ATHENA

Adaptive Trading & Hedging Engine with Neural Agents

Technical Architecture Document

Version 1.0
November 2025

Document Control	Details
Author	Rajesh Kumar
Organization	Swaas Systems Private Limited
Classification	Internal - Technical Architecture
Status	Draft for Review

Table of Contents

1. Executive Summary

ATHENA (Adaptive Trading & Hedging Engine with Neural Agents) is a next-generation multi-agent financial assistant that combines cutting-edge research from 2025 to create a self-evolving, memory-augmented, and highly efficient trading system. The architecture integrates five breakthrough research papers into a unified framework that enables autonomous task generation, experience-based learning, and seamless agent collaboration.

1.1 Core Innovation

Unlike traditional rule-based or single-model trading systems, ATHENA employs multiple specialized agents that communicate through latent representations rather than text, learn from experience through just-in-time memory retrieval, and continuously evolve their capabilities through self-generated training tasks. The system is built on AI2's OLMo 3, a fully open and transparent foundation model that enables complete traceability from trading decisions back to training data.

1.2 Key Differentiators

- Self-Evolution: Agents autonomously generate and learn from increasingly difficult trading scenarios
- Lossless Memory: Just-in-time retrieval eliminates information loss from memory compression
- Latent Communication: 10-100x faster inter-agent coordination through embedding exchange
- Continual Learning: Multi-timescale updates prevent catastrophic forgetting
- Full Transparency: OLMo 3's OlmoTrace enables regulatory-compliant decision auditing

2. Research Foundation

ATHENA is built upon five peer-reviewed research contributions from leading AI institutions. Each paper addresses a specific challenge in building robust, scalable multi-agent systems.

Paper	Source	Core Contribution	ATHENA Application
AgentEvolver	ModelScope/Alibaba	Self-questioning, self-navigating, self-attributing mechanisms for agent evolution	Training loop for agent improvement without human task curation
GAM	VectorSpaceLab	Just-in-time memory compilation with multi-hop research	Lossless experience retrieval across trading history
LatentMAS	Research Community	Latent thought generation and shared working memory	High-bandwidth agent communication
Nested Learning	Google/NeurIPS 2025	Multi-timescale parameter updates with continuum memory	Prevent catastrophic forgetting during adaptation
RepExp	Research Community	Representation-based exploration bonuses	Maintain strategy diversity, 3x sample efficiency

3. Foundation Model: OLMo 3

ATHENA uses AI2's OLMo 3 as its foundation model instead of alternatives like Qwen 2.5 or Llama 3.1. This choice is driven by several strategic and technical considerations that align with the requirements of a production financial system.

3.1 Model Selection Rationale

Criterion	OLMo 3	Qwen 2.5	Winner
True Openness	Full data, code, checkpoints, training logs	Weights only	OLMo 3
RL Research Support	OLMo-3-RL-Zero designed for RL	Standard fine-tuning	OLMo 3
Decision Traceability	OlmoTrace links to training data	None	OLMo 3
Training Efficiency	2.5x more efficient than Llama	Standard	OLMo 3
Context Window	65K tokens	128K tokens	Qwen 2.5
Model Sizes	7B, 32B	0.5B to 72B	Qwen 2.5

3.2 OLMo 3 Model Variants

ATHENA leverages different OLMo 3 variants for different purposes within the system:

- **OLMo 3-Base 32B:** Primary foundation for all agents, provides strong reasoning and code generation capabilities.
- **OLMo 3-Think 32B:** Used for Strategy Agent requiring explicit reasoning chains for complex decisions.
- **OLMo 3-Instruct 7B:** Used for Execution Agent optimized for tool use and API interactions.
- **OLMo 3-RL-Zero 7B:** Clean baseline for reinforcement learning experiments with decontaminated RL data.

4. System Architecture Overview

ATHENA's architecture consists of four interconnected layers: the Agent Layer (specialized trading agents), the Memory Layer (GAM-based experience storage), the Communication Layer (LatentMAS), and the Evolution Layer (AgentEvolver training loop). Each layer addresses specific challenges in building robust multi-agent financial systems.

4.1 Architectural Layers

Layer	Components	Research Basis	Primary Function
Agent Layer	Market Analyst, Risk Manager, Strategy Agent, Execution Agent, Coordinator	AgentEvolver	Specialized task execution
Memory Layer	Memorizer, Page-Store, Researcher	GAM	Lossless experience retrieval
Communication Layer	Latent Thought Generator, Shared Working Memory	LatentMAS	High-bandwidth coordination
Evolution Layer	Task Generator, Attribution Engine, Experience Pool	AgentEvolver + RepExp	Continuous improvement
Learning Layer	Continuum Memory System, Multi-scale Optimizers	Nested Learning	Prevent forgetting

4.2 Agent Specifications

Each agent in ATHENA is a specialized instance of OLMo 3 with a role-specific LoRA adapter. This approach maintains latent space compatibility for LatentMAS communication while enabling deep specialization.

