



SniperBot

A Research Engine for Small Wins

Whitepaper & Technical Reference

Version 2.6.0

October 16, 2026

An honest, technically-grounded analysis of a Solana-native trading research platform that pivoted from fresh meme-coin launches to established blue-chip crypto — with Jito bundle MEV protection and real on-chain P&L accounting.

This document is provided for educational and informational purposes only. SniperBot is research software, not a registered investment adviser, broker, or financial service. Nothing herein constitutes financial, investment, legal, or tax advice. Trading crypto-assets involves extreme risk of total capital loss.

Important Disclaimer

NOT FINANCIAL ADVICE — READ CAREFULLY

SniperBot is **research and simulation software**. It does not execute trades on your behalf without per-transaction authorization, manage your assets as a custodian, or act as an investment adviser. All output — risk scores, simulations, strategy presets, analytics, and alerts — is generated by deterministic and statistical models and is presented strictly as information to support your own independent research.

The platform now spans two materially different asset populations: (a) freshly-launched Pump.fun meme coins, of which the large majority are rugs, honeypots, or abandoned within minutes of launch; and (b) established blue-chip crypto-assets (SOL, USDC, USDT, WBTC, WETH, and others) with deep liquidity and real price discovery. The risk profile of these two populations differs by orders of magnitude, but neither is risk-free: blue-chips can and do draw down 50%+ in single trading days. Past simulation performance does not predict future results. Any capital you commit to live trading can be lost in full, including through MEV extraction, smart-contract bugs, counterparty default, or sudden liquidity withdrawal. **Never trade with funds you cannot afford to lose entirely.**

SniperBot is not registered with the SEC, CFTC, FinCEN, or any financial regulator. Depending on your jurisdiction, using automated software to interact with tokens may carry legal, tax, and licensing obligations that are entirely your responsibility to determine and satisfy. Consult a qualified attorney and tax professional before using this software with real funds.

By using SniperBot you acknowledge that the authors and operators accept no liability for any losses, damages, or legal consequences arising from your use of the software or reliance on its output.

Table of Contents

1. Executive Summary
2. Product Overview
3. Architecture
4. Strengths
5. Weaknesses
6. Opportunities
7. Threats
8. Technical Specifications
9. Risk Management
10. Regulatory & Legal Considerations
11. Conclusion

1. Executive Summary

SniperBot is a Solana-native research platform. The v1.0.0 release targeted the fastest, noisiest corner of the crypto market: freshly-launched meme coins on Pump.fun. The v2.0.0 release retained every meme-coin capability (the launch feed, risk engine, simulation studio, and execution path are unchanged) but added a **major strategic pivot**: the platform now also trades **established blue-chip crypto-assets** — SOL, USDC, USDT, WBTC, WETH, BONK, WIF, RAY, XRP, LINK, DOT, MATIC, LTC, AVAX, NEAR, SHIB, and DOGE — through the same Phantom + Jupiter + Jito + Helius execution path. The v2.1.0 release documented here is a comprehensive **security, auth, and ML hardening round** — closing the v2.0.0 auth-model gap (format-only wallet verification replaced by signed-message challenge→sign→verify), wiring the previously-decorative ML subsystem (BayesianTuner.observe + PresetBandit.update now fire), and adding server-side swap signature verification (verifySwapTx via Helius getTransaction) so LIVE trades are cryptographically verified on-chain before isLive=true is set. Thirty-plus surgical fixes were applied across four review rounds.

Three concrete engineering changes enable the pivot and materially improve the meme-coin path as well. First, **Jito bundle protection (E1)** is now implemented: signed transactions are submitted to Jito's mainnet block engine via submitViaJito(), bypassing the public mempool where MEV sandwich bots front-run retail swaps. Second, **server-side transaction confirmation via Helius** replaces client-side connection().confirmTransaction(); a new /api/wallet/confirm-tx route polls Helius getSignatureStatuses every 2s for 30s, eliminating the 429 rate-limit storms that plagued the public-RPC path. Third, **real on-chain P&L** is computed directly from the Jupiter swap response's outAmount (in lamports) and stored as realizedPnlSol in the database — eliminating the tier-accounting lies of the v1.0.0 path.

The platform's central thesis remains contrarian to the dominant meme-coin narrative — **consistent, small, risk-managed gains** compounded across many micro-trades are more attainable and more sustainable than lottery-ticket speculation. The five strategy presets — Small Wins (+5/+10/+15/+20% tiered take-profit), Balanced, Moonshot, **Momentum** (new; \$50K+ min liquidity, +20/+50/2x/3x tiers, 15% trailing, -15% SL), and **Blue Chip** (new; \$500K+ min liquidity, +5/+10/+20/+50% tiers, 10% trailing, -10% SL, 0.2 SOL buy amount) — encode that philosophy across both asset populations, with the Small Wins preset as the recommended default for meme coins and the Blue Chip preset as the recommended default for established coins.

Key Differentiators

SniperBot distinguishes itself from the sea of Telegram-call bots and copy-trading scripts through several concrete, verifiable design choices:

- **Real-data integration, not mock feeds.** The launch feed is streamed live from PumpPortal's WebSocket service; on-chain enrichment (mint authority, freeze authority, top holders, wallet balance) is pulled from a Helius RPC node; blue-chip prices are polled from the Kraken REST API every 15 seconds; transaction confirmation is polled from Helius every 2 seconds. There are no simulated prices dressed up as real ones.

- **A transparent, decomposable risk engine.** Every meme-coin candidate receives a 0–100 score composed of eight named checks — holder topology, effective insider ownership, volume-to-liquidity ratio, liquidity depth, age, authority flags, and more. The score and its components are visible to the user, not a black box. Blue-chip tokens bypass the risk gate (zero rug risk = no safety scoring needed) — a deliberate, auditable design choice.
- **Jito bundle MEV protection (E1).** Swaps are signed locally with Phantom's `signTransaction` and submitted to Jito's block engine as a single-tx bundle that bypasses the public mempool. A graceful fallback to `signAndSendTransaction` exists if the Jito endpoint is unreachable. The CSP allow-list includes `https://mainnet.block-engine.jito.wtf`.
- **Real on-chain P&L accounting.** The `LiveAutoSellArgs` contract carries `buyAmountSol`; the `LiveAutoSellResult` contract carries `realizedPnlSol`; the `onLiveAutoSell` callback computes `realPnlSol = (SOL received from sell) - buyAmountSol` directly from the Jupiter swap response's `outAmount` (in lamports, converted to SOL). The database stores the REAL on-chain P&L.
- **Self-tuning autonomy.** An auto-tune engine runs walk-forward optimization, Bayesian threshold tuning, and an ϵ -greedy bandit over the five strategy presets, selecting parameters based on observed regime rather than fixed guesses.
- **Simulation before execution.** A full Simulation Studio (forward simulation, Monte Carlo projection, historical backtest, stress scenarios, strategy comparison matrix) lets a user test an idea against real data before risking a single lamport — and works identically across meme-coin and blue-chip populations.
- **17 verified blue-chip mint addresses.** The `BLUE_CHIP_TOKENS` table is a hand-curated list of seventeen Solana mint addresses — each verified live via Helius RPC `getAccountInfo` — representing the deepest, lowest-rug-risk assets on the network: native SOL, USDC, USDT, wrapped BTC and ETH, and the major cross-chain wrapped majors (XRP, LINK, DOT, MATIC, LTC, AVAX, NEAR, SHIB, DOGE) plus established Solana ecosystem tokens (BONK, WIF, RAY).
- **Honest about its own limits.** The whitepaper you are reading devotes dedicated sections to weaknesses and threats — including the uncomfortable fact that the Jupiter free-tier API still lacks MEV protection (Jito helps but is not a complete solution), and that no private mempool / stake-relayer relationship exists for inclusion guarantees.

Who SniperBot Is For

SniperBot is designed for the technically curious retail participant who wants to engage with the Solana market with the rigor of a quant lab rather than the impulses of a casino. The v2.0.0 blue-chip pivot broadens the audience to include users whose primary interest is swing-trading established coins (BTC, ETH, SOL) on short timeframes — a use case the platform now serves natively via the Kraken price feed and Blue Chip preset. It assumes comfort with self-custodial wallets, a working understanding of Solana's transaction model, and the emotional discipline to follow a pre-committed risk plan. It is explicitly **not** for users seeking passive income, guaranteed returns, or someone else to manage their capital.

Document Scope

This whitepaper provides a complete technical and analytical reference for SniperBot as of its v2.6.0 release. It documents the product surface, internal architecture, measured strengths, known weaknesses, near-term opportunities, and a candid threat model spanning legal, security, market, and technical domains. It concludes with an honest readiness assessment. The document is intentionally non-promotional; its goal is to give a technically literate reader enough information to evaluate the platform on its merits.

2. Product Overview

SniperBot is structured as a single-page Next.js application organized around **eight specialized bots** — collectively the **Trading Desk** — backed by a Socket.io mini-service, a Prisma/SQLite persistence layer, and a Caddy gateway. The v2.0.0 release adds the BlueChipScanner bot (8th in the desk), the Jito bundle submission path, the server-side confirmation route, and the Kraken blue-chip polling loop. The following subsections describe each functional area in turn.

2.1 The 8-Bot Trading Desk

Rather than a single monolithic "sniper", SniperBot decomposes the trading workflow into eight cooperating specialists. Each bot owns a single concern, exposes its state through the shared dashboard, and can be armed or disarmed independently. This separation makes the system auditable: a user can see exactly which bot flagged a token and why, rather than reasoning about an opaque aggregate signal. The 8th bot — BlueChipScanner — is the v2.0.0 addition that powers the blue-chip pivot.

Bot	Role	Primary Inputs
TrendingScanner	Surfaces tokens gaining momentum across the launch feed and secondary markets.	PumpPortal feed, price/volume deltas
RugDetective	Runs the 8-check risk model and flags probable rugs, honeypots, and insider clusters.	Helius on-chain data, authority flags, holder topology
SocialHunter	Scans social channels for emerging narrative momentum and call channels.	X/Twitter, Telegram (roadmap)
WhaleTracker	Monitors large-wallet accumulation and distribution patterns on candidate tokens.	Holder deltas, wallet labels
RiskManager	Enforces position sizing, daily loss caps, circuit breakers, and narrative-concentration limits.	Portfolio state, config, kill-switch
SniperBot	Owns execution: builds the swap transaction, sets slippage/priority fee, routes via Jupiter, arms auto-sell, submits via Jito bundle.	Jupiter quote, ComputeBudget, Phantom signature, Jito endpoint
PerformanceOptimizer	Runs the auto-tune engine (walk-forward, Bayesian, ϵ -greedy bandit) and recommends parameter updates.	Historical fills, regime features
BlueChipScanner	v2.0.0 — Surfaces the 17 verified blue-chip mints in the live feed using real Kraken prices; bypasses the risk gate.	BLUE_CHIP_TOKENS table, Kraken REST API, Helius getAccountInfo verification

2.2 Real-Time Pump.fun Launch Feed (D1)

The platform's heartbeat (for the meme-coin path) is a persistent WebSocket connection to PumpPortal's WSS endpoint, which streams newly-launched Pump.fun tokens the instant they appear on-chain. A dedicated Socket.io mini-service (port 3003) terminates the upstream connection, normalizes the payload into a stable contract (*mint*, *symbol*, *name*, *marketCapUsd*, *liquidityUsd*, *holders*, *ageSeconds*, *priceUsd*, *launchedAt*), and re-emits a **token-launch** event to all connected browser clients.

This design isolates the browser from upstream connection churn and rate limits: the mini-service holds a single resilient connection and multiplexes events to many frontends. The frontend subscribes with `io('/?XTransformPort=3003')` so the Caddy gateway can forward traffic to the correct internal port without exposing absolute URLs. A companion **price-update** channel streams live mark-to-market for any token the user holds, enabling real-time unrealized P&L. **Blue-chip tokens are injected into the same feed by the Kraken polling loop** (see 2.14) so the user sees SOL, BTC, ETH, etc. in the Live Feed panel alongside fresh launches, with real Kraken prices as their mark.

2.3 On-Chain Enrichment via Helius RPC (D2)

Every candidate token is enriched with real on-chain state fetched from a Helius RPC node. The enrichment surface includes:

- **Mint authority** — if the mint authority is not revoked, the issuer can mint unlimited additional supply, a classic rug vector. Flags it red.
- **Freeze authority** — if the freeze authority is retained, the issuer can lock holders out of transferring their tokens. Flags it red.
- **Top holders** — the distribution of the top N wallets by percentage of supply. A handful of wallets holding >50% combined is a strong insider cluster signal.
- **Wallet balance** — the deployer / creator wallet's SOL and token balance, used to detect funded-but-unrugged setups and to watch for creator sell-offs in real time.
- **Blue-chip verification** — every entry in the BLUE_CHIP_TOKENS table is verified on-chain via `getAccountInfo`; the verified flag is the deterministic on-chain proof that the mint address is the canonical mint for the named asset (SOL, USDC, etc.).

This data is not synthetic. Helius is a production-grade Solana RPC provider, and the enrichment fields are the same ones a careful manual researcher would pull from Solscan — SniperBot simply does it automatically, within milliseconds of launch, for every token in the feed.

2.4 Risk Engine: 8-Check 0–100 Scoring

At the core of RugDetective is a transparent, decomposable scoring system that maps each candidate token to a 0–100 risk score. The score is not a black-box ML classifier; it is the weighted sum of eight named, inspectable checks. A user can open the risk dialog for any token and see exactly which checks fired and how much each contributed. The eight checks are:

#	Check	What it measures	Weight
1	Holder topology	Concentration of supply in top wallets; Gini-style dispersion	High
2	Effective insider ownership	Insider-attributable supply net of LP/burn; cluster heuristics	High
3	Volume / liquidity ratio	Trade volume relative to liquidity depth — flags wash-trading and thin books	Medium
4	Liquidity depth	Absolute liquidity USD and burn status of LP tokens	Medium
5	Age & launch phase	Seconds since deploy; flags tokens too new to have a track record	Low
6	Mint authority	Whether mint authority is revoked (revoked = safer)	High
7	Freeze authority	Whether freeze authority is revoked (revoked = safer)	High
8	Creator wallet history	Prior launches by the same deployer and their rug rate	Medium

The composite score gates downstream behavior: tokens below a configurable threshold are hidden from the feed or surfaced with a red banner; the SniperBot execution engine refuses to arm on tokens that fail the hard authority checks (#6 and #7) regardless of the composite score. This guarantees the platform will never auto-execute against a token whose issuer retains mint or freeze authority — the two most common rug mechanisms on Pump.fun. **Blue-chip tokens bypass the risk gate entirely:** the `blueChipToTokenLaunch()` helper produces synthetic `TokenLaunch` objects with all-safe risk fields, because a verified blue-chip mint has zero rug risk by construction — there is no safety scoring to do.

2.5 Execution Engine

When the user arms SniperBot on a scored candidate (or a blue-chip token after the user clicks Inspect → LIVE BUY), the execution engine constructs and submits the swap. It is built around five composable primitives, each of which materially affects fill quality and MEV exposure on Solana:

- **Dynamic slippage.** Slippage tolerance is not a fixed constant; it is derived from the token's measured volatility, liquidity depth, and the current network congestion proxy. Tighter on deep, stable books (blue-chip); wider on thin, fresh launches to avoid stuck transactions.
- **ComputeBudget instruction.** A dedicated `ComputeBudget` instruction is added to set the compute-unit limit and price, ensuring the swap does not silently fail on CU exhaustion — a common failure mode for complex aggregator routes.
- **Jupiter aggregator routing.** Swaps are routed through Jupiter's aggregator, which splits across DEX routes to minimize price impact. The platform relies on Jupiter's quote/swap API (free-tier lite-api) for route discovery and transaction construction.
- **Priority-fee oracle.** A priority fee (in lamports per compute unit) is attached based on a recent-priority-fee oracle so the transaction lands promptly under contention without overpaying in calm conditions.

- **Jito bundle submission (E1 — new in v2.0.0).** The signed transaction is submitted to Jito's mainnet block engine via `submitViaJito()` as a single-tx bundle, bypassing the public mempool where MEV sandwich bots front-run retail swaps. If the Jito endpoint is unreachable, the engine falls back to `signAndSendTransaction`. See 2.10 for full detail.

After a fill, SniperBot arms the **auto-sell** logic, which monitors the position against take-profit (TP), stop-loss (SL), and trailing-stop thresholds derived from the active strategy preset, and constructs the reverse swap when a threshold is crossed. For blue-chip positions, the Kraken polling loop (2.14) drives the price updates that trigger auto-sell.

2.6 Strategy Presets

Five presets encode distinct risk/reward philosophies. They are the user's high-level choice; the auto-tune engine adjusts their internal parameters. The v2.0.0 release adds the **Momentum** and **Blue Chip** presets.

Preset	Take-profit	Stop-loss	Position size	Use case
Small Wins	Tiered +5/+10/+15/+20% (partial exits)	Tight, ~10–15%	0.05 SOL fixed	Recommended default for meme coins; high-frequency micro-gains
Balanced	Tiered +50/+100/+200/+400%	~30%	0.10 SOL	Moderate conviction; fewer, larger
Moonshot	Tiered +200/+400/+900/+1900%	Wide / trailing 30%	0.15 SOL	Lottery-ticket allocation only
Momentum	Tiered +20/+50/+100/+200%	-15% SL, 15% trailing	0.10 SOL	Established tokens with proven momentum; \$50K+ min liquidity
Blue Chip	Tiered +5/+10/+20/+50%	-10% SL, 10% trailing	0.20 SOL	Swing-trade top-20 established coins; \$500K+ min liquidity, zero rug risk

The Small Wins preset is the philosophical anchor of the meme-coin path. Rather than waiting for one position to 10x (which most never do), it takes partial profits at four ascending thresholds, banking realized gains while letting a small tail ride. Compounded across hundreds of micro-trades, even a modest per-trade EV can produce a positive equity curve — *provided* the win rate and average win/loss ratio exceed the base-rate drag of the rug-prone Pump.fun universe (see Weaknesses). The Blue Chip preset, by contrast, plays an asset population with zero rug risk and deep liquidity — the limiting factor there is whether the user can time swings, not whether the asset will be rugged out from under them.

2.7 Auto-Tune Engine

PerformanceOptimizer is the platform's autonomous layer. It treats the five strategy presets and their internal thresholds as a parameter space to be searched, not guessed. Four techniques compose the engine:

- **Walk-forward optimization.** The engine trains parameters on a rolling in-sample window and evaluates them on the immediately following out-of-sample window, then rolls forward. This guards against overfitting to a single regime and approximates how parameters would have performed live.
- **Bayesian threshold tuning.** Risk-score thresholds, TP/SL levels, and position sizes are tuned via Bayesian optimization, which balances exploration of uncertain regions against exploitation of known-good ones — far more sample-efficient than grid search over a high-dimensional space.
- **ϵ -greedy bandit preset selection.** With probability ϵ the engine tries a non-default preset to gather information; otherwise it runs the empirically best-performing preset for the current regime. ϵ decays over time as confidence grows.
- **Market-regime detection.** A lightweight classifier segments the current market into regimes (e.g., high-volatility/trending, low-volatility/choppy, risk-off) using features derived from feed throughput, SOL price action, and aggregate liquidity. Parameter recommendations are conditioned on the detected regime.

The engine never silently changes live-trading parameters without the user's consent. It surfaces **recommendations** the user can accept or reject, preserving human-in-the-loop control over capital.

2.8 Simulation Studio

Simulation Studio is the platform's research sandbox — a place to test ideas against real data without risking capital. It comprises five modes:

- **Forward simulation on real data.** Replays the live launch feed from a chosen start point and applies a strategy's entry/exit logic as if it were live, producing a hypothetical equity curve and trade log.
- **Monte Carlo projection.** Resamples the trade log (with replacement) thousands of times to produce a distribution of possible equity curves, yielding confidence intervals on drawdown, terminal wealth, and ruin probability — not just a point estimate.
- **Historical backtest.** Replays an archived launch set through a strategy, with the same risk-scoring and enrichment pipeline applied retrospectively.
- **Stress-test scenarios.** Synthetic adverse scenarios — a 50% SOL crash intraday, a liquidity vacuum, a coordinated multi-token rug wave — applied to the strategy to measure tail behavior.
- **Strategy comparison matrix.** Side-by-side metrics (win rate, average win/loss, max drawdown, Sharpe-like ratio, terminal EV) across all presets and parameter variants, so the user can see which strategy *actually* won on the test data rather than which one felt right.

2.9 Live Trading

Live trading is an opt-in mode that the user must explicitly enable. When enabled, SniperBot integrates with the user's **Phantom wallet** (Solana's dominant self-custodial wallet) via the injected `window.solana` provider. The user remains the sole signer of every transaction; SniperBot constructs and proposes swaps but cannot move funds without the user's explicit per-transaction signature (or, where the user has granted it, an auto-sell signature within the armed session).

Swaps are routed through the **Jupiter aggregator**, committing real SOL. The signed transaction is submitted via the **Jito bundle** path (2.10) and confirmed via the **server-side Helius confirmation route**

(2.11). Auto-sell logic monitors every open position and constructs the reverse swap when TP, SL, or trailing-stop thresholds are crossed. For blue-chip positions, the Kraken polling loop (2.14) drives the price updates. All live fills, fees paid, and **realized P&L computed from on-chain** outAmount are persisted and surface in the analytics views alongside simulation results for direct comparison.

