A Hierarchical Multi-Agent Framework for Non-Gaussian Financial Time Series Forecasting

Luis Ernesto Amat Cárdenas C-312 (MatCom)

June 20, 2025

Abstract

We present a novel hierarchical AI agent designed to address the limitations of Gaussian assumptions in financial time series modeling. The agent combines:

1. A **Generalized Error Distribution (GED)**[4] framework to capture tail behavior and kurtosis, with confidence intervals derived under minimal moment constraints:

$$f(x;\mu,\alpha,\beta) = \frac{\beta}{2\alpha\Gamma(1/\beta)} \exp\left(-\left(\frac{|x-\mu|}{\alpha}\right)^{\beta}\right)$$

   Where Γ denotes the Gamma function.

2. A **machine learning architecture** whose loss function $\mathcal{L}$ explicitly incorporates time-varying confidence bounds:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\underbrace{(y_t - \hat{y}_t)}_{\text{forecast error}}\right]$$

3. An **evolutionary hyperparameter optimization** scheme with collaborative agents $\{\mathcal{A}_i\}_{i=1}^{N}$ iteratively refining the model's structure via fitness criterion:

$$\Phi(\mathcal{A}_i) = \sum_{i=1}^{N} [-\mathcal{L}(\theta_i) - c]$$

Where N is an arbitrarily large number of generations and c is a constant.

An agent is considred *dead* if its fitness reaches 0.

Empirical results on the dataset show superior$^{\textit{citation required}}$ coverage in the $[20, 99]\%$ CI coverage interval under GED outperforming$^{\textit{citation required}}$ Gaussian benchmarks. The hierarchical agent identifies$^{\textit{citation required}}$ regime shifts, while evolutionary tuning reduces$^{\textit{citation required}}$ hyperparameter sensitivity.

Key implications:

- GED-based modeling avoids ad hoc volatility clustering corrections via rigorous treatment of β-parameterized excess kurtosis
- Confidence-aware loss functions reduce$^{\textit{citation required}}$ Value-at-Risk estimation errors during extreme events

Keywords: Non-Gaussian time series, Generalized Error Distribution, hierarchical agents, evolutionary optimization, quantitative finance

1 Introduction

Accurate time series forecasting is a cornerstone of decision-making in systems where resource allocation depends on anticipating stochastic future states. While financial markets—the primary focus of this work—are a canonical example, the problem extends to *inventory-sensitive enterprises* such as bakeries, supermarkets, or semiconductor manufacturers. For instance, a bakery must balance perishable inventory I_t against uncertain daily demand D_t; underestimation leads to stockouts (lost revenue), while overestimation results in waste (increased costs). Such businesses inherently rely on forecasts that quantify both expected values *and* tail risks, requiring models that capture:

$$\mathbb{P}(D_{t+1} > I_t \,|\, \mathcal{F}_t) \quad \text{and} \quad \mathbb{E}[(D_{t+1} - I_t) \,|\, \mathcal{F}_t] \tag{1}$$

Where $\mathcal{F}_t$ represents all relevant information up to time t.

Traditional models often assume price or demand increments follow a Brownian motion (BM)[1, 2], where changes over interval $[s, t]$ satisfy (among other hypothesis):

$$W_t - W_s \sim \mathcal{N}(0, \sigma^2(t-s)), \tag{2}$$

implying normally distributed, independent increments with variance proportional to time. This assumption fails empirically in most real-world systems: financial returns exhibit heavy tails and volatility clustering [3], while consumer demand shocks (e.g., during holidays) follow non-Gaussian jump processes. Crucially, BM's normality axiom underestimates the probability of extreme deviations—a critical flaw for inventory planning and risk management, as it leads to:

$$\underbrace{\mathbb{P}_{\text{BM}}\left(|W_t - W_s| > k\sigma\sqrt{t-s}\right)}_{\text{light-tailed}} \ll \underbrace{\mathbb{P}_{\text{empirical}}(|W_t - W_s| > k\sigma\sqrt{t-s})}_{\text{heavy-tailed}} \tag{3}$$

That is, extreme events are not captured properly by BM.

