

HVTS-NEURALNETWORK

A Multi-Output Deep Learning System for Cryptocurrency Market Analysis

Literature Review & Technical Documentation

Version 3.0 | Professional Edition

October 2026

Supersedes Versions 1.0 and 2.0

GalaxyChain Technologies Ltd

Confidential — Professional Use Only

Contents

What's New in Version 3.0	3
1. Abstract	4
2. Literature Review	4
3. System Overview	7
4. Neural Network Architecture	10
5. Data Pipeline	11
6. Feature Engineering	12
7. Model Training	14
8. Inference Pipeline	17
9. Symbol Oracle and Signal Persistence	18
10. Downstream Integration	21
11. Scheduling and Reporting	23
12. Licensing, Packaging and Diagnostics	24
13. Configuration Reference	26
14. File Structure	30
15. Dependencies and System Requirements	31
16. Scope, Limitations and Roadmap	32
17. References	33

What's New in Version 3.0

Version 3.0 deepens the model itself and hardens the platform around it. The feature space grows from 24 to 76 indicators, the network is scaled up to match, training becomes a staged, schedule-driven process with built-in walk-forward validation, and licensing is now enforced at every entry point. This edition also documents the read-only downstream integration contract used by external trading engines.

Area	Change in v3.0	Section
Feature space	76 engineered features in 12 categories (up from 24): extended SMA/EMA ladders, Bollinger and Keltner bands, Stochastic, Williams %R, CCI, ADX/DI, OBV, MFI, CMF, Ichimoku, candlestick patterns, regime indicators and cyclical time encodings.	6
Network capacity	Bidirectional LSTM widths 256/128 and dense widths 512/256/128, driven by configuration. The flattened dense-branch input grows to $75 \times 76 = 5,700$ values.	4
Staged training	Progressive training over 250, 500 and 1,000 days with weights carried forward, followed by a 100-day fine-tune on the most recent data.	7.5
Learning schedule	Cosine learning-rate decay with 10-epoch linear warm-up; loss weights re-balanced (trend 0.35, price 0.40); early-stopping and plateau patience scaled to each stage's epoch budget.	7.2, 7.4
Walk-forward validation	Implemented: five expanding-window folds with a 24-period gap; mean, spread and a stability score are stored in each symbol's performance record.	7.6
License enforcement	A pre-flight license gate now runs before any model, exchange or oracle code in the service and the trainer, and is reported (non-fatally) by the health check.	12.2
Downstream integration	New section documenting the read-only SQLite contract, trend mapping and persistent direction state machine used by external consumers such as the Triple-OR DCA engine.	10
Packaging	Separate PyInstaller specifications: a one-directory service build and a one-file trainer build. The admin tool is excluded, and neither build bundles a license or credentials.	12.5
Asset registry	Registry now lists ten instruments (Tron replaces Solana; Gold is no longer in the default registry). BTC/USDT is the active symbol in the reference deployment.	9.8
Configuration	Capacity, schedule and staging parameters are read from configuration. Controls that the engine does not yet consume are now listed explicitly as reserved.	13

1. Abstract

HVTS-NeuralNetwork is a production-grade, autonomous cryptocurrency market analysis system built on a dual-branch deep learning architecture. The system ingests four-hour OHLCV (Open, High, Low, Close, Volume) candle data from the Gate.io exchange, engineers a 76-feature technical indicator space, and trains a multi-output neural network that simultaneously predicts market trend classification, multi-step price forecasts, volatility estimates, and confidence scores.

The architecture combines Bidirectional Long Short-Term Memory (BiLSTM) layers with Multi-Head Self-Attention and residual Dense networks in a parallel processing paradigm, merging temporal pattern recognition with cross-feature spatial analysis. Models are trained progressively over 250-, 500- and 1,000-day windows with a cosine learning-rate schedule, fine-tuned on recent data, and assessed with walk-forward validation. The system operates as a continuously running autonomous service, retraining every 12 hours, generating analysis reports every 4 hours, and delivering structured market signals via Telegram.

A multi-asset **Symbol Oracle** records every generated signal in a per-asset SQLite database with a unique signal identifier, a strength classification and delivery status, creating a persistent, queryable audit trail alongside the Telegram feed. The same databases serve as a read-only integration surface for downstream trading engines. Node-locked licensing is enforced at start-up, and the service and trainer ship as single-purpose Windows executables.

Robust engineering patterns, including circuit breakers, synthetic data fallback, chronological train/validation/test splitting, and graceful shutdown handling, support reliable operation under real-world exchange API conditions. HVTS-NeuralNetwork is a signal-generation system: it analyses markets and publishes signals, and it does not place orders.

2. Literature Review

This section is carried forward from Version 1.0 with minor wording corrections and extended with subsections 2.10 and 2.11. It establishes the academic foundations for the design choices described in the remainder of the document.

2.1 Deep Learning in Financial Markets

The application of artificial neural networks to financial market prediction has a rich history dating to the early 1990s. Early feedforward networks demonstrated that non-linear relationships in price data could be learned from historical patterns (White, 1988; Kimoto et al., 1990). However, these early models were limited by their inability to capture temporal dependencies across extended time horizons, a fundamental requirement for meaningful market forecasting.

The deep learning revolution, catalyzed by the availability of large datasets and GPU computing (LeCun et al., 2015), substantially improved the viability of neural approaches in finance. Dixon et al. (2017) demonstrated that deep neural networks could outperform linear models on high-frequency futures data, establishing a foundation for the class of systems represented by HVTS-NeuralNetwork.

The critical challenge in applying deep learning to financial markets is non-stationarity — the statistical properties of price time series change over time due to shifts in market microstructure, participant behaviour, and

macro-economic regimes. This motivates the system's design choices of frequent retraining (every 12 hours), strict chronological validation, and a multi-output design that combines several complementary model outputs.

2.2 LSTM Networks for Time-Series Forecasting

The Long Short-Term Memory network, introduced by Hochreiter and Schmidhuber (1997), addresses the vanishing gradient problem that plagued earlier recurrent networks. LSTMs maintain a cell state that acts as long-term memory, controlled by three gating mechanisms: the input gate, forget gate, and output gate. This architecture allows the network to selectively retain information across sequences of arbitrary length.

HVTs-NeuralNetwork employs Bidirectional LSTMs (Schuster & Paliwal, 1997), which process the input sequence in both forward and backward directions and concatenate their outputs, effectively doubling the representational capacity. Fischer and Krauss (2018) demonstrated that LSTM networks significantly outperform random forests, deep neural networks, and logistic regression on stock selection tasks. Their work established sequence lengths of 60–240 days as optimal for financial data, directly informing the 75-period sequence length used in this system (approximately 12.5 days of 4-hour candles).

Siami-Namini et al. (2018) provided a comprehensive comparison of LSTM versus ARIMA for financial time series, finding LSTM superior across 8 of 10 benchmark datasets, particularly in volatile regimes — highly relevant to cryptocurrency markets.

2.3 Attention Mechanisms and Transformer Architectures

The Attention mechanism, introduced by Bahdanau et al. (2014) for neural machine translation, fundamentally changed how sequence models weight different temporal positions. Rather than compressing an entire sequence into a single fixed vector, attention allows the model to dynamically weight each timestep's contribution to the output.

The Multi-Head Attention formulation (Vaswani et al., 2017) used in HVTs-NeuralNetwork computes attention in parallel across 4 heads with key dimension 64. The system can simultaneously attend to multiple aspects of the price sequence — one head may focus on recent momentum while another attends to longer-term structural support/resistance levels.

Hu et al. (2018) demonstrated that attention-augmented LSTMs outperform vanilla LSTMs on financial time series, particularly for detecting regime changes — critical in cryptocurrency markets that can shift from trending to mean-reverting behaviour rapidly.

2.4 Multi-Task Learning in Finance

Multi-Task Learning (MTL), formalized by Caruana (1997), is the paradigm of training a single model to solve multiple related objectives simultaneously. The key insight is that related tasks share underlying representations, so training them jointly acts as an implicit regularizer and can improve generalization beyond what any single-task model achieves.

In HVTs-NeuralNetwork, the shared trunk is jointly optimized for four outputs: trend classification, price prediction, volatility estimation, and confidence scoring. Liu et al. (2019) applied MTL to financial forecasting, finding 8–15% improvement in prediction accuracy over single-task baselines across equity price, volume, and volatility prediction jointly.