Agent	Base Model	Primary Function	Tools/APIs	Self-Evolution Focus
Market Analyst	OLMo 3-Base 32B + LoRA	Analyze market data, detect trends, generate signals	Yahoo Finance, Alpha Vantage, News APIs	Pattern recognition accuracy
Risk Manager	OLMo 3-Base 32B + LoRA	Calculate VaR, monitor exposure, set limits	Portfolio DB, Risk Models	Risk prediction precision
Strategy Agent	OLMo 3-Think 32B + LoRA	Formulate trading strategies with explicit reasoning	Backtesting Engine	Strategy formulation
Execution Agent	OLMo 3-Instruct 7B + LoRA	Execute trades, manage orders, handle slippage	Broker APIs (IBKR, Alpaca)	Execution efficiency
Coordinator	OLMo 3-Base 32B + LoRA	Route tasks, orchestrate multi-agent workflows	Agent Registry	Delegation accuracy

5. Memory System: GAM Integration

Traditional memory systems suffer from information loss due to compression. GAM (General Agentic Memory) solves this by maintaining complete history in a page-store while performing just-in-time (JIT) context generation at runtime. This enables multi-hop reasoning like 'Find all times VIX > 30 AND we held tech positions AND what was the outcome.'

5.1 GAM Architecture

Component	Function	Implementation
Memorizer	Creates lightweight summaries of trading sessions for quick indexing	Runs after each trading day to compress key events
Page-Store	Maintains complete, uncompressed history of all trades, decisions, and outcomes	Vector database (Qdrant/Pinecone) with full trajectory storage
Researcher	Performs iterative search with planning, searching, and reflection (max depth 3)	Multi-hop retrieval with up to 5 pages per query

5.2 GAM vs Traditional Retrieval

GAM replaces AgentEvolver's simple top-k experience retrieval with deep research capabilities:

Aspect	Traditional (Top-K)	GAM (JIT Research)
Retrieval Method	Single-hop cosine similarity	Multi-hop with reflection
Information Loss	High (compressed embeddings)	None (full page-store)
Complex Queries	Limited to keyword matching	Supports conditional logic
Reasoning Depth	Surface-level similarity	Iterative planning (depth 3)
Benchmark Performance	Baseline	+29.6% F1 on HotpotQA 448K

6. Communication Layer: LatentMAS

Traditional multi-agent systems communicate through text serialization, which is slow and lossy. LatentMAS enables agents to share 'intuitions' directly through latent embeddings, achieving 10-100x faster communication while preserving subtle information that would be lost in text.

6.1 LatentMAS Components

Component	Description	Financial Application
Latent Thought Generator	Extracts last-layer hidden embeddings as 'thoughts' instead of generating text	Market Analyst shares market 'feel' with Strategy Agent
Shared Working Memory	Vector buffer where agents deposit and retrieve latent thoughts	Common situational awareness across all agents
Latent Decoder	Interprets incoming latent thoughts from other agents	Execution Agent understands Risk Manager's constraints

6.2 Latent Space Compatibility

For LatentMAS to work, all agents must share a compatible latent space. ATHENA achieves this through:

1. **Same Base Model:** All agents use OLMo 3-Base 32B as foundation
2. **LoRA Specialization:** Role-specific adapters maintain shared embedding space
3. **Training-Free:** LatentMAS is a drop-in replacement requiring no additional training

7. Evolution Layer: AgentEvolver Integration

The Evolution Layer implements AgentEvolver's three core mechanisms to enable continuous agent improvement without human task curation. This creates a virtuous cycle where agents generate their own training data, learn from experience, and receive fine-grained credit for their decisions.

7.1 AgentEvolver Mechanisms

Mechanism	Description	Financial Implementation
Self-Questioning	Autonomous generation of diverse training tasks through environment exploration	Generate varied trading scenarios: easy (single asset analysis) to hard (multi-asset hedging with constraints)
Self-Navigating	Retrieve and apply relevant past experiences during inference	'When VIX > 25, recall March 2020 volatility trades' injected as in-context learning
Self-Attributing	LLM-based step-wise credit assignment using GOOD/BAD labels	Attribute profit/loss to specific decisions in trading trajectory

7.2 Training Loop

The complete training loop integrates all research components:

1. **Self-Questioning:** Generate diverse financial tasks (breadth Nb=3, depth Nd=17)
2. **Rollout:** Execute with GAM memory + LatentMAS communication (η=0.5 experience ratio)
3. **Self-Attributing:** Step-wise credit assignment (α=0.1 attribution weight)
4. **RepExp Bonus:** Add representation-based diversity bonus to prevent strategy collapse
5. **GRPO Update:** With Nested Learning multi-scale updates (fast + slow parameters)
6. **GAM Update:** Store successful trajectories in page-store

7.3 Reward Computation

ATHENA uses a composite reward function that combines outcome-based rewards with step-wise attribution:

```
r_composite[t] = α × r_attr[t] + (1 if t=T else 0) × r_outcome
```

Where α=0.1 (range 0.05-0.2), r_attr is the attribution score (+1 for GOOD, -1 for BAD), and r_outcome is the final trading profit/loss normalized.