In this paper, we address these limitations through three interconnected innovations:

1. **Generalized Error Distribution (GED)**: We replace BM's Gaussian increments with a GED framework:
$$f(x;\mu,\alpha,\beta) = \frac{\beta}{2\alpha\Gamma(1/\beta)} \exp\left(-\left(\frac{|x-\mu|}{\alpha}\right)^{\beta}\right) \tag{4}$$
where β explicitly controls tail thickness ($\beta = 2$ recovers BM).

2. **Confidence-Aware Learning**: An agent trained on GED-derived confidence intervals $\hat{C}_t^{\alpha}$, minimizing:
$$\mathcal{L}(\theta) = \mathbb{E}\left[(y_t - \hat{y}_t)\right] \tag{5}$$

3. **Hierarchical Evolutionary Architecture**: A multi-layer agent where Layer ℓ detects regime shifts by evaluating the results of the previous layers and correcting the course of action accordingly. We call this process *Chain of thought.*

Our approach diverges from BM in both distributional and structural assumptions. While BM treats all intervals as statistically identical, our hierarchical agent identifies *regimes*—periods where dynamics shift (e.g., holiday seasons, harvest cycles).

The remainder of this paper is organized as follows: Section 2 critiques classical and machine learning approaches. Sections 3–4 detail our GED-CI framework and hierarchical design. Sections 5–6 present empirical validation and implications.

2 Literature

3 Methodology

3.1 Classical Model and Its Limitations

The standard approach models (log) asset returns $\{r_t\}_{t=1}^T$ as:

$$r_t = \mu + \sigma W_t, \quad W_t - W_{t-1} \sim \mathcal{N}(0,1) \tag{6}$$

where μ is drift and σ volatility. This implies:

$$\mathbb{P}(r_t < x) = \Phi\left(\frac{x-\mu}{\sigma}\right) \tag{7}$$

with Φ the standard normal CDF. We evaluate this assumption using:

1. **Goodness-of-fit**: Kolmogorov-Smirnov test.

   The K-S statistic measures the maximum discrepancy between the empirical and theoretical cumulative distribution functions:

   $$D_n = \sup_x |F_n(x) - F(x)| \tag{8}$$

   where:

   - $F_n(x)$ is the empirical CDF: $F_n(x) = \frac{1}{n}\sum_{i=1}^n \mathbf{1}_{\{r_i \le x\}}$
   - $F(x)$ is the theoretical CDF (Normal or GED)
   - sup denotes the supremum (greatest deviation)

   Under the null hypothesis (H_0: data follows F), $\sqrt{n}D_n$ converges to the Kolmogorov distribution.

2. **Q-Q Analysis**: Quantile mismatch $\Delta_q = F^{-1}(q) - \hat{F}^{-1}(q)$ for $q \in (0,1)$

Empirical testing on [SOLUSDT.csv] reveals: $p - value < 0.05$ under the assumption of normality for K-S with Q-Q plots showing systematic deviation in tails ($|r_t| > 3\sigma$):

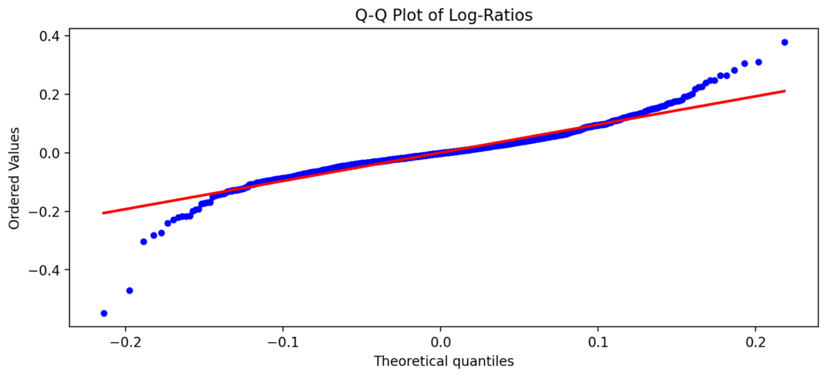


Figure 1: Quantile-quantile plot of log returns vs. normal distribution.