2.10 Jito Bundle Protection (E1)

The single most damaging weakness of v1.0.0 — MEV sandwich exposure on the public mempool — is materially remediated in v2.0.0 by Jito bundle submission. The flow is:

- **Local signing.** The user's Phantom wallet calls `signTransaction` (not `signAndSendTransaction`) — the signed bytes are returned to the SniperBot client without being broadcast to the public mempool. This is the structural change that makes Jito possible: the signed tx never enters the public mempool where MEV bots observe and front-run it.
- **Base58 encoding.** The signed transaction bytes are base58-encoded (via the `bs58` npm package, added in v2.0.0) for transport to the Jito endpoint.
- **Jito bundle submission.** The `submitViaJito()` function POSTs the base58-encoded signed transaction to `https://mainnet.block-engine.jito.wtf/api/v1/bundles` as a single-tx bundle. Jito's block engine includes the bundle atomically in a block, bypassing the public mempool entirely. MEV sandwich bots that watch the public mempool for retail swaps never see this transaction until it is already finalized in a block.
- **Graceful fallback.** If the Jito endpoint is unreachable (network failure, block engine down, rate-limit), the `signAndSubmit()` wrapper falls back to Phantom's `signAndSendTransaction`, which broadcasts via the standard public RPC. The fallback is logged so the user can see when MEV protection was and was not applied.
- **CSP allow-list.** The Content-Security-Policy directive `connect-src` now includes `https://mainnet.block-engine.jito.wtf` so the browser is permitted to make the cross-origin POST to the Jito endpoint without a CSP violation.

Jito bundle submission is the platform's most significant execution-path improvement. It does not make the platform immune to MEV — the Jupiter free-tier API still produces quotes that an observer could correlate with subsequent on-chain activity — but it eliminates the dominant public-mempool sandwich vector that made every v1.0.0 swap "food for MEV."

2.11 Server-Side TX Confirmation via Helius

v1.0.0 used client-side `connection().confirmTransaction()` to wait for transaction finality. Under meme-coin mania — when many users were retrying swaps in parallel against the public Solana RPC — this path frequently hit 429 (too many requests) errors, leaving the user unsure whether their swap had landed. v2.0.0 replaces this entirely:

- **New API route.** `GET /api/wallet/confirm-tx?signature=<base58>` is a server-side endpoint that polls Helius `getSignatureStatuses` every 2 seconds for up to 30 seconds (15 attempts).
- **Client-side replacement.** `confirmViaServer()` replaces `connection().confirmTransaction()` in the live swap pipeline. The client makes a single HTTP

request to the route and awaits the response; the route does the polling.

- **Helius rate limits.** Helius's paid tier has materially higher rate limits than the public Solana RPC, and the polling happens server-side where the Helius API key lives (never exposed to the browser).
- **Finality semantics.** The route returns `{ confirmed: true, slot, confirmationStatus }` when Helius reports the transaction as confirmed or finalized, or `{ confirmed: false, error: 'Transaction failed on-chain' }` if Helius reports an `err` field on the signature.

2.12 Real On-Chain P&L Computation

v1.0.0 computed realized P&L from tier accounting — "the user took profit at the +5% tier, so we credit a +5% gain." This was structurally dishonest: the actual SOL received from a sell depended on slippage, priority-fee burn, route selection, and dust left in LP, none of which the tier model captured. v2.0.0 eliminates the accounting lies by reading the real on-chain result:

- **LiveAutoSellArgs** now includes `buyAmountSol` — the original SOL spent on the buy, captured at buy time and threaded through the auto-sell call.
- **LiveAutoSellResult** now includes `realizedPnlSol` — the real on-chain P&L, computed by the callback.
- **The onLiveAutoSell callback** computes: `realPnlSol = (SOL received from sell) - buyAmountSol`. The "SOL received from sell" is read directly from the Jupiter swap response's `outAmount` field, which is denominated in lamports (1 SOL = 10^9 lamports) and converted to SOL by dividing by 10^9 (with the appropriate decimal adjustment for the output mint).
- **The DB stores the REAL on-chain P&L.** No more fake tier accounting — the value surfaced in the dashboard and analytics is the literal SOL delta the user's wallet actually realized, computed from the literal bytes of the swap response.

This change has an important consequence for the analytics views: the win-rate-vs-EV scatter, the P&L calendar, and the cumulative-P&L curve now reflect ground truth rather than a model of ground truth. A user who appears profitable on the dashboard IS profitable on-chain, by definition.

2.13 Momentum Strategy Preset

The Momentum preset (4th of 5) targets tokens that have graduated from the fresh-launch phase into established-but-still-liquid territory. Its parameters — \$50K+ minimum liquidity, +20/+50/+100/+200% tiered take-profit, 15% trailing stop, -15% stop-loss — are deliberately looser than Small Wins because the asset population is lower-variance: established tokens do not rug in the way fresh launches do. The preset is a stepping stone between the meme-coin and blue-chip presets — a user comfortable with Small Wins who wants larger swing potential without taking on the deep-liquidity blue-chip universe can run Momentum.

2.14 Blue Chip Strategy

The Blue Chip preset is the major v2.0.0 strategic pivot. Where v1.0.0 was framed as a meme-coin research engine, v2.0.0 reframes SniperBot as a **Solana-native swing-trading platform for established crypto-assets**. The pivot is enabled by three pieces:

- **17 verified Solana mint addresses.** The `BLUE_CHIP_TOKENS` table in `src/lib/bluechip-tokens.ts` is a hand-curated list of seventeen canonical mint addresses: native SOL, USDC, USDT, wrapped BTC and ETH (via Portal/Wormhole), and the major cross-chain wrapped majors (XRP, LINK, DOT, MATIC, LTC, AVAX, NEAR, SHIB, DOGE) plus established Solana ecosystem tokens (BONK, WIF, RAY). Every entry is verified live via Helius RPC `getAccountInfo`; the verification is the deterministic on-chain proof that the mint address is the canonical mint for the named asset.
- **Blue Chip preset parameters.** \$500K+ minimum liquidity, +5/+10/+20/+50% tiered take-profit, 10% trailing stop, -10% stop-loss, 0.2 SOL buy amount. These are deliberately conservative — established coins move 5-20% per swing, not 100-1000% per launch — so the tiers are tighter and the SL is closer.
- **Risk-gate bypass.** `blueChipToTokenLaunch()` creates synthetic `TokenLaunch` objects with all-safe risk fields (mint authority burned, freeze authority burned, deep liquidity, top holders non-concentrated). The 8-check risk model is bypassed because a verified blue-chip mint has zero rug risk by construction — there is no safety scoring to do. This is a deliberate, auditable design choice.

The Kraken API is expanded with twelve blue-chip pairs — SOLUSD, XBTUSD (Bitcoin), ETHUSD, XRPUSD, LINKUSD, DOTUSD, MATICUSD, LTCUSD, AVAXUSD, NEARUSD, ADAUSD, ATOMUSD — for the live price feed (see 2.14 for the polling loop). The `BlueChipScanner` bot (8th in the Trading Desk) owns the surfacing of these tokens in the live feed.

2.15 Blue-Chip LIVE Trading

Blue-chip live trading is the full round-trip execution path for established coins. The Kraken polling loop in `use-sniper-engine.ts` runs every 15 seconds and performs three duties:

- **Inject blue-chip tokens into engine.feed.** Each poll fetches the latest Kraken prices for the twelve supported pairs and, for each blue-chip token in the `BLUE_CHIP_TOKENS` table, calls `blueChipToTokenLaunch(token, price)` to produce a synthetic `TokenLaunch` object that the rest of the engine treats identically to a Pump.fun launch. The user sees SOL, BTC, ETH, etc. in the Live Feed panel with real Kraken prices as their mark.
- **Update engine.priceMap with real Kraken prices for open positions.** For any open blue-chip position, the loop updates the corresponding entry in `engine.priceMap` with the latest Kraken price. This is the input the auto-sell effect (TP/SL/trailing) reads — so a blue-chip position's TP/SL fires on real Kraken price changes, not on a stale or synthetic mark.
- **Auto-sell on TP/SL.** The same auto-sell effect that handles meme-coin positions also handles blue-chip positions: when the Kraken-driven price in the `priceMap` crosses the active preset's TP, SL, or trailing threshold, the reverse swap is constructed, signed via Phantom, submitted via Jito, confirmed via the server-side route, and the realized P&L computed from the Jupiter swap response's `outAmount`.

The full round-trip for a blue-chip trade is therefore: **Jupiter swap** → **Phantom sign** → **Jito bundle** → **server-side confirm** → **DB persist** → **Kraken price monitoring** → **auto-sell on TP/SL** → **on-chain P&L**. The user can click **Inspect** → **LIVE BUY** on any blue-chip token in the Live Feed to enter a position; the platform handles the rest, including the exit.

2.16 Portfolio Dashboard

The dashboard is the user's home base. It displays the **real on-chain SOL balance** of the connected wallet (fetched live from Helius RPC, not cached), the **unrealized P&L** across all open positions (streamed via the price-update channel for meme-coins and via the Kraken polling loop for blue-chips), the rolling **win rate**, and a **simulation-projection overlay** that plots the live equity curve against the Monte Carlo confidence band from Simulation Studio — so the user can see, at a glance, whether live results are tracking inside, above, or below the simulated expectation. **The dashboard now shows REAL on-chain P&L (computed from outAmount), not fake tier accounting.**

2.17 P&L Analytics

Beyond the dashboard's headline numbers, a dedicated analytics view provides the diagnostic depth a serious user needs:

- **Cumulative P&L curve** — realized + unrealized, over the user's full history, with a drawdown overlay. *Realized* now means the literal on-chain SOL delta computed from outAmount.
- **Drawdown chart** — peak-to-trough equity decline, the metric that actually determines whether a strategy is survivable.
- **Win-rate-vs-EV scatter** — plots each strategy variant's win rate against its expected value per trade, exposing the uncomfortable truth that high win rates can coexist with negative EV (many small wins, one catastrophic loss).
- **P&L calendar** — a GitHub-style calendar heatmap of daily P&L, surfacing streaks and cluster risk visually.
- **Strategy A/B simulator** — counterfactual comparison: "had I run Balanced instead of Small Wins, or Blue Chip instead of Momentum, over the last 30 days, what would my equity curve look like?"

2.18 Alerts, Reports & UX

SniperBot is designed to be observable even when the user is not at the screen. Notifications span four channels:

- **Email reports** — automated daily, weekly, and monthly summaries of fills, realized P&L, win rate, and notable risk events, delivered via nodemailer.
- **Telegram alerts** — push notifications to a configured Telegram chat for armed-state changes, fills, and risk-limit breaches.
- **Sound alerts** — in-browser audio cues for fills and TP/SL hits, useful when the dashboard is open in a background tab.
- **Web push notifications** — browser-level push for the most important events (kill-switch trip, daily-loss-cap hit).

On the UX side, the platform offers a **20-Principle Playbook** — a curated set of trading-discipline principles rendered as collapsible cards, intended to keep the user's behavior aligned with the platform's risk philosophy; a **Replay Mode** that lets a user scrub through any past session token-by-token to understand what the bots saw and did; **collapsible sections** throughout the UI so power users can expand detail and newcomers can keep a clean view; and **ELI8 ("Explain Like I'm 8")** explanations —

plain-language, jargon-free breakdowns of every non-trivial concept, available via a button on the relevant card.

3. Architecture

SniperBot is a single-route Next.js application with a small fleet of supporting services. The v2.0.0 architecture adds the Jito bundle submission path, the server-side Helius confirmation route, and the Kraken polling loop to the existing Socket.io feed mini-service, Prisma/SQLite persistence layer, and Caddy gateway. The architecture prioritizes operational simplicity (one external port, one gateway) without sacrificing the real-time and on-chain data integration that distinguishes the platform.

3.1 Technology Stack

Layer	Technology	Role
Frontend	Next.js 16 (App Router), React 19	Single-route dashboard; client components for wallet/WS/Jito interplay
Language	TypeScript 5 (strict)	End-to-end type safety across UI, API, and WS contracts
Styling	Tailwind CSS 4 + shadcn/ui (New York)	Responsive, themeable component system
Icons	Lucide React	Consistent iconography
State (client)	Zustand	Wallet/session/UI state
State (server)	TanStack Query	API data fetching, caching, invalidation
ORM	Prisma 6	Type-safe SQLite access
Database	SQLite	Config, sniped positions, real-P&L, logs, analytics
Realtime	Socket.io (mini-service, port 3003)	token-launch & price-update multiplexing
Gateway	Caddy	Single external port; XTransformPort routing to internal services
Runtime	Bun	Dev server, scripts, mini-service hot-reload
Charts	Recharts	P&L; curves, drawdown, calendars
Wallet	@solana/web3.js + Phantom (window.solana)	Self-custodial signing; user is sole signer
MEV protection	Jito bundle API (mainnet.block-engine.jito.wtf)	Single-tx bundles bypass public mempool; eliminate sandwich exposure
TX confirmation	Helius RPC (getSignatureStatuses)	Server-side polling via /api/wallet/confirm-tx; replaces client-side confirmTransaction
Blue-chip pricing	Kraken REST API (12 pairs)	15-second polling loop injects blue-chip prices into engine.feed and engine.priceMap
Auth	Wallet-as-auth (NextAuth available)	Wallet address is the principal identifier
Email	nodemailer	Daily/weekly/monthly reports
AI SDK	z-ai-web-dev-sdk (server-only)	ELI8 explanations, summary generation

3.2 Data Flow

The end-to-end flow for a single token, from launch to user decision, proceeds as follows (described textually; a diagram is deliberately omitted to keep the document reproducible from source). The same flow applies to blue-chip tokens, except that the "launch" event is replaced by a Kraken poll and the risk gate is bypassed.

- **1. Launch.** A new token is minted on Pump.fun. PumpPortal's WSS emits the launch event. (For blue-chips, the Kraken 15-second poll substitutes.)
- **2. Ingest.** The Socket.io mini-service (port 3003) receives the event, normalizes it to the shared contract, and emits *token-launch* to all connected browser clients. (For blue-chips, the engine's Kraken poll directly injects synthetic token-launches into `engine.feed`.)
- **3. Route.** The browser's `io( '/?XTransformPort=3003' )` connection receives the event via the Caddy gateway (no absolute URLs cross the wire).
- **4. Enrich.** The frontend triggers enrichment via a Next.js API route, which queries the Helius RPC for mint/freeze authority, top holders, and creator balance. The Helius key lives server-side only. Blue-chips skip enrichment (the mint was verified at table-build time).
- **5. Score.** RugDetective runs the 8-check model, producing the 0–100 score and per-check breakdown. Tokens failing hard authority checks are filtered or flagged. Blue-chips bypass the score entirely.
- **6. Present.** The token appears in the launch feed with its score, enrichment, and per-check detail (expandable). TrendingScanner, WhaleTracker, and BlueChipScanner annotations are layered on.
- **7. Decide.** The user arms SniperBot (or the auto-tune engine recommends arming). The execution engine builds the Jupiter swap with dynamic slippage, ComputeBudget, and a priority fee, and requests a Phantom signature via `signTransaction` (local signing).
- **8. Jito submit.** The signed bytes are base58-encoded and POSTed to the Jito mainnet block engine as a single-tx bundle. If Jito is unreachable, fall back to `signAndSendTransaction` via the standard RPC.
- **9. Server-side confirm.** The client calls `/api/wallet/confirm-tx?signature=...`, which polls Helius `getSignatureStatuses` every 2s for 30s and returns `{ confirmed: true, slot }` on success.
- **10. Monitor.** The fill is persisted with `buyAmountSol` captured. The price-update channel (meme-coins) or the Kraken polling loop (blue-chips) streams mark-to-market into `engine.priceMap`. Auto-sell watches TP/SL/trailing.
- **11. Exit & account.** On a threshold hit, auto-sell builds the reverse swap, the user signs (or the armed session auto-signs), the swap is submitted via Jito, confirmed server-side, and the `onLiveAutoSell` callback computes $realPnLSol = (SOL\ received) - buyAmountSol$ from the Jupiter response's `outAmount`. The REAL on-chain P&L is persisted and surfaced across dashboard, analytics, and reports.

3.3 Jito Bundle Flow

The Jito bundle submission path is the structural change that eliminates the v1.0.0 "food for MEV" weakness. The flow, in order:

- **(a)** Jupiter's `/swap/v1/swap` endpoint returns an unsigned versioned transaction (base64).
- **(b)** The SniperBot client deserializes the unsigned tx via `@solana/web3.js`.
- **(c)** Phantom's `signTransaction` is called — the user approves the swap, Phantom signs locally with the user's private key (which never leaves the wallet), and the signed bytes are returned to the SniperBot client. Crucially, `signTransaction` does NOT broadcast to the public mempool — the signed bytes are returned privately to the caller.
- **(d)** The signed bytes are base58-encoded via the `bs58` npm package (added to dependencies in v2.0.0).
- **(e)** `submitViaJito()` POSTs the base58 tx to `https://mainnet.block-engine.jito.wtf/api/v1/bundles` as a single-tx bundle. Jito's block engine includes the bundle atomically in a block, bypassing the public mempool.
- **(f)** The client calls the server-side confirm route (3.4) to await finality. MEV sandwich bots that watch the public mempool for retail swaps never see this transaction until it is already finalized in a block — they cannot front-run it.
- **(g)** If the Jito endpoint returns an error (network failure, block engine down, rate-limit), `signAndSubmit()` falls back to Phantom's `signAndSendTransaction`, which broadcasts via the standard public RPC. The fallback is logged so the user can see when MEV protection was and was not applied.

3.4 Server-Side Confirmation Flow

The server-side confirmation path replaces client-side `connection().confirmTransaction()` and eliminates the 429 rate-limit storms that plagued the v1.0.0 path. The flow:

- **(a)** After Jito (or the public-RPC fallback) accepts the signed tx, the client receives the base58 signature.
- **(b)** The client calls `GET /api/wallet/confirm-tx?signature=<base58>`.
- **(c)** The route rate-limits the calling IP (30 requests per minute), validates the signature string (≥ 64 chars), and selects the RPC URL — Helius (if `HELIUS_API_KEY` is set) or the public Solana RPC as a fallback.
- **(d)** The route polls `getSignatureStatuses` every 2 seconds for up to 30 seconds (15 attempts). Each poll has a 5-second `AbortSignal` timeout. If a poll fails, the route continues to the next attempt rather than aborting.
- **(e)** On the first poll where Helius reports the transaction's `confirmationStatus` as `confirmed` or `finalized` and `err` is null, the route returns `{ confirmed: true, slot, confirmationStatus }`.
- **(f)** If Helius reports an `err` field on any poll, the route returns `{ confirmed: false, error: 'Transaction failed on-chain' }`.
- **(g)** If all 15 attempts elapse without confirmation or error, the route returns `{ confirmed: false, error: 'tx not found within 30s' }`.

3.5 Kraken Polling Loop

The Kraken polling loop in `use-sniper-engine.ts` is a 15-second `setInterval` that drives the blue-chip side of the platform. Each tick:

- **(a)** Fetches `/api/market/kraken`, which in turn queries the Kraken REST API for the twelve blue-chip pairs: SOLUSD, XBTUSD (Bitcoin), ETHUSD, XRPUSD, LINKUSD, DOTUSD, MATICUSD, LTCUSD, AVAXUSD, NEARUSD, ADAUSD, ATOMUSD.
- **(b)** For each returned price, looks up the corresponding entry in the `BLUE_CHIP_TOKENS` table (matched by `krakenPair`) and calls `blueChipToTokenLaunch(token, price)` to produce a synthetic `TokenLaunch`. These are injected into `engine.feed` so the user sees SOL, BTC, ETH, etc. in the Live Feed panel.
- **(c)** For each open blue-chip position, updates the corresponding entry in `engine.priceMap` with the latest Kraken price. This is the input the auto-sell effect reads — so a blue-chip position's TP/SL fires on real Kraken price changes, not on a stale or synthetic mark.
- **(d)** For blue-chip mints that have open positions but no `priceMap` entry yet (e.g., the position was opened before the first poll), the loop directly fetches the Kraken price for that mint and seeds the `priceMap`.

3.6 Security Headers & Hardening

The Caddy gateway terminates TLS and emits a conservative set of security headers on every response. The intent is to harden the browser surface against the most common web attacks even though the wallet, not a cookie, is the authentication primitive.

Header	Value	Purpose
Content-Security-Policy	default-src 'self'; connect-src 'self' wss:... https://mainnet.block-engine.jito.wtf ...	Mitigates XSS and data exfiltration; restricts WS origins; allows Jito block-engine POSTs
Strict-Transport-Security	max-age=31536000; includeSubDomains	Forces TLS; prevents protocol-downgrade attacks
X-Frame-Options	DENY	Prevents clickjacking via iframe embedding
X-Content-Type-Options	nosniff	Prevents MIME-sniff confusion attacks
Referrer-Policy	strict-origin-when-cross-origin	Limits referrer leakage to upstream RPCs
Permissions-Policy	geolocation=(), microphone=()	Denies unused browser capabilities

API keys (Helius RPC, Jupiter, Jito, Telegram bot token, SMTP credentials, Kraken API key/secret) are held exclusively server-side as environment variables and never serialized into client bundles. The wallet's private key never touches SniperBot infrastructure — Phantom signs locally in the user's browser or extension, and SniperBot only ever sees the signed transaction bytes (which it forwards to Jito). The signed bytes themselves are valid for one specific swap on one specific account; they cannot be replayed

for a different transaction.

4. Strengths

An honest whitepaper does not bury its strengths, but it does present them in proportion. SniperBot's v2.0.0 strengths are concentrated in nine areas where the platform makes verifiable, non-trivial choices that distinguish it from the prevailing meme-coin tooling — including three new strengths landed since v1.0.0: Jito MEV protection, real on-chain P&L, and the blue-chip strategic pivot.

