Real-Time Risk Watcher Agents Design

The **Real-Time Risk Watcher** architecture is a multi-agent system designed for continuous financial risk monitoring. It consists of several specialized agents that work together to ingest data, compute risk metrics, analyze market signals, and provide human-readable explanations. These agents communicate via a publish/subscribe message bus, where each agent publishes its outputs to specific topics and subscribes to relevant input topics. This modular design allows the system to be scalable and real-time: as new data arrives (market prices, news, filings, etc.), each agent reacts and updates its analysis accordingly.

We assume integrations with external data sources such as *AlphaVantage* (for real-time market and fundamental data, as well as news feeds) and *EDGAR* (the SEC's company filings database) for raw inputs. We also leverage modern AI capabilities: for example, the ExplanationAgent uses the **LLaMA 5** large language model as its base to generate natural language insights, reflecting the growing role of LLMs in fintech by 2025:contentReference[oaicite:0]{index=0}. The UI (dashboard) subscribes to the outputs of these agents to display up-to-date risk information and alerts to users.

The following sections detail each agent in the architecture. For each agent, we describe its purpose, inputs/outputs (including data structures and message topics), internal logic and algorithms (with formulas or pseudocode where appropriate), and how its outputs are used by other agents or the user interface.

Agent	Role/Purpose	Key Inputs (Topics/Data)	Key Outputs (Topics/Data)
IngestionAgent	Fetches and publishes raw market and textual data in real-time.	External APIs (AlphaVantage prices, news; EDGAR filings); config of tickers.	Market data stream (e.g. `market_data` topic), news/filings feed (e.g. `news_feed` topic).
RiskMetricsAgent	Calculates core risk metrics (market risk indicators).	Market data (prices, possibly portfolio positions).	Risk metrics (e.g. VaR, volatility, beta) on `risk_metrics` topic.
SentimentAgent	Analyzes news and textual data for sentiment signals.	News/filings text (from `news_feed` or EDGAR data).	Sentiment scores or alerts per asset or market (`sentiment_signals` topic).
RegimeShiftDetector	Detects shifts in market regime (bull/bear, volatility regimes).	Market data (index prices, volatility metrics).	Regime state updates (`regime_state` topic).
CreditRiskAgent	Assesses credit risk of portfolio entities.	Fundamental data (financial ratios, debt levels from EDGAR).	Credit risk metrics (e.g. credit score or probability of default) on `credit_risk` topic.
OptionsGreeksAgent	Calculates derivative risk sensitivities (options Greeks).	Options market data (option chain quotes, positions).	Greeks values (per option and aggregated) on `options_greeks` topic.
ExplanationAgent	Generates natural-language explanations of the risk state using LLM.	All other agents' outputs (risk metrics, sentiment, regime, etc.).	Explanatory text or summarized report on `risk_explanation` topic (for UI).

Core Architectural Agents

IngestionAgent

Purpose & Role: The IngestionAgent is the pipeline's entry point, responsible for collecting real-time data from external sources and injecting it into the system. It ensures that all other agents have up-to-date information. In essence, this agent bridges the outside world (markets and data feeds) with the internal message bus. Its role includes fetching **market prices**, **news articles**, and **financial filings**, then normalizing and publishing them to appropriate internal topics for downstream consumption.

Inputs & Outputs: The IngestionAgent takes in configuration inputs (for example, a list of stock tickers or financial instruments to track, API keys, and polling frequency). It may also receive trigger messages (e.g. a request to fetch a specific dataset on demand), though primarily it operates on a schedule or event basis. It connects to external APIs:

- *AlphaVantage API* – for real-time and historical price data (OHLCV) for stocks, indices, etc., as well as technical indicators and news feeds. For example, it might call AlphaVantage's time-series endpoint to get the latest price for each ticker every minute.
- *News & Sentiment feeds* – using AlphaVantage's news API or other providers to get breaking news headlines and any sentiment metadata. (AlphaVantage offers a News & Sentiments API that aggregates news and provides sentiment scores:contentReference[oaicite:1]{index=1}.)
- *EDGAR filings* – monitoring the SEC EDGAR database for new 10-K/10-Q filings or updates. This might be done via EDGAR's RSS feed or API: the agent checks for newly filed reports for companies of interest and downloads the files or extracts key sections (like Risk Factors or financial statements).

After gathering data, the IngestionAgent outputs standardized messages to the internal bus. Key output topics and data formats include:

- `market_data`: streaming market quotes and possibly aggregated bars. Each message could be a JSON object like:

  ```
  { "symbol": "AAPL", "price": 150.23, "volume": 1203450, "timestamp": "2025-12-14T21:30:00Z" }
  ```

  This example shows a single-ticker update with fields for price, volume, timestamp, etc. The agent might publish such updates for each symbol as data arrives (e.g., per tick or per minute).
- `news_feed`: a stream of news items or filings. For instance, a JSON message might contain a headline or filing excerpt, a source, and timestamp:

  ```
  { "type": "news", "symbol": "MSFT", "headline": "Microsoft announces new product line in AI segment", "summary": "Microsoft unveiled ...", "sentiment
  ```

  Here `sentiment` could be left null at ingestion time (to be filled by the SentimentAgent). Similarly, for EDGAR filings, the message might have `"type": "filing"` and include a link or extracted key data (e.g., parsed financial ratios).

The IngestionAgent ensures data formats are consistent (e.g., uses ISO timestamps, uniform JSON schema) so that downstream agents can easily parse them. It also handles any necessary transformations (currency conversion if needed, filtering out irrelevant data, etc.). If data quality issues occur (missing values, API downtime), the IngestionAgent might log errors or send an alert message.