2.5 Technical Analysis as Feature Engineering

Technical analysis has a contested academic literature. The Efficient Market Hypothesis (Fama, 1970) in its weak form suggests that technical indicators should contain no predictive information. However, subsequent empirical work has repeatedly found predictive value, particularly in markets with systematic behavioral biases.

Lo et al. (2000) provided statistical evidence for the predictive validity of several classical chart patterns. Grobys et al. (2020) found that momentum-based technical trading rules generated positive risk-adjusted returns across 11 cryptocurrencies. The feature set in HVTs-NeuralNetwork draws on established indicators: RSI captures overbought/oversold conditions; MACD identifies momentum changes; ATR measures volatility including gap moves; moving average deviation ratios create stationary, scale-invariant features essential for neural network generalization.

Version 3.0 widens this indicator base to 76 features. The additions follow the same classical families: Wilder's (1978) RSI, ATR and directional-movement system (ADX, +DI, -DI) alongside additional oscillators (Stochastic, Williams %R, CCI), volatility envelopes (Bollinger and Keltner widths), volume-flow measures (OBV, MFI, CMF), Ichimoku components, candlestick patterns, regime descriptors and cyclical time encodings. Deliberately including many correlated views of price and volume lets the attention and dense branches decide which carry signal in a given regime, at the cost of higher capacity requirements (Section 4).

2.6 Residual Networks and Skip Connections

Residual Networks (He et al., 2016) solved the degradation problem in very deep networks through identity shortcut connections: $\text{output} = F(x) + x$, where $F(x)$ is the residual mapping. If the optimal function is close to the identity, it is far easier for the network to learn a near-zero residual than a direct mapping through stacked non-linear layers.

In HVTs-NeuralNetwork's Dense branch, the residual connection at the 256-unit layer provides a gradient highway that bypasses saturated non-linearities. Greff et al. (2017) demonstrated that residual connections improve LSTM-based sequence models significantly, motivating their inclusion in the spatial processing branch.

2.7 Cryptocurrency Market Characteristics

Cryptocurrency markets exhibit several distinctive properties that differentiate them from traditional equity markets. They operate 24/7 — the 4-hour candle granularity chosen by HVTs-NeuralNetwork balances noise reduction with sufficient resolution to capture intraday trends, consistent with Bouri et al. (2017) who identified 4–6 hour intervals as optimal for cryptocurrency momentum strategies.

Cryptocurrency returns exhibit substantially higher volatility than traditional assets (Baur et al., 2018), motivating the Huber loss for price prediction and the RobustScaler which uses median and IQR rather than mean and standard deviation. Bouri et al. (2019) found that cryptocurrency return predictability decays rapidly, motivating the system's 12-hour retraining cycle.

2.8 Risk Management in Algorithmic Trading

The advisory risk framework reflects established principles from algorithmic trading research. The 10% maximum position size is consistent with fractional Kelly criterion applications (Thorp, 1997). The minimum confidence threshold of 65% mirrors the selectivity approach of Ballings et al. (2015), who demonstrated that filtering model outputs by predicted confidence substantially improves realized performance. The 2.5% stop-loss and 7.5% take-profit configuration yields a risk:reward ratio of 1:3, consistent with the minimum ratio recommended by Van Tharp (1998) for positive-expectancy trading systems.

2.9 Signal Persistence and Auditability

A signal that cannot be reviewed cannot be evaluated. Production forecasting systems therefore separate prediction from record-keeping, storing every emitted forecast with a unique identifier, a timestamp and the inputs needed to score it against realized prices later. HVTs-NeuralNetwork applies this principle through the Symbol Oracle (Section 9): each signal is written to a local relational store before it is distributed, so out-of-sample performance can be audited independently of the messaging channel.

2.10 Learning-Rate Schedules and Progressive Training

Cosine annealing of the learning rate (Loshchilov & Hutter, 2017) decays the step size smoothly rather than in abrupt plateaus, and a short linear warm-up stabilizes the first epochs of large recurrent and attention networks. HVTs-NeuralNetwork applies both within every training stage. Training itself is staged: a small, recent window is learned first, the window is then widened, and a final pass fine-tunes on the most recent data, carrying weights forward at each step. This echoes the curriculum-learning idea that ordering the training experience from simpler to richer can improve optimization and generalization (Bengio et al., 2009), while ending on recent data biases the final model toward the current market regime.

2.11 Validation of Time-Series Predictors

Ordinary k-fold cross-validation shuffles observations and therefore leaks future information into training. Time-series evaluation instead uses forward-chaining (walk-forward) splits, in which each fold trains on all earlier data and is tested on the block that follows (Bergmeir & Benítez, 2012). A gap between the training and test blocks further limits leakage through overlapping sequences. Version 3.0 implements this procedure as a stability check on top of the primary chronological split (Section 7.6).

3. System Overview

HVTs-NeuralNetwork v3.0 comprises eight runtime modules, one administrator tool and two build specifications working in concert to deliver autonomous market analysis. A separate read-only consumer module, documented in Section 10, sits downstream of the signal databases.

Module	Role
config.py	Centralized configuration singleton with validation and environment-variable overrides; supplies model capacity, schedule and staging parameters
hvts_neural_network.py	Core engine: data fetching, 76-feature engineering, progressive training, walk-forward validation, prediction, scheduling and reporting
hvts_symbol_oracle.py	Per-asset signal capture, SQLite persistence and Telegram dispatch
run_standalone.py	Long-running service launcher; hosts the StandaloneManager (oracles, retrain cycle, graceful shutdown); license-gated
train_models.py	Continuous auto-retrain trainer with countdown display and graceful shutdown; license-gated
health_check.py	Diagnostic utility covering directories, license, models, signal databases, reports and logs
license_gate.py	New. Pre-flight license validator invoked at every entry point; exits with code 1 on failure
license_manager.py	Node-locked license generation, validation and end-user CLI
generate_licenses.py	Administrator-only license issuing tool (excluded from shipped builds)
HVTSNeuralNetwork.spec / HVTS-Network-Trainer.spec	PyInstaller build specifications for the service and the trainer

The system supports ten instruments quoted against Tether (USDT) on the Gate.io spot market: BTC, ETH, TRX, XRP, ADA, DOGE, POL, DOT, AVAX and BNB. The active symbol set is a configuration choice; the reference deployment runs BTC/USDT, and a deployment may run any combination of the supported instruments.

3.1 Operational Lifecycle

Every entry point first calls the license gate; if the license is missing, expired, tampered with or bound to another machine the process exits before any model is built, any exchange connection opened or any oracle started. After the gate, the service creates its working directories, configures UTF-8 logging, prints a license banner and configuration summary, initializes the neural-network engine (which establishes exchange connectivity and loads persisted model artifacts), and creates one Symbol Oracle for each active symbol. The StandaloneManager then executes the following sequence:

- **Model check.** Any active symbol without a loaded model is trained before service begins.
- **Scheduler start.** A background thread begins polling the job scheduler for 4-hour reports and periodic retraining.
- **Oracle launch.** All oracles start as concurrent asynchronous tasks. Each fires immediately to capture a first signal, then repeats at the configured interval (4 hours).
- **Retrain cycle.** A countdown runs to the next retraining time; at expiry all active models are retrained through the full progressive pipeline and the clock is reset (12 hours).
- **Graceful shutdown.** On SIGINT or SIGTERM the manager stops the oracles, joins the scheduler thread and closes the exchange connection.