4.1 Real-Data Integration

The single most important strength is that SniperBot is wired to real data sources rather than synthetic mocks dressed up as live feeds. PumpPortal's WSS provides genuine launch events; Helius provides genuine on-chain state and now also drives server-side transaction confirmation; Kraken provides genuine blue-chip prices every 15 seconds; Jito provides a genuine private-mempool-equivalent submission path. This means a user's research, simulations, and live decisions are made against the same ground truth a manual researcher would use — with the advantage of automation and latency measured in milliseconds rather than minutes. Many competing tools either fabricate data or quietly fall back to mocks when an upstream fails; SniperBot's design isolates upstream failure to a visible connection-error state rather than silently degrading to fiction.

4.2 A Sophisticated, Transparent Risk Engine

The 8-check risk model is both more rigorous and more auditable than the binary "safe/unsafe" labels common in this space. Because the score is decomposed into named, weighted checks, a user can reason about *why* a token scored as it did, disagree with a specific check's weight, and develop intuition about the actual rug vectors the model is sensitive to. The hard-gate on mint/freeze authority is a particularly important design choice: it makes the platform structurally incapable of auto-executing the two most common Pump.fun rug mechanisms, regardless of the composite score. The blue-chip risk-gate bypass is the symmetric design choice for the established-coin population: a verified blue-chip mint has zero rug risk by construction, so bypassing the risk engine is correct, not negligent.

4.3 Autonomous Self-Tuning

The auto-tune engine is a genuine differentiator. Walk-forward optimization, Bayesian tuning, and an ϵ -greedy bandit over the five presets are techniques drawn from quantitative finance, not from the meme-coin zeitgeist. They do not guarantee profitability — no method can overcome a structurally negative-EV regime — but they materially reduce the probability that the user is running stale parameters that were good last month and bad this month. Crucially, the engine preserves human-in-the-loop control: it recommends, the user decides, capital is never moved on the engine's sole authority. The engine now spans both asset populations, so a user running the Blue Chip preset benefits from the same walk-forward tuning as a user running Small Wins.

Active Learning Pipeline (new in v2.2.0)

v2.2.0 transforms the ML subsystem from functional-but-advisory to actively driving decisions.

The bot now learns from every 5 closed LIVE trades (reduced from 10 in v2.1.0 for faster adaptation) and applies those lessons to future trades. The full learning pipeline:

1. Observe. When a LIVE position closes, `observeClosedTrades()` in `auto-tune.ts` sorts the closed-trade history ascending by `closedAt`, slices the newest unobserved trades, and feeds each one to `BayesianTuner.observe(score, outcome)` (learns win/loss patterns per safety-score bucket via $\text{Beta}(\alpha, \beta)$ posterior updating) and `PresetBandit.update(presetId, roi)` (learns ROI per preset — unit-normalized via `pnl/buyAmount`, not raw SOL). The REKT threshold scales per-trade: `pnl <= -(t.amountSol × 0.9)` (was a flat -0.05 SOL that broke for buyAmounts > 0.05).

2. Adjust. The `analyzeAndTune()` function runs 6 overlays in sequence, each building on the prior:

Overlay	What it does	When it fires
Walk-forward	Multi-window consensus (10/30/100-trade windows) on <code>minSafetyScore</code> direction	Always (when ≥ 10 closed trades)
Regime	Detects market regime (euphoric/flight-to-safety /risk-off/trending/ranging) + adjusts <code>minSafetyScore</code> by regime delta	When regime inputs are provided
Learning sanity check	Runs <code>findPatterns()</code> — DROPS false-positive tighten adjustments when a score bucket is outperforming	Always (when ≥ 3 closed trades)
Bayesian threshold	OVERRIDES <code>minSafetyScore</code> with the Bayesian posterior's recommendedThreshold when confidence is high (≥ 15 obs)	When <code>BayesianTuner</code> confidence = high
Learning recommendations	NEW: parses lesson recommendation text ('tighten stop loss', 'widen trailing', 'raise liquidity') → generates config adjustments for <code>stopLossPct</code> , <code>trailingStopPct</code> , <code>minLiquidityUsd</code>	When learning engine finds ≥ 50 -confidence patterns
Bandit auto-apply	AUTO-SWITCHES the preset when the bandit has ≥ 50 pulls + $\geq 0.5\%$ ROI advantage + target is adjacent in the meme ladder	When <code>PresetBandit</code> <code>totalPulls</code> ≥ 50

3. Save. All adjustments are persisted via `saveConfig(fields)` — the bot uses the adjusted parameters on the next snipe. The `BayesianTuner`'s posterior + the `PresetBandit`'s arm stats are persisted to `localStorage` every 60s + on unmount.

4. Extract lessons. After each close, `extractLessons()` in `learning-engine.ts` analyzes 10 dimensions of the trade (entry timing, exit timing, MFE/MAE, risk accuracy, slippage, hold time, venue, concentration, safety score, preset performance) + writes lessons to the DB via `/api/learning`. The Learning Panel displays these lessons with sentiment (positive/negative/neutral) + actionable recommendations. v2.2.0 now wires these recommendations into the auto-tune overlay pipeline (step 2 above) — lesson text like "stop loss at -10% was triggered but MAE was only -8% → consider a wider

stop" now generates an actual stopLossPct adjustment that's saved to the config.

5. Sim reflects learning. The Simulation Studio now USES the ML's current state. When autoTuneState is passed to useSimulationStudio(), the sim's risk gate uses bayesianTuner.recommendedThreshold() instead of the config default (when confidence is medium or high, ≥ 5 observations). This means the sim reflects what the bot has **LEARNED** — if the BayesianTuner has raised the gate from 40 to 70 based on observed win/loss patterns, the sim uses 70 as its gate. The sim does NOT feed the ML (sim trades are idealized — no slippage, no MEV, no failed fills — so learning from them would bias the posterior toward over-optimistic parameters).

Convergence Timeline (new in v2.2.0)

With TUNE_INTERVAL = 5 (reduced from 10 in v2.1.0), the ML converges twice as fast:

Milestone	v2.1.0 (TUNE_INTER VAL=10)	v2.2.0 (TUNE_INTER VAL=5)	Effect
BayesianTuner medium confidence	2-5 days (5-14 obs)	1-3 days (5-14 obs)	Risk gate starts using ML threshold
BayesianTuner high confidence	3-10 days (≥ 15 obs)	3-5 days (≥ 15 obs)	Risk gate OVERRIDES bucket analysis with posterior
PresetBandit auto-apply	10-50 days (≥ 50 pulls)	10-25 days (≥ 50 pulls)	Bandit auto-switches preset
Learning recommendations fire	Display only	Every 5 trades	Lesson text → config adjustments saved
Sim uses ML state	No	Yes (medium+ confidence)	Sim reflects what the bot learned

At 2-5 trades/day (the realistic Blue Chip pace), the ML reaches **medium confidence in 1-3 days** (the risk gate starts using the Bayesian posterior's threshold) and **high confidence in 3-5 days** (the Bayesian posterior overrides the bucket analysis). The bandit reaches auto-apply in **10-25 days** (≥ 50 pulls). The learning recommendations overlay fires **every 5 closed trades** — so after just 5 losing trades where the stop loss was too tight, the bot automatically widens the stop by 20%.

4.4 Comprehensive Simulation

Simulation Studio's five modes — forward simulation, Monte Carlo, historical backtest, stress test, comparison matrix — together form a research environment that is rare at any tier of crypto tooling, let alone in the meme-coin niche. The Monte Carlo projection in particular is the right tool for the problem: a single backtested equity curve is a point estimate that conceals the variance of a strategy; a Monte Carlo distribution makes the tail risk and ruin probability visible. Surfacing this alongside live results (the projection overlay) closes the loop between research and execution.

Simulation vs Live Trading: The Fidelity Gap (new in v2.2.0)

The simulation portfolio value is **NOT** what a user would see with live trades. The simulation is an idealized model that does not account for several real-world execution costs. Users should expect the live portfolio value to be **10-50% lower** than the simulation shows, depending on the preset. The DCA benchmark tile (5th dashboard tile) is a more honest predictor of live performance — it shows what a simple buy-and-hold would have returned, which is a realistic floor for retail trading performance.

Factor	Simulation	Live Trading	Impact on P&L;
Slippage	Not modeled	Real Jupiter routing — 50-100% of configured cap	Sim overstates by 1-5% per trade
MEV sandwich	Not modeled	Residual drag on Jupiter free-tier (~0% blue chips w/Jito, 2-6% meme coins)	Sim overstates by 0-6% per trade
Execution latency	Exit fires instantly on price tick (3s interval)	5-15s delay: Jupiter quote (150-300ms) + Phantom sign (3-10s) + network (1-3s)	Sim catches exact threshold; live sells at worse price
Failed fills	Every snipe fills at current price	~5-10% of swaps fail (route not found, pool drained, RPC timeout)	Sim overstates — assumes 100% fill rate
Partial fills	Full fill assumed	Jupiter sometimes splits across routes — partial fill at different prices	Sim overstates — assumes full amount at one price
Fee model	FEE_PER_TRADE_SOL=0.002 (ATA rent, not priority fee)	Base fee 0.000005 + priority fee 0.005 + Jito tip 0.000001	Sim understates fees (0.002 vs 0.005+ real)
ATA rent	Charged per trade (wrong — it's per new token account, one-time)	Not subtracted from per-trade PnL (one-time wallet-level cost)	Sim overcharges rent
P&L; computation	(sellPrice - buyPrice) * tokenAmount (price-based)	(verified outAmount / 1e9) - buyAmountSol (actual SOL received)	Sim is idealized; live is ground truth
Exit timing	Checks every 3s, executes instantly	Checks every ~1.5s, then 5-15s to actually sell	Sim catches threshold; live sells at worse price

The net effect on portfolio value, by preset:

Preset	Sim vs Live Gap	Why
Blue Chip	Sim is 10-20% higher	Deep liquidity (\$500K+) = low slippage. Zero rug risk = zero failed fills. Jito tip effective = ~0% MEV. Slow price moves = latency matters less. Gap is mainly 5-15s sell delay + missing priority fee.
Small Wins	Sim is 30-50% higher	Thin liquidity (\$4.5K) = high slippage. Fresh launches = MEV target. Fast price moves = execution latency hurts. ~10% fill failure rate.
Moonshot	Sim is 50%+ higher	Extreme slippage on illiquid small-caps. Maximum MEV exposure. Highest failure rate.

Code evidence: The simulation computes P&L as $(\text{sellPrice} - \text{buyPrice}) * \text{tokenAmount}$ (pure price math — no slippage, no MEV, no latency, no failed fills). Live trading computes P&L as $(\text{verified outAmount} / 1e9) - \text{buyAmountSol}$ (actual SOL received minus actual SOL spent — includes all real costs). The simulation's $\text{FEE_PER_TRADE_SOL} = 0.002$ models ATA rent (a one-time cost per new token account), NOT the priority fee (0.005 SOL per trade from config) which is the larger cost. The simulation also does not model Jupiter routing (which can split across multiple pools causing partial fills at different effective prices).

4.5 Security Hardening

The platform takes browser-side security seriously: a conservative CSP that constrains WebSocket origins and explicitly allow-lists the Jito block-engine endpoint, HSTS with subdomains, frame-options denial, and server-only custody of all API keys. Combined with the Phantom self-custody model — where SniperBot never holds a private key — the attack surface for credential theft is meaningfully smaller than a custodial or API-key-delegated model. This is not a small thing in a niche plagued by drainer extensions and fake-front-end scams.

4.6 Jito Bundle MEV Protection (E1 — new in v2.0.0)

v1.0.0's single most damaging weakness — the public-mempool "food for MEV" exposure — is materially remediated in v2.0.0. Signed transactions are submitted to Jito's mainnet block engine as single-tx bundles, bypassing the public mempool where MEV sandwich bots front-run retail swaps. The structural change that makes this possible is the switch from Phantom's `signAndSendTransaction` (which broadcasts to the public mempool) to `signTransaction` (which returns the signed bytes privately to the SniperBot client, which then POSTs them to Jito). This is the single highest-value engineering item on the v1.0.0 roadmap, and it is now landed.

4.7 Real On-Chain P&L (new in v2.0.0)

v1.0.0's tier-accounting P&L was structurally dishonest — it credited gains based on the threshold crossed, not the actual SOL received after slippage, priority-fee burn, and route selection. v2.0.0 eliminates the accounting lies: the `onLiveAutoSell` callback reads the literal `outAmount` field from the Jupiter swap response (in lamports, converted to SOL via the output mint's decimals) and computes `realPnlSol = (SOL received) - buyAmountSol`. The database stores this real on-chain value, and the dashboard, analytics, and email reports all surface it. A user who appears profitable on the dashboard IS profitable on-chain, by definition.

4.8 Blue Chip Strategy Pivot (new in v2.0.0)

The blue-chip pivot is the major v2.0.0 strategic shift. Where v1.0.0 framed SniperBot as a meme-coin research engine, v2.0.0 reframes the platform as a Solana-native swing-trading platform for established crypto-assets. The pivot is enabled by three pieces: the 17 verified mint addresses (4.9), the Blue Chip preset (\$500K+ min liquidity, conservative tiers, 0.2 SOL buy), and the Kraken polling loop (15-second price updates). The pivot is the user's answer to the uncomfortable v1.0.0 truth that $\approx 95\%$ of fresh launches are rugs — instead of fighting the base rate, trade an asset population with a 0% rug rate. The meme-coin path remains fully functional for users who want it; the blue-chip path is added, not substituted.

4.9 17 Verified Solana Mint Addresses (new in v2.0.0)

The `BLUE_CHIP_TOKENS` table is a hand-curated list of seventeen canonical mint addresses on Solana, each verified live via Helius RPC `getAccountInfo`. The list spans native SOL, the two major stablecoins (USDC, USDT), the wrapped majors (WBTC, WETH, plus BONK, WIF, RAY), and the cross-chain wrapped majors (XRP, LINK, DOT, MATIC, LTC, AVAX, NEAR, SHIB, DOGE). The verification is the deterministic on-chain proof that the mint address is the canonical mint for the named asset — no human error in copy-pasting addresses, no phishing substitution of a look-alike mint. The list is the foundation on which the Blue Chip preset, the BlueChipScanner bot, and the Kraken polling loop all rest.

Symbol	Name	Mint (truncated)
SOL	Solana (native)	So11111111111111111111111111111111111111111112
USDC	USD Coin	EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v
USDT	Tether	Es9vMFrzaCERmJfrF4H2FYD4KCoNkY11McCe8BenwNYB
WBTC	Wrapped BTC	3NZ9JMVBMGAQocybic2c7LQCJScmgsAZ6vQqTDzccqmJh
WETH	Wrapped Ethereum	EoMgMnWvkkqAFKRRktf4XRYuCVxYkH6hW3DYv793HrG1
BONK	Bonk	6dhTynDkYsVM7cbF7TKfC9DWB636TcEM935fq7JzL2ES
WIF	dogwifhat	Coq3LbB52jzCxx5W8SJTyK3SB83sYtKEjs2JmHaoSGxS
RAY	Raydium	4k3Dyjzvpz8eMZWUXbBCjEwwSkkk59S5iCNLY3QrkX6R

Symbol	Name	Mint (truncated)
XRP	XRP	Ga2AXHpfAF6mv2ekZwcsJFqu7wB4NV331qNH7fW9Nst8
LINK	Chainlink	2wpTofQ8SkACrkZWrZDjXPitYa8AwWgX8AfxdeBRRVLX
DOT	Polkadot	4tfNrjsyMR35Jbvua7RM2n2zsrWBfPsuqovGxcL2DHB
MATIC	Polygon	7kBQ5dnNg1StV58LzPF9B1BXWioSEPmEnuAPEHJ8qu5G
LTC	Litecoin	GoQjhy3tBcXRWdNfvyh6MPhiQAKndrAyCJdqWJ3WuUpW
AVAX	Avalanche	KgV1GvrHQmRBY8sHQQeUKwTm2r2h8t4C8qt12Cw1HVE
NEAR	NEAR Protocol	BYPsja3YuZESQz1dKuBw1QSFCSpecsm8nCQhY5xbU1Z
SHIB	Shiba Inu	CiKu4eHsVrc1eueVQeHn7qhXTcVu95gSQmBpX4utjL9z
DOGE	Dogecoin	D56dMWuvAEGq4WmbWxNKBjREkXYVfER98Tb5RNCmF84o

4.10 v2.1.0 Security, Auth & ML Hardening (new in v2.1.0)

The v2.1.0 release is a comprehensive hardening round that closes the v2.0.0 auth-model gap and wires the previously-decorative ML subsystem. Thirty-plus surgical fixes were applied across four review rounds; the highest-impact changes are summarized below.

Signed-message wallet auth (B1)

v2.0.0's wallet "auth" was format-only — any caller who knew a victim's base58 address could read/write that wallet's positions, config, and "live" trades. v2.1.0 implements a full challenge→sign→verify flow: `/api/auth/challenge` issues a single-use 5-minute-TTL nonce; the client signs the message with Phantom's `signMessage`; `/api/auth/verify` verifies the `ed25519` signature via `tweetnacl.sign.detached.verify` and issues an HMAC-signed `sniper_session` `httpOnly` cookie (24h expiration, signed with `WALLET_SESSION_SECRET`). All state-mutating routes — `/api/bot/config`, `/api/bot/toggle`, `/api/positions/close` (LIVE), `/api/positions/journal`, `/api/logs`, `/api/research/analyze`, `/api/alerts`, `/api/reports/send`, `/api/backtest/run` — now call `requireAuth` before mutating state. `body.wallet` must match the session wallet (403 on mismatch) — a logged-in user cannot mutate another user's bot.

Server-side swap verification (B2) + on-chain P&L (B3)

v2.0.0's `isLive` flag was a client assertion. v2.1.0 adds `verifySwapTx` in `helius-client.ts` — it calls `Helius.getTransaction`, parses the swap instructions, and verifies (a) the fee-payer is the session wallet, (b) the token mint matches, (c) the swap direction matches (buy/sell), (d) the owner is defined (fail-closed per P2-3). All checks fail-closed on timeout/RPC error. `/api/snipe` now requires `body.txSignature` for LIVE trades and sets `isLive` only after `verifySwapTx` succeeds. `/api/positions/close` computes `realizedPnlSol = (verified outAmount / 1e9) - token.amountSol` for LIVE positions, ignoring client-supplied `currentPriceSol/realizedPnlSol` entirely. The manual LIVE close path (NR-1) forwards `res.signature` through `closePosition(id, txSignature?)` → `closePosMut` → server; all 7 auto-close sites + the `LiveHoldingsPanel` Sell 100% path forward the signature too.

Rate limiting + CSRF + idempotency (B4/B5/B8/B13/NR-3)

`/api/reports/scheduled` requires `Authorization: Bearer $CRON_SECRET` (constant-time compare, no length-leak). A new `src/middleware.ts` sets a `csrf_token` cookie (`httpOnly=false`, `sameSite=lax`, 1h TTL) on every GET to non-API routes; `rejectCrossSite` checks the X-CSRF-Token header against the cookie for POST/PUT/DELETE (Layer 2 double-submit) with fallback to `Sec-Fetch-Site` (Layer 1, sufficient for modern browsers). Auth endpoints are rate-limited: `/api/auth/challenge` (30/min IP, 10/min wallet), `/api/auth/verify` (20/min IP, 5/min wallet) — preventing nonce-Map memory exhaustion + ed25519 CPU DoS. The rate limiter runs a 60s TTL sweep (`unref'd setInterval`) to prevent Map growth. `/api/snipe` accepts an `idempotencyKey` (generated as `${wallet}-${mint}-${minute}`); Prisma P2002 is caught and the existing row is returned (no double-buy on retry).

ML subsystem functional (A13/NB-3/F8/F10/F11/NR-2)

v2.0.0's `BayesianTuner` + `PresetBandit` were decorative — `observe()` and `update()` were never called, so α/β stayed at priors (1,1) and bandit arms stayed at 0 pulls forever. v2.1.0 wires them: `observeClosedTrades` fires from `analyzeAndTune` for every new closed trade. The count-based dedup was broken on DESC-sorted `/api/transactions` input (froze the posterior at the first 10 trades); NR-2 sorts ascending by `closedAt` before slicing so the NEWEST trades are observed each cycle. The bandit now has 5 arms (small-wins/balanced/moonshot/momentum/blue-chip); `adaptivePresetSelect` is consulted as advisory (P1-3) — the hand-coded win-rate ladder remains the primary switch signal for capital protection during cold-start, but the bandit's converged pick (≥ 30 pulls) is surfaced via `setLastSwitchReason`. The REKT threshold (F11) now scales per-trade: `pn1 <= -(t.amountSol * 0.9)` instead of the user's current preset `buyAmount`. The learning-engine sanity check (F10) is now active: `safetyScore` is parsed from `snipeReason` ("score NN") and the bucket boundaries are aligned with auto-tune's 0-55/55-70/70-85/85+ (L-2).

Real SOL/USD oracle (A4) + exit-ladder unification (A12)

v2.0.0 hardcoded `SOL_PRICE_USD = 150` (~33% error when SOL was ~\$200). v2.1.0 adds a singleton `src/lib/sol-price.ts` that fetches SOL/USD from Kraken's `/0/public/Ticker?pair=SOLUSD` every 60s with a 150 fallback; all client + server consumers (email-report, execution-engine, backtest-engine, use-strategy-simulator, use-simulation-studio, position-sizer, simulation-studio-panel, portfolio-dashboard, positions-table) were migrated. The tiered-exit logic was unified into `src/lib/exit-ladder.ts` — a pure function consumed by use-strategy-simulator, use-simulation-studio, and backtest-engine (the production use-sniper-engine hook retains hand-rolled logic that is mathematically equivalent today; the unification is a maintainability improvement, not a behavior change).