Internal Logic & Algorithms: Internally, the IngestionAgent likely runs as a scheduler or event-driven fetcher. Pseudocode for its operation could look like:

```
# Pseudocode for IngestionAgent main loop for each asset in config.asset_list: schedule task to fetch market price every X seconds schedule task to fetch r
```

This outline shows the agent polling various sources. In a real implementation, it might use asynchronous IO or multi-threading to handle multiple data streams concurrently. For example, real-time price streams could be pushed via websocket or pub/sub from a provider – the agent would subscribe and then re-publish internally. The logic also includes parsing responses into the internal format (the `normalize_market_data`, `format_news_item` functions handle JSON shaping).

Output Usage: The data published by IngestionAgent is consumed by almost all other agents:

- The **RiskMetricsAgent** subscribes to `market_data` to update risk calculations whenever new prices come in.
- The **SentimentAgent** subscribes to `news_feed` (especially items of type "news" or textual content) to perform sentiment analysis on headlines or filings text.
- The **RegimeShiftDetector** may subscribe to broad market index data (e.g., S&P 500 from `market_data`) or volatility indices (which could also be ingested) to detect regime changes.
- The **CreditRiskAgent** uses fundamental data; if the IngestionAgent extracts financial ratios or indicators from EDGAR filings, these are fed to the CreditRiskAgent (possibly via the same `news_feed` topic or a dedicated `fundamentals_data` topic).
- The **OptionsGreeksAgent** might get underlying asset prices from `market_data` and option chain info from a special ingestion call (AlphaVantage also provides some options data if available, or another source).

In summary, IngestionAgent's outputs form the live data foundation that all analytic agents rely on. Without timely ingestion, the rest of the system cannot function in real-time.

RiskMetricsAgent

Purpose & Role: The RiskMetricsAgent is the core analytical engine that computes quantitative market risk indicators. Its primary role is to continuously evaluate how risky the tracked portfolio or set of assets is, given the latest market moves. Key metrics produced include volatility, Value-at-Risk (VaR), and other related measures (potentially Beta, correlations, or stress test results). These metrics give a sense of market risk exposure and are fundamental to the system's risk awareness.

For instance, VaR answers "what is the maximum expected loss over a given time horizon at a certain confidence level?":contentReference[oaicite:2]{index=2}. Volatility measures the degree of price variation (riskiness) of an asset. By computing these in real-time, the RiskMetricsAgent provides immediate insight into risk changes (e.g., if market volatility spikes or if the portfolio's VaR suddenly increases, indicating higher potential losses).

Inputs & Outputs: The RiskMetricsAgent subscribes to:

- `market_data`: it receives price updates for relevant assets. This could include prices for each asset in a portfolio or a set of assets being monitored. The agent typically maintains an internal buffer or time-series of recent prices/returns for calculations.
- `portfolio_positions` (if applicable): information on the holdings and weights of each asset in the portfolio. This could be a static configuration or a dynamic input if the portfolio changes. For example, a JSON could define positions like `{"AAPL": 50 shares, "MSFT": 30 shares, ...}`. If the system is focusing on a single asset or index risk, this may not be needed, but generally for VaR a portfolio context is considered.

Outputs are published on a `risk_metrics` topic (or similar). The output data structure can be a JSON containing various calculated metrics. For example:

```
{ "timestamp": "2025-12-14T21:35:00Z", "portfolio": "DefaultBook", "VaR_95": 250000.0, /* 95% one-day VaR in dollars */ "volatility_annualized": 0.18, /* e
```

This indicates at the current time, the portfolio's 95% one-day VaR is $250k, etc. The agent might also output component metrics per asset (e.g., contribution of each asset to VaR) or other statistics depending on sophistication.

Internal Logic & Algorithms: Upon receiving new market data, the RiskMetricsAgent updates its risk calculations. It typically performs the following steps:

1. *Update Price Histories:* The agent maintains a window of recent prices or returns for each asset. For example, it might keep the last N days of daily returns or last M intraday returns. Each new price allows computing the latest return and appending to the series (possibly dropping the oldest data point if using a rolling window).
2. *Compute Volatility:* Using the return series, it calculates volatility. If using daily returns, the formula might be: $$\sigma_{\text{daily}} = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(r_t - \bar{r})^2}$$ (standard deviation of returns). This can be annualized by $\sigma_{\text{annual}} = \sigma_{\text{daily}} \times \sqrt{252}$ (assuming ~252 trading days in a year). This gives an estimate of how volatile the asset or portfolio is. Higher σ means higher risk.
3. *Compute Value-at-Risk (VaR):* The agent estimates VaR at a given confidence (say 95% or 99%). There are multiple methods to do this:
   - **Historical VaR:** Take the historical return distribution (e.g., last 100 days of portfolio returns), sort it, and find the cutoff quantile. For 95% VaR, find the 5th percentile of returns (the loss level that is exceeded only 5% of the time). VaR is then reported as a positive number representing that loss. For example, if the 5th percentile of daily returns is -2%, and the portfolio is $10M, the 1-day 95% VaR = $200,000 (meaning 5% chance of losing $200k or more):contentReference[oaicite:3]{index=3}.
   - **Parametric (Variance-Covariance) VaR:** Assume returns are normally distributed. Compute the portfolio's mean (μ_p) and standard deviation (σ_p). Then VaR = $-\left(\mu_p - z_{p} \sigma_p\right) \times \text{PortfolioValue}$ for confidence p. Often μ_p is small and can be taken 0 for short horizons. z_p is the z-score for the p quantile (e.g., 1.65 for 95%, 2.33 for 99%). For example, if σ_p (daily) is 1.5% and $p=95\%$, VaR ≈ 1.65 * 1.5% = 2.475% of portfolio value (i.e., $247,500 on a $10M portfolio).
   - **Monte Carlo VaR:** Simulate many random scenarios for the portfolio value by modeling asset return distributions (e.g., generate 10,000 random price paths for each asset for one day, perhaps using historical vol and correlations). Then take the 5th percentile of the resulting profit/loss distribution. Monte Carlo can capture non-normal distributions at the cost of more computation.

