COUNCIL OF ALPHAS

Complete Technical Specification

An Evolutionary Multi-Agent Framework for Alpha Discovery

HackEurope 2026 | Team: The Greeks (Andreas + Markos)

February 21, 2026

An evolutionary multi-agent framework that prevents mode collapse in LLM strategy generation through enforced specialist diversity, niche-preserving selection, deterministic hybrid construction, and evidence-locked diagnostic refinement.

1. Data

Parameter	Value
Instrument	SOL-USD
Timeframe	15-minute candles
Source	Binance (parquet file)
Date Range	3 years
Index	open_time (UTC datetime, already set as index)

Columns Kept from Raw Binance Data

Core OHLCV: open, high, low, close, volume

Extended: quote_volume, count, taker_buy_volume

2. Regime Detection

Three dimensions used exclusively as diagnostic buckets — NOT as strategy entry conditions.

2.1 Session Regime

Label	UTC Hours
ASIA	00:00 – 07:59
LONDON	08:00 – 12:59
NY	13:00 – 20:59
OTHER	21:00 – 23:59

2.2 Trend Regime

Indicator: SMA(50) slope over 3-bar lookback (pct_change(3)). Threshold: ±0.0005

Label	Condition
UPTREND	slope > +0.0005
DOWNTREND	slope < -0.0005
CONSOLIDATION	-0.0005 ≤ slope ≤ +0.0005

2.3 Volatility Regime

ATR window: 24 bars (6 hours on 15m). Smoothing: SMA(20) of ATR.

Label	Condition
HIGH_VOL	ATR(24) > SMA20(ATR(24))

LOW_VOL	ATR(24) ≤ SMA20(ATR(24))

Total micro-buckets: 4 sessions × 3 trend states × 2 vol states = 24 micro-buckets

3. Triple Barrier Labeling (TBM)

3.1 Fixed Parameters (System-Wide)

Parameter	Value
Win multiplier	2.0 × ATR (Take Profit)
Loss multiplier	1.0 × ATR (Stop Loss)
Time horizon	50 bars (~12.5 hours)
ATR window	24 bars (6 hours)
Tie-break	stop_first (worst-case on whipsaw)

3.2 Label Values

Label	Meaning
+1.0	Long trade: TP hit (price went up)
-1.0	Short trade: TP hit (price went down)
0.0	Timeout: neither barrier hit in time_horizon
NaN	Whipsaw: both long AND short hit in same bar (untradable)

3.3 Logic

- Long scan and short scan run simultaneously for every candle
- Long takes priority: if long hits TP → +1, else check short → -1, else 0
- Whipsaw (both directions hit) → NaN label (excluded from ML training)
- Every single candle in the dataset gets labeled

3.4 Output Columns (appended to State Matrix)

Column	Description
tbm_label	Oracle label: +1, -1, 0, or NaN
tbm_long_pnl	Exact fractional return if long trade taken
tbm_long_exit_idx	Row index where long trade exits
tbm_long_duration	Candles the long trade was open
tbm_short_pnl	Exact fractional return if short trade taken
tbm_short_exit_idx	Row index where short trade exits

tbm_short_duration	Candles the short trade was open

4. State Matrix

A single pre-computed pandas DataFrame saved as parquet. Built ONCE, loaded on every subsequent run. Every downstream component reads from it — nothing is recomputed.

4.1 Complete Column Schema

Column	Source	Type
open, high, low, close, volume	Raw Binance	float
quote_volume, count, taker_buy_volume	Raw Binance	float
session	StateMatrixBuilder	str: ASIA/LONDON/NY/OTHER
trend_regime	StateMatrixBuilder	str: UPTREND/DOWNTREND/CONSOLIDATIC
vol_regime	StateMatrixBuilder	str: HIGH_VOL/LOW_VOL
tbm_label	labeling.py	float: +1/-1/0/NaN
tbm_long_pnl, tbm_long_exit_idx, tbm_long_durabe	labeling.py	float/int/int
tbm_short_pnl, tbm_short_exit_idx, tbm_short_duratin	labeling.py	float/int/int

4.2 StateMatrixBuilder Updated Parameters

```
StateMatrixBuilder(
    trend_slope_threshold=0.0005,  # updated from 0.001
    atr_window=24,                  # updated from 14 (= 6 hours on 15m)
    vol_sma_window=20,
    tbm_win=2.0,                    # fixed system-wide
    tbm_loss=1.0,                   # fixed system-wide
    tbm_time_horizon=50,
)
```

Intermediate columns (sma_50, sma_50_slope_3, ATR_24, ATR_24_SMA_20) are dropped before saving parquet.