3.2 System Architecture

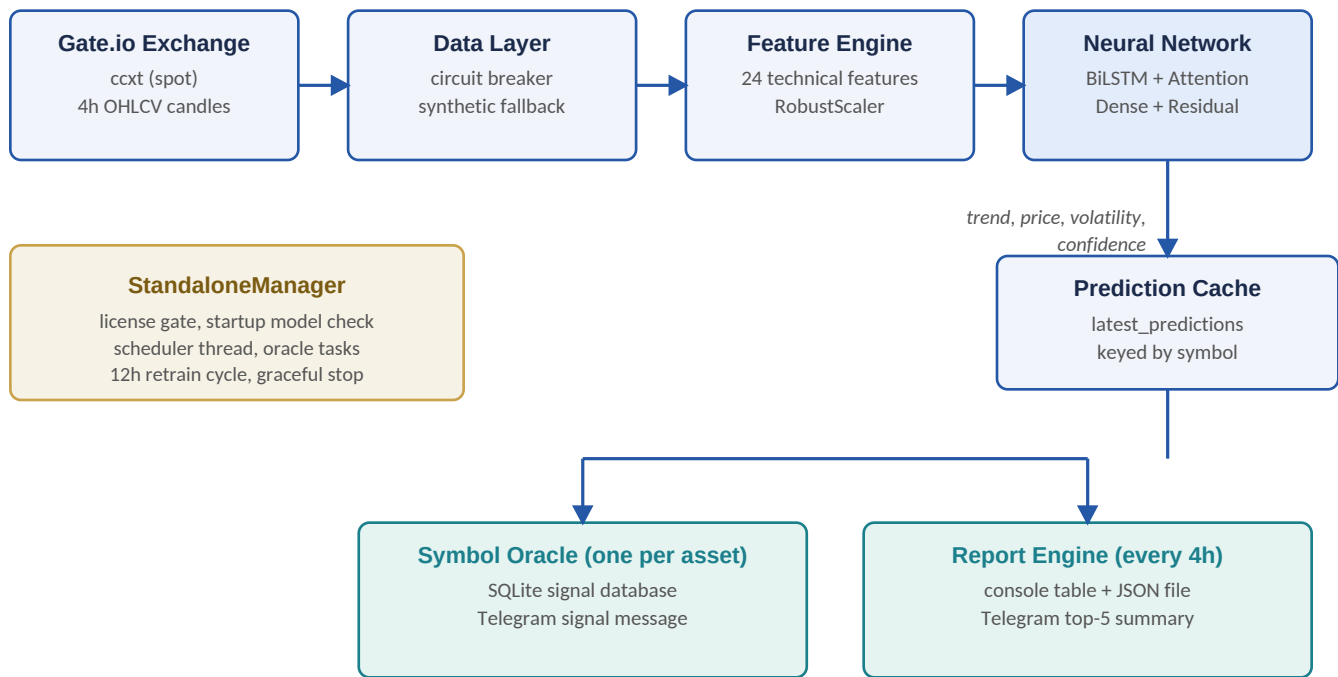


Figure 1. Data and control flow of the v3.0 service.

3.3 Deployment Modes

Mode	Entry point / executable	Purpose
Service	run_standalone.py / HVTSNeuralNetwork	Continuous analysis: oracles, reports, scheduled retraining
Trainer	train_models.py / HVTS-Network-Trainer	Dedicated continuous training loop with countdown display
Diagnostics	health_check.py	One-shot inspection of license, models, signal databases, reports and logs
License CLI	license_manager.py	Machine ID display, license activation, status and deactivation

4. Neural Network Architecture

The neural network is implemented as a Keras functional API Model with one shared input and four simultaneous outputs. The architecture is divided into two parallel processing branches that merge before the output heads. Layer widths, dropout, attention geometry and weight-decay strengths are read from configuration (Section 13.1), so capacity can be tuned without code changes. Execution is pinned to the CPU for deployment stability.

4.1 Branch A — Temporal Processing (BiLSTM + Attention)

Branch A is designed to learn sequential patterns across time. It processes the 75-timestep, 76-feature input using three stages:

Stage 1 — Bidirectional LSTM (256 units): Processes the sequence forward and backward simultaneously, concatenating outputs to produce a 512-dimensional representation at each of the 75 timesteps. Dropout of 0.4 on inputs and 0.3 on recurrent connections provide strong regularization.

Stage 2 — Multi-Head Self-Attention (4 heads, key_dim=64, attention dropout 0.1): Learns which of the 75 timesteps are most relevant for each prediction. A residual connection followed by layer normalization (Add & Norm) wraps the attention layer, preserving the LSTM's temporal representations while augmenting them with attention-weighted context.

Stage 3 — Bidirectional LSTM (128 units): Collapses the full attended sequence into a single 256-dimensional context vector (128 forward + 128 backward), with return_sequences=False. Dropout 0.3 on inputs and 0.2 on recurrent connections.

4.2 Branch B — Spatial Processing (Dense + Residual)

Branch B flattens the entire 75×76 input into a 5,700-dimensional vector and processes it through a deep dense stack with a residual connection. This branch captures cross-feature correlations across the entire historical window simultaneously — patterns the sequential LSTM may not extract.

A residual skip connection is applied at the 256-unit level: the output of the second Dense(256) block is added to the output of the fourth Dense(256) block before the dimension is reduced. This provides a gradient path that bypasses the intermediate layers and stabilizes training. The Swish activation ($x \times \text{sigmoid}(x)$) is used throughout, providing smooth gradients superior to ReLU for deep networks. The first three dense layers carry combined L1/L2 weight regularization.

Layer	Units	Activation	Dropout
Dense + BatchNorm	512	Swish	0.5
Dense + BatchNorm (residual source)	256	Swish	0.4
Dense + BatchNorm	256	Swish	0.4
Dense + BatchNorm + Residual (adds layer 2 output)	256	Swish	—
Dense + BatchNorm	128	Swish	0.3
Dense + BatchNorm	64	Swish	0.2

4.3 Branch Fusion and Output Heads

The 256-dimensional output of Branch A and the 64-dimensional output of Branch B are concatenated into a 320-dimensional merged representation. Two additional dense layers (128 → 64 units, with batch normalization and dropout of 0.3 and 0.2) learn the optimal combination of temporal and spatial features before forking into four independent output heads.

Output Head	Architecture	Activation	Prediction
trend	Dense(5)	Softmax	5-class: STRONG_BEARISH to STRONG_BULLISH
price	Dense(5)	Linear	Fractional price change at forecast steps 1-5
volatility	Dense(1)	Linear	Expected mean absolute return
confidence	Dense(2)	Softmax	Two-class: high / low market-stability confidence

4.4 Capacity and Regularization Balance

Version 3.0 increases capacity roughly in proportion to the richer input: wider recurrent layers, a 512-unit entry layer in the dense branch and a larger fusion vector. To offset the added risk of overfitting on noisy financial data, the base dropout rate is held at 0.4 (with derived rates in each block), L2 weight decay is 5e-5, and batch normalization follows every dense layer. All derived dropout rates scale from the single dropout_rate parameter.

5. Data Pipeline

5.1 Exchange Connectivity

The system connects to Gate.io via the ccxt (CryptoCurrency eXchange Trading) library in synchronous mode, wrapped in `asyncio.to_thread()` to avoid blocking the async event loop. Rate limiting is enabled and the connection uses a 30-second timeout on the spot market. Symbol formatting converts ccxt's slash notation to Gate.io's underscore notation (BTC/USDT → BTC_USDT). Data is requested as 4-hour OHLCV candles; the training request is sized at six candles per day of the stage lookback (250, 500 and 1,000 days in progressive mode; up to approximately 6,000 candles at the full window). Forward-fill and backward-fill handle gaps from exchange downtime.

5.2 Synthetic Data Fallback

When the exchange is unavailable, the system generates statistically realistic synthetic data using a Geometric Brownian Motion model. Returns are drawn from a normal distribution with a small upward drift ($\mu = 0.0001$) and per-asset calibrated volatility. Prices are computed as the cumulative sum of log-returns exponentiated. Intraday variation is added using normally distributed open/high/low offsets, and volume uses a lognormal distribution modulated by a sinusoidal intraday pattern. This ensures training and testing can proceed in offline environments.

5.3 Circuit Breaker Pattern

The circuit breaker pattern (Nygard, 2007), borrowed from distributed systems engineering, prevents cascading failures when an exchange endpoint is unreliable. After 3 consecutive failures for a given symbol, the circuit is opened for 6 hours, during which synthetic data is used automatically. This prevents repeated failing API calls that would waste rate-limit budget and slow processing of healthy symbols.

5.4 Data Sufficiency Gates

Stage	Requirement	Outcome if not met
Exchange fetch	At least 100 candles returned	Falls back to synthetic data
Training (each stage)	At least 300 candles available	Stage aborted; training reported as failed for that symbol
Sequence construction (each stage)	At least 100 training sequences	Stage aborted; training reported as failed for that symbol
Inference	90 days fetched; at least 75 feature rows after indicator warm-up (about 120 rows are consumed by the longest rolling windows)	Neutral default prediction returned

6. Feature Engineering

The feature engineering stage computes 76 input features from raw OHLCV data across twelve categories. Infinite values produced by division edge cases are replaced with missing values and all incomplete rows are dropped, so the network never sees undefined inputs. The RobustScaler is fit on training sequences only to prevent target leakage. The engine logs a warning if fewer than 70 features are produced.