   The agent may implement one or more of these methods. Monte Carlo might be used if the portfolio has nonlinear instruments or if a more robust risk estimation is needed beyond normal assumptions.
4. *Other Metrics:* The RiskMetricsAgent can also compute things like:
   - **Expected Shortfall (CVaR):** the average loss in the worst q% cases (e.g., average loss in worst 5% scenarios, providing insight beyond VaR's cutoff).
   - **Beta:** if a benchmark index (like S&P 500) return series is available (via IngestionAgent), the agent can run a regression of portfolio returns vs. index returns to estimate beta (slope). In practice: $\beta = \frac{\mathrm{Cov}(R_{\text{portfolio}}, R_{\text{index}})}{\mathrm{Var}(R_{\text{index}})}$. A beta >1 means the portfolio is more volatile than the market.
   - **Stress Tests:** scenario-based calculations, e.g., "If the market drops 5% today, portfolio loses X" – achieved by applying a shock to prices and revaluating positions (especially if options are present).
5. *Publish Results:* After calculations, the agent publishes the metrics to `risk_metrics` topic as shown above. It might also include flags if something is noteworthy, e.g., if current VaR is the highest in 1 year, or if volatility doubled from last week, etc., to aid in alerts.

In code form, a simplified logic focusing on VaR/vol could be:

```
# Simplified pseudocode for RiskMetricsAgent on new data on market_data_update(price_dict): update_price_history(price_dict) # store latest price, compute
```

This pseudocode assumes portfolio_value and returns array are known. In reality, the agent would carefully manage data structures for asset weights and historical returns.

Output Interpretation & Usage: The outputs from RiskMetricsAgent are critical for both the UI and other agents:

- The **UI dashboard** will display these metrics directly to end-users. For example, it might show a gauge or chart for current VaR, a time-series of volatility, etc. If VaR exceeds a user-defined threshold, the UI could flag a warning (like a red indicator).
- The **ExplanationAgent** consumes `risk_metrics` outputs to include quantitative facts in its narrative. For instance, it might say "Today's 95% VaR is $250k, which is higher than yesterday's $200k, indicating increased risk." The ExplanationAgent might also use trends in these metrics to explain why a change occurred (e.g., "Volatility has risen, contributing to a higher VaR").
- Other agents might indirectly use these metrics: e.g., the RegimeShiftDetector could take volatility from RiskMetricsAgent instead of calculating its own, or the OptionsGreeksAgent might use overall portfolio delta (if computed) to gauge risk. Primarily, however, RiskMetricsAgent's output is a standalone summary of market risk.

In summary, RiskMetricsAgent provides the quantitative backbone of the system's risk assessment. Its metrics are foundational numbers that inform decisions and explanations throughout the Real-Time Risk Watcher.

Custom Analytical Agents

SentimentAgent

Purpose & Role: The SentimentAgent gauges the qualitative market sentiment by analyzing textual data (news, social media, and filings). While RiskMetricsAgent focuses on numerical market risk, SentimentAgent adds a complementary view: how do news and narratives potentially impact risk? For example, a slew of negative news articles about a company could foreshadow stock drops or volatility spikes that numeric models might not fully capture immediately. Thus, the SentimentAgent's role is to provide signals like "market sentiment is turning bearish" or "this stock has very positive news sentiment right now." This helps the system anticipate risks or opportunities that come from public perception and information flow, not just price movements.

Inputs & Outputs: The SentimentAgent subscribes to the `news_feed` topic (published by IngestionAgent). These inputs include:

- News headlines and summaries for relevant tickers or the market as a whole.
- Chunks of text from EDGAR filings (for example, the Management Discussion & Analysis section of a 10-K, or the Risk Factors section, which often contain language that can be sentiment-scored).
- Potentially social media or analyst reports if those were ingested (not explicitly in our current inputs, but the architecture could be extended).

The output of SentimentAgent is typically a sentiment score or alert on a `sentiment_signals` topic. The output data structure might be:

```
{ "symbol": "MSFT", "sentiment_score": -0.75, "sentiment_trend": "decreasing", "summary": "News sentiment has turned sharply negative after earnings miss.'
```

Here, `sentiment_score` could be a normalized metric (e.g., -1 = extremely negative, +1 = extremely positive, 0 = neutral). We might include a `trend` or change indicator (if the agent tracks sentiment over time). The summary or reasoning field can indicate why (e.g., "multiple headlines mention earnings miss and layoffs"). In some cases, the output might be simpler, e.g., just a score per article or an average score per symbol per time window.

Internal Logic & Algorithms: The SentimentAgent uses Natural Language Processing (NLP) techniques to analyze text for sentiment:

- When a new news item or filing text arrives, the agent cleans the text (remove stopwords, etc.) if needed and then evaluates sentiment. This could be done via a pre-trained sentiment model. In finance, a specialized model like *FinBERT* (a BERT-based model fine-tuned on financial text sentiment) might be used to get more accurate results on finance context (since words like "liability" or "debt" have nuanced meaning).
- If the input already contains a sentiment from an external source (for example, AlphaVantage's News API might already provide a sentiment score:contentReference[oaicite:4]{index=4}), the agent can use that directly or as a baseline, possibly refining it.
- The agent might implement a dictionary-based fallback (e.g., count positive vs negative words from a sentiment lexicon) if a model is not available. For example, it could calculate sentiment_score = (count_positive_words - count_negative_words) / total_words as a simple metric.
- For each item, produce a score. If multiple news items come in a short period about the same symbol, the agent could aggregate them (e.g., average score of last 10 articles) to smooth the noise and detect trends.
- Optionally, it can perform topic recognition to understand what the news is about (e.g., "product launch" vs "scandal") because risk impact might differ. But this is an advanced extension.
- It publishes the sentiment result. Pseudocode snippet:

  ```
  # On receiving a news_feed item text = item.headline + " " + item.summary if item.source == "EDGAR": text = extract_relevant_sections(item.full_text)
  ```

  This shows using a `sentiment_model` (which could be an LLM or a smaller ML model) to get a score.