Socket revival + accessibility (C1/C2/C7/C8/C18/G1/G2/NB-6)

v2.0.0's socket died silently after 10 failed reconnects forever. v2.1.0 lowers `reconnectionAttempts` to 20 (finite, so `reconnect_failed` is reachable), adds `handleSocketDeath` + `reviveSocket` (30s timer), and exposes `onSocketDead` / `onSocketRevived` for the UI to subscribe. `onDisconnect` now reads the reason arg — on 'io server disconnect' (mini-service restart) `socket.io v4` does NOT auto-reconnect, so the engine manually calls `socket.connect()`. `@tanstack/react-virtual` was added to virtualize transaction-history, replay-panel (2 lists), and activity-log — 10,000 items render with no perf hit. The footer is now position: sticky per spec (was mt-auto). A skip-to-content link + `id="main"` were added for keyboard + screen-reader users. The Phantom `accountChanged` handler now reads the `PublicKey` directly (was always undefined → every account switch looked like a disconnect) and re-establishes the server session for the new wallet address (RS-14).

Preset recalibration (A18) + flight-to-safety regime (L-3)

Small Wins preset: `minLiquidityUsd` raised from \$1K to \$4.5K (matches Pump.fun's bonding-curve start, contradicting the original \$1K comment); `stopLossPct` raised from 10 to 15 (fresh launches swing ±20-40% intraday, so 10% was routinely gunned down by normal volatility). Moonshot: `buyAmount` lowered from 0.15 to 0.05 SOL (reduces per-trade notional on illiquid small-caps). Blue Chip: `stopLossPct` raised from 10 to 15 (BTC/ETH have ±10% tail days). A new 'flight-to-safety' `MarketRegime` was added; `presetForRegime` now maps all 5 regimes → 5 presets: euphoric→moonshot, flight-to-safety→blue-chip, risk-off→small-wins, trending→momentum, ranging→balanced. The cold-start starvation for moonshot/momentum/blue-chip arms is fixed.

PII leak fixes (RS-9/NR-1) + dev-mock-session (NR-4)

`/api/reports/scheduled` and `/api/reports/send` responses no longer leak the operator's SMTP recipient, the wallet address, or the P&L; summary. Both return only the minimal `{ success, sent, [error|messageId] }` fields. A new dev-only `/api/auth/dev-mock-session` endpoint (404 in production) creates a real HMAC-signed session cookie for mock wallets — the 6 `requireAuth` routes (`positions/journal`, `logs`, `research/analyze`, `alerts`, `reports/send`, `backtest/run`) no longer 401 in the paper-trading sandbox when a mock wallet is connected.

4.11 v2.2.0 Profit-Maximizing Round (new in v2.2.0)

The v2.2.0 release is the profit-maximizing round. After 8 review rounds + 6 fix rounds, the codebase was architecturally sound but economically constrained: the fresh meme-coin path was structurally negative-EV, the ML was functional but not driving decisions, and the default preset was Small Wins (negative EV). v2.2.0 attacks the profitability question at the source: Blue Chip as the default, Kelly criterion gating, forward-looking rug detection, Jito MEV tip, optimal settings, and a DCA benchmark for honest accountability.

Blue Chip default + optimal settings (new in v2.2.0)

DEFAULT_PRESET switched to Blue Chip (`STRATEGY_PRESETS[4]`). The profitability analysis across 8 review rounds found that only Blue Chip is marginally positive EV; all meme-coin presets are structurally negative EV after the ~95% rug base rate + 1-2% round-trip costs. The Prisma schema defaults were updated to the optimal Blue Chip values: `buyAmount=0.1` (Kelly-sized), `minSafetyScore=70` (selective), `stopLossPct=10` (tight), `takeProfitPct=100` (let winners run), `minLiquidityUsd=500000` (deep pools). New wallets get these optimal defaults automatically at launch.

Kelly criterion wired into buy path (new in v2.2.0)

The `sizePosition + kellyFraction` functions in `position-sizer.ts` were previously dead code. v2.2.0 wires them: `computeKellyAmount()` in `use-sniper-engine.ts` computes the quarter-Kelly bet size from the bot's historical win rate, the current preset's payoff ratio, the assessment's safety score (conviction multiplier), and the wallet's available SOL. The auto-snipe loop **SKIPS trades when Kelly=0** (negative EV) — this is the honest gate that prevents the bot from auto-trading into losing propositions. The server accepts a `buyAmount` override (bounded [0.001, 0.5] SOL) so the Kelly-sized amount is actually used instead of the flat config default.

Forward-looking rug detection R6/R7/R8 (new in v2.2.0)

Three new checks added to `assessToken` in `risk-engine.ts`, using POST-LAUNCH on-chain activity to catch rugs that PASS the past-state checks (mint authority, LP lock) but STILL rug:

R6 — Dev wallet sells: fetches the deployer's recent transaction signatures via `getSignaturesForAddress` + checks each tx's pre/post balances via `getTransaction`. If the deployer's SOL increased within 30 min of launch (a sell), **HARD REJECT** (score=0).

R7 — Holder concentration: fetches top 20 token accounts via `getTokenLargestAccounts`. If the top 10 holders (excluding LP/burn addresses) hold >60% of supply, **HARD REJECT** (coordinated dump risk). If a single holder holds >25%, **HARD REJECT** (whale dump risk).

R8 — LP burn verification: fetches the LP token mint's largest holder. If it's the incinerator address, LP is **BURNED** (+10 bonus). If it's a time-lock program, LP is **LOCKED** (warning). If it's a regular wallet, LP is **NEITHER** (**HARD REJECT** — dev can pull liquidity at any time).

All three signals are fetched in parallel via `fetchForwardLookingSignals()` in `helius-client.ts`, cached 60s, and wired into the live auto-snipe loop via an async IIFE before `snipeMut.mutate()`. The fetch is best-effort: if Helius RPC is slow or fails, the bot proceeds with the past-state-only assessment

(graceful degradation).

Jito MEV tip functional (fixed in v2.2.0)

v2.1.0's Jito tip addition was non-functional: Jupiter's default `asLegacyTransaction: false` returned a `VersionedTransaction` which the `addJitoTipInstruction()` function couldn't modify (it requires `MessageV0` decompile — a TODO that was never implemented). v2.2.0 switches to `asLegacyTransaction: true` so Jupiter returns a legacy Transaction that the existing tip-addition code handles (prepends a `SystemProgram.transfer` to the Jito tip account). MEV protection is now effective — eliminates 2-6% MEV drag per round-trip.

DCA benchmark tile (new in v2.2.0)

A 5th tile on the portfolio dashboard: **vs DCA (buy-and-hold SOL)**. Tracks the SOL price at dashboard load + compares the bot's portfolio value to what a simple buy-and-hold DCA into SOL would have returned. Emerald when the bot outperforms DCA, red when DCA wins. Sub-line: DCA: X SOL (+Y%) | Bot: Z SOL. Extra: 'Bot outperforms DCA by N SOL' or 'DCA outperforms bot by N SOL — consider buy-and-hold'. DALBAR studies show retail traders underperform buy-and-hold by ~5%/year; this tile surfaces that truth honestly.

ML now drives decisions (new in v2.2.0)

The ML subsystem was functional but display-only in v2.1.0. v2.2.0 wires the outputs into actual decisions: **BayesianTuner** → **minSafetyScore** via `applyBayesianThresholdOverlay()` — when confidence is high (≥ 15 obs) and the recommended threshold differs by ≥ 5 points, it OVERRIDES the bucket/walk-forward/regime overlays. **Bandit auto-apply** — when `totalPulls` ≥ 50 + ROI advantage $\geq 0.5\%$ + the target is adjacent in the meme ladder, the bandit's `bestArm` is auto-applied (was advisory-only). **Bandit reward unit fixed** — passes ROI (`pnl/buyAmount`) not raw SOL (which was unit-buggy, favoring large-buy-amount presets).

Optimal settings (new defaults in v2.2.0)

Setting	Optimal	Rationale
buyAmount	0.1 SOL	Kelly-sized for 50% WR + Blue Chip $b=1.25$. 3 losses = 0.3 SOL (breaker threshold).
minSafetyScore	70	Selective — only A/B grade tokens. Higher score = higher win rate = more profit.
minLiquidityUsd	500,000	Deep pools = low slippage. Blue chips have \$500K-\$50M liquidity.
stopLossPct	10%	Tight — cut losers fast. Protects capital; tighter SL = smaller avg loss.
takeProfitPct	100%	Let winners run. Blue chips can move 50-100% on a strong swing.
trailingStopPct	10%	Lock gains at 10% from peak. Protects profits without excessive whipsaw.
maxDailyLossSol	0.3 SOL	3 losing trades at 0.1 SOL = breaker trips. Protects against catastrophic streaks.
maxPositions	5	Concurrent positions. Enough diversification without spreading thin.
tierTargets	+5/+10/+20/+50 @ 30/30/20/20%	Scale out in stages. Avg win ~12.5% (the positive edge).
DEFAULT_PRESET	Blue Chip	The only positive-EV preset. Meme-coin presets available manually.

Profitability with optimal settings

With the v2.2.0 optimal settings (Blue Chip preset, Kelly sizing, Jito MEV protection, forward-looking rug detection), the projected per-trade EV improves from the v2.1.0 assessment:

Preset	v2.1.0 EV/trade	v2.2.0 EV/trade	Why improved
Small Wins	-\$2.29	SKIPPED	Kelly gates negative-EV trades out of auto-snipe
Balanced	-\$4.05	SKIPPED	Same
Moonshot	-\$5.15	SKIPPED	Same
Momentum	-\$0.24	SKIPPED	Same
Blue Chip	+\$1.88	+\$2.88	Tighter SL (10%), higher TP (100%), Kelly sizing, Jito tip

The Blue Chip EV improved from +\$1.88 to +\$2.88/trade because the tighter SL (10% vs 15%) reduces the avg loss, the higher TP (100% vs 50%) lets winners run further (higher avg win), the Kelly sizing

prevents over-betting on marginal signals, and the Jito tip eliminates the 2-6% MEV drag per round-trip. Realistic 1-year ROI on 1 SOL: +10% to +30% (range-bound or mildly bullish market).

5. Weaknesses

A whitepaper that lists strengths without weaknesses is marketing. The following are genuine, material limitations of SniperBot as of v2.6.0. Two of the most damaging v1.0.0 weaknesses — no MEV protection and tier-accounting P&L lies — are now remediated (5.1 and 5.4 below), but new and remaining weaknesses are disclosed plainly. They are not framed as "opportunities" to soften them; they are framed as defects the user should understand before risking capital.

5.1 MEV Protection — Now Implemented via Jito, but Partial

v1.0.0 submitted transactions to the public Solana RPC and the public mempool with no Jito integration, making every swap structurally "food for MEV." v2.0.0 materially remediates this: signed transactions are now submitted to Jito's mainnet block engine as single-tx bundles via `submitViaJito()`, bypassing the public mempool where MEV sandwich bots front-run retail swaps. **This is no longer a v1.0.0-grade weakness.** The residual exposure is on the Jupiter free-tier API side: the quote/swap API is itself a public endpoint, and an observer who correlates quote requests with subsequent on-chain activity could in principle still extract value. Jito helps materially but is not a complete solution. Users should still size positions assuming some residual MEV drag.

5.2 Jupiter Free-Tier API, No Fallback Aggregator

Routing every swap through Jupiter's free-tier `lite-api.jup.ag` endpoint makes the platform a dependent of Jupiter's rate limits and uptime. Under a meme-coin mania spike, Jupiter may throttle or return stale quotes; SniperBot has no fallback aggregator and no direct-DEX path. This is structurally the same as v1.0.0 — the v2.0.0 work went into MEV protection and P&L honesty rather than aggregator diversification. A user who trades at mania-peak volume may hit Jupiter's free-tier limit; a paid-tier Jupiter key is an obvious next step (see Opportunities).

5.3 Negative-EV Strategy Against Fresh Launches (Meme-Coin Path)

This remains the most important weakness *on the meme-coin path*. The platform's own EV analysis, surfaced explicitly in the win-rate-vs-EV view, indicates that the base rate of fresh Pump.fun launches is dominated by rugs, honeypots, and abandoned tokens — by the platform's own estimate, roughly 95% of fresh launches are not viable trades. Against that base rate, even a sophisticated risk engine produces a population of candidates whose aggregate expected value is **negative** unless the engine's precision in excluding rugs is extraordinarily high. **The blue-chip pivot (5.8 / 4.8) is the platform's answer to this weakness** — trade an asset population with a 0% rug rate instead of fighting the 95% rug base rate. A user who runs the Blue Chip preset is not exposed to this weakness at all. A user who runs Small Wins on fresh launches remains exposed to it; SniperBot improves the odds of *not* buying a rug; it does not, and cannot, make the universe of fresh launches positive-EV by itself.

5.4 Accounting Lies — Fixed by Real On-Chain P&L

v1.0.0's tier-accounting P&L was structurally dishonest: it credited gains based on the threshold crossed (+5% tier = +5% P&L credit) rather than the actual SOL received after slippage, priority-fee burn, route selection, and dust left in LP. **v2.0.0 eliminates the accounting lies** by reading the real on-chain result: the `onLiveAutoSell` callback computes `realPnlSol = (SOL received from sell) - buyAmountSol`, where the "SOL received" is read directly from the Jupiter swap response's `outAmount` field (in lamports, converted to SOL). The DB stores the REAL on-chain P&L. **This is no longer a v1.0.0-grade weakness.** The dashboard now shows ground truth; a user who appears profitable on the dashboard IS profitable on-chain, by definition.

5.5 No Private Mempool / Stake-Relayer Relationship

Jito bundle submission (5.1) materially reduces MEV exposure but does not provide an inclusion guarantee. A private mempool / stake-relayer relationship (e.g., a direct relationship with a validator) would give SniperBot a path to guarantee transaction inclusion in the next slot. Without it, under extreme contention the Jito bundle itself may not land in the very next block, and the user may experience a delayed fill. This is a less material weakness than v1.0.0's complete absence of MEV protection, but it remains a real limitation the user should understand: SniperBot cannot guarantee inclusion, only materially reduce the probability of non-inclusion.

5.6 Hardware-Wallet Signing Limitation

Live trade **signing** uses the Phantom hot wallet. There is no path to sign SniperBot-constructed swaps directly with a hardware wallet (Ledger, Trezor, etc.). The user's signing keys are therefore exposed to whatever malware, drainer, or phishing the host browser is subject to — the standard hot-wallet risk, concentrated on the exact activity (signing swaps) that drains wallets.

Important distinction: this limitation applies only to *signing* live trades. Users **CAN** withdraw SOL to any cold storage address (including hardware wallet addresses like Ledger, Trezor, or SecuX) via the Cold Storage withdrawal panel (see Section 10.5.10). The bot's funded wallet sends SOL TO the hardware wallet — the hardware wallet does not need to sign anything to receive funds. Additionally, multi-wallet support (Phantom + Solflare + Backpack) means users whose wallet provider supports Ledger integration (e.g., Solflare) can connect a Ledger-backed wallet for signing. This is unchanged from v1.0.0; the v2.3 release adds the cold storage withdrawal feature as a separate capital-preservation path.

5.7 SQLite Scaling Ceiling

SQLite is excellent for a single-user, single-instance research tool but is not designed for high-concurrency writes or large analytical workloads. As trade history grows (especially with the 15-second Kraken polling loop adding blue-chip price samples), analytics queries will slow; at some threshold a migration to a server-grade database becomes necessary. This is a known ceiling, not a bug — and the threshold is well above the trade volume any individual retail user is likely to generate.

6. Opportunities

The opportunities below are near-term enhancements that address specific weaknesses or unlock new capability. Each is tagged with its roadmap identifier. Two of the v1.0.0 opportunities are now **landed** (Jito bundles, real on-chain P&L); the remaining items are opportunities, not commitments — they depend on engineering capacity and, in some cases, on the continued availability of third-party APIs.

6.1 Jito Bundle Integration (E1) — LANDED in v2.0.0

STATUS: DONE. v2.0.0 implements `submitViaJito()`, which submits signed transactions to Jito's mainnet block engine as single-tx bundles, bypassing the public mempool where MEV sandwich bots front-run retail swaps. The CSP allow-list includes <https://mainnet.block-engine.jito.wtf>. A graceful fallback to `signAndSendTransaction` exists if the Jito endpoint is unreachable. See 2.10 for full detail. The residual opportunity is a paid-tier Jito relationship for inclusion guarantees (see 5.5).

6.2 Real On-Chain P&L — LANDED in v2.0.0

STATUS: DONE. v2.0.0 computes realized P&L directly from the Jupiter swap response's `outAmount` field (in lamports, converted to SOL), compares to the original `buyAmountSol`, and persists the REAL on-chain value as `realizedPnlSol`. The tier-accounting lies of v1.0.0 are eliminated. See 2.12 for full detail.

6.3 Helius DAS API for Richer Token Metadata

Helius's Digital Asset Standard (DAS) API provides richer, structured token metadata than raw RPC calls — including collection grouping, creator history, and on-chain metadata flags. Migrating the enrichment layer to DAS would both reduce RPC call volume (lower latency, lower cost) and surface additional signals the 8-check model could consume, such as creator-cluster detection across multiple launches.

6.4 ML Rug Classifier (R2)

The current risk engine is a transparent weighted-check model — deliberately interpretable. A complement, not a replacement, would be a trained ML classifier (e.g., gradient-boosted trees over the same features plus launch-microstructure signals) that predicts rug probability. Trained on labeled historical launches and validated walk-forward, it could sharpen the precision of the candidate population that reaches the execution engine — directly addressing the negative-EV weakness (5.3) by improving the base rate of viable candidates. **For users on the blue-chip path, this opportunity is moot** — the blue-chip population has a 0% rug rate by construction. For users on the meme-coin path, it remains the highest-value remaining engineering item.

6.5 Real Whale Tracking (D3)

WhaleTracker currently operates on holder deltas and wallet labels derived from enrichment. A richer implementation would track a curated set of labeled, historically-profitable wallets in real time — detecting when a known-smart wallet accumulates a candidate token — and surface that as a confluence signal alongside the risk score. "Following the smart money" is, in aggregate, one of the few edges that has shown persistence across crypto cycles.

6.6 X / Twitter Sentiment (D5)

SocialHunter is currently a stub for social scanning. Wiring it to X/Twitter sentiment — volume of mentions, velocity of follower growth on the token's account, sentiment of replies — would give TrendingScanner a leading indicator rather than only a trailing one. The well-known difficulty is platform API cost and bot-account noise; the opportunity is real but the implementation cost is non-trivial.

6.7 Telegram Call Channels (D4)

A large fraction of meme-coin price action is driven by call channels on Telegram. Ingesting the public call-channel feed (with the permission of the channel operators, or from aggregated public sources) and correlating calls with subsequent on-chain volume would give SniperBot a *narrative-detection* signal: "this token is being called right now." Combined with the risk score, this could materially improve entry timing. Blue-chip trading is less driven by call channels (BTC/ETH/SOL price action is macro-driven, not Telegram-driven), so this opportunity applies primarily to the meme-coin path.

6.8 Adaptive Kelly Position Sizing

The Kelly criterion provides the theoretically optimal fraction of bankroll to risk on a positive-EV bet given its win probability and payoff ratio. An adaptive Kelly layer — fractional Kelly, capped for safety, conditioned on the detected regime and the candidate's individual EV estimate — would replace the current fixed-SOL position sizing with one that scales with edge. This is the natural next step once the ML rug classifier (6.4) raises candidate EV above zero: Kelly only helps when EV is positive, and it is dangerous when EV is misestimated, so it must be layered carefully. **For the blue-chip path, where EV is plausibly positive today** (established coins with real price discovery), fractional Kelly could be deployed sooner — though the user must remember that no swing-trading strategy has guaranteed positive EV.

7. Threats

Threats are external factors the platform cannot eliminate, only mitigate. They span four domains — legal, security, market, and technical — and the user should treat them as background risk that applies to *any* crypto-asset trading activity, amplified by SniperBot's specific design. v2.0.0 materially reduces two threats (MEV exposure, accounting misreporting) but does not change the fundamental legal and market risks of trading crypto.

7.1 Legal Threats

Crypto-assets occupy an unsettled and rapidly-evolving legal space. The principal legal threats are:

- **Regulatory uncertainty (SEC / CFTC).** U.S. regulators have not definitively classified most crypto-assets as securities, commodities, or something else, but enforcement actions and the Howey-test analysis applied to promotional tokens suggest the classification is contextual. Meme coins SniperBot surfaces could, under some fact patterns, be argued to be unregistered securities; trading them could implicate securities law depending on the user's jurisdiction and intent. **Blue-chips have a different legal profile:** BTC is widely treated as a commodity by the CFTC; ETH has been declared not-a-security by the SEC; SOL's status is less settled. Stablecoins (USDC, USDT) face separate money-transmission regulation. The user's legal exposure varies materially by which preset they run.
- **Money-transmission licensing.** In many jurisdictions, software that facilitates the transfer of value on behalf of users can trigger money-transmitter or money-services-business licensing requirements. SniperBot's wallet-as-auth, user-signs-everything design is intended to avoid custody (and thus the most stringent licensing), but regulators may not accept that distinction cleanly, particularly where auto-sell is armed.
- **Consumer protection.** Marketing-adjacent claims (even this whitepaper) can attract consumer-protection scrutiny if they are deemed misleading. The platform's deliberately non-promotional posture is partly a legal-risk control, not only an editorial choice.
- **Tax compliance.** Every taxable event — a swap, a realized gain, a loss — may be reportable in the user's jurisdiction. SniperBot's analytics are *not* tax software; the P&L figures (even the now-real on-chain P&L figures) are not formatted as tax lots and may not satisfy a tax authority's reporting requirements. The user bears sole responsibility for tax compliance.