7.4 Key Hyperparameters

Parameter	Value	Description
α (attribution weight)	0.1 (range 0.05-0.2)	Balance between step-wise credit and outcome reward
η (experience ratio)	0.5	50% vanilla rollouts, 50% experience-guided rollouts
ε_high (selective boost)	0.6	Threshold for applying attribution bonus to high-rated steps
Top-k experiences	5	Number of relevant experiences retrieved per query
GAM max depth	3	Maximum reflection iterations for multi-hop retrieval
GAM max pages	5	Maximum pages retrieved per research query

8. Continual Learning: Nested Learning Integration

Financial markets evolve continuously, requiring agents to learn new strategies (e.g., Q4 tax-loss harvesting) without forgetting existing capabilities (e.g., momentum trading). Nested Learning provides the theoretical framework for multi-timescale updates that prevent catastrophic forgetting.

8.1 Memory Types

ATHENA distinguishes between two complementary memory systems:

Memory Type	Implementation	Contents
Procedural Memory (CMS)	Nested Learning Continuum Memory System - internal model weights at different update frequencies	'How to trade' - strategies, patterns, decision rules encoded in parameters
Episodic Memory (GAM)	GAM page-store - external experience storage with JIT retrieval	'What happened' - specific trades, outcomes, market conditions

8.2 Multi-Timescale Updates

Parameters are partitioned into fast-adapting and slow-adapting groups:

- **Fast Parameters:** LoRA adapters, attention heads - update frequently to adapt to current market regime
- **Slow Parameters:** Base model weights - update infrequently to preserve core capabilities
- **Frozen Parameters:** OLMo 3 foundation layers - never updated to maintain reasoning and language abilities

9. Data Flow Architecture

This section describes how data flows through ATHENA during both inference (live trading) and training (evolution) phases.

9.1 Inference Flow (Live Trading)

1. Market data arrives from external APIs (Yahoo Finance, Alpha Vantage)
2. Coordinator receives query and routes to appropriate agent(s)
3. Each agent queries GAM Researcher for relevant historical context
4. Agents exchange latent thoughts via LatentMAS Shared Working Memory
5. Strategy Agent formulates recommendation with explicit reasoning chain
6. Risk Manager validates against portfolio constraints
7. Execution Agent places orders via broker APIs
8. Complete trajectory logged to GAM page-store

9.2 Training Flow (Evolution)

1. Self-Questioning: Task Generator creates diverse trading scenarios
2. Rollout: Agents execute tasks with experience injection (η=0.5)
3. Outcome: Trading simulation returns profit/loss
4. Attribution: LLM labels each step as GOOD/BAD with reasoning
5. Reward: Composite reward computed (α=0.1 attribution weight)
6. Diversity Bonus: RepExp adds representation-based exploration bonus
7. Update: GRPO with Nested Learning multi-timescale updates
8. Memory: Successful trajectories added to GAM page-store

10. Technology Stack

Category	Technology	Purpose
Foundation Model	OLMo 3 (7B/32B variants)	Base LLM with full transparency
Fine-tuning	LoRA + QLoRA	Parameter-efficient agent specialization
Training Framework	PyTorch + Hugging Face Transformers	Model training and inference
RL Framework	TRL (Transformer Reinforcement Learning)	GRPO implementation
Vector Database	Qdrant / Pinecone	GAM page-store
Embeddings	sentence-transformers (all-MiniLM-L6-v2)	Experience and thought embeddings
Orchestration	Apache Airflow	Training pipeline scheduling
Compute	NVIDIA H100 / A100 GPUs	Model training and inference
Cloud	AWS / GCP	Infrastructure hosting
Data APIs	Yahoo Finance, Alpha Vantage, IEX Cloud	Market data ingestion
Broker APIs	Interactive Brokers, Alpaca	Trade execution

11. Implementation Roadmap

11.1 Phase 1: MVP (Weeks 1-6)

Goal: Basic multi-agent system with AgentEvolver + GAM

- Set up OLMo 3-Base 32B with LoRA adapters for each agent
- Implement AgentEvolver self-questioning with financial environment profile
- Deploy GAM page-store with Qdrant
- Build basic Coordinator for task routing
- Milestone: Agent successfully generates and learns from self-created tasks