6.1 Feature Summary Table

Category	Count	Features
Price and momentum	13	returns, log_returns, price_range, rsi_14, rsi_28, macd, macd_signal, macd_histogram, roc_5, roc_10, roc_20, momentum_5, momentum_10
Moving averages	20	For periods 5, 10, 20, 50, 100: ma_N, price_vs_ma_N, ema_N, price_vs_ema_N
Volatility	8	volatility_20, volatility_50, atr, bb_upper, bb_lower, bb_width, bb_percent, keltner_width
Volume	6	volume_ma_20, volume_ratio, obv, obv_ma, mfi, cmf
Oscillators and trend strength	7	stoch_k, stoch_d, williams_r, cci, adx, plus_di, minus_di
Market regime	5	trend_strength, trend_regime, volatility_regime, volume_regime, rsi_divergence
Microstructure proxies	2	bid_ask_spread, volume_profile

Category	Count	Features
Ichimoku	3	tenkan_sen, kijun_sen, ichimoku_cross
Candlestick patterns	4	doji, hammer, bullish_engulfing, bearish_engulfing
Cyclical time	4	hour_sin, hour_cos, dow_sin, dow_cos
Sentiment proxies	2	buying_pressure, selling_pressure
Volatility descriptors	2	volatility_skew, volatility_clustering
Total	76	

6.2 Indicator Definitions

Indicator	Definition / parameters
rsi_14 / rsi_28	Relative Strength Index with simple rolling averages of gains and losses (14 and 28 periods)
macd family	EMA(12) – EMA(26); signal = EMA(9) of MACD; histogram = MACD – signal
roc_N / momentum_N	Percentage change over N periods (5, 10, 20) / absolute price difference over N periods (5, 10)
bb_width / bb_percent	Bollinger bands (20, 2σ): width = (upper – lower) / mean; %B = (close – lower) / (upper – lower)
keltner_width	2 × ATR(14) / SMA(20)
obv / obv_ma	Cumulative signed volume; its 20-period mean
mfi / cmf	Money Flow Index (14) and Chaikin Money Flow (20)
stoch_k / stoch_d	Stochastic %K (14, smoothed over 3) and %D (3-period mean of %K)
williams_r / cci	Williams %R (14); Commodity Channel Index (20, constant 0.015)
adx / plus_di / minus_di	Directional movement system over 14 periods using rolling means
trend_strength	abs(polyfit slope over 20 closes)
trend_regime	1 if current trend_strength exceeds its 100-period mean, otherwise 0
volatility_regime / volume_regime	100-period z-scores of volatility_20 and volume_ratio
rsi_divergence	(RSI – its 20-period mean) × (close – its 20-period mean)
tenkan_sen / kijun_sen	Midpoints of the 9- and 26-period high/low ranges; ichimoku_cross = 1 when tenkan exceeds kijun
doji / hammer / engulfing	Binary candlestick flags: doji (body below 10% of range), hammer, and bullish / bearish two-candle engulfing
buying / selling_pressure	(close – low) / range and (high – close) / range
volatility_clustering	10-period rolling mean of absolute returns

6.3 Training Targets

Targets are derived from the future close over the forecast horizon (5 steps, or 20 hours). The trend label is a five-class discretization of the horizon return:

Class	Label	Horizon return
0	STRONG_BEARISH	below -2.0%
1	BEARISH	-2.0% to -0.5%
2	NEUTRAL	-0.5% to +0.5%
3	BULLISH	+0.5% to +2.0%
4	STRONG_BULLISH	above +2.0%

- **Price target:** fractional change of the close at each of steps 1-5 relative to the reference close.
- **Volatility target:** mean absolute return over a 5-candle window, shifted forward by the horizon.
- **Confidence target:** $\exp(-5 \times \text{volatility}_{20})$, a smooth score that approaches 1 in calm regimes and decays toward 0 as realized volatility rises. The two-class confidence head is trained on $[\text{score}, 1 - \text{score}]$.

7. Model Training

7.1 Training Procedure

Training uses a strict temporal split to prevent data leakage: 75% training, 15% validation, 10% test — applied in chronological order. `shuffle=False` is mandatory; shuffling would violate temporal causality by allowing future data to appear in training batches alongside past data.

Sequences are constructed with a sliding window of length 75, advancing one timestep at a time. Each input sequence $X[i]$ corresponds to timesteps $[i : i+75]$ and predicts targets at $[i + 75 + \text{forecast_horizon} - 1]$. Each window is flattened, scaled with a `RobustScaler` fit exclusively on training data, and reshaped to (75, 76) before being applied to the validation and test sets. A fresh scaler and a fresh feature build are produced at every stage because the data window, and therefore the feature distribution, changes between stages. Training runs on the CPU with batch size 64.

7.2 Loss Functions and Weighting

Output	Loss Function	Weight	Rationale
Trend	<code>SparseCategoricalCrossentropy</code>	0.35	Standard multi-class classification
Price	Huber ($\delta=1.0$)	0.40	Robust to crypto outlier price moves; highest weight
Volatility	<code>LogCosh</code>	0.15	Smooth MAE approximation, outlier-tolerant
Confidence	<code>BinaryCrossentropy</code>	0.10	Binary high/low confidence classification

7.3 Regularization Strategies

Six independent regularization mechanisms combat overfitting on noisy financial data:

- **Dropout (0.2–0.5):** Stochastic neuron deactivation, rate decreasing with depth in the Dense branch.
- **Recurrent Dropout (0.2–0.3):** Applied to LSTM recurrent connections independently from input dropout.
- **L1/L2 Weight Decay:** L1=1e-6 (sparsity), L2=5e-5 (weight magnitude penalty) on the first three Dense layers.
- **Batch Normalization:** Normalizes layer activations per mini-batch, stabilizing and accelerating training.
- **Early Stopping:** Halts training when val_loss stops improving, restoring the best weights. Patience scales with the stage's epoch budget.
- **Gradient Clipping:** clipnorm=1.0 in the Adam optimizer prevents gradient explosions through recurrent layers.

7.4 Learning-Rate Schedule and Callbacks

The optimizer is Adam with an initial learning rate of 0.0003. A **cosine decay schedule with linear warm-up** is applied per stage: the rate rises linearly over the first 10 epochs, then follows a half-cosine down to zero across the remainder of that stage's epoch budget. Three callbacks run alongside the schedule:

- **EarlyStopping** (patience = max(10, epochs/8), restore_best_weights=True, monitored on val_loss) prevents overtraining.
- **ReduceLROnPlateau** (patience = max(6, epochs/12), factor 0.5, min_lr 1e-7) provides a safety net for plateaus.
- **ModelCheckpoint** (save_best_only=True on val_loss) writes the best model to disk in .keras format during the stages that produce final artifacts.

7.5 Progressive Training

When progressive training is enabled (the default), each training run is a four-stage pipeline in which the network's weights are carried forward from stage to stage instead of being re-initialized. Intermediate stages learn quickly on shorter windows; the full-window stage receives the largest epoch budget; and a final fine-tune concentrates on the most recent market regime.

Stage	Data window	Max epochs	Early-stop patience	Purpose
1	250 days	50	10	Warm-up: new network learns coarse structure on a recent window
2	500 days	50	10	Scale-up: weights carried forward onto a wider window
3	1,000 days	200	25	Full-window training; scaler, features, metrics and model saved
4	100 most recent days	40	10	Fine-tune on the current regime; final artifacts saved

Stage epoch budgets derive from the configured maximum (200): intermediate stages use max(40, epochs/4), the full-window stage uses the configured maximum, and the fine-tune uses max(30, epochs/5) on a window of max(min(first stage / 2, 100), 30) days. If progressive training is disabled, a single full-window pass is run instead. Both paths write to the same artifact files, so loading and prediction are unaffected.

7.6 Walk-Forward Validation

After a progressive run completes, the engine can assess stability with an expanding-window walk-forward test (enabled by default). The 1,000-day dataset is split into five forward-chaining folds with a gap of 24 periods between the training and test blocks. For each fold a fresh network is trained for 20 epochs on the earlier data and scored on the block that follows. Fold metrics are summarized as:

- **mean and standard deviation** of trend accuracy across folds;
- **mean and standard deviation** of price MAE (as a fraction);
- **stability score** = mean trend accuracy ÷ (standard deviation + 1e-6), higher meaning more consistent behaviour across regimes.

The summary is written into the symbol's performance record under `walk_forward`. The check is non-fatal: if it fails, a warning is logged and the trained model is unaffected. Because folds are shortened for speed and use the trend and price targets only, the walk-forward numbers are a stability indicator and are not directly comparable with the full-training test metrics.

7.7 Auto-Retrain Service

The trainer keeps every model current without operator intervention.