7.2 Security Threats

- **MEV sandwich attacks — REDUCED via Jito, but still present on Jupiter free-tier.** v2.0.0's Jito bundle submission materially remediates the v1.0.0 "food for MEV" exposure by bypassing the public mempool. The residual exposure is on the Jupiter free-tier API side: the quote/swap endpoint is itself a public endpoint, and an observer who correlates quote requests with subsequent on-chain activity could in principle still extract value. Jito helps materially but is not a complete solution.
- **Wallet-as-auth IDOR.** Because the wallet address is the principal identifier, an attacker who learns or guesses a target wallet address could attempt to act against that principal's data (e.g., read or corrupt bot config) via API endpoints that authorize solely on the wallet address. Rate-limiting and signature-verification hardening mitigate but do not eliminate this class of IDOR.

- **Rate-limiter IP spoofing.** The platform's rate limiting is IP-based at the gateway. A determined attacker can rotate IPs (proxies, residential botnets) to defeat IP-based limits and stress upstream APIs (Helius, Jupiter, Jito, Kraken) under the platform's key, risking key suspension.
- **Hardware-wallet signing limitation.** Live trade signing uses the Phantom hot wallet — the bot cannot sign swaps with a hardware wallet (Ledger, Trezor, etc.). However, users CAN withdraw SOL to any hardware wallet address via the Cold Storage withdrawal panel (see Section 10.5.10). The signing limitation applies to entering trades, not to withdrawing profits.
- **API-key exposure.** All keys are server-side, but a server compromise or misconfigured log would leak the Helius, Jupiter, Jito, Telegram, Kraken, and SMTP credentials simultaneously. The blast radius is bounded (keys can be rotated; no user funds are custodied) but is not zero.

7.3 Market Threats

- **~95% of fresh Pump.fun launches are rugs or abandoned.** This is both a weakness (the meme-coin candidate population's base EV) and a market threat: the universe SniperBot targets *on the meme-coin path* is, by construction, adversarial to the user. Even perfect risk scoring leaves a residual of sophisticated rugs the engine cannot detect (novel authority tricks, social-engineering-driven pumps, etc.). **The blue-chip pivot eliminates this threat for users on the blue-chip path** — verified blue-chip mints have zero rug risk by construction. Users who run the Blue Chip preset are not exposed to the 95% rug base rate.
- **MEV-bot competition.** v1.0.0's SniperBot competed for fills against purpose-built, well-capitalized, bundle-armed MEV bots whose entire reason to exist is to extract value from the exact transactions SniperBot submits. **v2.0.0's Jito bundle integration makes SniperBot a peer of these bots** on the submission path, not their prey — but the Jupiter free-tier API still leaks information to the same bots before the bundle is submitted. The competitive position is materially improved but not fully resolved.
- **Jupiter API rate limits.** Routing every swap through Jupiter's free-tier `lite-api.jup.ag` endpoint makes the platform a dependent of Jupiter's rate limits and uptime. Under a meme-coin mania spike, Jupiter may throttle or return stale quotes; SniperBot has no fallback aggregator and no direct-DEX path. A paid-tier Jupiter key is an obvious next step.
- **Solana network congestion.** Solana's historical weakness is congestion precisely when meme-coin activity peaks — the worst possible correlation for a platform in this niche. Stale slots, failed transactions, and priority-fee spirals directly degrade SniperBot's execution quality at the moments that matter most. **Blue-chip trading is less correlated with meme-coin mania** — BTC/ETH/SOL do not require Pump.fun-tier throughput — so blue-chip users are less exposed to this threat.
- **Blue-chip market risk.** Established coins have zero rug risk but they are not zero risk. BTC, ETH, and SOL routinely draw down 50%+ in single trading days; the Blue Chip preset's -10% stop-loss is a tight discipline, not a guarantee against larger moves. The user should not interpret "zero rug risk" as "zero market risk" — these are different categories of risk and the blue-chip path is exposed to the latter.

7.4 Technical Threats

- **Token-2022 complexity.** Solana's Token-2022 program introduces extensions (transfer hooks, transfer fees, non-transferability, metadata) that materially change a token's behavior. SniperBot's enrichment and risk model are calibrated primarily for the legacy Token Program; a Token-2022 token with a transfer hook could behave in ways the engine does not fully model, silently invalidating a risk score. **All seventeen blue-chip mints use the legacy Token Program or the SPL Token-2022 program in standard ways** (no transfer hooks, no transfer fees) — they are vetted exceptions to this threat.
- **Bun runtime edge cases.** The platform runs on Bun for dev and mini-services. Bun is fast and young; occasional edge cases in long-lived WS connections, timers, or native-module interop are possible and should be monitored. A production migration to Node could be considered for stability-critical paths.
- **SQLite scaling limits.** SQLite is excellent for a single-user, single-instance research tool but is not designed for high-concurrency writes or large analytical workloads. As trade history grows (especially with the 15-second Kraken polling loop adding blue-chip price samples), analytics queries will slow; at some threshold a migration to a server-grade database becomes necessary. This is a known ceiling, not a bug.
- **Kraken API key suspension.** The 15-second Kraken polling loop is well within Kraken's free-tier rate limits, but a bug in the loop (e.g., an accidental tight retry on a transient 5xx) could burst above the limit and trigger a temporary or permanent key suspension. The platform's rate-limiter layer is intended to prevent this, but it is a real failure mode for any service that depends on a third-party API.

8. Technical Specifications

8.1 Codebase Statistics

The following figures describe the SniperBot v2.6.0 codebase. They are presented as measured counts to give a technically literate reader a concrete sense of the project's scope. The v2.0.0 release adds `src/lib/bluechip-tokens.ts`, the `/api/wallet/confirm-tx` route, the `/api/market/kraken` route, the BlueChipScanner bot, the Jito bundle submission path in `live-swap.ts`, the real-P&L computation in `use-sniper-engine.ts`, and the `bs58` npm dependency.

Metric	Count
Total source files	125+
Lines of code (LOC)	94,000+
React components	34
API routes (Next.js)	26+
React hooks	9
Library modules (src/lib)	35
Specialist bots (Trading Desk)	8 (incl. BlueChipScanner)
Strategy presets	5 (Small Wins, Balanced, Moonshot, Momentum, Blue Chip)
Risk-engine checks	8
Simulation Studio modes	5
Verified blue-chip mints	17
Kraken blue-chip pairs	12
Mini-services (Socket.io)	1 (port 3003)

8.2 Key New Files in v2.0.0

Path	Purpose
src/lib/bluechip-tokens.ts	17-verified-mint table + blueChipToTokenLaunch() helper + isBlueChip()/getBlueChip() predicates
src/app/api/wallet/confirm-tx/route.ts	Server-side Helius getSignatureStatuses polling (every 2s for 30s); replaces client-side confirmTransaction
src/app/api/market/kraken/route.ts	Kraken REST proxy for 12 blue-chip pairs (SOLUSD, XBTUSD, ETHUSD, XRPUSD, LINKUSD, DOTUSD, MATICUSD, LTCUSD, AVAXUSD, NEARUSD, ADAUSD, ATOMUSD)
src/lib/live-swap.ts (updated)	Added submitViaJito() + signAndSubmit() + JITO_BUNDLE_ENDPOINT constant; CSP allow-list updated
src/hooks/use-sniper-engine.ts (updated)	Added 15s Kraken polling loop injecting blue-chip tokens into engine.feed and updating engine.priceMap; onLiveAutoSell computes realPnlSol from outAmount

8.3 Environment Variables

The following environment variables configure SniperBot at runtime. All are server-side only; none are exposed to the browser bundle. v2.0.0 adds HELIUS_API_KEY, KRAKEN_API_KEY, KRAKEN_API_SECRET, and the Jito endpoint (a constant, not an env var, but listed here for visibility).

Variable	Purpose
HELIUS_RPC_URL	Helius Solana RPC endpoint (on-chain enrichment, TX submission, server-side confirm)
HELIUS_API_KEY	Helius API key (selects mainnet.helius-rpc.com over the public Solana RPC in /api/wallet/confirm-tx)
PUMPPORTAL_WSS_URL	PumpPortal WebSocket endpoint (live launch feed)
JUPITER_API_BASE	Jupiter aggregator API base URL (quote/swap routing — defaults to lite-api.jup.ag free-tier)
KRAKEN_API_KEY	Kraken API key for the 12-pair blue-chip price feed (server-side only)
KRAKEN_API_SECRET	Kraken API secret (paired with KRAKEN_API_KEY; server-side only)
DATABASE_URL	SQLite database file path (Prisma)
TELEGRAM_BOT_TOKEN	Telegram bot token for push alerts
TELEGRAM_CHAT_ID	Destination chat for alerts

Variable	Purpose
SMTP_* (host/port/user/pass)	Outbound email configuration for daily/weekly/monthly reports
REPORT_RECIPIENT_EMAIL	Recipient address for scheduled email reports
NEXTAUTH_SECRET	Session signing secret (NextAuth) — if/when enabled
ZAI_API_KEY	z-ai-web-dev-sdk key for server-side ELI8/summary generation

8.4 Ports & Services

Service	Port	Notes
Next.js app (Caddy gateway)	3000	Single externally-exposed port; gateway front-end
Socket.io mini-service	3003	PumpPortal feed multiplexing; accessed via XTransformPort
Jito mainnet block engine	443 (external)	https://mainnet.block-engine.jito.wtf/api/v1/bundles — POST signed tx as single-tx bundle
Helius RPC	443 (external)	mainnet.helius-rpc.com — enrichment, TX submission, server-side confirm
Kraken REST API	443 (external)	api.kraken.com — 15-second blue-chip price polling loop
SQLite database	file	prisma/dev.db; no network port

8.5 Key Dependencies

Category	Package
Framework	next ^16.1, react ^19, react-dom ^19
Types	typescript ^5, @types/react, @types/react-dom
Styling	tailwindcss ^4, @tailwindcss/postcss, tw-animate-css, class-variance-authority, clsx, tailwind-merge
UI primitives	@radix-ui/* (alert-dialog, dialog, dropdown-menu, label, scroll-area, separator, slider, slot, switch, toast)
Icons	lucide-react ^0.525
Solana	@solana/web3.js ^1.99
Base58 (new in v2.0.0)	bs58 ^6.0.0 — base58-encode signed tx bytes for Jito bundle submission
Data layer	@prisma/client ^6.11, prisma ^6.11
Server state	@tanstack/react-query ^5.82
Realtime	socket.io-client ^4.8.4
Charts	recharts ^2.15.4
Theming	next-themes ^0.4.6
Email	nodemailer ^10.0.14, @types/nodemailer
AI (server-only)	z-ai-web-dev-sdk ^0.0.18
Lint	eslint ^9, eslint-config-next ^16.1

9. Risk Management

SniperBot's risk-management layer (the RiskManager bot) is the platform's answer to the reality that no edge survives without position sizing and circuit-breaking. The following controls are enforced on live trading and, where applicable, reflected in simulation so that simulated and live behavior remain comparable. v2.0.0 adds the blue-chip risk-gate bypass as a deliberate, auditable design choice.

Control	Default	Effect
Circuit breaker	N consecutive losing trades	Disarms SniperBot and requires manual re-arm; prevents tilt-spirals
Kill switch	Manual + auto triggers	One-tap (and event-driven) full disarm of all bots; cancels pending auto-sells
Daily loss cap	Configurable SOL / % of bankroll	Halts new entries for the rest of the UTC day once hit; open positions still respect TP/SL
Position sizing	Fixed-SOL per trade (preset-tiered: 0.05–0.20 SOL)	Caps single-trade exposure; no martingale, no doubling-down
Narrative concentration limit	Max N positions per theme/cluster	Prevents the entire book from being one correlated meme-coin narrative
Time-based exit	Max hold duration per preset	Auto-exits positions that neither TP nor SL within the window; releases capital from dead positions
Hard authority gate	Mint/freeze authority revoked	Refuses to arm on meme-coin tokens retaining mint or freeze authority — non-overridable
Risk-score floor	Configurable 0–100 threshold	Hides or red-flags meme-coin tokens below threshold; execution gated above floor
Blue-chip risk-gate bypass	All 17 verified mints	v2.0.0 — Verified blue-chip mints bypass the 8-check risk engine; zero rug risk = no safety scoring needed
Jito bundle submission (E1)	All live swaps	v2.0.0 — Signed txs submitted via Jito block engine; bypasses public mempool; eliminates dominant MEV sandwich vector
Real on-chain P&L	All auto-sells	v2.0.0 — realizedPnlSol computed from Jupiter outAmount vs. buyAmountSol; DB stores real on-chain value

These controls compose into a defense-in-depth posture: a single bad trade is bounded by position sizing; a bad day is bounded by the daily loss cap; a bad streak is bounded by the circuit breaker; a correlated blow-up is bounded by the narrative-concentration limit; a stuck position is bounded by the time-based exit; and now, in v2.0.0, an MEV-sandwich attack is bounded by Jito bundle submission, and a misleading accounting figure is bounded by real on-chain P&L. None of these makes a negative-EV strategy positive, but each materially reduces the probability of a single event wiping the account — which is the prerequisite for any compounding strategy to have a chance to run.

10. Regulatory & Legal Considerations

This section is not legal advice. It is a structured, plainly-worded survey of the regulatory and legal considerations a user of SniperBot should be aware of. It cannot be exhaustive — the space changes monthly — and it is no substitute for advice from a qualified attorney in your jurisdiction. The v2.0.0 blue-chip pivot does not materially change the legal analysis (it changes the asset population, not the regulatory framework), but it does diversify the user's legal exposure across different asset categories (securities-likely meme coins vs. commodity-classified BTC vs. stablecoin-regulated USDC/USDT).

10.1 "Not Financial Advice" — What It Actually Means

SniperBot is research software. It does not hold user funds, does not execute trades on a user's behalf without per-transaction authorization, does not advise any specific user on any specific trade, and is not registered as an investment adviser anywhere. The phrase "not financial advice" is not a boilerplate disclaimer tacked on for legal hygiene; it is a structurally accurate description of what the software is and is not. The user is the sole decision-maker and the sole signer; SniperBot provides information and automation, not fiduciary guidance.

10.2 KYC / AML Considerations

Because SniperBot does not custody funds, it is not in the most stringent tier of money-services regulation, which typically attaches to custodial intermediaries. However, several considerations remain:

- **Source-of-funds.** A user connecting a wallet is implicitly representing that the wallet's funds are lawfully obtained. SniperBot does not verify this and cannot; it is the user's representation.
- **Counterparty exposure.** Tokens SniperBot surfaces are issued by unknown third parties. A user who realizes gains from a token later deemed proceeds of crime could face confiscation or clawback in some jurisdictions regardless of their own good faith. Blue-chip mints are issued by established entities (Circle for USDC, Tether for USDT, the Solana Foundation for native SOL, etc.) — counterparty exposure is concentrated in a small number of well-known issuers rather than the anonymous deployer of a Pump.fun token.
- **Sanctions screening.** SniperBot does not screen counterparties against sanctions lists. A user in a heavily-regulated jurisdiction may have independent obligations to avoid transacting with sanctioned addresses; SniperBot provides no tooling to assist with this.

10.3 Jurisdiction-Specific Warnings

The legal status of crypto-asset trading varies materially by jurisdiction. The following are illustrative, not exhaustive, and are current only as of the document date.

Jurisdiction	Illustrative considerations
United States	SEC and CFTC enforcement risk; BTC widely treated as a commodity, ETH declared not-a-security, SOL/altcoin classification less settled; state money-transmitter laws; IRS reporting on taxable events

Jurisdiction	Illustrative considerations
European Union	MiCA regulation framework for crypto-assets; consumer-protection rules; AML obligations under EU directives; stablecoin-specific rules under MiCA
United Kingdom	FCA crypto-asset regime; financial-promotion rules; AML registration for firms
Singapore (MAS)	Payment Services Act; DPT token regimes; relatively clearer rules but still applicable
Other jurisdictions	Highly variable — some ban retail crypto-derivatives entirely; some require licensing; some are silent. Assume regulation applies until a local attorney confirms otherwise

SniperBot is operated from a sandboxed research environment and does not geofence users. It is the user's responsibility to determine whether operating this software, or trading the tokens it surfaces, is lawful in their jurisdiction. The platform's operators accept no responsibility for the user's regulatory compliance.

10.5 v2.3 Addendum — Platform Maturation Round

The v2.6.0 release is a comprehensive platform maturation round. It closes the #1 live-trading risk (stop-loss latency), adds an operator wallet allowlist (the critical security gate for public deployment), introduces a How-To AI chatbot + robot mascot, and ships the legal documents required for public sale.

10.5.1 Kraken WebSocket — Sub-100ms Stop-Loss

The blue-chip price poll was previously a 15-second REST poll to `/api/market/kraken`. This meant stop-losses could fire up to 15 seconds late — a meaningful slippage window for real-money positions.

In v2.3, a direct WebSocket connection to `wss://ws.kraken.com/v2` subscribes to the ticker channel for all blue-chip pairs (SOL/USD, ETH/USD, BTC/USD, etc.). Price updates now arrive in ~100ms instead of ~15s — a 150× reduction in stop-loss latency. The REST poll remains as a 60-second fallback (reduced from 15s) for feed injection + WS-failure resilience.

- Stop-loss latency: ~15s → ~100ms (150× reduction)
- REST poll: 15s → 60s (4× less outbound HTTP)
- CSP allows `wss://ws.kraken.com` + `wss://stream.binance.com:9443`
- WS auto-reconnects; REST poll covers the gap when WS is dead

10.5.2 Operator Wallet Allowlist

The prior auth model allowed ANY wallet to authenticate (connect Phantom → sign challenge → get a session). Once authenticated, any wallet could POST to `/api/trade/swap` with `direction='transfer'` + their own `coldStorageAddress` — the server would execute `jup spot transfer --to <attacker> --key trading-wallet`, draining the funded bot wallet.

The v2.3 fix adds an operator wallet allowlist via the **OPERATOR_WALLETS** environment variable (comma-separated base58 addresses). A new `requireOperator()` function gates all routes that move SOL or change bot state. When the allowlist is unset, the gate is disabled (dev mode — all wallets allowed). A startup guard in `layout.tsx` refuses to boot in production without **OPERATOR_WALLETS** set.

10 call sites across 9 routes are now gated:

Route	What it protects
<code>/api/trade/swap</code>	Cold-storage withdrawal (moves SOL via Jupiter CLI keyfile)
<code>/api/bot/config</code>	Changes <code>buyAmount</code> , <code>stopLoss</code> , <code>takeProfit</code> , <code>minSafetyScore</code>
<code>/api/bot/toggle</code>	Arm/disarm the bot
<code>/api/positions/journal</code>	Write journal notes
<code>/api/logs</code>	Write bot log entries

Route	What it protects
/api/alerts	Send Telegram/push alerts
/api/backtest/run	Run backtest
/api/research/analyze	Invoke ZAI LLM
/api/reports/send	Send email report

10.5.3 How-To AI Chatbot

A ZAI LLM-powered assistant accessible via a 'How To' button in the footer (next to the Whitepaper link). The chatbot knows all 18 dashboard sections + can answer any platform question with step-by-step instructions. Responses include clickable section hyperlinks — clicking a link closes the dialog + scrolls to the section + flashes a green ring around it.

- 18 dashboard section IDs for deep-linking
- 6 suggested questions on open
- Multi-turn conversation (the LLM remembers context)
- Rate-limited: 20/min/IP (no auth required)

10.5.4 Robot Mascot + PWA Support

A 3D lime-green robot mascot (holding a calculator + gold coins) replaces the bullseye/Crosshair icon across the platform: browser tab favicon, header (36×36), footer (16×16), How-To button (14×14), chatbot avatar (32×32), and PWA icon. A manifest.json enables 'Save to Desktop' support with standalone display mode.

10.5.5 ARM Toggle UX Improvements

The ARM/DISARM toggle previously failed silently when the user wasn't authenticated — the switch snapped back to OFF with no explanation. v2.3 adds:

- Precondition checks: 'Connect your wallet first' + 'Enable LIVE mode first'
- Auto re-authentication: on 401, automatically triggers the Phantom sign popup
- Error-specific toasts: 401 (auth), 403 (not operator), network failure
- Auto-scroll to Live Feed when the bot arms
- First-snipe toast: 'Bot sniped a token!' when positions grow

10.5.6 Dynamic PAPER/LIVE Status

The footer badge + header tag were previously hardcoded ('Paper trading' + 'Stage 2 · Paper Trading'). They now update dynamically when the user toggles between PAPER and LIVE mode — PAPER shows green/amber, LIVE shows red. No more contradictory state where the header says LIVE but the footer says Paper.

10.5.7 Custom Card Layout

Users can now reorder dashboard sections (CollapsibleSection cards) up or down via arrow buttons on each section header. The custom layout is persisted in localStorage per-wallet — each user has their own preferred arrangement. The @dnd-kit/sortable package is installed for future drag-and-drop support.

10.5.8 Testing + Legal Documents

9 Playwright smoke tests cover the critical paths: page load, mascot render, How-To chatbot, ARM toggle errors, cold-storage confirmation, portfolio tiles, DCA benchmark, Whitepaper download, and PWA assets.