11.2 Phase 2: Communication (Weeks 7-10)

Goal: Add LatentMAS for efficient agent communication

- Implement latent thought generator (last-layer embedding extraction)
- Build shared working memory buffer
- Replace text-based agent communication with latent exchange
- Benchmark communication latency improvement
- Milestone: 10x+ speedup in multi-agent coordination

11.3 Phase 3: Diversity (Weeks 11-14)

Goal: Add RepExp for strategy diversity

- Implement representation-based exploration bonuses
- Add elliptic bonus computation from hidden states
- Tune diversity hyperparameters for financial domain
- Milestone: Maintained strategy diversity across 100+ training epochs

11.4 Phase 4: Continual Learning (Weeks 15-20)

Goal: Add Nested Learning for multi-timescale updates

- Partition parameters into fast/slow/frozen groups
- Implement Continuum Memory System (CMS)
- Test adaptation to new market regimes without forgetting
- Milestone: Learn Q4 tax-loss harvesting while retaining momentum strategies

12. Success Metrics

Metric	Baseline	Target	Measurement
Task Completion Rate	Single-agent: 60%	> 85%	% of generated tasks completed successfully
Communication Latency	Text-based: 500ms	< 50ms	Time for agent-to-agent message
Memory Retrieval F1	Top-k: 0.45	> 0.75	Relevance of retrieved experiences
Strategy Diversity	After 50 epochs: 0.3	> 0.7	pass@k maintained for large k
Catastrophic Forgetting	Standard: 40% degradation	< 10%	Performance on old tasks after new learning
Sample Efficiency	Standard RL: 1x	> 3x	Tasks to reach performance threshold

13. Risk Assessment

Risk	Likelihood	Impact	Mitigation
Computational Cost	High	Medium	Start with 7B models, use QLoRA, implement phased rollout
LatentMAS Compatibility	Medium	High	All agents share OLMo 3 base, verify embedding alignment
GAM Latency	Medium	Medium	Cache frequent queries, limit reflection depth for time-critical
Reward Hacking	Medium	High	Human oversight, conservative position limits, diverse reward signals
Market Regime Shift	High	High	Nested Learning multi-timescale, explicit regime detection
Integration Complexity	High	Medium	Phased implementation with clear milestones and rollback

14. References

1. *AgentEvolver: Self-Evolving Agents through Self-Questioning, Self-Navigating, and Self-Attributing.* ModelScope/Alibaba, 2025. arxiv.org/abs/2511.10395
2. *GAM: General Agentic Memory via Deep Research.* VectorSpaceLab, 2025. arxiv.org/abs/2511.18423
3. *LatentMAS: Enabling Large Language Model-Based Multi-Agent Systems to Collaborate in Latent Space.* 2025. arxiv.org/abs/2511.20639
4. *Nested Learning: Learning to Learn from Experience.* Google DeepMind, NeurIPS 2025.
5. *Representation-Based Exploration for Diverse LLM Solutions.* 2025. arxiv.org/abs/2510.11686
6. *OLMo 3: Charting a Path Through the Model Flow.* Allen Institute for AI, 2025. allenai.org/olmo

Appendix A: Financial Environment Profile

The following environment profile is used by the Self-Questioning mechanism to generate diverse financial tasks:

A.1 Entities

- **portfolio:** holdings, cash_balance, total_value, risk_score
- **market_data:** price, volume, volatility, trend
- **indicators:** rsi, macd, bollinger_bands, moving_averages
- **events:** earnings, fed_meeting, economic_release, news
- **risk_metrics:** var, sharpe_ratio, max_drawdown, beta

A.2 Operations

analyze_market, buy, sell, hold, rebalance, hedge, calculate_risk, backtest

A.3 Difficulty Levels

- **Easy:** 2 entities, 2 attributes, 1 operation (e.g., 'Analyze AAPL trend')
- **Medium:** 3 entities, 3 attributes, 2 operations (e.g., 'Buy if RSI < 30 and MACD bullish')
- **Hard:** 4 entities, 4 attributes, 3 operations (e.g., 'Rebalance portfolio considering earnings, VaR limit, and Fed meeting')

Appendix B: Code Repositories

Component	Repository	License
AgentEvolver	github.com/modelscope/AgentEvolver	Apache 2.0
GAM	github.com/VectorSpaceLab/general-agentic-memory	MIT
OLMo 3	huggingface.co/allenai/OLMo-3-32B	Apache 2.0
TRL (GRPO)	github.com/huggingface/trl	Apache 2.0
sentence-transformers	github.com/UKPLab/sentence-transformers	Apache 2.0

— End of Document —