🔄Auto-Learning Architecture

Safe, controlled learning from live market data
Designed for non-stationary markets (Gold & Silver)

Purpose

This document defines how the **AI Strategy Engine** learns from incoming market data **without degrading model quality**.

The goal is to: - Adapt to changing market behaviour - Improve probability estimates over time - Preserve stability and explainability

This is **not** real-time self-retraining.

Core Principle (Non-Negotiable)

🚫**The LLM must never retrain itself directly on live market prices.**

Reasons: - Markets are non-stationary - Live noise causes overfitting - One bad regime can permanently damage the model - Catastrophic forgetting is irreversible

Instead, learning is split into **three controlled layers**.

Layer 1 — Continuous Market Ingestion (Always On)

What Happens

Incoming data from market APIs is continuously ingested:

- OHLCV (multi-timeframe)
- Spread & liquidity metrics
- Volatility measures
- Session transitions (Asia / London / NY)
- Event timestamps (news, rollovers)

Rules

- No model weights are updated
- Data feeds feature engineering and memory only

Outcome

Provides **fresh context** without risk.

Layer 2 — Experience Memory (RAG Auto-Learning)

This is the **primary auto-learning mechanism**.

What Is Stored

Each inference + outcome becomes an experience record:

```
{
  "context": "Silver London open fake breakdown below Asia low",
  "features": {
    "session": "London",
    "volatility": "High",
    "spread": "Wide"
  },
  "decision": "Long reversal",
  "targets": "+0.30%",
  "outcome": "Failed",
  "MAE": "-0.18%",
  "MFE": "+0.12%"
}
```

How It Learns

- Each record is embedded into a vector database
- Future inferences retrieve **similar historical cases**
- Reasoning improves immediately without retraining

The system gets smarter **without changing its brain**.

Layer 3 — Scheduled Model Updates (Gated Fine-Tuning)

This is the **only layer where model weights change**.

When It Runs

- Offline only
- Weekly or monthly
- Never during live trading

Data Used

- Curated, labelled experiences
- Filtered for data quality
- Balanced across regimes

Pipeline

```
Live Data → Outcomes → Validation
         → Dataset Curation
         → LoRA Fine-Tune
         → Evaluation
         → Manual Deploy
```

Safety Gates

A model update is rejected unless it: - Improves expectancy - Reduces drawdown - Preserves known behaviours (regression tests)

No automatic deployment is allowed.

What the System Learns

Learns

- Session behaviour drift
- Regime transitions
- Target probability accuracy
- Strategy decay over time

Does NOT Learn

- Exact future prices
- Tick-level prediction
- Emotional reactions to losses

Probability Adaptation (Fast Feedback)

Instead of retraining, the system adapts via:

- Probability decay
- Distribution widening / tightening
- Confidence throttling

- Trade frequency reduction

Example

```
If volatility regime shifts rapidly:
 → widen probability curves
 → reduce target confidence
 → limit trades per session
```

This provides **near-real-time adaptation** safely.

Drift & Degradation Detection

The system continuously monitors:

- Target hit-rate vs expectation
- Confidence vs realised outcomes
- Regime misclassification frequency

Trigger Actions

- Reduce confidence
- Flag retraining candidate
- Lock strategy output

Integration with System Components

Node.js

- Market API ingestion
- Trade journaling
- Outcome labelling
- Dataset versioning

Java

- Regime detection updates
- Feature drift analysis
- Confidence decay logic

LLM

- Uses RAG memory immediately
- Receives gated LoRA updates only

What NOT to Do (Critical)

🚫Online backpropagation on live ticks
🚫Weight updates per trade
🚫Training on unlabelled outcomes
🚫Immediate learning from losses
🚫Deleting historical regimes

Markets punish impatient learners.

Summary

✔️Yes, the system **auto-learns from live market data**
✔️Learning is **safe, explainable, and reversible**
✔️Adaptation happens faster than retraining

This architecture is designed for **real trading systems**, not experiments.