Load from parquet if exists. Build and save if not. force_rebuild=True to override.

5. Specialist Agents

4 LLM agents (Claude Sonnet, temp=0), each locked to a distinct strategy family.

5.1 Families & Indicator Subsets

Family	Allowed Indicators
Trend	ema, hma, macd, adx, slope
Momentum	rsi, cci, roc, mfi, zscore
Volatility	natr, bollinger_bands, keltner_channels, choppiness_index
Volume	vwap, obv, cmf

5.2 Random Indicator Sampling

Before each strategy generation, IndicatorSampler randomly samples a subset of the family's allowed indicators (min=2, max=4). Sampled subset injected into prompt — LLM cannot use what it cannot see. Prevents intra-specialist mode collapse.

5.3 Strategy Base Class

```
class Strategy(Indicators):
    name: str = "unnamed"
    family: str = "unknown"   # trend/momentum/volatility/volume/hybrid
    description: str = ""     # one sentence: what does this strategy do

    def generate_signals(self, data: pd.DataFrame) -> pd.Series:
        # Returns: 1 (long), -1 (short), 0 (flat)
        # Index must match data.index exactly
        raise NotImplementedError
```

IMPORTANT: tbm_win and tbm_loss are NOT on Strategy objects — fixed system-wide in config.py

5.4 Generation Rules

- PoC: 1-3 strategies per specialist (MAX_STRATEGIES_PER_SPECIALIST in config.py)
- Score immediately after each generation (not batched)
- Max 3 code generation attempts per strategy with error feedback injection
- Parallel generation across specialists using asyncio.gather(return_exceptions=True)
- 60-second timeout per strategy execution

5.5 Code Validation Steps

Step	Check	Action on Failure
1	Syntax: compile(code)	Retry with SyntaxError message
2	Execution: strategy.generate_signals(sample_data)	Retry with exception message

3	Return type: valid pd.Series of 1/-1/0	Retry with type feedback
4	Trade count: must produce > 0 trades	Retry with guidance to loosen conditions

6. Backtesting

Custom VectorizedBacktester — no VectorBT dependency. Numba-accelerated.

Parameter	Value
Fee	0.00075 (0.075% per trade, Binance taker)
Capital lock	No overlapping trades allowed
TBM	Reads pre-computed columns from State Matrix

Trade Log Output Schema

Column	Description
entry_ts, exit_ts	Entry and exit timestamps
entry_index, exit_index	Row indices
duration	Candles open
side	1 (long) or -1 (short)
net_trade_return	PnL after fee deduction
session, trend_regime, vol_regime	Regime tags from entry candle

7. Diagnostics Engine

Hierarchical bucket system. 5 metrics across all regime combinations. 6-tile fingerprint replaced entirely by this bucket system.

7.1 Hierarchy (60 rows total)

Level	Description	Rows
GLOBAL	All trades combined	1
1D	By session, trend, or vol (separate)	4 + 3 + 2 = 9
2D	Session×Trend, Session×Vol, Trend×Vol	12 + 8 + 6 = 26
3D	Session × Trend × Vol (all combinations)	24
TOTAL		60

7.2 Output Schema

Column	Description
granularity	GLOBAL / 1D / 2D / 3D
session, trend_regime, vol_regime	Regime values or ALL
trade_count	N trades in bucket
win_rate	% winning trades
sharpe	mean_return / std_return (ddof=0, NaN if std=0 or n<2)
max_consecutive_losses	Longest losing streak (chronological)
sufficient_evidence	True if trade_count >= 30

Critic receives ONLY rows where sufficient_evidence = True

8. Fitness Function

```
Score = Global_Sharpe × ln(N) × Coverage
```

Component Definitions

Global_Sharpe: Sharpe from the GLOBAL row of diagnostics table.

ln(N): Natural log of total trade count from GLOBAL row. Rewards sample size with diminishing returns.

Coverage (trade-weighted):

```
active = 3D buckets where sufficient_evidence == True
profitable_trades = sum(trade_count for buckets where sharpe > 0)
total_trades = sum(trade_count for all active buckets)
coverage = profitable_trades / total_trades
```

Hard Eliminations (return -999 before scoring)

```
if global_row.sufficient_evidence == False: return -999
if global_row.sharpe <= 0: return -999
if len(active_3d_buckets) == 0: return -999
```

Why This Formula

Component	What it catches
Global_Sharpe	Is this strategy actually profitable risk-adjusted?
ln(N)	Do we have enough evidence? Diminishing returns above ~1000 trades
Coverage	Is it profitable across the regimes where it actually TRADES?

