Intelligent Trading Signal Generator: A Comprehensive Technical Report on End-to-End System Architecture, Implementation, and Quantitative Methodology

Executive Summary

The financial services landscape is defined by a stark bifurcation in capability. On one side, institutional entities—hedge funds, proprietary trading firms, and high-frequency trading (HFT) desks—leverage infrastructure costing millions of dollars annually to secure information asymmetry. These entities utilize alternative data, low-latency execution grids, and sophisticated machine learning models to generate "alpha," or returns in excess of a benchmark. On the other side, the retail investor and the independent analyst are historically constrained by delayed data, rudimentary tools, and a lack of access to the computational power necessary to synthesize the vast unstructured datasets that move modern markets.

The "Intelligent Trading Signal Generator" project proposed herein addresses this disparity. It is not an automated trading bot, which executes orders blindly and incurs significant operational risk. Rather, it is an institutional-grade Decision Support System (DSS). It is designed to ingest multi-modal data streams—quantitative market ticks and qualitative linguistic sentiment—to generate high-probability trade theses backed by rigorous statistical evidence.

This report serves as an exhaustive blueprint for constructing this system. It details the utilization of a modern, cost-effective technology stack including **Alpha Vantage** and **Yahoo Finance** for market data, **PostgreSQL** with **TimescaleDB** for high-performance time-series storage, **FinBERT** and **GPT-4** for semantic analysis, and **Plotly Dash** for visualization. Furthermore, it delineates the mathematical foundations of the risk management modules, specifically the implementation of **Monte Carlo simulations** and **Value at Risk (VaR)** models. By following this guide, a developer or analyst can build a sustainable, scalable platform that mirrors the workflow of a quantitative hedge fund, bridging the gap between retail limitations and institutional capability.

1. Introduction: The Quantitative Landscape and Problem Statement

1.1 The Alpha Generation Gap

In the context of modern finance, "alpha" represents the active return on an investment, gauging the performance of an investment against a market index or benchmark which is considered to represent the market's movement as a whole. Institutional players generate alpha not merely through superior intuition but through superior *synthesis*. They do not just look at price; they analyze satellite imagery of parking lots to predict retail earnings, scrape Twitter for sentiment shifts, and parse millions of regulatory filings instantly.[1]

For the independent developer or retail investor, the "Real Problem" is the inability to replicate this synthesis manually. A human trader can watch a chart for technical signals (RSI, MACD) or read news for fundamental signals (Earnings beats), but rarely can they do both simultaneously across a universe of hundreds of assets with perfect consistency. The Intelligent Trading Signal Generator solves this by automating the *monitoring* and *synthesis* phases of the trading lifecycle, leaving the final *execution* decision to the human operator.

1.2 Project Scope and Objectives

The objective is to engineer a system that functions as a "Digital Quant Analyst." The system must meet the following high-level requirements:

1. **Multi-Source Ingestion:** It must autonomously fetch and normalize data from disparate sources, including structured time-series (prices) and unstructured text (news, earnings calls).[2]
2. **Hybrid Analysis:** It must implement a "Confluence Engine" that requires both technical confirmation and fundamental support before issuing a signal, thereby reducing the false-positive rate common in pure technical strategies.[4]
3. **Explainability (White Box AI):** Unlike neural networks that output a black-box prediction, this system must utilize Large Language Models (LLMs) to articulate a "Trade Thesis," explaining *why* a signal was generated (e.g., "RSI Divergence coupled with positive forward guidance in Q3 transcript").[6]
4. **Rigorous Validation:** It must include a backtesting framework that accounts for look-ahead bias and survivorship bias, providing realistic performance metrics like Sharpe Ratio and Maximum Drawdown.[7]
5. **Sustainable Architecture:** The system must be deployable on commodity cloud hardware (e.g., DigitalOcean, Render) with minimal recurring costs, utilizing efficient caching and database partitioning strategies.[9]

2. System Architecture and Design Philosophy

2.1 Architectural Pattern: Event-Driven Microservices

Designing a financial analytics platform requires handling asynchronous data flows. Market data arrives in streams (ticks), while news arrives in bursts. A monolithic architecture, where a single process handles ingestion, analysis, and serving the UI, is prone to blocking. For instance, if the system is busy parsing a 50-page earnings transcript, it might miss ingesting real-time price packets, leading to data gaps.