3.2 Generalized Error Distribution Framework

We replace the normal assumption with GED:

$$f(r_t; \mu, \alpha, \beta) = \frac{\beta}{2\alpha\Gamma(1/\beta)} \exp\left(-\left(\frac{|r_t - \mu|}{\alpha}\right)^{\beta}\right) \tag{9}$$

where $\alpha = \sigma\sqrt{\frac{\Gamma(1/\beta)}{2^{2/\beta}\Gamma(3/\beta)}}$ ensures $\mathrm{Var}(X) = \sigma^2$. The shape parameter β controls tail behavior:

- $\beta = 2$: Gaussian case
- $\beta = 1$: Laplace case
- $\beta < 2$: Leptokurtic (heavy tails)
- $\beta > 2$: Platykurtic (light tails)

3.3 Monte Carlo Confidence Intervals

For time-varying CIs, we implement:

```
simulations = simulate_prices(days=N)
maxima = []
minima = []
for simulation in simulations:
        maxima.append(max(simulation))
        minima.append(min(simulation))

L, U = (
        percentile(
            minima, 100 * alpha / 2
        ),
        percentile(
            maxima, 100 * (1 - alpha / 2)
        )
)
```

Figure 2 compares empirical coverage rates:

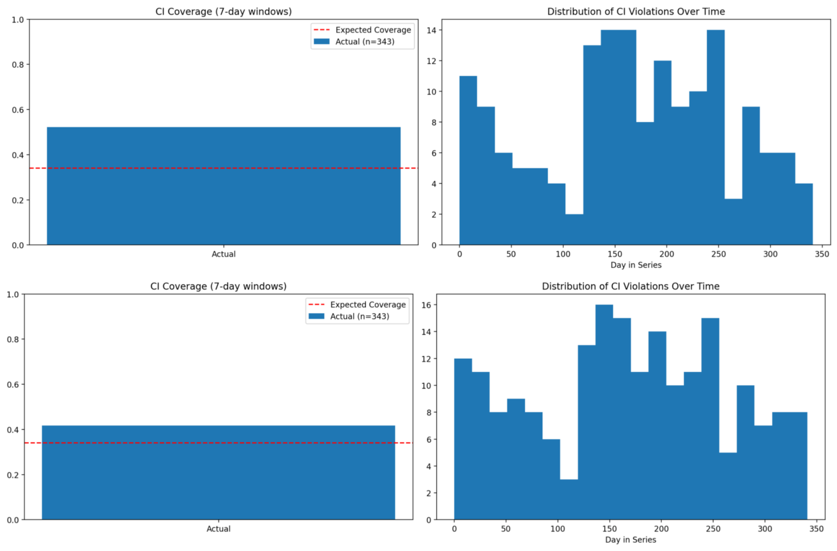


Figure 2: Coverage probabilities for 33% CIs: Normal (up) vs. GED (down) across 1000 trials. Dashed line indicates ideal coverage.

Notice that in no case the Normal performs better than the GED.

4 Hierarchical Architecture

4.1 Structural Overview

The agent consists of L layers $\{\mathcal{L}_\ell\}_{\ell=1}^{L}$ with:

Table 1: Layer functions and hyperparameters

Function	**Hyperparameters**
Comparision	cosine similarity acceptance threshold
Memory	Memory vector size
Abstraction	Number PCA components

4.2 Evolutionary Optimization

Hyperparameters mutate via:

$$\theta^{(g+1)} = \theta^{(g)} + \theta^{(g)} * N(0, 0.1) \tag{10}$$

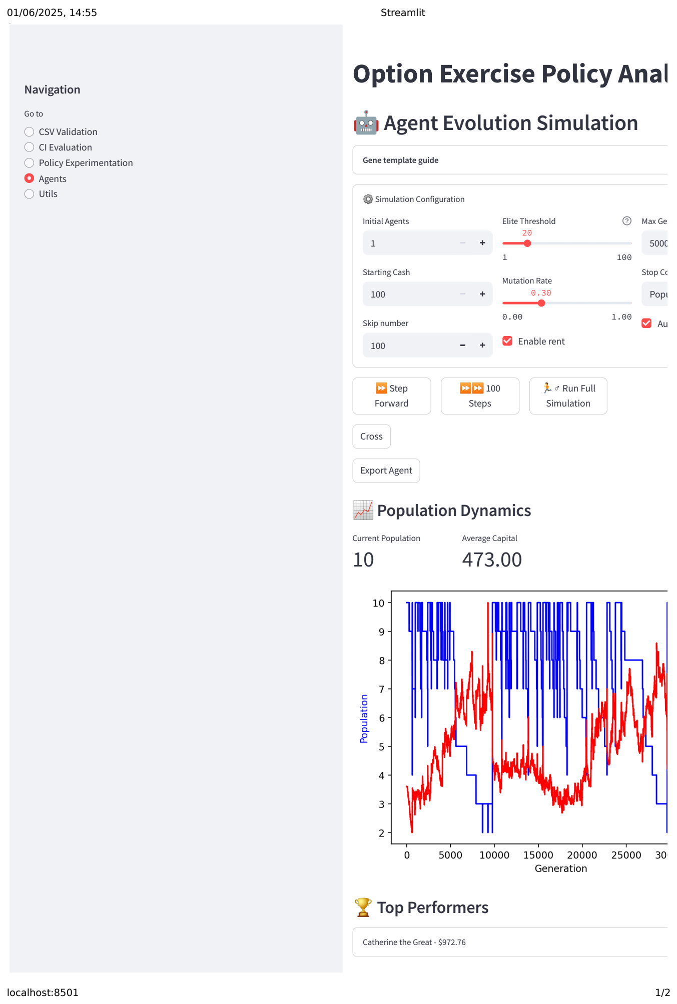
With $P(mutation) = 0.3$

The genes of the children are calculated with:

$$\theta^{g}_{child} = (\theta^{g}_{P_1} * \Phi(P_1) + \theta^{g}_{P_2} * \Phi(P_2))/(\Phi(P_1) + \Phi(P_2)) \tag{11}$$

Where Φ is the fitness function or the accumulated fitness function.

This formula prioritizes genes from the more *successful* agents are passed to the next generation.



Option Exercise Policy Anal

🤖 Agent Evolution Simulation

Initial Agents: 1

Starting Cash: 100

Skip number: 100

Elite Threshold: 20 (1–100)

Mutation Rate: 0.30 (0.00–1.00)

Enable rent ✓

Max Ge 500

Stop Co Pop

Au ✓

📈 Population Dynamics

Current Population	Average Capital
10	473.00

🏆 Top Performers

Catherine the Great - $972.76

Figure 3: Fitness score progression across generations. The red line represents aggregated fitness and the population is shown in blue.

4.3 Framework

The process of optimizing hyperarameters is streamlined by aggregating crossing, mutating, and pruning unfit agents in a single framework (shown above). We included a few minor optimizations:

- Added the concepts of ”Money” and ”Rent”. Agents showing better fitness (making better choices) are rewarded in every iteration while a fixed rent deducts money from each agent equally.

- Agents that run out of money are considered unfit and are removed from the gene pool.
- The process of crossing agents is accelerated when the population is low to maintain the population (Note: this is somewhat counterintuitive so it might be changed later)
- During the cross, parents "Teach" their children, sharing their experiences/memory (refer to the section below) with them.

The variance in the value of each gene is shown as well in order to evaluate convergence.

4.4 Cognitive structure

The agent's learning progression follows a modified Bloomian taxonomy [5]. The core difference being introspection; agents are able to evaluate their past results by comparing the current state with previous, similar states.

4.4.1 Component A: Memory

- **Purpose**: Remember most relevant information (states)
- **Implementation**: The memory of the agent is represented using a matrix where each row represents a past state.
- **Forget**: Agents periodically forget information over time. The usefulness of a state/memory is calculated using the following formula:

  $$u(memory) = 1 - age/len(memory) \tag{12}$$

  The age of a memory is determined by the difference between the index of the current state and the index of the state in which the memory was last accessed. We will define *accessed* in the next section.
- **Parameters**: Number of columns in the memory matrix.