Mathematically, if using a regression-based sentiment: `score = \frac{positive\_probability - negative\_probability}{1}` (just to give an idea that many models output a probability of positive/negative which can be converted to a score). If using FinBERT, it might directly output a sentiment class (positive/negative/neutral) with confidence — which the agent can map to numeric scores (e.g., +1, -1, 0 or some continuous value).

Output Interpretation: The sentiment outputs are used as follows:

- **UI:** The dashboard could show a sentiment gauge for each major asset or an overall "market mood" indicator. For example, if many important stocks have negative sentiment scores, it might display a warning that "Market news sentiment is bearish today." The UI could also list top news with their sentiment color-coded (green for positive, red for negative).
- **Risk Analysis:** While sentiment isn't a traditional risk metric, it provides early warning. The ExplanationAgent will incorporate sentiment to contextualize risk metrics. For instance, if VaR spiked and at the same time the SentimentAgent reports highly negative news, the ExplanationAgent might explain that "Risk is up (VaR increased) likely due to bad news sentiment in the market." Conversely, positive sentiment might mitigate fear.
- No other agent directly uses sentiment in calculations (RiskMetricsAgent doesn't change numbers based on sentiment), but the RegimeShiftDetector could indirectly factor in if it looked at a composite indicator. Primarily, sentiment is a parallel track of information feeding into the overall picture that the ExplanationAgent and UI present.

In short, the SentimentAgent gives the system a qualitative edge, detecting risks that come from human reaction and information, which complements the pure numbers.

RegimeShiftDetector

Purpose & Role: The RegimeShiftDetector agent monitors the market's state to determine if there are structural shifts in market behavior. "Regimes" in finance refer to periods with distinct characteristics — for example, a low-volatility bull market versus a high-volatility bear market. The purpose of this agent is to signal when the market transitions from one regime to another. Such changes are critical: many risk models (like VaR) might behave differently in a calm regime vs a turbulent regime. By detecting a regime shift, the system can warn users that "things are different now" and possibly adjust how other metrics are interpreted.

Inputs & Outputs: The RegimeShiftDetector subscribes to data that reflect broad market conditions:

- Typically an **equity index time series** (e.g., S&P 500 price or returns via `market_data` on a ticker like "^GSPC" if provided, or an ETF like SPY).
- **Volatility indices** or metrics, e.g., the VIX index (if available via ingestion) or the volatility calculated by RiskMetricsAgent for the market. High volatility is often a hallmark of a regime shift to turmoil.
- Potentially macro indicators or interest rates (if ingested), as drastic changes there (like a Fed rate hike cycle) could constitute a regime shift for markets.

The output is a regime label or state published on `regime_state` topic. It might look like:

```
{ "timestamp": "2025-12-14T21:45:00Z", "regime": "High-Volatility Bear", "confidence": 0.90, "details": "Market has shifted to a high-risk regime with fal
```

Here, `regime` is a categorical description. The agent might define regimes such as "Bull (uptrend/low vol)", "Bear (downtrend/high vol)", "Stagnant (sideways/low vol)", etc. The `confidence` field indicates how sure the model is (especially if using a probabilistic model like an HMM). The details or reason might be optional, explaining criteria (e.g., "volatility > X and 20-day trend is down"). In simpler form, the output might just toggle a binary state (e.g., regime = bull or bear).

Internal Logic & Algorithms: Several approaches can be used to detect regime shifts:

- *Threshold/Rule-based:* The agent could use simple rules on metrics. For example: If market volatility (annualized) exceeds say 25% and the index is > 10% down from recent highs, declare regime "Bear". If volatility is < 15% and index is near highs, declare "Bull". Intermediate conditions could be "Neutral" or "Correction". This approach is straightforward and transparent. Pseudocode:

  ```
  vol = current_market_volatility() trend = index_return_last_30d() if vol > 0.25 and trend < -0.05: regime = "High-Volatility Bear" elif vol < 0.15 an
  ```

  This will publish a new state when it changes.
- *Statistical Regime Detection (e.g., Hidden Markov Models):* A more advanced method is to train a Hidden Markov Model (HMM) on historical returns or volatility to identify hidden states that correspond to regimes:contentReference[oaicite:5]{index=5}. An HMM will assume the market switches between, say, two or three hidden states, each with its own characteristics (mean return, volatility). The agent would continuously feed recent return data into the HMM and compute the probability of being in each state. For example, state 1 might correspond to a low-volatility bull regime (small positive mean, low variance), state 2 to a high-volatility bear (negative mean, high variance):contentReference[oaicite:6]{index=6}. The agent would output the state with highest probability as the current regime. - The HMM approach provides a confidence (the probability of each state) and can detect more subtle shifts. It involves algorithms like Baum-Welch for training on past data and the Viterbi algorithm or filtering for state estimation in real-time. Formula-wise, an HMM is defined by state transition probabilities $P(s_t|s_{t-1})$ and emission probabilities P(observation | state):contentReference[oaicite:7]{index=7}. Here observations are returns, and states are regimes.
- *Change-point Detection:* Another technique is to apply statistical tests for structural breaks in the time series (like Chow test for mean shifts, or CUSUM for variance changes). When a significant change is detected, mark a regime shift point. For real-time, a sequential algorithm (like CUSUM) could raise an alert when cumulative changes exceed a threshold.