- **Cycle.** Each session trains every configured symbol sequentially through the full progressive pipeline, then sleeps until the next scheduled time (default 12 hours, set by `RETRAIN_INTERVAL_HOURS`).
- **Per-model reporting.** After each successful training run the service prints test-set trend accuracy, price MAE and direction accuracy, followed by a session summary of successes, failures and the next training time.
- **Countdown display.** A tqdm progress bar shows the time remaining to the next cycle, with an automatic plain-text countdown if tqdm is not installed.
- **Fault isolation.** A failure on one symbol is logged and the session continues with the remaining symbols; if no model trains in a session the service warns and retries at the next scheduled time.
- **Graceful shutdown.** `SIGINT` and `SIGTERM` handlers stop the loop between symbols or during the countdown and print a session summary.

Two retraining policies coexist. The service launcher and the dedicated trainer retrain all active models every 12 hours. The engine's own scheduler job, which runs on the same interval, retrains a symbol only when its model is missing or its test-set trend accuracy has fallen below 40%. Reference timings are on the order of one to three hours for a full training cycle on CPU hardware.

7.8 Evaluation Metrics and Artifacts

After the final stage, the model is evaluated on the training, validation and test partitions. Results are stored per symbol in a performance file together with the training timestamp, stage name, feature count and sample counts.

Metric	Definition
trend_accuracy	Fraction of correct five-class trend predictions
price_mae_pct / price_rmse_pct	Mean absolute error and root-mean-square error of the five-step price-change vector, in percent
direction_accuracy	Fraction of samples where the sign of the step-5 predicted change matches the realized sign
volatility_mae	Mean absolute error of the volatility head
confidence_accuracy	Agreement of the confidence head with the target at the 0.5 threshold
avg_confidence	Mean confidence output across the partition

Four artifacts are persisted per symbol in the model directory: the trained network (`{SYMBOL}_advanced_model.keras`), the fitted scaler (`_advanced_scaler.pkl`), the ordered feature list (`_advanced_features.pkl`) and the performance record (`_performance.json`). On startup, models are loaded without compilation and then recompiled; a corrupted artifact set causes that symbol to be retrained at the next scheduled opportunity.

8. Inference Pipeline

At prediction time, the most recent 90 days of data are fetched, 76 features are engineered, and the last 75-step sequence is extracted. Features are scaled using the saved per-symbol `RobustScaler` (fit during training). The scaled sequence is passed through the trained model to produce four simultaneous outputs.

8.1 Output Decoding

- **Trend:** probabilities are decoded via `argmax` to one of five labels: `STRONG_BULLISH`, `BULLISH`, `NEUTRAL`, `BEARISH`, `STRONG_BEARISH`.
- **Price:** predicted fractional changes are multiplied by the current price to give absolute target prices at each of the five forecast steps (4, 8, 12, 16 and 20 hours).
- **Volatility:** the scalar forecast of mean absolute return is mapped to a qualitative risk level (Section 8.2).
- **Confidence:** the first softmax component is reported as a 0–100% score. Because the head is trained on a volatility-derived stability target (Section 6.3), it expresses how orderly the market regime is expected to be rather than a calibrated probability that the trend call is correct.

8.2 Risk Classification

Risk level	Forecast volatility (mean absolute return)
LOW	below 1.0%
MEDIUM	1.0% to 3.0%
HIGH	above 3.0%
UNKNOWN	assigned to default predictions when no model output is available

8.3 Prediction Record

Each prediction is cached in `latest_predictions`, keyed by symbol, and contains: symbol, current price, five target prices and percentage changes, trend label, trend confidence, volatility forecast, risk level, expected returns at steps 1–3, model confidence, timestamp and an analysis-type tag. The cache is consumed by the reporting subsystem every 4 hours and by the Symbol Oracle on demand.

8.4 Degraded-Mode Behaviour

If a model is not yet trained, data is insufficient, feature creation fails or an error occurs, the engine returns a neutral default prediction (trend NEUTRAL, confidence 0.10, risk UNKNOWN) carrying an `error_reason`. Reports exclude predictions with no valid price, so default records never appear among the top signals.

9. Symbol Oracle and Signal Persistence

9.1 Purpose

The Symbol Oracle is the module that turns transient predictions into durable, auditable signals. For each active asset an oracle instance requests a fresh prediction from the engine, assigns it a unique identifier, classifies its strength, writes it to a dedicated SQLite database, and sends a formatted message to Telegram. The database write and the Telegram dispatch are independent: a messaging failure never prevents the signal from being recorded.

9.2 Execution Model

- One oracle is created per active symbol and runs as its own asynchronous task, so assets are processed concurrently.
- Each oracle fires immediately at startup, then repeats every signal interval (default 4 hours, aligned with the report cycle).
- Between dispatches the oracle sleeps in 10-second slices and checks the shutdown flag, so the service stops promptly on request.
- Each oracle exposes status (dispatch count, last and next dispatch, database path) and a method to read back its most recent signals.

9.3 Storage Layout

Databases are created automatically on first use. On Windows the directory is `C:\hvts_signals`; on Linux and macOS it is `~/hvts_signals`. Each asset has its own file and table, named from the asset's display name.

Element	Pattern	Example (Bitcoin)
Database file	HVTS{Name}DB.db	HVTSBitcoinDB.db
Table	{Name}NeuralNetwork	BitcoinNeuralNetwork

9.4 Signal Schema

Column	Type	Description
id	INTEGER PK	Auto-increment row identifier
signal_id	TEXT UNIQUE	UUID assigned to the signal; first 8 characters shown in Telegram
timestamp	TEXT	Prediction timestamp
current_price	REAL	Price at time of analysis
trend / trend_label	TEXT	Machine label (e.g. STRONG_BULLISH) and display label
trend_confidence / model_confidence	REAL	Confidence outputs (0-1)
risk_level	TEXT	LOW, MEDIUM, HIGH or UNKNOWN
volatility_forecast	REAL	Forecast mean absolute return
expected_return_1h / _2h / _3h	REAL	Expected returns at forecast steps 1, 2 and 3 (4-hour steps; column names retained from earlier versions)
target_price_1 ... target_price_5	REAL	Absolute target prices at +4h, +8h, +12h, +16h, +20h
signal_strength	TEXT	Strength class (Section 9.5)
analysis_type	TEXT	Source tag of the prediction
telegram_sent / telegram_sent_at	INTEGER / TEXT	Delivery flag and UTC time, set after successful dispatch
error_reason	TEXT	Populated when the prediction is a degraded default
created_at	TEXT	UTC time the row was written

9.5 Signal Strength Classification

Strength combines the trend label with the model confidence and is evaluated in the order below.

Class	Rule	Display
VERY_STRONG	Strong trend label (STRONG_BULLISH or STRONG_BEARISH) and confidence of at least 75%	Very strong
STRONG	Strong trend label, or confidence of at least 70%	Strong
MODERATE	Confidence of at least 55%	Moderate
WEAK	NEUTRAL trend with confidence below 55%	Weak
UNCERTAIN	All remaining cases	Uncertain

9.6 Telegram Signal Message

Each signal is delivered as a structured MarkdownV2 message containing: the asset name and signal identifier, timestamp, direction with a trend-specific emoji, strength badge, confidence, current price, colour-coded risk badge, volatility forecast, expected returns at +4h, +8h and +12h, the five target prices with their percentage moves, and a footer indicating the next signal time. All dynamic content passes through the single-pass escaping routine described in Section 11.3.

9.7 Audit and Retrieval

Because each signal carries a unique identifier, a UTC creation time and its delivery status, the signal databases support after-the-fact evaluation: realized prices can be joined to the stored target prices to measure forecast error by horizon, trend class or strength class. Standard SQLite tooling can query the files directly, and the oracle's retrieval method returns the latest N signals ordered by creation time.

9.8 Asset Registry

Instrument	Display name	Database file
BTC/USDT	Bitcoin	HVTsBitcoinDB.db
ETH/USDT	Ethereum	HVTsEthereumDB.db
TRX/USDT	Tron	HVTsTronDB.db
XRP/USDT	Ripple	HVTsRippleDB.db
ADA/USDT	Cardano	HVTsCardanoDB.db
DOGE/USDT	Dogecoin	HVTsDogecoinDB.db
POL/USDT	Polygon	HVTsPolygonDB.db
DOT/USDT	Polkadot	HVTsPolkadotDB.db
AVAX/USDT	Avalanche	HVTsAvalancheDB.db
BNB/USDT	BinanceCoin	HVTsBinanceCoinDB.db

An oracle is instantiated only for instruments that are both in this registry and in the active symbol configuration. The reference deployment currently activates BTC/USDT; the remaining instruments are enabled by adding them to the configuration.