Therefore, the system adopts an **Event-Driven Architecture (EDA)**. In this model, distinct services communicate via a message broker.

- **The Producer:** The "Ingestion Service" fetches data and pushes a message (e.g., NEW_CANDLE_AAPL or NEW_ARTICLE_NVDA) to the broker.[11]
- **The Broker: Redis** is selected as the broker due to its in-memory speed and support for Pub/Sub patterns. It acts as the central nervous system, buffering messages during high-load periods.[12]
- **The Consumer:** Specialized "Worker" services listen for specific events. The "Technical Worker" computes indicators when new price data arrives. The "NLP Worker" runs inference when new text arrives.

This decoupling allows for **Independent Scaling**. If news volume spikes during earnings season, we can spin up more NLP Workers without touching the Price Ingestion logic.[14]

2.2 Technology Stack Selection and Justification

The selection of the technology stack balances performance, cost, and developer productivity, adhering to the requirement for a "sustainable" project.

Layer	**Technology**	**Selection Rationale**
Language	**Python 3.10+**	The undisputed standard for quantitative finance. Libraries like pandas, numpy, and scipy provide optimized C-bindings for math, while transformers and langchain lead in AI.[4]
Orchestration	**Celery**	A robust distributed task queue for Python. It manages the asynchronous workers and handles retries/failures gracefully, which is critical for unstable API connections.[14]

Database	**PostgreSQL + TimescaleDB**	Financial data is time-series data. Standard SQL tables degrade in performance as row counts hit millions. TimescaleDB extends Postgres with "hypertables," automatic partitioning, and compression, maintaining fast query speeds for historical analysis.[16]
Data Source	**Alpha Vantage / OpenBB**	Alpha Vantage offers a reliable free tier for technical indicators. OpenBB acts as an aggregator, allowing the system to switch providers (e.g., to Yahoo Finance or FMP) without rewriting code.[18]
NLP Model	**FinBERT & GPT-4**	FinBERT (based on BERT) is fine-tuned on financial text for sentiment scoring. GPT-4 is used for high-level reasoning and summarizing earnings calls via RAG (Retrieval Augmented Generation).[20]
Visualization	**Plotly Dash**	Allows for the creation of interactive, web-based financial dashboards entirely in Python. It supports complex charting (candlesticks, heatmaps) essential for trading UIs.[22]
Infrastructure	**Docker & DigitalOcean**	Docker ensures environment consistency. DigitalOcean offers

		predictable pricing for VPS (Droplets), which is more cost-effective for a continuously running service than serverless functions that might time out during long calculations.[9]

2.3 The Data Pipeline Workflow

The end-to-end data flow operates as follows:

1. **Scheduler (Celery Beat):** Triggers fetch_market_data every minute and fetch_news every 15 minutes.
2. **Ingestion Worker:** Calls the Alpha Vantage API. If the API limit is hit, it falls back to Yahoo Finance (yfinance). Data is normalized into a Pandas DataFrame.
3. **Persistence:** The raw data is upserted into the TimescaleDB hypertable.
4. **Analysis Trigger:** The successful database commit triggers a analyze_asset task.
5. **Hybrid Analysis:** The worker fetches the latest 100 candles and recent news from the DB, runs TA-Lib indicators and FinBERT scoring, and checks for signal confluence.
6. **Alert Generation:** If a signal is found, a trade_idea object is created, risk metrics are calculated via Monte Carlo, and the result is pushed to the Frontend Database for display.[25]

3. Data Engineering: Ingestion and Storage

3.1 Navigating the API Landscape

A significant challenge in building a sustainable project is "API Decay"—the tendency for free data sources to become paid or restricted over time. The report identifies a multi-tiered strategy to mitigate this risk.

