotone increasing into the peak
right side is monotone decreasing away from the peak
finite probabilities plus tails sum to 1.0
```

Additional geometry tests assert that CASHCAT, HOOD, and SNKD stay spot-centered under high volatility and that CASHCAT keeps its clean low-price width above zero.

23. What This Paper Does Not Claim

This paper does not claim that Black-Scholes is perfect. It claims that Black-Scholes remains the approved deterministic baseline and that the deployed kernel now avoids the specific long-duration equal-dollar bucket distortion observed in CASHCAT-like markets.

This paper does not claim every displayed ticket price is exact raw probability. Sub-cent compression, the executable floor, rounding, and the `\$1` cap are product denomination rules. Raw model probability and final executable quote must remain separately logged and auditable.

This paper does not claim every finite grid captures 93 percent mass at absurd volatility. If the spot-centering cap is active, the finite grid may capture less and the tails carry more. That is the correct outcome when the alternative would make the finite book unreadable or push spot away from the middle.

This paper does not claim on-chain pushes can land without gas. Deployment, health, and model correctness are separate from wallet runway. Gas shortfall is an operational blocker, not evidence that the pricing formula failed.

This paper does not claim legacy markets all have the new 9-row shape. Existing priced markets are repriced on their immutable created grids until close, even if they are unsold. New active markets use the current 9+2 structure.

Finally, and most importantly for anyone auditing the running system rather than the model: this paper does not claim to describe the whole relayer. Its scope is the pricing kernel and the grid geometry that kernel prices onto. The relayer that carries those quotes to chain is a larger system, and the following parts of it are deliberately out of scope here, each governed by its own document or module rather than by this paper:

repricing cadence	src/relayer/cadence.py, cadence_estimator.py and docs/PRICING-CADENCE-SPEC.md
quote transport	push_tickets_price / _v2 / _v3 in push_price.py, including the V5 timestamped-quote path that carries an on-chain sequence number and freshness check
settlement batching	resolve_lotteries_batch and MAX_RESOLVES_PER_BATCH in src/relayer/transaction_policy.py
gas sizing	_bounded_gas_limit in src/relayer/relayer.py
market creation	src/relayer/create_lottery.py and docs/MARKET-CREATION-RUNBOOK.md
points and revenue	the points_*, revenue_*, and entitlement_* modules

Section 19 and Appendix C describe quote freshness as an operational property. That remains true, but a reader should know it is operational in a specific and scheduled sense: repricing runs on a cadence ladder that tightens as a market approaches close, and the V3 path enforces freshness on-chain. The cadence specification, not this paper, is the authority on when a quote is pushed. This paper is the authority on what that quote contains.

24. Reviewer Checklist

A reviewer should be able to verify:

1. Active new markets use 9 finite buckets plus 2 grouped tails.
2. The finite count derives from one relayer authority.
3. 1-day uses 23h wagering plus 1h lock.
4. 3-day uses 60h wagering plus 12h lock.
5. 7-day uses 6d wagering plus 1d lock.
6. Pricing model time is maturity-based.
7. Tradability stops at closeTimestamp.
8. No finite bucket starts at zero.
9. Spot is in the middle finite bucket at launch.
10. Bucket width is a clean 1/2/5 tick times a power of ten.
11. Low-price assets apply the spot-centering cap when needed.
12. The finite kernel uses centered drift.
13. Total horizon volatility is clamped to [0.01, 0.45].
14. Finite masses are log-width equalized.
15. Equalized finite probabilities preserve the same finite mass.
16. Downside and upside tails use the same centered/capped distribution.
17. Finite rows plus tails partition to 1.0 at the model layer.
18. Product margin is explicit and currently 1.0.
19. Final ticket quote is capped at \$1.000.

20. Minimum executable quote is \$0.001.
21. Sub-cent compression uses anchor 1e-10 and band top \$0.009.
22. Priced markets reprice on the immutable on-chain grid, even if unsold.
23. An already priced market cannot recenter because spot leaves the finite grid.
24. Read-back count is not confused with new-market finite count.
25. One-day maxQuoteAge is zero by launch policy.
26. Quote freshness is monitored operationally.
27. Contract blocks new exposure after close.
28. Contract blocks buys/matches while suspended.
29. Relay gas runway is monitored separately from model health.
30. Frontend points to the same deployed contract/subgraph generation.
31. Regression tests include CASHCAT, HOOD, and SNDK stress cases.

25. Failure Modes and Required Response

Market creation or promotion should fail closed for:

- unsupported token
- unsupported duration index
- missing price source
- empty price history
- non-positive spot
- non-finite spot
- non-positive volatility
- non-finite volatility
- derived bucket width below precision
- unreadable or uncentered launch grid
- finite-mass failure when centering cap is not active
- first price push failure
- subgraph read-back failure
- attempted repricing that changes an already priced grid
- frontend version mismatch
- relayer gas below threshold
- source-data outage
- unresolved contract/subgraph deployment drift

The response should match the failure owner. If the input data is bad, fix the source. If geometry fails, skip or redesign the market. If gas is low, fund the relayer wallet. If the subgraph is stale, wait for indexing or reconcile the deployment. Manual bucket overrides should not be used to force a market through review.

26. Implementation Map

The public paper maps to the live relayer as follows:

- pricing kernel:
C:\Users\Connect\Desktop\bullseye-price-relayer\src\models\black_scholes.py
- market geometry:
C:\Users\Connect\Desktop\bullseye-price-relayer\src\relayer\market_geometry.py
- pricing constants:
C:\Users\Connect\Desktop\bullseye-price-relayer\src\constants\constants.py
- implementation-facing spec:
C:\Users\Connect\Desktop\bullseye-price-relayer\docs\pricing-model.md
- public frontend paper source:
C:\Users\Connect\Desktop\bullseye-frontend\docs\bullseye-pricing-paper-source.md
- public linked PDF:
C:\Users\Connect\Desktop\bullseye-frontend\public\docs\bullseye-pricing-paper.pdf

The linked PDF should always be regenerated from the Markdown source after this paper changes.

27. Conclusion

The current Bullseye pricing architecture is probability-first, partition-complete, centered, and operationally bounded. It still uses Black-Scholes as the foundation, but it no longer exposes raw textbook d^2 interval mass directly as finite bucket probability in cases where that creates visible product distortion.

The current system now does five things that the earlier paper did not fully capture:

1. It centers the visible finite distribution on spot.
2. It caps total horizon volatility at 0.45.
3. It log-width equalizes finite bucket masses.
4. It freezes a market's grid after first pricing, even before sales.
5. It verifies the shape across CASHCAT, HOOD, SNDK, and other price scales.

New 1-day, 3-day, and 7-day markets all use 9 finite buckets plus two grouped tails. Longer or more volatile markets express uncertainty through wider clean buckets and honest tails. Low-priced markets keep the spot range in the middle and do not start finite buckets at zero. Already priced markets keep their published ranges until close; quote updates move prices, not outcome definitions. The relayer owns pricing and monitoring. The contract owns close, suspension, grid immutability, and resolution safety.

That is the current source-of-truth structure for Bullseye range pricing.

Appendix A. Old Formula vs Current Kernel

The old direct finite formula was:

```
d2(K) = (ln(S / K) - 0.5 * sigma^2 * T) / (sigma * sqrt(T))
P(L <= S_T < U) = Phi(d2(L)) - Phi(d2(U))
```

That is a valid Black-Scholes terminal stock probability object in isolation, but it was not sufficient for the visible Bullseye finite book. Two issues appeared in production review:

```
textbook drift: median shifts below spot as sigma^2 * T grows
equal-dollar intervals: low-price buckets cover larger log-return width
```

The current finite kernel is:

```
v = min(max(sigma * sqrt(T), 0.01), 0.45)
d(K) = ln(S / K) / v
mass_i = Phi(d(K_low_i)) - Phi(d(K_high_i))
score_i = mass_i / ln(K_high_i / K_low_i)
finite_quote_i = sum(mass_i) * score_i / sum(score_i)
```

The tails remain direct outside mass from the same centered/capped distribution. This is why the finite visible rows can be neutral and readable without losing complete partition accounting.

Appendix B. Why Same Count Can Fit Volatile Assets

A volatile asset does not require an unlimited number of rows. It requires a wider span. Bullseye fixes the count for product clarity and lets width adapt:

```
more volatility -> wider clean buckets and/or larger tails
more duration -> wider clean buckets and/or larger tails
lower volatility -> narrower clean buckets
same product class -> same visible count
```

This is similar to practical strike listings in ordinary option markets. Users see human-sized ranges. The relevant range span changes with asset and expiry.

The alternative designs are worse:

```
fixed dollar width for every asset -> volatile assets look broken
unlimited rows for volatile assets -> UI becomes inconsistent
manual bucket override -> source of truth drifts again
```

The approved structure is fixed count, dynamic clean width, and grouped tails.

Appendix C. Operational Freshness

One-day `maxQuoteAge = 0` is not a permission to ignore freshness. It is a decision to keep quote-age enforcement operational at launch instead of contract-enforced.

The reason is practical. Some assets can remain inside the same bucket for hours. Requiring a heartbeat transaction every few minutes can create gas spam, and if the heartbeat fails, the market can abruptly stop accepting buys even when the price is still inside the same range. The launch structure avoids that brittleness.

Required operational checks are:

- latest quote age by market
- source quote age
- source fetch failures
- last successful ticket push
- last successful settlement push
- gas balance
- nonce/replacement failures
- suspension status
- cron health
- subgraph indexing lag

If those checks fail, the response is to suspend purchases or fix operations, not to add hidden contract complexity after deployment.

Appendix D. Audit Questions for a New Market

For every newly created market, the operator should be able to answer:

- What token was approved?
- What duration index was used?
- What live/lock pair did that index select?
- What finite bucket count did the class select?
- What spot price was used?
- What price source produced it?
- What historical window produced sigma?
- What annualized sigma was computed?
- What total model horizon was used?
- What raw total vol was computed?
- Was total vol clamped to $[0.01, 0.45]$, and at which end?
- What raw bucket width was computed?
- What clean bucket width was selected?
- Was spot-centering cap applied?
- What first finite lower bound resulted?
- What finite end resulted?
- Is spot in the middle finite bucket?
- What expected finite mass was captured?
- What downside tail probability was computed?
- What upside tail probability was computed?
- Did finite plus tails sum to 1.0?
- What contract bucketSize was encoded?
- Did the first push include exactly 9 finite prices?
- Did the grouped-tail push include both tails?
- Can the subgraph read the same market?
- Can the frontend display the exact bucket bounds?
- Does the relayer have enough gas runway?

If these questions cannot be answered from logs, tests, and source code, the market should not be treated as production-grade.