10. Downstream Integration

10.1 Purpose and Access Model

The signal databases written by the Symbol Oracle double as an integration surface. External trading engines, such as the Triple-OR Signal Protocol DCA engine, consume the neural signal through a lightweight reader module that opens the database in read-only mode. There is no network API and no model inference on the consumer side: training, prediction and persistence all happen in HVTS-NeuralNetwork, and the consumer only observes the most recent stored signal.

Property	Value
Access mode	SQLite read-only URI (mode=ro) with a 5-second busy timeout
Database / table	HVTSBitcoinDB.db / BitcoinNeuralNetwork (default path C:\hvts_signals; overridable)
Direction output	bullish or bearish; neutral only before the first non-neutral signal is ever observed
Persistence	Sidecar state file (neural_state.json)
Cache	In-process, 30-second time-to-live
Master switch	NEURAL_ENABLED environment variable

10.2 Database Contract

The consumer requires only two columns; the oracle schema (Section 9.4) is a superset of this contract, so no changes are needed on the producer side. The reader selects the single most recent row ordered by `created_at`.

Column	Required	Purpose
trend	Yes	Raw prediction label (see Section 10.3)
created_at	Yes	Ordering key; must increase with each signal
confidence	Optional	Detected at runtime if present (the oracle stores trend_confidence and model_confidence as fractions)
model_version	Optional	Informational only

10.3 Trend Mapping

Raw trend	Mapped direction
STRONG_BULLISH, BULLISH	bullish
NEUTRAL	neutral (ignored once a direction is established)
BEARISH, STRONG_BEARISH	bearish
Any other value	neutral (safe default)

The mapping is applied to the raw stored string, so the oracle's upper-case labels (STRONG_BULLISH and so on) are consumed as written.

10.4 Persistent Direction State Machine

The reader wraps the raw signal in a persistent state machine whose purpose is to remove *neutral* from the decision path after first initialization, so that a transient neutral reading or a database outage never reads as a bearish signal.

Case	Condition	Action
1	No direction yet and the raw trend is non-neutral	Initialize the direction, set flip count to 1, persist
1b	No direction yet and the raw trend is neutral	Search history for the latest non-neutral row; initialize from it if found
2	Direction set and the raw trend is neutral	Ignore; keep the current direction
3	Raw trend is non-neutral and opposite to the direction	Flip: increment the flip count, persist, log a warning
4	Raw trend matches the direction	Confirm; upgrade the stored raw trend to the STRONG variant when applicable

The state file records the effective direction, lifetime flip count, last raw trend, last update time and the creation time of the row that last changed the state. Deleting it while the consumer is stopped forces a re-bootstrap from the most recent non-neutral row.

10.5 Caching and Graceful Degradation

The reader caches its result for 30 seconds, so a consumer polling every minute performs at most one read per half-minute while the state machine still evaluates on every call. Every failure path returns a usable value and never blocks the consumer:

Failure	Behaviour
Neural leg disabled	Treated as neutral permanently (abstain)
Database file or table missing	Returns neutral and keeps the last known direction if one exists
Query error or lock	Returns neutral with an error source; direction retained
Empty table	Returns neutral; history fallback also empty

Consumers must treat a neutral neural reading as **abstain**, not as bearish.

10.6 Configuration

Variable	Default	Meaning
NEURAL_DB_PATH	C:\hvts_signals\HVTSBitcoinDB.db	Absolute path to the SQLite file
NEURAL_STATE_FILE	neural_state.json	Sidecar persistence file
NEURAL_TABLE	BitcoinNeuralNetwork	Table name
NEURAL_ENABLED	true	Master toggle for the neural leg
NEURAL_CACHE_TTL	30	Seconds between database re-reads

11. Scheduling and Reporting

11.1 Execution Contexts

The service layer operates several concurrent execution contexts: the main asynchronous event loop handles exchange I/O, the oracle tasks and Telegram API calls, while a dedicated daemon thread runs the *schedule* library polling loop at 60-second resolution for periodic job dispatch (reports and engine-level retraining).

11.2 Report Cycle

Each 4-hour report cycle generates three outputs simultaneously:

- **Console and log report:** a formatted table with a per-symbol model performance summary (test accuracy, price MAE, last-trained date), the top predictions ranked by confidence, and system status.
- **JSON file:** a timestamped file in `neural_network_reports/` containing the report identifier, symbols analysed, full predictions, performance metrics and system status.
- **Telegram summary:** a MarkdownV2 message with the top five signals by confidence and the average test accuracy across models.

11.3 Telegram Reliability

Telegram MarkdownV2 requires strict escaping of reserved characters. All dynamic content, including numeric values, symbol names, return strings and the pipe separator, passes through a single-pass escaping function that inserts one backslash before each reserved character and never re-processes its own output. This eliminates the double-escaping defect and the BadRequest errors caused by unescaped characters. On a BadRequest the system logs the error together with a preview of the failing message to speed diagnosis.

11.4 Console and Logging Robustness

On Windows the console and log streams are wrapped for UTF-8 so that status symbols render correctly and cannot raise encoding errors. Logs are written to timestamped files under `logs/` and mirrored to the console.

12. Licensing, Packaging and Diagnostics

12.1 License Model

HVTS-NeuralNetwork includes a node-locked licensing subsystem. A license is a compact signed token: a JSON payload authenticated with an HMAC-SHA256 signature and encoded as URL-safe base64. Any alteration of the payload invalidates the signature.

Payload field	Meaning
version	License format version (currently 1.0)
email	Licensee identifier
created	UTC issue time
features	List of enabled features (default: all)
machine_id	Optional. Binds the license to one machine
expiry	Optional. UTC expiry time; omitted for permanent licenses

Validation checks, in order, the signature, the format version, the expiry time and (where present) the machine binding. The machine identifier is a 16-character digest derived from a composite of host-level identifiers: hostname, operating-system hardware identifiers (volume or baseboard serial on Windows, machine-id on Linux, platform UUID on macOS) and a processor descriptor.

12.2 License Gate

Version 3.0 enforces licensing through a dedicated pre-flight module, `license_gate.py`. It is invoked exactly once at each process entry point, before any neural-network, exchange or messaging code is loaded, and it deliberately avoids heavy imports so that it behaves identically in source and frozen builds.

Entry point	Behaviour
<code>run_standalone.py</code>	<code>require_license()</code> is the first statement of <code>main()</code> . On failure the process exits with code 1 before any model, exchange connection or oracle task exists. A license banner is printed after successful validation.
<code>train_models.py</code>	Identical gate and banner; the trainer refuses to start without a valid license.
<code>health_check.py</code>	Read-only, non-fatal probe. Reports ACTIVE or INACTIVE with the licensee, expiry, machine binding and features, and adds a recommendation if the license is inactive.

When validation fails the gate prints a diagnostic block containing the reason (missing file, expiry, wrong machine or invalid signature), the current Machine ID, the expected license file location and the exact commands needed to activate or inspect a license.

12.3 End-User Commands

Command	Function
<code>python license_manager.py --info</code>	Display machine details and the Machine ID to send for license issuance
<code>python license_manager.py --activate [KEY]</code>	Validate and store a license key (prompts if no key supplied)
<code>python license_manager.py --status</code>	Show licensee, expiry, days remaining and features
<code>python license_manager.py --deactivate</code>	Remove the stored license after confirmation
<code>python license_gate.py</code>	Self-test the gate (exit code 0 if valid, 1 if not)

The active license is stored in `license.key` beside the executable or in the working directory.

12.4 Administrator Tooling

`generate_licenses.py` is an administrator-only tool that must be kept separate from distributed builds, and the build specification explicitly excludes it. It supports permanent or expiring licenses, machine-locked or portable licenses, feature scoping, an interactive prompt mode, and CSV batch generation (columns: email, machine_id, expiry_days, features). Each license is also saved to a file named from the licensee's email.

12.5 Executable Packaging

Two PyInstaller specifications build the Windows deliverables. Neither bundles a license key or credentials: each customer places their own `license.key` and `.env` beside the executable.