Primary Source: Alpha Vantage
Alpha Vantage provides institutional-quality data but has introduced tighter rate limits (e.g., 25 requests/day on the free tier).18 This is insufficient for real-time monitoring of a large portfolio.

- *Mitigation Strategy:* The system implements an "Adaptive Polling" mechanism. It prioritizes high-volatility assets for frequent updates while relegating stable assets to hourly updates.

Secondary Source: Yahoo Finance (yfinance)
The yfinance library scrapes Yahoo Finance. While it has no official rate limit, it is prone to breaking changes in the HTML structure.28

- *Usage:* Use yfinance for "History Seeding"—downloading the past 5 years of data for backtesting—to save Alpha Vantage credits for live intraday updates.[30]

Alternative Source: OpenBB Platform
The OpenBB SDK abstracts the data provider. By writing code against the OpenBB interface, the backend can switch from Alpha Vantage to Financial Modeling Prep (FMP) or Polygon.io simply by changing an API key in the configuration, ensuring long-term sustainability.19

3.2 Database Schema Design

The choice of schema determines the speed of backtesting and signal generation. We utilize a **Star Schema** variant optimized for time-series.

Table 1: asset_metadata (Dimension Table)
Stores static information.

SQL

```sql
CREATE TABLE asset_metadata (
  symbol VARCHAR(10) PRIMARY KEY,
  name VARCHAR(255),
  sector VARCHAR(100), -- e.g., Technology
  industry VARCHAR(100), -- e.g., Semiconductors
  currency VARCHAR(10) DEFAULT 'USD'
);
```

Table 2: market_candles (Fact Table - Hypertable)
Stores the core OHLCV (Open, High, Low, Close, Volume) data.

SQL

```sql
CREATE TABLE market_candles (
  time TIMESTAMPTZ NOT NULL,
  symbol VARCHAR(10) REFERENCES asset_metadata(symbol),
  open DOUBLE PRECISION,
  high DOUBLE PRECISION,
  low DOUBLE PRECISION,
  close DOUBLE PRECISION,
  volume BIGINT,
  PRIMARY KEY (time, symbol)
);
-- Convert to TimescaleDB Hypertable
SELECT create_hypertable('market_candles', 'time');
```

Design Insight: The composite primary key (time, symbol) allows for rapid retrieval of all assets for a specific timestamp (market scans) or all history for a specific asset (backtesting). The create_hypertable command instructs TimescaleDB to partition this table on disk, drastically speeding up time-range queries compared to a standard B-Tree indexed table.[16]

Table 3: news_sentiment (Fact Table)
Stores the unstructured data in a structured format.

SQL

```sql
CREATE TABLE news_sentiment (
  time TIMESTAMPTZ NOT NULL,
  symbol VARCHAR(10),
  title TEXT,
  source VARCHAR(50),
  url TEXT,
  finbert_score DOUBLE PRECISION, -- Range: -1.0 to 1.0
  sentiment_label VARCHAR(10) -- 'positive', 'neutral', 'negative'
);
SELECT create_hypertable('news_sentiment', 'time');
```

4. The Quantitative Engine: Technical Analysis

The "Quantitative Engine" is responsible for the deterministic analysis of price action. It leverages **TA-Lib**, an industry-standard library written in C, to perform calculations with high speed and precision.[32]

4.1 Mathematical Models and Indicators

To generate "Intelligent" signals, we move beyond basic crossovers and implement sophisticated indicator logic.

4.1.1 Relative Strength Index (RSI)
The RSI measures the velocity of directional price movement.

$$RSI = 100 - \frac{100}{1 + RS}$$

Where RS (Relative Strength) is the average of n days' up closes divided by the average of n days' down closes.