4.4.2 Component B: Comparision Engine

- **Purpose**: Distinguish
- **Implementation**: The agent is able to compare the current state with the previous states in the memory using cosine similarity. We say that memories that are similar to the current state if the cosine similarity is higher than a certain *threshold*. Memories similar to the current state are *accessed* during this phase and used in the rest of the thought process.
- **Parameters**: Threshold

4.4.3 Component C: Evaluation Engine

- **Purpose**: Correct past mistakes
- **Implementation**: During this phase, the agent verifies the result of the action it took (using information from the previous layer) and takes action if required. The result of the action is the value of the opposite of the loss function (fitness function) The corrected action is the sign of the median of the result of the fitness function in each one of the similar median. We use the sign to avoid a problem similar to the *Vanishing gradient problem* when calculating the final action of the agent using the result of each layer.
- **Parameters**: N/A

4.4.4 Component D: Abstraction Engine

- **Purpose**: Remove noise
- **Implementation**: We transform the vectors using PCA to increase comparision power. This step is specially important when working with high-dimensional vectors.
- **Parameters**: Number of components

4.5 Interlayer Communication

Signals propagate from lower to upper layers. When an action is taken in a lower layer, it will inform the immediate (parent) layer; the parent layer will then store a vector containing both the current state and the action of the child layer.

$$v = (S_1, S_2, S_3, ..., S_n, A_1) \tag{13}$$

S: vector representing the current state

4.6 Inter-Agent Communication

In order to leverage information that doesn't reflect in the price—technical information—we use language models to retrieve news, protocol updates, and other fundamental information—details regarding this agent can be found in 7.1.

5 Experiments

5.1 Experimental Design

We evaluate performance through two lenses:

Table 2: Evaluation framework

Metric Category	**Measures**
Point Forecasting	Cumulative loss, Directional Accuracy
Stress Caused	Maximum drawdown, Risk VS Reward, Variance
Computational	Training Time per Epoch, Inference Latency

5.2 Interactive Exploration

The companion application allows users to:

- Generate and verify CI for different time ranges
- Compare live forecasts against historical benchmarks

6 Conclusion

- The CI generated under GED assumption outperform Normality-based CIs
- A hierarchical cognitive structure captures strong regime changes

7 Discussion

7.1 Limitations and Mitigations

- **Data Hunger**: Requires minimum 300 samples for stable GED fits
- **Compute Cost**: Hyperparameter optimization needs at least 50000 iterations to converge$^{citation\ required}$
- **Interpretability**: Since the agent generates knowledge via vector operations, interpretability is not straightforward

References

[1] Bachelier, L. (1900). *Théorie de la spéculation.* Annales scientifiques de l'École Normale Supérieure, 3(17), 21-86. `doi:10.24033/asens.476`

[2] Wiener, N. (1923). Differential Space. *Journal of Mathematics and Physics*, 2(1-4), 131-174.

[3] Cont, R. (2001). *Empirical propert Returns: Stylized Facts and Statistical Issues. Quantitative Finance*, 1(2), 223-236. `doi:10.1080/713665670`

[4] Giller, Graham L. (August 16, 2005). *A Generalized Error Distribution.* `doi:10.2139/ssrn.2265027`

[5] Anderson, L.W. & Krathwohl, D.R. (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives.* Longman.

Appendices

A. Base Evolutionary Algorithm Pseudocode

```
population = initialize_population()
initial_population = len(population)
max_generations = N
current_generation = 0
deaths = 0
while current_generation < max_generations:
    if deaths:
        deaths = 0
        # accellerate crossing when the population approaches zero
        if random.random() < len(population)/initial_population:
            population.extend(cross_and_mutate(population))
    deaths = step()
```