Executable	Specification	Mode	Entry script
HVTSNeuralNetwork	HVTSNeuralNetwork.spec	One-directory, console	<code>run_standalone.py</code>
HVTS-Network-Trainer	HVTS-Network-Trainer.spec	One-file, console	<code>train_models.py</code>

- **Service build:** TensorFlow, scikit-learn, ccxt, Telegram and standard-library modules are declared as hidden imports together with the license modules; the admin generator is excluded so the signing path is not embedded.
- **Trainer build:** heavy dependencies are gathered with PyInstaller's `collect_all`; the project's license modules are bundled; UPX compression is disabled because it corrupts TensorFlow, scikit-learn and ccxt libraries; selected unused TensorFlow components are excluded to trim size.
- **Cold start:** the one-file trainer unpacks to a temporary folder on every launch (typically 20–60 seconds); a one-directory variant is documented in the specification for faster starts.
- **Console retained** in both builds so logs and the countdown bar remain visible.

12.6 Health Check Utility

`health_check.py` performs a one-shot inspection and prints recommended actions. It verifies that the logs, models and reports directories exist (creating any that are missing); reports the license state; counts model, performance and scaler files and lists the five most recent models; locates every `HVTS*.db` signal database and reports its size, modification time, signal count and latest signal time; summarizes the most recent report and log; and finally recommends the next step (activate a license, train models, run the service, or none) based on what it finds.

13. Configuration Reference

All configuration is centralized in `config.py` as a singleton `Config` class, loaded from a `.env` file via `python-dotenv` and validated on every startup. Environment variable overrides are available for the key parameters listed in Section 13.5. Parameters are grouped by whether the current engine consumes them (Sections 13.1–13.3) or only declares them for future use (Section 13.4).

13.1 Neural Network Parameters

Parameter	Default	Description
<code>sequence_length</code>	75	Input timesteps (4h candles per sequence)
<code>forecast_horizon</code>	5	Steps ahead to predict (20 hours total)
<code>training_days</code>	1000	Full-window historical lookback (~2.7 years)
<code>epochs</code>	200	Maximum epochs for the full-window stage
<code>batch_size</code>	64	Mini-batch size for gradient computation
<code>learning_rate</code>	0.0003	Initial Adam learning rate (peak of the cosine schedule)
<code>lr_warmup_epochs</code>	10	Linear warm-up epochs
<code>lstm_units</code>	[256, 128]	Widths of the first and second BiLSTM layers
<code>dense_units</code>	[512, 256, 128]	Dense-branch widths (the 64-unit tail is derived as half of the last)
<code>dropout_rate</code>	0.4	Base dropout from which per-block rates derive
<code>attention_heads</code> / <code>attention_key_dim</code>	4 / 64	Multi-head attention geometry
<code>l1_regularization</code> / <code>l2_regularization</code>	1e-6 / 5e-5	Weight decay strengths

The engine currently fixes the sequence length (75) and forecast horizon (5) as constants that mirror these defaults. Additional entries (`early_stopping_patience`, `reduce_lr_patience`, `reduce_lr_factor`, `min_learning_rate`, `gradient_clip_norm`, `lr_schedule_total_epochs`) are retained for reference: the engine derives early-stopping and plateau patience from each stage's epoch budget, fixes gradient clipping at 1.0, and stretches the cosine schedule over each stage's own epochs.

13.2 Scheduling and Risk Parameters

Parameter	Default	Description
ANALYSIS_INTERVAL_HOURS	4	Signal frequency (the oracle interval)
RETRAIN_INTERVAL_HOURS	12	Model retraining frequency
NEURAL_NETWORK_REPORT_INTERVAL_HOURS	4	Report generation frequency (the engine currently uses a constant 4-hour interval)
SYMBOL_REFRESH_INTERVAL_HOURS	12	Symbol refresh frequency (declared)
MAX_POSITION_SIZE	0.10	Advisory maximum portfolio allocation per signal
STOP_LOSS_PERCENTAGE	0.025	Advisory stop loss at 2.5%
TAKE_PROFIT_PERCENTAGE	0.075	Advisory take profit at 7.5%
MIN_CONFIDENCE_THRESHOLD	0.65	Advisory minimum confidence to act
STRONG_SIGNAL_THRESHOLD	0.80	Confidence level regarded as a strong signal
MAX_CORRELATED_POSITIONS	3	Advisory cap on positions in correlated assets
MAX_OPEN_POSITIONS	5	Advisory cap on total open positions

13.3 Training Workflow Controls

Parameter	Default	Description
PROGRESSIVE_TRAINING	True	Run the staged pipeline of Section 7.5
PROGRESSIVE_STAGES	[250, 500, 1000]	Data windows (days) of the progressive stages
ENABLE_WALK_FORWARD_VALIDATION	True	Run walk-forward validation after progressive training
WALK_FORWARD_SPLITS / WALK_FORWARD_GAP	5 / 24	Number of folds and gap between train and test blocks (periods)
MIN_SYMBOLS / MAX_SYMBOLS	3 / 50	Recommended minimum (warning) and hard maximum (error) symbol count
MIN_FEATURE_COUNT_WARNING	70	Feature-count warning level (the engine applies the same value as a constant)

13.4 Reserved Controls

Parameter	Default	Description
ENABLE_ENSEMBLE / ENSEMBLE_WEIGHTS	True	Ensemble weights: trend 0.35, price 0.30, volatility 0.20, confidence 0.15
USE_DATA_CACHE / CACHE_TTL_HOURS	True / 24	Disk caching of market data
MIN_DATA_QUALITY_SCORE	0.95	Minimum data-quality score
PARALLEL_FETCH_LIMIT	5	Maximum parallel data fetches
ENABLE_SYNTHETIC_FALLBACK / ENABLE_CIRCUIT_BREAKER	True / True	Recovery switches (the engine always applies fallback and the breaker)
CIRCUIT_BREAKER_THRESHOLD / TIMEOUT_HOURS	3 / 6	Failures before opening / open duration (engine constants mirror these)
MODEL_RETRAIN_THRESHOLD	0.70	Accuracy floor policy (the engine's conditional retrain uses 0.40)
MAX_MODEL_VERSIONS / SAVE_BEST_ONLY / SAVE_FREQUENCY_EPOCHS	5 / True / 10	Model version retention and checkpointing
ENABLE_PERFORMANCE_LOGGING / RETENTION_DAYS	True / 30	Performance log controls
ALERT_ON_PERFORMANCE_DEGRADE / THRESHOLD	True / 0.15	Alert if accuracy drops by 15%
USE_XLA_COMPILATION / USE_MIXED_PRECISION	True / False	Performance options (execution is CPU-only)

Note on parameter status. The risk parameters in 13.2 are advisory values published for downstream execution systems; HVTS-NeuralNetwork does not place orders. The controls in 13.4 are declared as the planned control surface and are not yet enforced by the current engine (see Section 16). All parameters in 13.1 and 13.3 are consumed at runtime.

13.5 Environment Variables

Variable	Purpose
GATE_API_KEY, GATE_SECRET	Gate.io exchange credentials
TELEGRAM_BOT_TOKEN, TELEGRAM_CHAT_ID	Telegram delivery credentials and destination chat
TRADING_SYMBOLS	Comma-separated override of the active symbol list
TRAINING_DAYS	Override of the training lookback
SEQUENCE_LENGTH, FORECAST_HORIZON	Overrides recorded in configuration (the engine constants currently apply 75 and 5)
ANALYSIS_INTERVAL_HOURS	Override of the signal interval
LOG_LEVEL	DEBUG, INFO, WARNING or ERROR
NEURAL_DB_PATH, NEURAL_STATE_FILE, NEURAL_TABLE, NEURAL_ENABLED, NEURAL_CACHE_TTL	Downstream consumer settings (Section 10.6)

13.6 Validation Rules

On every startup the configuration is checked: the symbol count must not exceed the maximum; forecast horizon must not exceed sequence length; batch size must lie between 1 and 512; learning rate must be above 0 and no greater than 0.1; position size must be between 0 and 1; stop-loss and take-profit must lie strictly between 0 and 1; and the analysis interval must be at least one hour. Warnings are issued for fewer than the recommended minimum symbols, very short sequences, a retrain interval shorter than the analysis interval, and progressive stages that exceed the training lookback.