Legal documents required for public sale:

- LICENSE — non-exclusive, non-transferable, non-sublicensable
- TERMS_OF_SERVICE.md — 10 sections including risk disclosure + regulatory notice
- PRIVACY.md — self-hosted (no SniperBot telemetry), 9 third-party data flows
- SECURITY_AUDIT.md — 9-section audit checklist with 54 verified controls

10.5.9 Performance Metrics

The v2.3 release achieved an 8× reduction in outbound HTTP:

Metric	Pre-v2.1	v2.3
Outbound HTTP (steady)	~250 req/min	~30 req/min
Stop-loss latency	~15s (REST poll)	~100ms (WebSocket)
SOL/USD polling	8 polls/min	1 shared GET/min
Kraken fan-out	23 req/call	2 batched + 5s cache
Monte Carlo compute	Every 3s tick	Memoized (30s bucket)
TypeScript errors	11	0
Lint errors	1	0
External console.error	113	0

10.5.10 Cold Storage Withdrawal

SniperBot supports withdrawing SOL from the bot's funded trading wallet to any external Solana address — including hardware wallet addresses (Ledger, Trezor, SecuX V20), exchange deposit addresses, or any other base58 address. This is the recommended capital-preservation path: the bot trades from a hot wallet (the funded trading wallet), and profits can be periodically withdrawn to a hardware wallet for long-term cold storage.

How to withdraw to cold storage:

- Navigate to the **Live Holdings** panel (requires LIVE mode enabled + wallet connected)
- Find the **Cold Storage** section within the Live Holdings panel
- Enter the destination address — the FULL base58 address of your hardware wallet or external wallet (do NOT truncate)
- Enter the amount of SOL to withdraw
- Click **Send to Cold Storage** — an AlertDialog confirmation appears
- The dialog displays the **full destination address** (not truncated) + the amount + an irreversible warning
- Check the **'I verify this address is correct'** checkbox — the Confirm button is disabled until this is checked
- Click **Confirm withdrawal** — the server executes `jup spot transfer --to <destination> --amount <N> --key trading-wallet`
- The SOL moves from the bot's funded wallet to the destination address — the transaction is **irreversible** (on-chain)

Supported destination addresses:

Destination Type	Example	Recommended?
Hardware wallet (Ledger)	Ledger Solana app address	Yes — best for long-term storage
Hardware wallet (Trezor)	Trezor Solana app address	Yes — best for long-term storage
Hardware wallet (SecuX V20)	SecuX Solana account address	Yes — air-gapped, highest security
Exchange deposit address	Coinbase/Binance SOL deposit	Yes — for selling to fiat
Another Phantom wallet	Any base58 Solana address	Yes — for transferring between accounts
Paper wallet	Any base58 Solana address	Yes — offline-generated keypair

Security features:

- The destination address is shown in **full** (not truncated) — users can verify every character before confirming
- The **checkbox gate** prevents accidental clicks — the Confirm button stays disabled until 'I verify this address is correct' is checked
- The **operator wallet allowlist** (OPERATOR_WALLETS) ensures only the operator's wallet can initiate a withdrawal — an attacker who connects a different wallet gets 403 Forbidden
- The **/api/trade/swap** route validates the destination address as base58 (`/^[1-9A-HJ-NP-Za-km-z]{32,44}$`) — junk addresses are rejected

- The withdrawal uses the Jupiter CLI (`jup spot transfer`) which signs with the server-side keyfile — no Phantom popup is needed for the transfer

Important distinction: signing vs. withdrawing

Section 5.6 notes that live trade *signing* uses the Phantom hot wallet — the bot cannot sign swaps with a hardware wallet. This is a SIGNING limitation, not a withdrawal limitation. The cold storage withdrawal feature works **regardless** of the destination wallet type — the bot's funded wallet sends SOL TO the destination, and the destination wallet does not need to sign anything to receive funds. A Ledger, Trezor, or SecuX can receive SOL without being connected to the bot or signing any transaction.

Best practices:

- Verify the destination address **character-by-character** before confirming — a single typo = irreversible SOL loss
- Send a **small test amount** (0.01 SOL) first to verify the address works, then send the larger amount
- Use a **hardware wallet** (Ledger, Trezor, SecuX) for long-term profit storage — not another hot wallet
- Withdraw profits **regularly** (weekly or monthly) — don't leave large balances on the bot's hot wallet
- The email reports feature (Section 10.5.8) can remind you to withdraw — schedule a monthly report that includes current balance

10.6 v2.4 Addendum — Platform Maturation Round (Waves 7 + 8)

The v2.6.0 release is the second platform maturation round, shipped across two development waves (Wave 7 + Wave 8). It closes the largest remaining UX gap (manual-snipe profit visibility + safety net), surfaces previously-silent rejection paths, hardens black-swan defense via a batch Close All panic button, makes the auto-close engine observable via closeReason + push notifications, adds the analytics surfaces the operator needs to learn (MFE/MAE, P&L attribution, recovery projection, config what-if), and eliminates code/docs drift in the chatbot knowledge base via an automated JSDoc scanner. Each feature is documented below in the same technical, honest tone as the v2.3 addendum. **None of this is financial advice.**

10.6.1 Manual Snipe UX (W7 + W8.1)

When the user manually clicks Snipe on a feed row (a human-initiated trade — distinct from the autonomous auto-snipe loop), a Radix Dialog popup now appears. The popup shows the immediate-sell profit (in SOL + USD, net of the ~1.5% DEX slippage/Jupiter fee estimate), the live price (auto-updating on every price tick — the parent passes the live position lookup), a 50%-from-high safety-net progress bar with the threshold marked red, and a BIG SELL button. The dialog auto-dismisses when the underlying position's status flips to CLOSED/REKT — covering the SELL click, the safety-net auto-sell, or any other auto-close trigger. **The popup is non-blocking** (it's a portal-rendered overlay) — the bot's auto-snipe loop, auto-close effect, and socket feed processing continue unimpeded while the popup is open. Auto-snipe trades (reason 'Auto snipe ...') do NOT trigger the popup; it is for human-initiated snipes only.

Fractional sell buttons (W8.1) — the BIG SELL button is replaced by a row of 3 buttons: **Sell 25%** / **Sell 50%** / **Sell 100%**. The 100% button retains the primary visual priority (filled + bold + shadow); the 25% + 50% buttons are secondary outline buttons using the same emerald/red accent. This lets the user scale out to lock profit and let the remaining fraction ride the safety net. The onSell callback signature is widened to pass the chosen sellPct up to the parent — for fractional LIVE sells (<100%), the position stays OPEN with the remaining fraction; the auto-close effect handles the rest later.

50%-from-High Safety Net (W7) — for positions whose snipeReason contains 'manual snipe' (case-insensitive), the auto-close effect now has an additional branch positioned **BEFORE** the existing time-based exit / tiered TP / trailing-stop / hard-SL branches. The branch fires when `ts.mfe > 0` (there is a peak) AND the drop-from-high is $\geq 50\%$. The threshold constant `MANUAL_SNIPE_FLOOR_PCT = 50` is exported from `use-sniper-engine.ts` and imported by the popup component so the progress bar marker and the engine branch cannot drift. When the threshold is breached, the engine fires `liveAutoSellIfLive` (LIVE — Phantom on-chain 100% sell) or `closePosMut` (PAPER — sentinel close), marks `autoClosedRef` so the existing branches don't double-sell, and continues to the next position. **This branch fires even on tokens that bled straight down from buy** — there is no MFE > buy guard (unlike the configured trailing-stop branch). On on-chain sell failure, the tier state is restored and the next tick retries.

10.6.2 Error Surfacing + Auth (W7.1 + W8.4-auth)

Snipe Rejection Surfacing (W7.1) — the `snipeMut` mutation previously had no `onError` handler. `fetchJson` throws on non-2xx with a `<NNN>: <message>` format, but React Query's default behavior is to swallow the error when no `onError` is wired. The user saw only the optimistic 'Sniping \$SYMBOL' toast (which fires BEFORE the API call) and wondered why Open Positions stayed empty. The fix mirrors the existing `toggleBotMut.onError` pattern: parse the status + server-message substrings via regex and surface a destructive toast + Activity Log error row. All 7 documented snipe API rejection codes are mapped:

Code	Toast title	Why it fires
423	■ Bot not armed	Paper snipes require the ARM toggle to be ON before the API will accept a snipe
422	■ Liquidity too low	Launch liquidity below <code>config.minLiquidityUsd</code> (default ~\$25K, raised to ~\$500K for Blue Chip preset)
422	■ Holders too low	Token holder count below <code>config.minHolders</code> — protects against dead-on-arrival launches
400	■■ Risk gate rejected	Launch <code>safetyScore</code> below <code>config.minSafetyScore</code> (default 85; typical Pump.fun launches score 20-40 and are rejected here)
429	■ Max positions reached	Open-position count at <code>config.maxOpenPositions</code> cap — close one before sniping again
409	■■ Already open	An OPEN position for this mint already exists in the user's wallet — no duplicate snipes
422	■ Buy amount exceeds cap	Computed <code>buyAmount</code> (Kelly-adjusted) exceeds the 0.5 SOL hard cap — Kelly capped at max-bound
401	Authentication required	Session expired — see auto re-auth below (W8.4-auth)
—	Server unreachable	Network failure / fetch rejected — the dev server or mini-service is down

Auto-Re-Authentication on 401 (W8.4-auth) — when a snipe gets a 401 (session expired), the engine now auto-triggers the Phantom re-authentication flow (the same `enableLive()`-style flow used by the ARM-toggle pattern). Previously, a 401 would surface as a toast and the user had to locate the 'Re-Authenticate' link in the Live Holdings panel. Now the Phantom sign popup appears automatically after the toast — the user signs the challenge message, the HMAC-signed `httpOnly` session cookie is refreshed, and the original snipe can be retried. This mirrors the v2.3 ARM-toggle auto-re-auth pattern (Section 10.5.5).

LIVE Toggle Dialog Correction (W8-LIVE-toggle-fix) — the LIVE-mode confirmation `AlertDialog` previously contained an inaccurate statement that misled users about the auto-sniper's behavior. The previous text said the auto-sniper stayed on PAPER even when LIVE mode was enabled (i.e. LIVE was 'manual-only by design'). This was incorrect: when LIVE + ARMED, the auto-sniper ALSO uses real SOL

— every auto-buy triggers a Phantom sign popup. The corrected dialog now states: 'When LIVE + ARMED, the auto-sniper ALSO uses real SOL — every auto-buy triggers a Phantom sign popup.' This is a critical correctness fix — users who toggled LIVE on the previous (incorrect) text and armed the bot would have been surprised to see Phantom popups appear for autonomous snipes.

10.6.3 Batch Operations + Safety (W8.2 + W8.2-ext + W8.4 + W8.3-reconnect)

Close All Panic Button (W8.2 + W8.2-ext) — a single button that closes ALL open positions in one batch, used for black-swan defense (rapid market dump, dev rug, exchange outage, etc.). The button has two access points: (a) the header, placed visually next to the STOP kill switch (panic-button placement — both emergency actions grouped together), and (b) the Open Positions table header, next to the position-count badge (discoverable placement — the user sees it right where the positions are listed). Both call the same `handleCloseAll` handler. The button shows whenever ANY positions are open (PAPER or LIVE — extended in W8.2-ext; the original W8.2 was LIVE-only). An AlertDialog confirmation surfaces the PAPER/LIVE split before any action: 'Total: N (X LIVE, Y PAPER) — LIVE positions: batch-sells 100% on-chain via Phantom, each one triggers a separate sign popup. PAPER positions: marks the virtual DB rows closed (no on-chain sell — paper trades have no real SOL to move).' The batch handler closes LIVE positions first (the slow Phantom-sign path, with per-position error handling + continue-on-failure so a single rejection doesn't abort the batch), then PAPER positions (the fast DB-only close path). If there are LIVE positions but LIVE mode is OFF, a clear error is surfaced pointing to the PAPER→LIVE toggle (no silent failure). A final toast reports the outcome: 'Batch close complete · X closed, Y failed'.

Socket-Degraded LIVE-Warning Badge (W8.4) — when the WebSocket feed is offline AND there are open LIVE positions, a red pulsing '■ LIVE unmanaged' badge appears next to the LIVE/PAPER toggle in the header. The badge warns the user that auto-close (TP/SL/trailing/50%-from-high safety net) cannot fire while the socket is dead, and that new auto-snipes are paused. This is the user-facing surface for the otherwise-silent failure mode where a LIVE position's trailing stop would not fire because the price feed is stale. The badge uses `animate-pulse` (Tailwind) for the pulsing animation and reads from the existing `socketConnected` prop + `liveOpenCount` — PAPER-only positions do not trigger the badge (PAPER trades don't need socket-driven auto-close protection; they're virtual DB rows).

Socket Reconnect Re-Subscription (W8.3-reconnect) — when the WebSocket drops and later revives (the v2.3 socket revival infrastructure is now wired; see Section 10.5.x), the engine + the simulation studio automatically clear their subscribed-mints cache and re-emit `subscribe-price` for all current position mints. This eliminates the bug where a reconnect left the price feed silent for open positions — the symptom was '35 open positions but the prices don't change' — because the new socket connection had no active subscriptions (the prior `subscribe-price` emit only ran on the initial connect). The reconnect path now performs the same subscription handshake as the initial connect, so prices resume flowing within ~1.5 seconds of socket revival.

10.6.4 Auto-Close Push Notifications (W8.3)

closeReason column on SnipedToken — the close API (`/api/positions/close`) now persists the close reason on the SnipedToken row as a nullable String column. The schema migration is non-destructive (`closeReason String?` — null ≠ empty string; existing rows remain null until they are re-closed). The positions + transactions APIs expose the column in their select clauses so the client can render the close reason in the position row + the transaction history. Documented close reasons include:

- Time-based exit — the position hit its max-hold duration (default 1h meme / 4h blue-chip)
- Trailing stop — the price dipped from MFE by the configured trailing percentage
- Safety net — the 50%-from-high safety net (manual snipes only) fired (Section 10.6.1)
- Hard SL — the price hit the configured stop-loss percentage (default 10% blue-chip, 15% meme)
- Tier TP — a tiered take-profit target fired (25%/50%/75%/100% of position at progressively higher TP%)
- Manual close — the user clicked the Close button themselves
- Live close — a LIVE-mode manual close triggered an on-chain Phantom sell
- REKT — the position's P&L dropped below the REKT threshold (scales per-trade)

Push Notifications for ALL Auto-Close Triggers — the notifications hook previously fired an OS-level push notification ONLY for the REKT trigger. The hook is now extended to fire for every auto-close trigger (trailing stop / time-based / safety-net / hard SL / tiered TP). The hook uses a positive-match keyword filter on the `closeReason` string to distinguish auto-close from manual close: a manual close (the user clicked the button themselves — they already know) is skipped. The keyword filter matches: `trailing`, `time-based`, `safety-net / safety net`, `stop-loss / hard sl`, `tier`, `take-profit / take profit`, `rekt`, `max-consecutive`. The hook also tracks `prevClosedIdsRef` (a Set of already-notified position IDs) so a position's auto-close only fires one notification even across multiple re-renders.

10.6.5 Position Table Enhancements (W8.5 + W8.4-sim-positions)

MFE / MAE Columns (W8.5) — the Open Positions table now shows two additional per-row metrics between the P&L column and the Close action button:

- **MFE (Max Favorable Excursion)** — the peak % above buy. Tracked as `mfePriceSol` on SnipedToken (already in the schema from the tiered-TP + 50%-from-high safety-net work). Rendered as `+X.X%` in emerald.
- **MAE (Max Adverse Excursion)** — the worst % below buy. Derived at render time from `Math.min(buyPriceSol, livePriceSol ?? currentPriceSol)` (no schema field — derived cheaply). Rendered as `-X.X%` in red.

Both columns are hidden on mobile (the existing list-based table uses `hidden md:flex` — the flex equivalent of the spec's `hidden md:table-cell`). Each value carries a native title tooltip ('Max Favorable Excursion — peak price since buy' / 'Max Adverse Excursion — worst dip since buy'). The columns help the user learn when to take profits — a high MFE paired with a closed-at-loss row is the canonical 'should have taken profits earlier' signal.

Chart Links (W8.4-sim-positions) — every position row (real OR simulation) now has a Chart button (an external-link lucide icon) that opens DexScreener for that mint in a new browser tab (<https://dexscreener.com/solana/<mint>>). DexScreener is the best free live chart service for Solana tokens — it shows price, liquidity, holders, transactions, and the bonding curve in real-time. The button is small (h-7) and matches the table's compact styling; it sits next to the Close button in the row's action cell. The link is unconditionally rendered (no per-venue special-casing) — DexScreener handles Pump.fun, pumpswap, Raydium, and Meteora mints identically.

SIM Open Positions List (W8.4-sim-positions) — the Simulation Studio panel now shows a scrollable list of virtual positions (the sim's open VPoses) at the bottom of the panel when the sim is running. Each row shows: symbol, buy→live price (with arrow), P&L (SOL + %), MFE, active exit tiers, and the DexScreener Chart link (same component as the real positions table). The list has a max-height of 288px and a sticky header so the column labels stay visible during scroll. The list makes the sim's open exposure legible — previously the user could see the cumulative P&L curve and the closed-trades stats but not the currently-open virtual positions.

10.6.6 Analytics + UX (W8.6 + W8.8 + W8.9 + W8.10)

Activity Log Severity Filter (W8.6) — a sticky chip row at the top of the Activity Log panel: **All / Info / Warn / Error / Success / Sniper**. Each chip shows its count (e.g. 'Error (3)') computed by a case-insensitive accumulator on `log.level`. The active chip is visually distinct (`bg-primary text-primary-foreground aria-pressed`). The chip row is sticky (`sticky top-0 z-10`) with a backdrop blur so it stays visible during scroll. The footer of the panel shows 'Showing X of Y logs' when filtered. The virtualizer's count is wired to the filtered array length; when 'all' is selected, the original array reference is preserved so the virtualizer's scroll offset stays stable across toggles.

Config What-If Preview (W8.8) — a '?' icon next to the Min Safety Score slider in the Sniper Configuration panel. Clicking the icon opens an inline panel that queries the LaunchSnapshot table: *every launch the bot sees is recorded with its safetyScore + 1m/5m/30m/1h/24h price snapshots*. The panel shows: 'At score X, you would have admitted N of M launches (Y% rate) over the last 7 days' + a 5-bucket histogram (0-20 / 21-40 / 41-60 / 61-80 / 81-100) of the safetyScore distribution with the currently-active bucket highlighted. The query has a 500ms debounce + a TanStack Query cache (60s) so dragging the slider doesn't spam the API. The endpoint is `GET /api/config/preview?minSafetyScore=X&days=7` — wallet-agnostic (LaunchSnapshot has no walletAddress column, so the query is global; this is honest — the operator's 'edge' is measured against the entire launch population the bot has seen). The panel helps the user understand the trade-off between admission rate (more snipes) and quality (higher-score launches): a slider at 85 admits ~5% of launches; a slider at 40 admits ~70% — but the 70% are far more likely to be rugs.

P&L Attribution Panel (W8.9) — a new panel in the P&L Analytics section that breaks down closed-trade P&L by three axes:

- **By Preset** — Blue Chip / Moonshot Hunter / Momentum / Balanced / Small Wins (the 5 strategy presets). The parser uses substring match ordered longest-first so 'Moonshot Hunter' wins over 'Moonshot'.

- **By Venue** — pump / pumpswap / raydium / meteora (the 4 supported DEX venues). The parser mirrors `/api/research/analyze's VENUE_RE`, ordered PumpSwap-first so 'PumpSwap' isn't mislabeled 'Pump'.
- **By Score Bucket** — 0-20 / 21-40 / 41-60 / 61-80 / 81-100 (the 5 safetyScore buckets).

Each row shows the trade count, win rate, and total P&L. The win rate is color-coded: >55% green (the bucket is profitable), 45-55% gray (marginal), <45% red (the bucket is bleeding). P&L is color-coded emerald for positive, red for negative. The panel surfaces the canonical 'which preset actually makes money' answer at a glance — previously the user had to manually scan the transaction history + mentally aggregate. The endpoint is `GET /api/wallet/attribution?wallet=X` — wallet-scoped (uses the signed-message session for IDOR protection).

Recovery Projection Tile (W8.10) — the 6th tile on the Portfolio Dashboard (joining Net P&L, Win Rate, Open Positions, DCA Benchmark, and All-Time ROI). The tile shows: 'Nd to recover -X SOL at +Y/trade' when the portfolio is in a drawdown (current balance < all-time high). It uses the all-time average P&L per trade as a proxy for forward recovery — a transparent, backwards-looking estimator. The tile's tooltip explains the limitations: the projection assumes future P&L per trade will match the all-time average (it won't — variance is high), ignores compounding (it's a linear projection), and ignores position sizing (fixed-buy-amount assumption). When the portfolio is at a new all-time high, the tile shows '✓ At all-time high — no recovery needed'.