UPTREND-only specialist scores well: Coverage ≈ 1.0 (all its active trades profitable). Not penalized for absence in other regimes — only penalized for losses where it is present.

Same formula used for both champions (Stage 3) and hybrids (Stage 5) — rankings directly comparable.

9. Niche Selection

Select top 1 champion per family. Prevents competitive exclusion.

Rule	Detail
Selection	Top 1 strategy per family by fitness score
Threshold	Score > 0 (must pass hard eliminations)
No viable champion	Family eliminated, Architect proceeds with remaining
Minimum champions	Pipeline continues with as few as 1 champion

Pipeline Order

Specialists generate → Fitness scored → Niche selection → HybridBuilder → Scientist → Final fitness → Ranking

10. HybridBuilder (Pure Python — No LLM)

Deterministic Python class. Takes up to 4 champions. Produces exactly 3 hybrids.

Hybrid 1 — Regime Router

For each of the 24 regime combinations, assign the champion with the highest Sharpe in that specific 3D bucket (sufficient_evidence=True). Fallback: if no champion has sufficient evidence → use champion with best GLOBAL Sharpe.

Hybrid 2 — Consensus Gate

All 4 champions vote. Fire long if >= 3/4 agree on long. Fire short if >= 3/4 agree on short.

```
votes = trend_signal + momentum_signal + vol_signal + volume_signal
long_signal = (votes >= 3)
short_signal = (votes <= -3)
```

Hybrid 3 — Weighted Combination

Champions weighted by fitness score. Weighted sum's sign determines direction.

```
weights = [fitness_trend, fitness_momentum, fitness_vol, fitness_volume]
weighted_sum = sum(w * s for w, s in zip(weights, signals))
signal = np.sign(weighted_sum)
```

All hybrids are inline Strategy subclasses (not compositional). Parent logic copied with clear section comments. family = 'hybrid' for all three.

11. Scientist / Critic Loop

11.1 Loop Structure (per hybrid, independent)

Step	Action
1	Backtest → trade log
2	DiagnosticsEngine → 60-row bucket table
3	Fitness check → if -999: UNVIABLE, stop
4	Critic (Opus, temp=0) → structured diagnosis
5	If UNVIABLE → discard. If CONTINUE → Refiner
6	Refiner (Sonnet, temp=0) → updated code (one change only)
7	Validation gate → accept / revert / early-exit
8	Repeat max 5 iterations

11.2 Iteration Rules

Rule	Detail
Max iterations	5
Early exit	2 consecutive iterations with improvement < 0.05 Sharpe → stop, keep best
Revert	If new score < previous score → revert to previous version
Guarantee	Monotonic improvement: v_n >= v_{n-1} always
UNVIABLE fallback	If all 3 hybrids UNVIABLE → fall back to best champion

11.3 UNVIABLE Conditions

Condition	Threshold
GLOBAL Sharpe < -0.5	With sufficient_evidence=True
Zero profitable 3D buckets	No 3D bucket with sufficient_evidence=True AND Sharpe > 0
Consecutive losses	GLOBAL max_consecutive_losses > 20

11.4 Critic Prompt (Claude Opus, temp=0)

```
You are the Evidence-Locked Critic for the Council of Alphas framework.

ROLE: Diagnose underperformance using ONLY computed numbers. Never guess.
Every claim must cite exact bucket and exact number from the table.

SCAN ORDER: GLOBAL → 1D → 2D → 3D

CONSTRAINTS:
- No structural rewrites
- No changing indicators or family
- One surgical fix only (parameter, threshold, or single condition)
```

```
- Every claim must cite: [bucket] | sharpe=[x] | n=[x]

OUTPUT FORMAT (exactly this, nothing else):
PRIMARY_FAILURE: [exact bucket] | sharpe=[value] | n=[value]
ROOT_CAUSE: [one sentence citing the code]
SURGICAL_FIX: [exact code change]
EXPECTED_IMPACT: [which metric improves and why]
VERDICT: CONTINUE | UNVIABLE
```

11.5 Refiner Prompt (Claude Sonnet, temp=0)

```
You are the Surgical Refiner. Apply exactly ONE fix from the Critic.
Do NOT restructure, change indicators, add or remove logic.
Return ONLY the complete updated Python class.
No explanation. No markdown. No imports.
```