In implementation, the agent might combine methods: e.g., use an HMM but also enforce some rules to avoid whipsaws. The regime definitions can be pre-set or learned. For clarity, if using an HMM with two states, we might define: state0 = Bull (high mean, low var), state1 = Bear (low/neg mean, high var). The agent would output which state is active and maybe a numeric indicator (like 0 or 1) in the message for simplicity, along with a human-readable label.

Output Interpretation: Knowing the current regime is valuable for:

- **RiskMetricsAgent adjustment:** The system (or the human user) can interpret the risk metrics in light of the regime. For example, a VaR of $250k might be more concerning in a low-volatility regime (since it's unusually high) than in a high-vol regime (where high VaR is expected). The RiskMetricsAgent itself might not change calculation based on regime (though in some designs it could, e.g., use longer window in stable regime and shorter in volatile regime), but the interpretation is affected.
- **UI & Alerts:** The UI can prominently display the regime ("Current Market Regime: Bearish (High Volatility)") so users immediately know the context. If a regime shift is detected, the UI can flash an alert "Market regime has changed!" because such shifts often precede or coincide with big changes in risk. This helps users contextualize other metrics and possibly take action (e.g., de-risking a portfolio during a newly detected bear regime).
- **ExplanationAgent:** The ExplanationAgent will incorporate the regime info in its narrative. E.g., "We are in a high-volatility bear regime, so the elevated VaR is not surprising. You should be cautious as markets are more turbulent than usual." It may also explain transitions: "The model indicates the market transitioned from a calm to a volatile state."

Overall, the RegimeShiftDetector adds a higher-level perspective on market conditions to the Real-Time Risk Watcher, allowing it to adapt its warnings and explanations to different market phases.

CreditRiskAgent

Purpose & Role: The CreditRiskAgent focuses on credit-related risk factors, i.e., the risk that a company (or bond issuer in the portfolio) might default or experience financial distress. While market risk (VaR, volatility) deals with price fluctuations, credit risk deals with the fundamental solvency of companies. This agent is particularly relevant if the portfolio includes corporate bonds or if we are concerned about the credit health of companies whose stocks we hold (since a deteriorating credit situation can tank a stock as well). Its role is to provide metrics like a credit score, default probability, or at least flags for companies with weakening financial health, adding a fundamental layer to the risk assessment.

Inputs & Outputs: The CreditRiskAgent takes in fundamental data about companies:

- *Financial statement data:* Key figures such as total debt, equity, earnings, cash flow, interest expense, etc. These can come from EDGAR filings processed by the IngestionAgent. For example, after a 10-Q is filed, IngestionAgent might publish extracted ratios or financials on a `fundamentals_data` topic that CreditRiskAgent subscribes to. Alternatively, AlphaVantage's Fundamental API provides metrics (like P/E, debt/equity) which could be used:contentReference[oaicite:8]{index=8}.
- *Market data relevant to credit:* If available, credit spreads (e.g., yield of company's bonds vs treasury) or credit ratings from rating agencies. However, these may not be directly provided; a simplified approach is to derive an implicit signal (for instance, if the stock price is plunging and debt is high, credit risk is likely rising).

The output is typically a credit risk score or assessment per company (or aggregated if the portfolio has many companies). For example, on a `credit_risk` topic:

```
{ "company": "ACME Corp", "credit_score": 5, "rating_equivalent": "BB", "default_probability": 0.02, "analysis": "High debt and declining earnings; elevate
```

Here we imagine a custom `credit_score` (say 1 to 10, where 10 is best credit, 1 is worst), a mapped rating (BB roughly corresponds to a speculative grade rating), and perhaps an estimated 2% probability of default (over a certain horizon, e.g., one year). The analysis field can provide a brief on why the score is what it is.

Internal Logic & Algorithms: The CreditRiskAgent can use various models:

- *Ratio-based scoring:* A straightforward way is to calculate financial ratios that indicate credit health:
  - Leverage ratios: e.g., Debt/Equity, Debt/EBITDA.
  - Coverage ratios: e.g., EBITDA/Interest (interest coverage), or Cash Flow/Interest.
  - Profitability and liquidity: e.g., Net Income/Revenue (profit margin), Current Ratio, etc.

The agent could apply threshold rules or weighted sums of these ratios. For example, if Debt/Equity is very high and interest coverage is very low, the agent would output a poor credit score.

- *Altman Z-score:* A classic model for predicting bankruptcy risk is the Altman Z-score:contentReference[oaicite:9]{index=9}. It's a linear formula using five financial ratios: $$Z = 1.2X_1 + 1.4X_2 + 3.3X_3 + 0.6X_4 + 1.0X_5,$$ where:
  - X_1 = Working Capital / Total Assets
  - X_2 = Retained Earnings / Total Assets
  - X_3 = EBIT / Total Assets
  - X_4 = Market Value of Equity / Total Liabilities
  - X_5 = Sales / Total Assets

  The agent can compute this Z-score if it has the necessary inputs. A Z-score below a certain value (e.g., Z < 1.8) indicates high bankruptcy risk, whereas a high Z (above ~3) indicates healthy status. For example, if the agent finds Z = 1.5 for a company, it would flag it as a credit concern (likely default zone):contentReference[oaicite:10]{index=10}. This formula gives a single number summary of credit risk.
- *Machine learning classification:* In a more advanced setup, the agent might use a trained model (like a logistic regression or an ML classifier) that inputs financial features and outputs a default probability or credit category. The model could be trained on historical cases of default vs non-default companies.
- *Market-implied signals:* If the company has bonds trading, a widening credit spread (bond yield minus risk-free rate) is a direct market measure of credit risk. Lacking that, a plummeting stock price can be a proxy (as equity value approaching debt value implies higher default risk in structural models). The agent could incorporate recent stock performance as a minor input to adjust the score (e.g., if stock down 50% in a month, worsen the credit outlook somewhat).