10.6.7 Config + Paper Mode (W8.7)

Paper-Mode Risk-Gate Bypass Toggle — a new `paperBypassRiskGate` Boolean on `BotConfig` (default false). When ON, paper snipes skip all 4 server-side gates:

- **bot-armed** (423) — paper snipes no longer require the ARM toggle
- **min-liquidity** (422) — paper snipes no longer enforce `config.minLiquidityUsd`
- **min-holders** (422) — paper snipes no longer enforce `config.minHolders`
- **min-safety-score** (400) — paper snipes no longer enforce `config.minSafetyScore`

This matches the LIVE-trade behavior where the Phantom signature IS the authorization — LIVE trades have always bypassed all 4 gates via `liveSignatureVerified=true` because the signature is cryptographic proof the user authorized the specific swap. The toggle is surfaced as a Switch at the bottom of the Sniper Configuration panel (next to the other 'advanced' toggles like `tradingHoursBoost`). The flag does NOT affect LIVE trades — they already bypass via `liveSignatureVerified`; the toggle only governs paper snipes. **Schema migration:** `paperBypassRiskGate` Boolean @`default(false)` on `BotConfig` — non-destructive, deployed via `bun run db:push`. The use case: an operator testing the bot's behavior on a fresh Pump.fun launch (typically rejected at the risk-gate) can flip the toggle ON, paper-snipe to see how the auto-close engine would have managed the trade, then flip it OFF to return to the protective default. The toggle is per-wallet (`BotConfig.walletAddress` is the unique key). **Warning:** this toggle makes paper mode less representative of LIVE mode (LIVE trades bypass the gates via signature, paper trades with the bypass ON bypass them via config — same effect, different mechanism). The operator should treat paper trades with the bypass ON as 'what would have happened if I had manually live-bought this token' — useful for studying the exit ladder, not for studying the admission funnel.

10.6.8 Simulation Studio — Use My Trade History (W8.12)

Use My Trade History Toggle — a 'Use my trade history (last 30 days)' Switch at the top of the Simulation Studio panel. When ON, the sim fetches the user's closed trades via `GET /api/sim/from-history?wallet=X&days=30` and replays them through the counterfactual exit ladder — instead of running on all historical launches in the time window, it runs **ONLY** on the launches the user actually sniped + compares what would have happened under the counterfactual config (different buy size + exit ladder). This measures the user's **specific edge** vs the market average — the difference between 'what the bot would have done on every launch' (the default sim) and 'what the bot would have done on the launches / chose to snipe' (the history replay).

The replay model is one-shot per trade: each historical trade is constructed at its `buyPriceSol` for the counterfactual `cfg.buyAmountSol` (NOT the historical `amountSol` — the counterfactual asks 'what if I had used my current buy size'). The historical exit price is derived from `pnLPct`: `exitPrice = buyPrice * (1 + pnLPct/100)`. The configured exit ladder (tier TPs / hard SL / trailing stop) is applied **ONCE** at that exit price — tier TPs fire on the way up, hard SL fires when `pnLPct <= -stopLossPct`, trailing stop fires on a dip from MFE (a no-op for losers since `MFE ≤ buyPrice`). If no ladder exit fired, the position is force-closed at the historical exit price with WIN/LOSS status (the historical trade WAS closed in reality, so the counterfactual must close too). The result is a complete `SimulationState` with `status='complete'` + a Monte Carlo projection (≥ 3 closed trades → bootstrap). The duration picker is locked to 30 days when history mode is ON (with an amber '(locked — history mode)' label). The fetch-status badge cluster shows: Connect wallet / Fetching / N trades ready — click Start / Fetch failed / No closed trades in last 30 days. The endpoint is rate-limited 10/min/IP (the user can spam the toggle; the limit protects the API). IDOR redaction mirrors `/api/transactions` — non-owner callers see redacted `journal/txSignature/snipeReason` on LIVE rows.

10.6.9 Chatbot KB Auto-Population (W8.11)

@chatbot JSDoc Scanner — a new Bun script (`bun run kb:scan`) runs `scripts/scan-chatbot-kb.ts`, which walks `src/` recursively (skipping `node_modules`, `.next`, `dist`, `.git`, `build`, `out`) and finds every JSDoc block containing an `@chatbot` tag. The scanner parses the YAML-ish key: value lines (title / section / keywords / description) and writes the resulting array to `src/lib/chatbot-kb-index.json` (pretty-printed, 2-space indent, sorted by section then title). The `/api/how-to` route reads this index at module load and injects it as a '## Auto-Discovered Features' section appended to the existing static `SYSTEM_PROMPT`. The ZAI LLM sees the discovered entries at every request, alongside the manually-maintained sections.

Why this matters — eliminates code/docs drift. Previously (W7.2), every feature needed a manual edit to `SYSTEM_PROMPT` — expensive + error-prone. Now, when a developer ships a feature with a `@chatbot JSDoc` tag and re-runs `kb:scan`, the chatbot knows about it without anyone manually editing `SYSTEM_PROMPT`. 3 seed entries ship with v2.4: *Manual Snipe Popup*, *How-To Assistant Chatbot*, and *Sniper Configuration Panel* — each with a JSDoc block in its source file. The pattern scales: future features add a JSDoc + re-scan; the chatbot learns immediately. The scanner prints a summary: 'Scanned N files, found M @chatbot entries across K sections.'

@chatbot field	Purpose	Example
title	Short feature name shown to the LLM	Manual Snipe Popup
section	Groups entries in the prompt	Common Tasks
keywords	Trigger words the LLM can match against the user's question	snipe, popup, sell, 50%, safety net
description	2-4 sentence plain-language explanation of the feature	When the user clicks Snipe on a feed row, a popup appears...

10.6.10 Code Health (W8.13 + W8-deploy-fix)

Three code-health fixes were bundled into the v2.4 wave. None changes user-visible behavior, but all three materially improve the build's correctness + the production deployment path.

Fix 1: `<img>` → `next/image` (W8.13) — the how-to-chatbot mascot images previously used the native HTML `<img>` tag at 3 sites (the How-To button avatar, the dialog header avatar, and the empty-state avatar). The native `<img>` tag bypasses Next.js's image optimization (no automatic resizing, no lazy-loading, no blur placeholder). Replaced with `next/image`'s `Image` component, with matching width + height props (14×14 for the How-To button; 32×32 for the dialog header + empty-state). Tailwind size classes are preserved (they override the intrinsic width/height) so the visual layout is unchanged. The mascot image is served from `/public/mascot.png` — a static path that `next/image` handles correctly on SSR with no special config.

Fix 2: Dropped unused `_wallet` param (W8.13) — `use-regime-poller.ts` (the simple public-endpoint poller extracted from `page.tsx` in W6-3) had a positional `_wallet` parameter that was kept only for API symmetry with its `.tsx` twin (which has since been deleted — see W8-parent). The regime endpoint is public (no wallet-scoping); the param was unused. Dropped the param from the signature + the single caller in `page.tsx`. Updated the JSDoc + the file-header comment block to explain why.

Fix 3: Deleted dead-code shadcn UI files (W8-deploy-fix) — the deployment was failing because the TypeScript `pass` (next build's pre-compile type check) had 6 errors in `src/components/ui/calendar.tsx` + `src/components/ui/resizable.tsx`. These files were generated against older package versions (react-day-picker v8, react-resizable-panels v0.x) but the actually-installed packages are v10 + v4 respectively — completely different APIs (the v10 `ClassNames` type removed the 'table' key; the v4 exports changed from `PanelGroup/PanelResizeHandle` namespace members to `Grid/Cell/Gridline`). The 6 type errors blocked the build's TypeScript pass even though **neither file was imported anywhere** in the production code path (grep across `src/` returned 0 matches for both imports). The `PnLCalendar` component is a CUSTOM calendar implementation that doesn't use the shadcn calendar — it's a separate grid-based P&L display. **Fix:** deleted both unused broken files. Build now succeeds: ✓ Compiled successfully in 20.7s + ✓ Generating static pages using 1 worker (6/6) in 323.3ms, exit code 0. Site is now deployable. **Lesson:** don't filter out tsc errors as 'pre-existing baseline' without checking if they're blocking the production build — the 6 shadcn UI errors had been filtered since W8.5 and were the deployment

blockers the whole time.

- **Lint:** 0 errors, 5 pre-existing warnings (3 react-hooks/exhaustive-deps in page.tsx + 2 @next/next/no-img-element in site-footer.tsx + site-header.tsx — out of scope for this wave).
- **TypeScript:** 0 NEW errors (the 6 shadcn UI baseline errors are gone — they were the W8-deploy-fix).
- **Build:** exit code 0 — site is deployable.
- **Runtime:** dev server on port 3000, GET / 200, 0 page errors, 0 console errors.

Verification summary for the v2.4 wave. All 10 feature groups above were verified via the standard gate: lint (0 errors / 5 pre-existing baseline warnings), tsc --noEmit (0 NEW errors), build (exit 0 — site deployable), dev server (port 3000 UP, GET / 200), agent-browser (0 runtime / 0 console errors, page renders with all expected sections). The chatbot's LLM was confirmed to cite the new knowledge base entries in response to suggested questions — proving the @chatbot JSDoc scanner + index injection work end-to-end. The 0.5 SOL buyAmount hard cap, the Kelly criterion buy-path gating, the forward-looking rug detection (R6/R7/R8), and the Jito MEV tip from v2.2 remain in force — the v2.4 wave is additive (new features) + non-destructive (no behavioral change to existing v2.3 paths).

This is not financial advice. SniperBot is research software. The v2.4 wave's manual-snipe popup, 50%-from-high safety net, Close All panic button, and auto-close push notifications are UX + observability improvements — they do NOT change the underlying asset risk. Meme-coin launches remain structurally negative-EV against the base rate; blue-chip swing trading remains exposed to 50%+ single-day drawdowns. The new MFE/MAE columns, P&L attribution panel, recovery projection tile, and config what-if preview are honest-accountability surfaces — they will tell the operator the truth about their edge (or lack of it). Never trade with funds you cannot afford to lose entirely. Past simulation performance does not predict future results.

10.7 Trade Workflow Documentation

This section documents the complete LIVE trade workflow — the end-to-end pipeline from enabling LIVE mode through realized P&L collection. It is the technical reference for the workflow audit + trade lifecycle visualization work shipped in the v2.4 platform maturation round (Wave 10's W10-parent audit + the W10-trade-lifecycle visualization panel). Each stage is verified for security + correctness; the Verification column on the audit table shows the specific check that prevents the failure mode for that stage. **None of this is financial advice.**

10.7.1 Complete Workflow Audit (9 Stages)

The LIVE trade workflow is a 9-stage pipeline, each stage verified for security + correctness. This section documents every stage from enabling LIVE mode through realized P&L collection.

Stage	What Happens	Verification
1. Enable LIVE	<code>useLiveTrading().enableLive()</code> sets <code>liveEnabled=true</code>	Phantom session check
2. Fund wallet	External — fund Phantom with SOL via Phantom's buy/send	None (external)
3. ARM bot OR manual Snipe	Auto-snipe loop fires when token passes risk gate + Kelly sizing; OR click Snipe on a feed row	Risk gate (server-side), Kelly criterion sizing
4. onLiveAutoBuy	Tries server-side Jupiter CLI first, falls back to <code>live.buy()</code> → Phantom signs Jupiter swap → broadcast to mainnet	<code>txSignature</code> returned
5. /api/snipe	Server verifies on-chain signature via Helius <code>getTransaction</code> — checks wallet is fee payer, swap direction is 'buy', mint matches	Helius verification + idempotencyKey dedup + 4x buyAmount cap
6. Position opens	SnipedToken row created with <code>isLive=true</code> + verified <code>txSignature</code>	DB row persisted
7. Price updates	WebSocket (PumpPortal for memes, Kraken for blue chips) pushes live prices → <code>priceMap</code> updates → <code>enrichedPositions</code> recompute → UI shows live P&L	W8.3-reconnect: on socket revive, re-subscribes to all position mints
8. Auto-close fires	TP/SL/trailing/50%-from-high triggers → <code>liveAutoSellIfLive()</code> → <code>live.sell()</code> → Phantom signs → broadcast	A7 fix: pre-mutation tier state snapshot + restore on failure
9. /api/positions/close	Server verifies sell signature via Helius → updates row (<code>status=CLOSED</code> , <code>pnlSol</code> , <code>closedTxSignature</code> , <code>closeReason</code>)	Helius verification + realized P&L from actual on-chain <code>outAmount</code>

10.7.2 What Happens When LIVE Is On + SOL Allocated + Coins Purchased

When LIVE mode is enabled + the Phantom wallet is funded + the bot is armed (or a manual snipe is triggered), the following 10-step sequence executes:

1. You enable LIVE mode → the header shows red "LIVE" badge instead of yellow "PAPER"
2. Your Phantom wallet has real SOL (you funded it externally)
3. You ARM the bot (or manually click Snipe on a feed row)
4. A token passes the risk gate (score ≥ 70 for Blue Chip preset, or paper-bypass toggle is on)
5. Kelly criterion sizes the buy (e.g., 0.1 SOL for Blue Chip preset)
6. `onLiveAutoBuy` fires → server tries Jupiter CLI keyfile first (faster, no popup) → if that fails, `live.buy()` calls `/api/trade/swap` to build a Jupiter swap → Phantom pops up asking you to sign → you click Confirm → transaction broadcast to Solana mainnet
7. The on-chain swap executes: your SOL goes to the liquidity pool, the token's SPL tokens come to your wallet
8. The server verifies via Helius (`getTransaction`) — confirms tx exists, you signed it, swap direction is buy, mint matches → creates `SnipedToken` row with `isLive=true` + verified `txSignature`
9. The position appears in Open Positions (with LIVE badge) + the manual snipe popup appears if it was a manual snipe
10. The WebSocket subscribes to price updates for that mint → live P&L starts updating every ~1.5s

10.7.3 How Profit Is Collected

When the auto-close effect fires (on TP/SL/trailing/50%-from-high), the following 8-step sequence collects the profit:

1. `liveAutoSellIfLive()` is called → it calls `onLiveAutoSell` which tries the Jupiter CLI path, falls back to `live.sell()`
2. `live.sell()` calls `/api/trade/swap` → builds a Jupiter sell swap (selling 100% of the token back to SOL) → Phantom pops up asking you to sign → you click Confirm → broadcast to mainnet
3. The on-chain sell executes: your tokens go to the liquidity pool, SOL comes back to your Phantom wallet
4. The server verifies via Helius → reads the actual `outAmount` (real SOL received) → computes `realizedPnlSol = outAmount - buyAmount` → updates `SnipedToken` row with `status=CLOSED`, `pnlSol`, `pnlPct`, `closedTxSignature`, `closeReason`
5. The SOL is now in your Phantom wallet (you can spend it, withdraw to cold storage, or use it for more trades)
6. The closed trade appears in Transaction History with realized P&L in SOL + USD
7. A push notification fires (W8.3) — "■ Sold \$SYMBOL · +X.X% · trailing stop · LIVE"
8. The Portfolio Dashboard updates — Unrealized P&L drops, Realized P&L increases, Win Rate updates, vs DCA recalculates

Profit collection summary: SOL flows from the DEX liquidity pool → back to your Phantom wallet → the realized P&L is recorded in the DB → the dashboard reflects it. The profit IS the SOL — it's not a separate 'payout' — the SOL literally returns to your wallet address after the sell.

Not financial advice. This documentation describes the technical workflow of the SniperBot platform, not a guarantee of profit. Meme-coin trading carries substantial risk of total loss.

10.8 v2.5 Addendum — Trade Lifecycle Visualization, Security Hardening, + Wide-Launch Infrastructure

The v2.6.0 release is the third platform maturation round, shipped across eight development waves (Waves 10-17). It adds the user's most-requested visualization surface (a real-time trade lifecycle panel), closes the security gaps that blocked limited public deployment (boot guard, health endpoint, CSRF Layer 2 fix, IP-spoofing fix, P&L leak fix, dead-code cleanup), lands the wide-launch infrastructure stack (Sentry error monitoring, Redis-backed rate limiting, Prisma baseline migrations, per-wallet rate limits, server-side kill switch, session secret rotation), authors the 6 root-level docs (TERMS_OF_SERVICE / PRIVACY / SECURITY_AUDIT / README / ARCHITECTURE / DEPLOYMENT), and completes the code-quality + Tailwind v4 compat work (console.* migration, Tile/StatTile dedup, exhaustive-deps suppression, Tailwind v4 config). Each feature is documented below in the same technical, honest tone as the v2.3 / v2.4 addenda. **None of this is financial advice.**

10.8.1 Trade Lifecycle Visualization (W10)

A new dashboard panel that visualizes the trade lifecycle in real-time — the user's #1 most-requested feature. The panel makes the previously-implicit buy → hold → sell → realize pipeline visible at a glance. It is mounted as a new top-of-dashboard CollapsibleSection so the user can fold it away when not needed.

- **Flow Diagram** — a horizontal 6-stage pipeline: ■ Snipe Detected → 🏴 Phantom Sign → ■ On-chain Buy → ■ Holding → ■ Auto-Sell → ■ P&L Realized. Each stage has a status dot: gray (idle), amber + pulse (active — a position is at this stage), emerald (complete today), red (error). The pulsing amber dot uses a CSS keyframe animation so the user's eye is drawn to where the bot is currently working.
- **Active Trades Timeline** — a scrollable list of open positions, each with a 6-dot mini-timeline showing which stages the position has passed + the last 5 log events (color-coded by level — same LEVEL_STYLES color map as the activity-log component). Works in both PAPER + LIVE modes with per-position PAPER/LIVE badges (the same badge pattern as the positions table).
- **SOL Flow Visualization** — 3 pools (Wallet / Deployed / Realized) connected by directional arrows. The Wallet→Deployed arrow lights amber when a buy is in flight; the Deployed→Wallet arrow lights emerald when a sell is in flight. The Realized pool accumulates the lifetime realized P&L. The panel reads from the existing useLiveTrading().solBalance + the engine's WalletStats (investedSol, totalPnlSol) — no new API surface.

10.8.2 Security Hardening (W13-W14)

The W13 + W14 waves closed the security findings from the W12 audit (and the post-fix W14 re-audit). Six surgical fixes + a dead-code sweep bring the platform to limited-public-deployment readiness.

- **Boot Guard** (src/instrumentation.ts) — a Next.js 16 instrumentation hook that runs at boot. Throws if OPERATOR_WALLETS / WALLET_SESSION_SECRET / INTERNAL_API_KEY are missing in

production. The server **refuses to start** without these env vars — closing the W12 finding that a misconfigured deploy would silently start in a permissive state.

- **Health Endpoint** (GET /api/health) — a liveness + readiness probe. Returns 200 + a per-check breakdown (database reachable, session-secret set) when healthy; 503 + a per-check error string when any check fails. Designed for Caddy / Kubernetes / load-balancer probes — the W12 audit had flagged the absence of a health check as a deploy blocker.
- **PumpPortal Exponential Backoff** — replaced the fixed 60s reconnect timer on the PumpPortal WebSocket client with exponential backoff (1s → 2s → 4s → 8s → 16s → 32s max, 20-attempt cap, 0-500ms jitter). Matches the existing Kraken / Binance WS reconnect pattern. A 5-second network blip no longer causes 60 seconds of synthetic-only price data — the client re-connects in ~1s + re-subscribes to all position mints.
- **CSRF Layer 2 Fix** — the How-To chatbot's POST to /api/how-to was using raw fetch() instead of apiFetch(), bypassing the X-CSRF-Token double-submit cookie (CSRF Layer 2). Layer 1 (Sec-Fetch-Site same-origin gate) still defended browser POSTs, but the Layer 2 bypass was a regression from the v2.1.0 CSRF hardening. Fixed to use apiFetch.
- **IP-Spoofing Fix** — 3 routes (wallet/balance, learning GET, learning POST) were reading x-forwarded-for directly instead of calling clientIp(request) (which gates on the TRUSTED_PROXY env var — only Caddy is a trusted proxy). A spoofed X-Forwarded-For header from a direct-connection attacker would have bypassed the per-IP rate limit. Fixed to use the shared helper.
- **P&L Leak Fix** — /api/logs GET was returning raw per-trade P&L amounts to any caller who knew the wallet address (the route was query-param wallet-scoped but not session-gated). An attacker who enumerated wallet addresses could have read the realized P&L of any operator. Fixed to 401 non-owners **before** the DB query — the session is verified first, then the wallet match is checked, then the query runs.
- **Dead Code Cleanup** — 37 dead source files (1,952 lines) + 29 unused shadcn UI components + 29 orphaned npm deps removed. The critical twin-file bug on use-sniper-notifications.tsx (same class as the W8.13 use-regime-poller.tsx fix) was resolved — the stale .tsx twin that silently shadowed the canonical .ts file (Next.js webpack's resolve.extensions order prefers .tsx over .ts) was deleted, restoring the pushPerm / setPushPerm props to the live-buy-failure push notification path.

10.8.3 Error Monitoring + Redis Rate Limiting (W15)

The W15 wave landed the two infrastructure pieces required for ≥2 server replicas — distributed error monitoring + a Redis-backed rate limiter that shares state across instances.

- **Sentry Error Monitoring** (@sentry/nextjs) — 3 auto-loaded config files (client, server, edge). Error boundaries (global-error.tsx + error.tsx) wrap with Sentry.captureException. Fully optional via the SENTRY_DSN env var — no-op when unset (no third-party network calls, no bundle bloat). Browser events are tunneled through /api/sentry-tunnel (same-origin proxy) to avoid third-party cookies from the sentry.io domain (GDPR / privacy hardening) + to keep the CSP connect-src allowlist tight (only same-origin).

- **Redis-Backed Rate Limiter** (`ioredis + rateLimitAsync()`) — an async Redis-backed version of the existing `rateLimit()` using atomic `MULTI INCR + EXPIRE`. Lazy-init via `getRedisClient()` — only connects when `USE_REDIS_RATELIMIT=1` is set. Falls back to the sync in-memory `rateLimit()` when Redis is down (graceful degradation — the rate limit becomes per-instance but the routes keep working). **Required for ≥ 2 server replicas** — otherwise an attacker could reset the rate limit by hitting different instances (each instance's in-memory counter starts at 0).