14. File Structure

```

project_root/
|-- config.py                # Configuration singleton
|-- hvts_neural_network.py   # Core engine (76 features, progressive training)
|-- hvts_symbol_oracle.py    # Symbol Oracle (signals, SQLite, Telegram)
|-- run_standalone.py        # Live service launcher (license-gated)
|-- train_models.py          # Auto-retrain trainer (license-gated)
|-- health_check.py          # Diagnostic utility (license-aware)
|-- license_gate.py          # Pre-flight license validator
|-- license_manager.py       # Licensing engine and end-user CLI
|-- generate_licenses.py     # Admin license tool (do not distribute)
|-- neural_signal_reader.py   # Downstream read-only consumer
|-- HVTSNeuralNetwork.spec    # Service build specification
|-- HVTS-Network-Trainer.spec # Trainer build specification
|-- requirements.txt          # Python dependencies
|-- .env                     # API keys (excluded from VCS)
|-- license.key              # Active license token (created on activation)
|-- neural_state.json        # Downstream direction state (created on first read)
|
|-- hvts_neural_models/      # Persisted model artifacts
|   |-- BTC_USDT_advanced_model.keras
|   |-- BTC_USDT_advanced_scaler.pkl
|   |-- BTC_USDT_advanced_features.pkl
|   |-- BTC_USDT_performance.json # includes walk_forward summary
|   `-- ... (one set of 4 files per symbol)
|
|-- neural_network_reports/   # JSON analysis reports
|   `-- hvts_nn_report_YYYYMMDD_HHMMSS.json
|
|-- logs/                    # Timestamped system logs
|   |-- hvts_nn_standalone_YYYYMMDD_HHMMSS.log
|   `-- training_YYYYMMDD_HHMMSS.log
|
`-- dist/                    # PyInstaller output
    |-- HVTSNeuralNetwork/    # one-directory service build
    |   |-- HVTSNeuralNetwork.exe
    |   |-- _internal/
    |   |-- license.key        # supplied per customer
    |   `-- .env               # supplied per customer
    `-- HVTS-Network-Trainer.exe # one-file trainer build

C:\hvts_signals\ (Windows) | ~/hvts_signals/ (Linux, macOS)
|-- HVTSBitcoinDB.db        # table: BitcoinNeuralNetwork
`-- HVTS{Name}DB.db         # one database per active symbol

```

15. Dependencies and System Requirements

Package	Reference version	Purpose
tensorflow	2.13.0 (≥ 2.12)	Neural network framework (LSTM, Attention, Keras functional API)
numpy	1.24.3	Numerical computation, array operations
pandas	2.0.3	Data manipulation, time series handling
scikit-learn	1.3.0	RobustScaler, TimeSeriesSplit, evaluation metrics
ccxt	4.1.22	Cryptocurrency exchange connectivity (Gate.io)
joblib	1.3.1	Scaler and feature list serialization to disk
schedule	≥ 1.2	Periodic job scheduling in background thread
python-telegram-bot	20.3	Telegram bot API for report and signal delivery
python-dotenv	1.0.0	Environment variable loading from .env file
tqdm	4.65.0	Countdown progress bar (optional; plain-text fallback provided)
sqlite3 / hmac / hashlib	standard library	Signal databases; license signing and validation
pywin32 (optional, Windows)	—	Volume serial number for machine identification
pyinstaller (build only)	—	Executable packaging
optuna	3.3.0 (listed)	Reserved for hyperparameter search; not used by the current engine

The licensing subsystem and the downstream reader use only the Python standard library. Reference system requirements for a VPS deployment:

Resource	Minimum	Recommended
CPU	2 vCores	4+ vCores
RAM	4 GB	8+ GB
Disk	10 GB	20+ GB
Operating system	Windows 10 / Server or Ubuntu 20.04+	Windows Server 2022 or Ubuntu 22.04 LTS
Python (source installs)	3.9+	3.10 or 3.11

16. Scope, Limitations and Roadmap

16.1 Scope

HVTS-NeuralNetwork analyses markets and publishes signals. It does not connect to trading accounts for order placement, and the risk parameters in this document are advisory inputs for whichever execution layer consumes the signals. The system is a decision-support tool, and nothing in this document or in any signal constitutes financial advice.

16.2 Performance Expectations

The configuration records design targets of 55–65% trend accuracy, 60–70% direction accuracy and 8–12% price MAE on held-out data. These are targets, not guarantees. The trend task has five classes, so a random classifier would score about 20%, and financial series are non-stationary: realized accuracy varies by asset and regime and should be monitored from the stored performance records, the walk-forward stability score and the signal databases.

16.3 Known Limitations

- **Confidence semantics.** The confidence output reflects forecast market stability, not the probability that a given trend call is right (Section 8.1).
- **Data source.** When the exchange is unreachable or a circuit is open, analysis can run on synthetic data intended for offline training and testing. Operators should treat any period with fallback warnings in the logs accordingly.
- **Training occupies the service.** A retraining cycle runs on the same process as the oracles; signals that fall due during a long training run are delayed until it completes.
- **Feature non-stationarity.** Several features are price-level or cumulative quantities (moving averages, Bollinger bands, Ichimoku levels, OBV). The RobustScaler normalizes them over the training window, but they remain non-stationary.
- **Validation scope.** Reported metrics come from one chronological split. Walk-forward folds use shortened training and the trend and price targets only, so they indicate stability rather than replace the primary metrics.
- **Reserved controls.** Ensemble inference, data caching, model versioning and performance alerting are declared in configuration but not yet applied (Section 13.4).

16.4 Roadmap

Item	Description
Ensemble inference	Combine model outputs using the configured weights
Data cache and quality gate	Disk caching with TTL and the configured data-quality threshold
Data-source tagging	Record whether each signal was produced from live or fallback data, and optionally withhold dispatch for fallback data
Non-blocking retraining	Run training in a worker thread or process so signal dispatch continues during long cycles
Model versioning and alerts	Retain recent model versions and alert on accuracy degradation
Hyperparameter search	Automated search over capacity, dropout and learning-rate settings
License hardening	Key management and signing-key separation for distributed builds
Multi-asset activation	Enable and tune the full ten-instrument registry in production

17. References

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473.
- Ballings, M., Van den Poel, D., Hespeels, N., & Gryp, R. (2015). Evaluating multiple classifiers for stock price direction prediction. *Expert Systems with Applications*, 42(20), 7046–7056.
- Baur, D. G., Hong, K., & Lee, A. D. (2018). Bitcoin: Medium of exchange or speculative assets? *Journal of International Financial Markets*, 54, 177–189.
- Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. *ICML 2009*, 41–48.
- Bergmeir, C., & Benítez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191, 192–213.
- Bouri, E., Molnár, P., Azzi, G., Roubaud, D., & Hagfors, L. I. (2017). On the hedge and safe haven properties of Bitcoin. *Finance Research Letters*, 20, 192–198.
- Brock, W., Lakonishok, J., & LeBaron, B. (1992). Simple technical trading rules and the stochastic properties of stock returns. *Journal of Finance*, 47(5), 1731–1764.
- Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1), 41–75.
- Dixon, M., Klabjan, D., & Bang, J. H. (2017). Classification-based financial markets prediction using deep neural networks. *Algorithmic Finance*, 6(3–4), 67–77.
- Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, 25(2), 383–417.
- Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2), 654–669.
- Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10), 2222–2232.
- Grobys, K., Ahmed, S., & Sapkota, N. (2020). Technical trading rules in the cryptocurrency market. *Finance Research Letters*, 32, 101396.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *CVPR 2016*, 770–778.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

- Hu, Z., Liu, W., Bian, J., Liu, X., & Liu, T.-Y. (2018). Listening to chaotic whispers: A deep learning framework for news-oriented stock trend prediction. WSDM 2018.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.
- Lo, A. W., Mamaysky, H., & Wang, J. (2000). Foundations of technical analysis. *Journal of Finance*, 55(4), 1705–1770.
- Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic gradient descent with warm restarts. ICLR 2017.
- Nygaard, M. T. (2007). Release It! Design and Deploy Production-Ready Software. Pragmatic Bookshelf.
- Ruder, S. (2017). An overview of multi-task learning in deep neural networks. arXiv:1706.05098.
- Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), 2673–2681.
- Siami-Namini, S., Tavakoli, N., & Namin, A. S. (2018). A comparison of ARIMA and LSTM in forecasting time series. ICMLA 2018, 1394–1401.
- Thorp, E. O. (1997). The Kelly criterion in blackjack, sports betting, and the stock market. *Handbook of Asset and Liability Management*, 1, 385–428.
- Van Tharp, R. (1998). Trade Your Way to Financial Freedom. McGraw-Hill.
- Vaswani, A., et al. (2017). Attention is all you need. NeurIPS 2017, 5998–6008.
- White, H. (1988). Economic prediction using neural networks: The case of IBM daily stock returns. ICNN 1988.
- Wilder, J. W. (1978). New Concepts in Technical Trading Systems. Trend Research.