In implementation, suppose the agent uses Altman Z plus a couple of extra checks:

```
# Pseudocode for CreditRiskAgent when new financial data arrives data = get_fundamental_data(company) Z = compute_altman_Z(data) score = map_Z_to_score(Z)
```

This shows a combination of Altman Z and a heuristic. `map_Z_to_score` might translate Z to a 1-10 scale, and `estimate_PD` could use an established mapping (for instance, a Z-score of 1.5 might correspond to ~20% default chance in 2 years, just illustrative). `map_score_to_rating` can map numerical scores to letter ratings (like score 8-10 = investment grade (AAA/AA), 5-7 = BBB/BB, etc., 1-4 = B or worse).

Output Usage: The credit risk outputs are used in these ways:

- **UI Display:** For each company (especially ones in the user's portfolio), the UI can show a credit risk indicator. For example, a table listing each holding with a credit score or a colored dot (green = good, red = bad). If a company's credit risk deteriorates significantly (score drops or default probability rises), the system can alert the user, since that could presage price drops or actual default on bonds.
- **Integration in Explanations:** The ExplanationAgent will include credit insights to give a holistic risk picture. E.g., "Company XYZ's credit risk has worsened (Credit score down to 4, indicating junk status), which heightens the portfolio's risk beyond market volatility." This adds fundamental context — the agent might suggest caution if credit risk is a concern even if market risk seems moderate.
- **Scenario Analysis:** If the system does any scenario or MCDA (multi-criteria decision analysis), credit risk would be one of the criteria. For instance, an overall risk score could weigh both market risk and credit risk. If a portfolio has mostly high credit-quality names, the system might interpret the risk differently than if it's loaded with highly leveraged companies.

In summary, CreditRiskAgent brings fundamental credit perspective, ensuring that the Real-Time Risk Watcher doesn't ignore slow-burning risks like financial health, which are not captured by price volatility alone.

OptionsGreeksAgent

Purpose & Role: The OptionsGreeksAgent addresses the risk arising from options and other derivatives in the portfolio. "Greeks" are sensitivity measures that tell us how an option's value (and thus the portfolio value) changes with respect to various underlying factors (underlying asset price, volatility, time, etc.). If the portfolio includes options (calls/puts) or other derivative instruments, market risk alone (VaR) might be insufficient because derivatives have nonlinear payoffs. The OptionsGreeksAgent provides an instantaneous read on these sensitivities:

- **Delta:** sensitivity to underlying price changes (essentially, equivalent stock exposure).
- **Gamma:** sensitivity of Delta to underlying price changes (how delta changes if underlying moves — indicates nonlinearity).
- **Theta:** sensitivity to time decay (how much value an option loses per day, all else equal).
- **Vega:** sensitivity to volatility (how option value changes if underlying volatility changes).
- **Rho:** sensitivity to interest rate changes (less critical for many, but included for completeness).

The role of this agent is to compute these for each option or for the overall portfolio and output them so that the risk from options can be understood. For example, a high negative delta means the portfolio will lose heavily if the underlying goes up, etc. By monitoring Greeks, the system can alert the user to exposures like "Your portfolio has a large gamma risk — meaning a big move in the underlying could drastically change your P/L."

Inputs & Outputs: Inputs include:

- *Option positions data:* a list of derivative holdings (e.g., "Long 10 AAPL Jan 2026 150-strike Call options"). This could be part of the portfolio config or a separate topic `portfolio_positions` where options are listed with details (underlying, strike, expiry, type, position size).
- *Market data for underlyings:* The latest price of the underlying assets (stocks, indices) for those options, from `market_data`.
- *Volatility data:* either the current implied volatility for those option contracts or an estimate (maybe RiskMetricsAgent's volatility as a proxy if IV is not directly available). In practice, one would use market implied vol or historical vol as input to formulas.
- *Interest rate:* a risk-free rate input (could be a constant or from an economic data feed) for options pricing calculations (especially for rho and theoretical values).

Outputs: The agent publishes on `options_greeks` topic. Depending on design, it could output per-option Greeks as well as aggregated portfolio Greeks. For example:

```
{ "timestamp": "2025-12-14T21:55:00Z", "underlying": "AAPL", "option": "AAPL_20260119_150C", /* Jan 19 2026 $150 Call */ "position": 10, "delta": 5.0, "gam
```

This indicates holding 10 call options results in a net delta of 5 (equivalent to being long 5 shares effectively), gamma 0.2 (delta will change by 0.2 if AAPL moves $1), etc., per option contract or total? Here likely total: if one call had delta 0.5, 10 calls => delta 5.0. The agent might also publish a summary like:

```
{ "portfolio_total": true, "timestamp": "2025-12-14T21:55:00Z", "total_delta": 5.0, "total_gamma": 0.2, "total_theta": -0.05, "total_vega": 1.3 }
```

for the whole portfolio (summing across all options positions). The output format and fields can be adjusted depending on needs (e.g., including per underlying totals, etc.).

Internal Logic & Algorithms: The OptionsGreeksAgent uses options pricing formulas or numerical approximations:

- If we assume vanilla options, it can use the Black-Scholes analytical formulas for Greeks:
  - Delta (call) = $N(d_1)$, Delta (put) = $N(d_1)$ - 1, where N is the standard normal CDF:contentReference[oaicite:11]{index=11}.
  - Gamma = $\frac{N'(d_1)}{S \sigma \sqrt{T}}$ (same for call/put):contentReference[oaicite:12]{index=12}.