12. Orchestration

12.1 Pipeline Flow

```
PIPELINE START
■■■ 1. LOAD DATA (Binance SOL-USD 15m parquet)
■■■ 2. BUILD / LOAD STATE MATRIX
■   ■■■ If parquet exists → load
■   ■■■ If not → StateMatrixBuilder.build() → save parquet
■■■ 3. SPECIATION (parallel, asyncio.gather)
■   ■■■ 4 specialists × 1-3 strategies each (scored immediately)
■■■ 4. NICHE SELECTION (top 1 per family, score > 0)
■■■ 5. HYBRID BUILDING (pure Python, 3 hybrids)
■■■ 6. SCIENTIST LOOP (sequential, max 5 iterations per hybrid)
■■■ 7. FINAL RANKING (fitness score, fallback to best champion)
■■■ OUTPUT → Streamlit UI
```

12.2 Key Configuration Constants

```
MAX_STRATEGIES_PER_SPECIALIST = 3
MAX_GENERATION_ATTEMPTS = 3
STRATEGY_TIMEOUT_SECONDS = 60
MAX_SCIENTIST_ITERATIONS = 5
MIN_IMPROVEMENT_THRESHOLD = 0.05
TBM_WIN = 2.0
TBM_LOSS = 1.0
TBM_TIME_HORIZON = 50
TBM_ATR_WINDOW = 24
BACKTEST_FEE = 0.00075
TREND_SLOPE_THRESHOLD = 0.0005
```

12.3 Model Assignment

Role	Model	Temperature
Specialist Agents (×4)	Claude Sonnet	0
Architect	PYTHON ONLY — no LLM	N/A
Scientist Critic	Claude Opus	0
Scientist Refiner	Claude Sonnet	0

13. Output / UI (Streamlit — Andreas)

Panel	Content
1 — Pipeline Status	Live real-time log of every pipeline event
2 — Champion Leaderboard	Family, Name, Fitness, Sharpe, Win Rate, Trades, Coverage
3 — Diagnostics Heatmap	Plotly heatmap: rows=Session×Trend, cols=Vol, color=Sharpe
4 — Scientist Loop Trace	Per hybrid: iteration history, Critic diagnoses, fixes applied

5 — Final Ranked Results	Lineage tree + Cumulative PnL chart (winning strategies only)

Cumulative PnL chart: Plotly line chart, one line per surviving strategy, X=trade number, Y=cumulative return %. Multiple survivors on same chart.

If all hybrids UNVIABLE: show best champion's PnL chart instead.

Use st.session_state to preserve pipeline results across Streamlit reruns.

14. Repository Structure

```
council_of_alphas/
■■■ data/
■   ■■■ btc_usd_15m_3y.parquet        # Raw Binance data (provided)
■   ■■■ state_matrix.parquet          # Auto-generated on first run
■
■■■ docs/
■   ■■■ MASTER_SPEC.md                # Complete specification
■   ■■■ BUILD_GUIDE.md                # Step-by-step build guide
■   ■■■ ARCHITECTURE.mermaid          # Pipeline diagram
■   ■■■ Council_of_Alphas_SPEC.pdf    # This document
■
■■■ config.py                         # ALL constants (build first)
■■■ strategy_base.py                  # Updated base class
■■■ indicator_sampler.py              # Updated (cleaned categories)
■■■ whitelist_indicators.py           # Ready (do not modify)
■■■ labeling.py                       # Ready (do not modify)
■■■ backtesting.py                    # Ready (do not modify)
■■■ diagnostics.py                    # Ready (do not modify)
■■■ state_builder.py                  # Update parameters only
■
■■■ fitness.py                        # Build: compute_fitness()
■■■ prompt_builder.py                 # Build: build_specialist_prompt()
■■■ specialist_agent.py               # Build: async generate_strategy()
■■■ niche_selector.py                 # Build: select_champions()
■■■ hybrid_builder.py                 # Build: HybridBuilder class
■■■ critic_agent.py                   # Build: run_critic()
■■■ refiner_agent.py                  # Build: run_refiner()
■■■ scientist.py                      # Build: run_scientist()
■■■ orchestrator.py                   # Build: Orchestrator class
■■■ app.py                            # Build: Streamlit UI (Andreas)
```

Build Order (strict dependency)

config.py → strategy_base.py → indicator_sampler.py → fitness.py → prompt_builder.py → specialist_agent.py → niche_selector.py → hybrid_builder.py → critic_agent.py → refiner_agent.py → scientist.py → orchestrator.py → app.py