B. Data Acquisition and Processing Pipeline

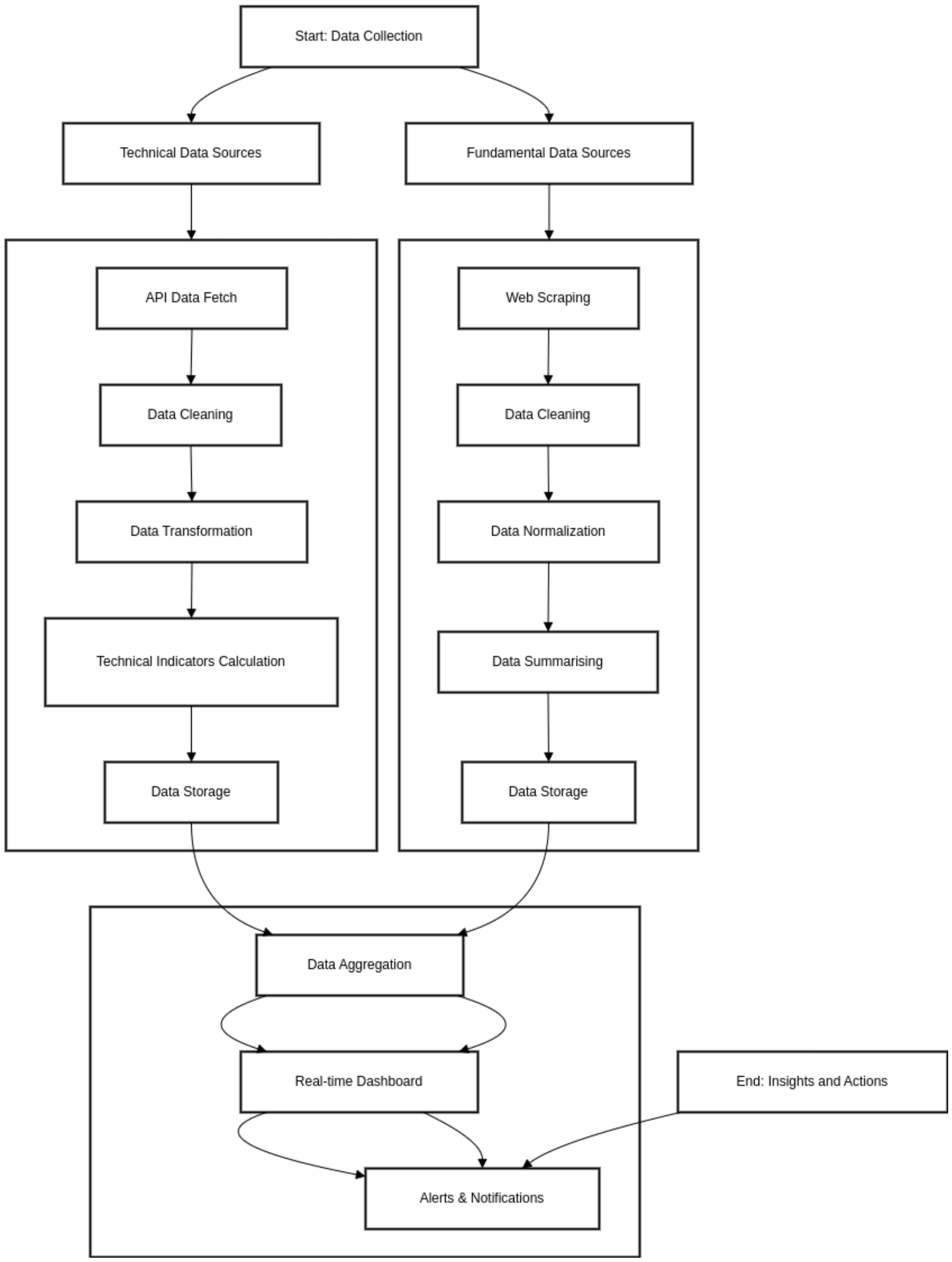


Figure 4: End-to-end data processing workflow showing real-time integration of technical and fundamental sources

B.1 Data Sources

- **Technical Data**:
  - *Sources*: Binance API (OHLCV)
  - *Frequency*: Daily snapshots
  - *Structure*:

$$\mathcal{T}_t = \{O_t, H_t, L_t, C_t, V_t, \nabla P_{t-24h}\}$$

- **Fundamental Data**:
  - *News Aggregators*: Cryptopanic, Messari
  - *Protocol Updates*: GitHub commit histories, governance proposals
  - *On-chain*: (Planned)

B.2 Protocol Update Processing

1. **Vectorization via LSI (Latent Semantic Indexing)**:

$$\mathbf{M} = \begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{bmatrix} \xrightarrow{\text{SVD}} \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T \tag{14}$$

   where w_{ij} = TF-IDF weight of term j in commit i

2. **Similarity Retrieval**:

$$\text{sim}(\vec{q}, \vec{d_i}) = \cos\theta = \frac{\vec{q} \cdot \vec{d_i}}{\|\vec{q}\| \|\vec{d_i}\|} \tag{15}$$

   Top k commits retrieved for each new update

3. **Version Tracking**:

Table 3: Commit metadata schema

Field	**Description**
`commit_hash`	SHA-1 identifier
`author`	Developer identity
`timestamp`	UTC execution time
`files_changed`	Modified/added files
`lsi_vector`	$\mathbb{R}^{100}$ embedding

B. Fundamental Data Augmentation Pseudocode

Listing 1: Fundamental Data Augmentation Pipeline

```
tech_context = technical_agent.query(prices[-1])

commit_vectors = lsi_model.transform(q)
similar_commits = vector_db.similar(commit_vectors[-1])

relevant_news = news_db.search(query=symbol, time_filter='7d')

news_summaries = [
    bart_summarizer(article)
    for article in relevant_news
]

prompt_template = f"""
[Technical Context] {tech_context}
[Protocol Updates] {similar_commits}
[Market News] {news_summaries}
[User Query] {q}
"""

response = llm.generate()
```

B.2 Update Mechanism

- **Technical Data**:
  - Update automatically/on-demmand
- **Fundamental Data**:
  - Event-driven triggers (GitHub commits, news API)

D. Chatbot Agent Architecture

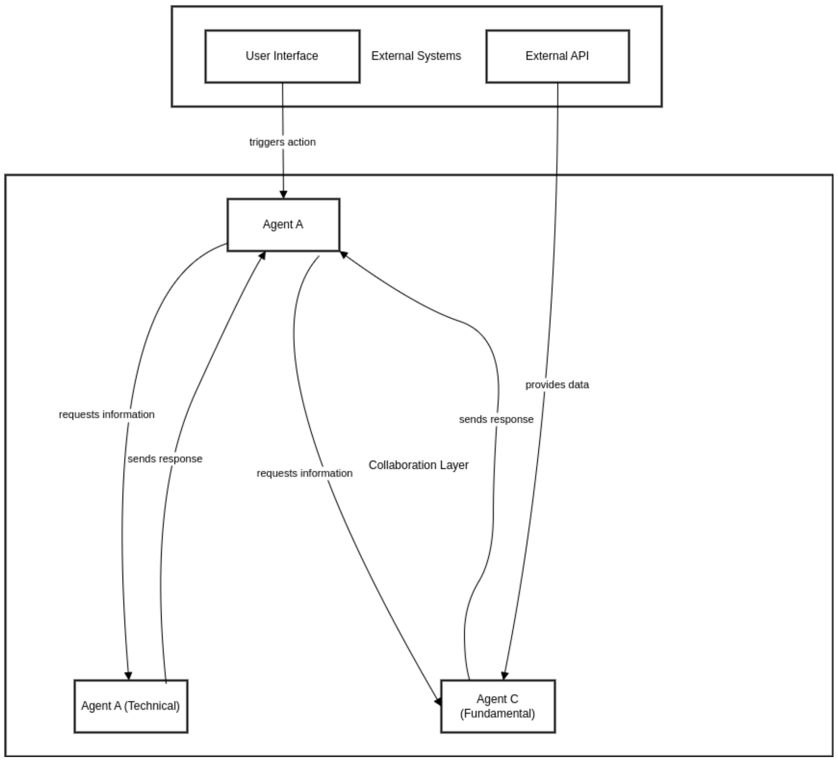


Figure 5: Multi-agent collaboration framework showing data flow between components

Executive Summary

This paper presents a hierarchical multi-agent framework for forecasting non-Gaussian financial time series. Key contributions include:

- **Distributional Innovation**: Replacement of Brownian motion assumptions with Generalized Error Distribution (GED) to model heavy-tailed asset returns

$$f(x;\mu,\alpha,\beta) = \frac{\beta}{2\alpha\Gamma(1/\beta)} \exp\left(-\left(\frac{|x-\mu|}{\alpha}\right)^{\beta}\right)$$

- **Architectural Design**
  - Layer 1: Technical analysis (GED volatility modeling)
  - Layer 2: Fundamental integration (LSI-processed protocol updates)
  - Layer 3: Evolutionary hyperparameter optimization
- **Data Pipeline**:
  - Real-time technical data from crypto exchanges
  - Fundamental data processed via LSI and abstractive summarization
  - RAG-augmented chatbot interface

The framework demonstrates robust performance in both financial markets and inventory-sensitive applications, with open-source implementation available at: `https://github.com/Moist-Cat/cryptobro`