- Vega = $S N'(d_1) \sqrt{T}$:contentReference[oaicite:13]{index=13} (same for call/put).
- Theta = for call: $-\frac{S N'(d_1)\sigma}{2\sqrt{T}} - r K e^{-rT} N(d_2)$:contentReference[oaicite:14]{index=14} (put has a plus in the second term).
- Rho = for call: $K T e^{-rT} N(d_2)$ (put is $-K T e^{-rT} N(-d_2)$):contentReference[oaicite:15]{index=15}.

These come from differentiating the Black-Scholes pricing formula. The agent can plug in S (current price), K (strike), T (time to expiry in years), σ (volatility), r (interest rate), and compute d1, d2, then each Greek. For example: $$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)T}{\sigma \sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T}.$$ Then Delta = N(d1), etc.

- If dealing with more complex derivatives (or if an option is far in/out-of-money where numerical stability issues might arise), the agent could compute Greeks by finite difference: - e.g., Delta ≈ (Price(S+Δ) - Price(S-Δ)) / (2Δ) using a small Δ in underlying price. - This requires the ability to price the option, which for vanilla options is straightforward via Black-Scholes. For other derivatives (like path-dependent ones), a pricing model might be needed or a library.
- The agent will iterate over each option position: - Retrieve or calculate implied vol for that option. Possibly, if no direct IV, use the RiskMetricsAgent's volatility for the underlying as a proxy. - Compute Greeks using formulas. - Multiply by position size (if long 10 calls, multiply each Greek by 10, noting that for puts delta will be negative because N(d1)-1 is negative). - Sum up contributions to portfolio totals.
- It then publishes each or aggregated results. Pseudocode:

```
for position in portfolio.options: S = market_price[position.underlying] K = position.strike T = year_fraction(now, position.expiry) type = position.
```

(Note: In practice, one would include the contract size like 100 shares/contract for equity options.)

The mathematical formulas ensure the agent captures how sensitive the portfolio is to small changes. For example, if total delta = 5 (as in output above), that means if underlying stock rises $1, the portfolio gains ~$5. If gamma is significant, that delta itself will change as the underlying moves, indicating nonlinear risk.

Output Interpretation: The Greeks output is used by:

- **Risk oversight via UI:** The UI might present the aggregate Greeks to the user. For instance, "Net Delta: +5 (equivalent to long 5 shares)", "Net Gamma: 0.2 (portfolio has positive convexity)", etc. This helps experienced users understand their exposure. If any Greek is very large in magnitude, the UI can highlight it. E.g., a very large negative delta means the portfolio is short the market heavily; a large gamma could mean the portfolio is very sensitive to large moves (which could make VaR underestimation a concern).
- **ExplanationAgent:** In generating explanations, the LLM might incorporate Greek information especially if there's a risk story. For example, "Your portfolio has a high Theta bleed of -$500/day due to short-dated options, meaning it will lose $500 daily if other factors remain constant." Or "Net Delta is near zero, indicating a market-neutral position, but the high Gamma (X) means if the market moves a lot, your delta will surge and you could face large gains or losses." These nuances can be important to communicate to the user to fully understand risk beyond VaR.
- **RiskMetricsAgent synergy:** The RiskMetricsAgent's VaR might not fully capture option non-linear risk unless it does very fine-grained simulation. Knowing the Greeks, a risk manager (or automated system) could decide to adjust VaR or run stress scenarios for option moves. For instance, if Gamma is high, the ExplanationAgent or UI might suggest looking at a scenario of a big underlying move, as VaR (which often looks at small sigma moves) might understate that risk.

In short, OptionsGreeksAgent ensures that derivative exposures are transparent and quantified. Its output, combined with traditional risk metrics, provides a comprehensive view of portfolio risk.

ExplanationAgent

Purpose & Role: The ExplanationAgent is a higher-level "AI analyst" that turns all the technical outputs from the other agents into human-friendly insights. While the other agents generate numbers, signals, and flags, the ExplanationAgent's role is to answer the question: *"What does all this mean?"* for the end user. It synthesizes data from RiskMetrics, Sentiment, Regime, Credit, and OptionsGreeks agents to produce a narrative report or explanation of the current risk situation. This helps users (especially those who are not risk experts) understand the significance of the metrics and any recommended actions or focus areas. It effectively uses an LLM (Large Language Model) to achieve this, which in our architecture is based on **LLaMA 5**, a state-of-the-art model assumed to be fine-tuned for financial contexts.

Inputs & Outputs: The ExplanationAgent subscribes to the outputs of all other analytical agents:

- `risk_metrics`: for the latest VaR, volatility, etc.
- `sentiment_signals`: for current sentiment state.
- `regime_state`: for the market regime identification.
- `credit_risk`: for any credit risk alerts or scores.
- `options_greeks`: for derivative exposure information.

Additionally, it may take input from the user or UI context (for example, if a user clicks "Explain this" or if it's scheduled to produce a daily summary, etc.). It could also have access to historical data if it needs to describe trends ("VaR increased 25% since last week" – it would find that by looking at past values, maybe via a query to a database or cache of prior outputs). The output is natural language text, published on a `risk_explanation` topic (or directly passed to the UI). This could be a paragraph or multi-paragraph report. For example:

```
{ "timestamp": "2025-12-14T22:00:00Z", "explanation_text": "The portfolio's risk has risen today. The 95% VaR is $250K, up from $200K yesterday, due to inc
```

This JSON contains the explanation text (in practice, the ExplanationAgent might output just the text, and the UI will timestamp it). The content is a cohesive story incorporating multiple inputs.