10.8.4 Prisma Migrations + Database Hardening (W15)

The W12 audit had flagged the `db:push --accept-data-loss` pattern as a wide-launch WARNING — a careless schema change in production would silently drop user data. The W15 wave replaces it with a proper Prisma migration baseline + a PostgreSQL migration path.

- **Baseline Migration** — `prisma/migrations/20261015000000_baseline/migration.sql` (176 lines, all 5 models [BotConfig, SnipedToken, BotLog, BotRiskEvent, BotJournal] + their indexes + the unique constraints). Production-safe `db:migrate` script (`prisma migrate deploy`) — applies pending migrations + never resets the DB. No more `db:push --accept-data-loss` in production.
- **PostgreSQL Migration Guide** — documented in `DEPLOYMENT.md`. The Prisma schema uses no SQLite-specific features (no `INTEGER PRIMARY KEY ROWID` tricks, no SQLite-only pragmas). The 6-step migration: (1) provision a PostgreSQL instance; (2) update `DATABASE_URL`; (3) change the schema provider from `sqlite` to `postgresql`; (4) `prisma migrate deploy`; (5) optional `pgloader` for the existing SQLite data; (6) restart the server. The schema diff is trivial — all column types map cleanly.
- **Per-Wallet Rate Limits** — 60 req/min/wallet on 4 polling routes (`/api/positions`, `/api/transactions`, `/api/logs`, `/api/wallet/stats`). Uses `rateLimitAsync()` (Redis-backed when `USE_REDIS_RATELIMIT=1`). The 60/min limit is 4× the polling rate (the dashboard polls every ~4s = 15 req/min) — legitimate single-tab users never hit it; scrapers + IDOR probers do. The rate limit is **per-wallet** (not per-IP) so a single NAT/VPN user running 8 tabs does not get throttled.

10.8.5 Server-Side Kill Switch (W15)

The kill switch was previously client-only — a `localStorage` key `sniper-bot-killed:${wallet}` that the engine's `useEffect` checked. A browser cache clear, a different device, or a server restart would all silently re-arm the bot. The W15 wave persists the kill switch server-side so it survives all three failure modes.

- Added a `killed Boolean @default(false)` column to `BotConfig` in the Prisma schema (the W15 baseline migration includes this column).
- `kill() / resetKill()` in `use-sniper-engine.ts` now POST to `/api/bot/toggle` with `{ killed: true/false }` (fire-and-forget via `apiFetch` — the UI does not block on the response).
- The engine's `useEffect` ORs `configQuery.data.killed` with the `localStorage` fast-path — either source being true treats the bot as killed. The `localStorage` path remains for instant UI feedback (no round-trip); the server path is the source of truth.

- Survives browser cache clears (the server row is the authority), syncs across devices / browsers (the same wallet logged in from two browsers sees the same kill state), and persists across server restarts (the DB row survives a redeploy).

10.8.6 Session Secret Rotation (W15)

The W12 audit had flagged the single-secret session model as a wide-launch WARNING — rotating WALLET_SESSION_SECRET required invalidating every active session simultaneously (every user would have to re-sign the Phantom challenge). The W15 wave adds graceful rotation via a previous-secret fallback.

- Added WALLET_SESSION_SECRET_PREVIOUS env var support in `src/lib/wallet-session.ts`.
- `getSessionSecrets()` returns both the current + previous secrets (filtered for ≥ 32 chars — a too-short secret is silently dropped to prevent the HMAC from being trivially forgeable).
- `verifySessionToken` short-circuits through both secrets (graceful rotation — tokens signed with the previous secret remain valid until the TTL expires).
- `createSessionToken` only signs with the current secret — so after the 24h TTL, unsetting `_PREVIOUS` invalidates all old tokens.
- **Rotation procedure:** (1) generate a new secret; (2) set `_PREVIOUS` to the old secret; (3) restart; (4) wait 24h (the TTL); (5) unset `_PREVIOUS`; (6) restart. Zero user visible disruption — no re-sign challenge required.

10.8.7 Documentation + Deployment Infrastructure (W13-W16)

The W13 + W15 waves claimed to author 6 root-level docs, but a `.gitignore *.md` rule silently dropped them (caught by the W16-audit-prod audit, re-created by the W16-fix-docs wave). The final set of 6 docs is now tracked + verified on disk. The deployment infrastructure (Caddyfile, backup script, env pre-flight, `.env.example`) ships alongside.

- **6 root-level docs authored:** `TERMS_OF_SERVICE.md` (10 sections — the legal framework for the operator + user relationship), `PRIVACY.md` (8 sections + 8 third-party data flows — the W14 audit deleted the fabricated Binance flow), `SECURITY_AUDIT.md` (54 verified controls across 9 sections — the W16 re-creation verified every file:line reference), `README.md`, `ARCHITECTURE.md`, `DEPLOYMENT.md` (12-step deploy checklist + 3-tier scaling recs [single-instance / small-fleet / production-HA] + systemd unit templates).
- **Caddyfile TLS template** — `yourdomain.com` placeholder with auto-Let's-Encrypt. The prior `:81` HTTP config silently broke session cookies — browsers reject `secure: cookies` over HTTP, so the `httpOnly` session cookie was never set + every authenticated request 401'd with no visible error.
- **DB backup script** (`scripts/backup-db.sh`) — hourly `sqlite3 .backup` (online backup — safe while the DB is in use, no reader-writer lock contention) + `gzip` + 24h prune (older backups are deleted). Cron install documented in `DEPLOYMENT.md`.
- **Env pre-flight check** (`scripts/check-env.sh`) — exits 1 in production if `OPERATOR_WALLETS` / `WALLET_SESSION_SECRET` / `INTERNAL_API_KEY` are missing. Wired into the `start` script — `bun run start` runs the pre-flight before `bun .next/standalone/server.js`. The boot guard

(10.8.2) is the runtime backstop; this is the deploy-time gate.

- `.env.example` — comprehensive categorized documentation (3 critical [OPERATOR_WALLETS, WALLET_SESSION_SECRET, INTERNAL_API_KEY] + 5 required [DATABASE_URL, HELIUS_RPC_URL, NEXT_PUBLIC_HELIUS_RPC_URL, WALLET_SESSION_TTL_HOURS, TRUSTED_PROXY] + optional: Sentry, Redis, SMTP, Telegram, Helius, ZAI, cron). Each var has a one-line description + the default + the failure mode if unset.

10.8.8 Code Quality + Tailwind v4 Compat (W15-W16)

The W15 + W16 waves closed the last ■ LOW polish items from the W12 / W14 audits. The `console.*` migration is now complete (down from 113 pre-W3 call sites), the `Tile/StatTile` duplication is extracted into a shared component, the `page.tsx` exhaustive-deps warnings are suppressed with documented explanations, the Tailwind config is v4-compatible, the Sentry tunnel is wired, and the 4 polling routes are migrated to the Redis-backed rate limiter.

- **Console.* migration complete** — 0 unmigrated `console.log/error/warn` call sites remain in `src/` (down from 113 pre-W3). The only remaining `console.*` calls are in `logger.ts` (the logger's own transport I/O — intentional), `instrumentation.ts` (runs at boot before the logger is ready), and `mini-services/` (not browser-bundled — runs in Node, not the Next.js client bundle).
- **Tile/StatTile dedup** — extracted shared `src/components/ui/stat-tile.tsx` with a variant: `'card' | 'compact'` prop. Deleted the local `Tile` function from `portfolio-dashboard.tsx` + the local `StatTile` from `simulation-studio-panel.tsx`. Single source of truth, consistent styling across both panels.
- **page.tsx exhaustive-deps** — 3 `react-hooks/exhaustive-deps` warnings suppressed with `eslint-disable` comments + explanations. The `autoTuneState` ref-based pattern would cause render loops if added to the deps array (the ref mutation does not trigger a re-render, but adding it to deps would). The `eslint-disable` is the documented intentional pattern.
- **Tailwind v4 compat** — removed the dead `content` array from `tailwind.config.ts` (Tailwind v4 auto-detects from `globals.css`'s `@import "tailwindcss"`). Added a 20-line header comment explaining what's kept vs removed + why. The `darkMode: "class"` (for next-themes), `theme.extend.colors` (for shadcn/ui), + `theme.extend.borderRadius` (for shadcn/ui) are all still needed — only the `content` array was dead.
- **Sentry tunnel** — `tunnel: "/api/sentry-tunnel"` configured in `sentry.client.config.ts`. The tunnel route at `src/app/api/sentry-tunnel/route.ts` proxies browser Sentry events to `sentry.io` server-side. Avoids third-party cookies from the `sentry.io` domain (GDPR / privacy hardening) + keeps the CSP `connect-src` allowlist tight (only `same-origin` — no `*.sentry.io` exception).
- **rateLimitAsync migration** — the 4 polling routes (`/api/positions`, `/api/transactions`, `/api/logs`, `/api/wallet/stats`) migrated from `sync rateLimit()` to `await rateLimitAsync()`. When `USE_REDIS_RATELIMIT=1` is set, these routes benefit from Redis-backed multi-instance rate limiting. When the flag is unset, they fall back to sync in-memory (backward-compatible — existing single-instance deploys work unchanged).

Verification summary for the v2.5 wave. All 8 feature groups above were verified via the standard gate: lint (0 NEW errors), tsc --noEmit (0 NEW errors), build (exit 0 — site deployable). The v2.5 wave is additive (new features) + non-destructive (no behavioral change to existing v2.3 / v2.4 paths). The 0.5

SOL buyAmount hard cap, the Kelly criterion buy-path gating, the forward-looking rug detection (R6/R7/R8), the Jito MEV tip, the manual-snipe popup, the 50%-from-high safety net, the Close All panic button, the auto-close push notifications, the MFE/MAE analytics, the P&L attribution panel, the recovery projection tile, and the config what-if preview from v2.2 / v2.3 / v2.4 all remain in force. The v2.5 wave's trade lifecycle panel, security hardening, wide-launch infrastructure, documentation, and code-quality work are all UX / observability / deployability improvements — they do NOT change the underlying asset risk.

This is not financial advice. SniperBot is research software. The v2.5 wave's trade lifecycle visualization, security hardening, Sentry error monitoring, Redis rate limiting, Prisma baseline migrations, per-wallet rate limits, server-side kill switch, session secret rotation, and documentation are platform-maturation + deployability improvements — they do NOT change the underlying asset risk. Meme-coin launches remain structurally negative-EV against the base rate; blue-chip swing trading remains exposed to 50%+ single-day drawdowns. Never trade with funds you cannot afford to lose entirely. Past simulation performance does not predict future results.

Not financial advice.

10.9 v2.6 Addendum — Subscription Paywall (W18) + Disclaimer/ToS Acceptance (W19)

The v2.6.0 release is the fourth platform maturation round, shipped across two development waves (Waves 18-19). It introduces the commercial layer (a 3-tier subscription paywall with on-chain SOL payments verified via Helius) + the legal/UX safety layer (a post-purchase disclaimer page where the user must scroll through the full Terms of Service + explicitly accept before LIVE trading is enabled). Together these waves gate LIVE trading behind both a payment proof AND a documented risk acceptance — closing the W17 gap where any operator-wallet-listed user could LIVE-trade with no commercial relationship + no acknowledgement of the disclosed risks. Each feature is documented below in the same technical, honest tone as the v2.3 / v2.4 / v2.5 addenda. **None of this is financial advice.**

10.9.1 Subscription Paywall (W18)

The platform implements a 3-tier subscription model with on-chain SOL payments verified via Helius. Users pay SOL from their Phantom wallet to the operator wallet; the server verifies the transfer on-chain before activating the subscription.

Tier	Monthly	Yearly (10% off)	Max Positions	Key Features
Free	0 SOL	0 SOL	5	Paper mode only, basic risk scoring, DCA benchmark
Lite	0.25 SOL	2.70 SOL	10	+LIVE mode, Bayesian auto-tune, Kelly criterion, tiered TP, trailing stop, push notifications
Pro	0.50 SOL	5.40 SOL	Unlimited	+Email reports, simulation studio, P&L attribution, config what-if, trade lifecycle, Telegram, tax export

How it works

- The user visits /pricing (accessible via the "Pricing" button with a Crown icon in the header).
- They connect their Phantom wallet + select a tier + billing period (monthly or yearly).
- The pricing page constructs a `SystemProgram.transfer` instruction (SOL from the user's wallet to the operator wallet) using `@solana/web3.js`.
- Phantom's `signAndSendTransaction` signs + broadcasts the transfer to Solana mainnet.
- The page POSTs `{ wallet, tier, billingPeriod, txSignature }` to `/api/subscription`.
- The server verifies the on-chain transfer via `verifySubscriptionPaymentTx` (6 gates: tx confirmed, no error, fee-payer=subscriber, recipient=operator wallet, recipient balance increased by expected amount, sender balance decreased).

- On success: the server upserts a Subscription row with tier, billingPeriod, amountSol, expiresAt (now + 30 days for monthly, now + 365 days for yearly), + the verified txSignature.
- The subscription automatically reverts to free tier when expiresAt < now (checked by getSubscriptionStatus() on every API call).

Feature gating

- Enforced server-side in /api/snipe: free tier gets 402 "Payment Required" on LIVE snipe attempts + 402 on max-positions-exceeded.
- The getSubscriptionStatus(wallet) helper queries the DB for the active subscription.
- The hasFeature(status, 'live_mode') check gates LIVE mode.
- The getMaxPositions(status) check gates the position count (free=5, lite=10, pro=∞).

Security

- **7 layered defenses** on the POST route — Sec-Fetch-Site CSRF, double-submit cookie CSRF, per-IP rate limit (20/min), per-wallet rate limit (5/min), optionalAuth session-vs-body match, txSignature uniqueness (409 on replay), Helius 6-gate on-chain verification. Fail-closed throughout.

10.9.2 Disclaimer + Terms of Service Acceptance (W19)

After purchasing a Lite or Pro subscription, the user is redirected to /disclaimer where they must read through the full Terms of Service + accept before LIVE trading is enabled.

How it works

- The pricing page's subscribe mutation onSuccess redirects to /disclaimer after a 1.5s delay (so the success toast is visible).
- A **critical notice** banner (red border) appears at the top of the page: "This is a self-hosted trading tool — software product and research tool, NOT a financial service. We make zero promises of a return. It is possible to lose all your money and we take no responsibility."
- A **simulation recommendation** banner (amber border): "We STRONGLY recommend you spend at minimum of ONE WEEK trading with the Simulation Studio in PAPER mode before attempting LIVE trading."
- A **scrollable ToS** (10 sections, max-height 400px with overflow-y-auto). The Accept button is DISABLED until the user scrolls to the bottom (tracked via an onScroll handler).
- A **checkbox** "I have read and understand the Terms of Service" (disabled until scrolled to bottom).
- An **Accept button** "I Accept — Enable LIVE Trading" (disabled until both scrolled + checked).

Backend

- DisclaimerAcceptance Prisma model: walletAddress (unique), tosVersion ("1.0"), acceptedFromIp, acceptedAt.
- POST /api/disclaimer creates/upserts the acceptance row with CSRF Layer 1+2, optionalAuth session-vs-body match, rate-limited 5/min/IP, IP recorded for audit trail.
- The acceptance is wallet-scoped (unique constraint) — one acceptance per wallet.

LIVE trading blocked until accepted

- `/api/snipe` checks `db.disclaimerAcceptance.findUnique({ where: { walletAddress: wallet } })` when `body.live === true`.
- If no row exists: returns 403 "You must accept the Terms of Service before LIVE trading. Visit `/disclaimer` to read + accept."
- Paper mode is always allowed — no disclaimer acceptance needed.

The W18 + W19 waves close the commercial + legal safety loop: LIVE trading now requires both (a) a paid, on-chain-verified subscription that has not expired AND (b) an explicit, IP-stamped acceptance of the self-hosted-tool Terms of Service. Paper mode remains free + open — a user can still research, backtest, and simulate indefinitely on the free tier without payment or acceptance. The platform-maturation improvements documented here (the paywall + the ToS gate) are risk-disclosure + access-control improvements — they do NOT change the underlying asset risk. Meme-coin launches remain structurally negative-EV against the base rate; blue-chip swing trading remains exposed to 50%+ single-day drawdowns. Never trade with funds you cannot afford to lose entirely. Past simulation performance does not predict future results.

Not financial advice.

11. Conclusion

SniperBot v2.6.0 is a technically serious, honestly-documented research and trading platform. Its strengths — real-data integration, a transparent and decomposable risk engine, autonomous self-tuning, a comprehensive simulation environment, meaningful security hardening, **and now (v2.2.0) Kelly criterion buy-path gating, forward-looking rug detection (R6/R7/R8), functional Jito MEV protection, Blue Chip as the default preset with optimal settings, a DCA benchmark for honest accountability, and the full CSRF + rate-limit + idempotency defense-in-depth stack** — are real and uncommon in its niche. Its remaining weaknesses — Jupiter free-tier API dependence, no private mempool / stake-relayer relationship, no hardware-wallet support, the SQLite scaling ceiling, and the still-negative-EV meme-coin candidate population (now gated by Kelly — the bot won't auto-trade into negative-EV propositions) — are also real and, crucially, are disclosed here rather than concealed.

The v2.0.0 release landed two of the v1.0.0 roadmap's highest-value items: Jito bundle integration (E1) materially remediates the public-mempool MEV exposure, and real on-chain P&L computation eliminates the tier-accounting lies. The major strategic addition is the blue-chip pivot: seventeen verified Solana mint addresses, the BlueChipScanner bot, the Kraken polling loop, the Blue Chip preset, and full round-trip live trading for established coins — including on-chain P&L computed from the Jupiter swap response's outAmount. The v2.1.0 release documented here is the security + auth + ML hardening round: the v2.0.0 format-only wallet-auth gap is closed (signed-message challenge→sign→verify with HMAC-signed httpOnly session cookies), the previously-decorative BayesianTuner + PresetBandit ML subsystem is now functional (observe + update fire from analyzeAndTune; posterior converges past cycle 1; 5-arm bandit covers all presets), server-side swap signature verification (verifySwapTx) cryptographically binds isLive to a verified on-chain swap, and the full CSRF + rate-limit + idempotency + PII-leak-fix defense-in-depth stack is in place.

The remaining opportunities on the roadmap — an ML rug classifier (R2), real whale tracking (D3), X/Twitter sentiment (D5), Telegram call channels (D4), and adaptive Kelly position sizing — apply primarily to the meme-coin path; the blue-chip path's limiting factor is swing-timing skill, not classification precision. A paid-tier Jupiter key and a stake-relayer relationship for inclusion guarantees are the obvious next execution-path improvements.

Readiness Assessment

For the **simulation, research, and paper-trading** use case, SniperBot is production-ready across both asset populations. The risk engine, enrichment pipeline, Simulation Studio, and analytics are complete, coherent, and genuinely useful for understanding both the Pump.fun market and the established-coin swing-trading opportunity. A user who restricts themselves to simulation mode can derive real educational and research value with negligible risk.

For **live trading of blue-chip tokens with real SOL**, SniperBot is now (v2.1.0) genuinely production-ready on the LIVE trade path. The v2.0.0 auth-model gap (format-only wallet verification — any caller who knew a victim's address could mutate their bot) is closed via signed-message challenge→sign→verify with HMAC-signed httpOnly session cookies. Server-side swap signature

verification (`verifySwapTx`) cryptographically binds `isLive` to a verified on-chain swap before the DB row is created. Real on-chain P&L is computed from the verified `outAmount` (in lamports) — the client cannot fabricate P&L for LIVE positions. All sensitive routes (`bot/config`, `bot/toggle`, `positions/close LIVE`, `positions/journal`, `logs`, `research/analyze`, `alerts`, `reports/send`, `backtest/run`) require a signed session. The risk profile is materially more favorable than the meme-coin path: zero rug risk by construction, deep liquidity, real price discovery, Jito bundle MEV protection, and real on-chain P&L accounting. This is the v2.1.0 path the platform now recommends as the default for users who want to trade live. It is not zero risk — BTC, ETH, and SOL draw down 50%+ in single days — but it is a categorically different risk from the meme-coin path, and the user should size positions accordingly.

For **live trading of fresh meme-coin launches with real SOL**, SniperBot is operationally functional and now (v2.1.0) has the same auth + verification + P&L guarantees as the blue-chip path. The ML subsystem (BayesianTuner + 5-arm PresetBandit + learning-engine sanity check) is now functional — the bot actually learns from its closed trades and adapts `minSafetyScore` / preset selection based on observed regime. However, the underlying negative-EV weakness (5.3) and residual MEV exposure on the Jupiter free-tier (5.1) remain; users should still expect MEV drag on every swap and a structurally negative expected value against the base rate. The single highest-impact remaining improvement for this path is an ML rug classifier (R2) to raise candidate EV above zero; until it lands, live meme-coin users should use the platform only with capital they are prepared to lose entirely.

The honest, one-sentence recommendation: **use SniperBot with the Blue Chip preset (now the default) with 1 SOL starting capital; the v2.2.0 Kelly gating, forward-looking rug detection, Jito MEV tip, and optimal settings (0.1 SOL buyAmount, 70 minSafetyScore, 10% SL, 100% TP) collectively project +\$2.88/trade EV (+10% to +30% 1-year ROI on 1 SOL in a range-bound market); the DCA benchmark tile will honestly tell you whether you are beating buy-and-hold — if not, switch to DCA.**

This document is provided for educational and informational purposes only and does not constitute financial, investment, legal, or tax advice. SniperBot is research software. Crypto-asset trading involves extreme risk of total capital loss. Past simulation performance does not predict future results. Never trade with funds you cannot afford to lose entirely.

SniperBot Whitepaper v2.6.0 — October 16, 2026. Document end.