Internal Logic & Algorithms: The ExplanationAgent's core is the LLM (LLaMA 5 in this case). The internal logic involves constructing a prompt for the model that includes the relevant data and an instruction to produce a helpful explanation. Steps:

1. *Data Aggregation:* The agent gathers the latest outputs from all other agents. This could be done by maintaining an internal state or cache that updates whenever it receives a message, so at any time it has the most recent risk_metrics, sentiment, etc. Alternatively, when triggered to explain, it can query a data store or wait for the next incoming messages and collate them.
2. *Prompt Construction:* It then forms a prompt for LLaMA 5. For example, the prompt might be:

```
System: You are a financial risk analysis assistant. Provide a clear, concise explanation of the current risk status for a portfolio, using the given
ACME Corp: credit score 4/10 (high risk)
Other holdings: no issues.
Options:
Net Delta: ~0 (delta-neutral)
Net Gamma: High (portfolio very sensitive to large moves)
Net Theta: -$500/day (will lose value over time if unchanged)
Now explain what this means for the portfolio risk and any concerns or recommendations.
```

The above is a conceptual illustration; the actual format might be JSON or a structured text inserted into the LLM prompt. The system role instructs the model to be an expert risk explainer, and the user prompt provides the data in an organized way.

3. *LLM Generation:* The ExplanationAgent invokes the LLaMA 5 model with this prompt (which might be done via an API or local model inference). The model, being a sophisticated LLM, generates a coherent explanation. For example, it may output something akin to the explanation_text shown earlier. Because LLaMA 5

is presumably quite advanced, it can understand the financial context and produce an explanation that is both accurate and accessible (perhaps even suggesting what to monitor next).

4. *Post-processing:* The agent may do minor tweaks to the LLM's output – e.g., ensure no hallucinated facts (since all input data is provided, this risk is small), or format the text (maybe add bullet points or a warning highlight if needed). Generally, the raw text from the LLM will suffice.
5. *Publishing:* Finally, it wraps the text into the output structure and publishes to `risk_explanation`. Alternatively, it could send it directly to the UI component responsible for displaying explanations.

In code form, it might be:

```
# Simplified pseudocode for ExplanationAgent def on_trigger_explanation(): data = collect_latest_data() # gather strings/numbers from other agents prompt =
```

The trigger for explanation could be timed (e.g., generate a report every hour or end-of-day) or event-driven (e.g., VaR spiked beyond threshold, so explain now). It can also be on user demand (user clicks "Explain" in UI, which sends a request that the agent listens for).

Output Interpretation & Usage: The ExplanationAgent's output is directly aimed at the end-user:

- **UI Display:** The text is shown in a dashboard panel, report section, or even spoken aloud if it were a voice assistant. The user can read this explanation to understand the numbers they see elsewhere on the dashboard. It essentially translates data into insight. The UI might format it nicely (paragraphs, bullet points for recommendations, etc.).
- **User Guidance:** A good explanation not only states what is happening but might also suggest actions or points of attention ("caution is advised" or "monitor credit risk on ACME Corp"). This guides the user on how to interpret the risk. The ExplanationAgent could be extended to answer follow-up questions from the user as well (interactive Q&A using the LLM). For now, its main job is the proactive summary.
- **Internal QA:** Interestingly, the process of generating explanations can also serve as a consistency check. If some data point looks off (say the LLM is confused because one agent's output is contradictory to another's), that might indicate a data issue. While not explicitly designed for this, an anomalous or incoherent explanation might prompt developers to review the inputs. In a future iteration, one could incorporate a feedback loop where the ExplanationAgent flags if something "doesn't make sense".

Because we use LLaMA 5, which is a powerful model, the explanations can be quite nuanced and tailor-made. The use of an LLM is in line with industry trends of using AI to enhance decision support:contentReference[oaicite:16]{index=16}. Importantly, all the heavy lifting of calculation is done by other agents; the LLM's job is purely interpretation. This separation keeps the system's analytical correctness grounded (since it's based on formulaic outputs) and uses the LLM for what it's best at – language and abstraction.

Multi-Criteria Decision Analysis (MCDA) Note: It's worth mentioning that the system could employ an MCDA approach internally (either within the ExplanationAgent or as a separate logic) to consolidate the various risk dimensions into an overall risk score or ranking. For instance, it could weight market risk, credit risk, and sentiment to derive a single "Composite Risk Index". A simple weighted formula could be: $$\text{CompositeRiskScore} = w_{mr} \times \text{VaR\_score} + w_{cr} \times \text{CreditRisk\_score} + w_{st} \times \text{Sentiment\_score} + \dots$$ where each component score is a normalized value of that risk (VaR_score could be the current VaR divided by a benchmark VaR, credit score already on 1-10 scale, sentiment_score scaled between -1 to 1, etc.), and w are weights summing to 1 reflecting importance. The system might then classify overall risk as High/Medium/Low based on this composite score. This is essentially an MCDA approach to combine multiple criteria:contentReference[oaicite:17]{index=17}. In our architecture, we haven't explicitly included a separate MCDA agent; however, this computation can be done as part of ExplanationAgent's reasoning or simply for display. For example, ExplanationAgent could calculate this internally to decide what aspect to emphasize (if composite is heavily driven by credit risk, focus explanation there). Such a formula or decision matrix can be tuned by risk managers to reflect their priorities.

With all the agents described, the Real-Time Risk Watcher presents a comprehensive picture: each specialized agent monitors a facet of risk in real-time, and the ExplanationAgent ties it all together using LLaMA 5's advanced language capabilities. This results in a developer-friendly yet richly informative system. New agents can be added similarly if new risk factors need to be monitored (for example, a *LiquidityRiskAgent* could be envisioned to monitor market liquidity metrics). The architecture is extensible and each agent remains focused, communicating through clear input/output contracts (message schemas and topics). The design above can be directly implemented by deploying each agent as a microservice or process, connected via a message broker (like Kafka or RabbitMQ), ensuring real-time flow of information and robustness of the overall system.