- *Implementation Strategy:* The system detects **RSI Divergence**. A "Bullish Divergence" occurs when the price records a lower low, but the RSI records a higher low. This indicates that the selling momentum is exhausting, often preceding a reversal. This is a far more potent signal than the standard "RSI < 30" threshold.[33]

4.1.2 Moving Average Convergence Divergence (MACD)

The MACD is a trend-following momentum indicator.

$$MACD = EMA_{12}(Price) - EMA_{26}(Price)$$

$$Signal = EMA_{9}(MACD)$$

- *Implementation Strategy:* The system monitors the **Histogram** ($MACD - Signal$). A "Zero Cross" of the histogram confirms a trend change. We specifically look for situations where the MACD line crosses *above* the Signal line while both are deep in negative territory (oversold), suggesting a high-potential reversal.[34]

4.1.3 Bollinger Bands

These bands map the volatility of the asset.

$$Upper = SMA_{20} + (2 \times \sigma)$$

$$Lower = SMA_{20} - (2 \times \sigma)$$

- *Implementation Strategy:* **The Squeeze**. When the bandwidth ($Upper - Lower$) reaches a historical minimum (e.g., lowest in 6 months), it indicates a period of low volatility. Efficient Market Hypothesis suggests volatility is mean-reverting; thus, a "Squeeze" predicts an explosive move. The system flags this as a "Breakout Watch".[4]

4.2 Signal Logic Implementation

The implementation uses pandas for vectorization, ensuring the logic scales to thousands of tickers.

Python

```python
import talib
import pandas as pd

def analyze_technicals(df):
    # Inputs: DataFrame with 'close', 'high', 'low'

    # 1. Calculate Indicators
    df['rsi'] = talib.RSI(df['close'], timeperiod=14)
    df['macd'], df['macd_signal'], df['macd_hist'] = talib.MACD(df['close'])
    df['upper'], df['middle'], df['lower'] = talib.BBANDS(df['close'])

    # 2. Logic: The "Golden Setup"
    # Condition A: RSI is oversold (< 30)
    # Condition B: Price touched Lower Bollinger Band
    # Condition C: MACD Histogram is ticking up (momentum shifting)

    df['signal'] = 0
    condition = (
        (df['rsi'] < 30) &
        (df['low'] <= df['lower']) &
        (df['macd_hist'] > df['macd_hist'].shift(1))
    )

    df.loc[condition, 'signal'] = 1 # Buy Signal
    return df
```

This function returns a DataFrame with a signal column. However, in this system, a technical signal 1 is *not* a buy order—it is merely a proposal that must be ratified by the Semantic Engine.

5. The Semantic Engine: NLP and LLM Integration

This layer differentiates the project from a standard trading bot. It ingests unstructured text to provide context, sentiment, and fundamental justification.

5.1 Sentiment Quantification with FinBERT

General-purpose sentiment models often fail in finance. For example, "The company's cost of revenue decreased" is a *positive* event for a stock, but a generic model might flag "decreased" as negative. **FinBERT** is a BERT model pre-trained on a massive corpus of financial documents, enabling it to capture these nuances.[20]

Implementation:
The pipeline uses the Hugging Face transformers library.

1. **Ingestion:** Fetch news headlines for the asset (e.g., via NewsAPI or scraping Yahoo

News).
2. **Tokenization:** Convert text to FinBERT-compatible tokens.
3. **Inference:**
   ```
   Python
   from transformers import pipeline
   nlp = pipeline("sentiment-analysis", model="ProsusAI/finbert")
   results = nlp()
   # Output: [{'label': 'positive', 'score': 0.98}]
   ```
4. **Aggregation:** The scores of all articles in the last 24 hours are aggregated into a weighted average. This sentiment_score (-1 to +1) acts as a filter. If a Technical Signal says "Buy" but sentiment_score is -0.8 (extremely negative news), the signal is suppressed to prevent "catching a falling knife".[35]

5.2 Deep Analysis with LLMs (GPT-4) and RAG

While FinBERT gives a score, it cannot explain *why*. For this, we use **GPT-4** combined with **Retrieval Augmented Generation (RAG)** to analyze Earnings Call Transcripts.

The RAG Architecture:

1. **Document Loader:** The system downloads the transcript JSON from the Financial Modeling Prep API.[37]
2. **Chunking & Embedding:** The text is split into chunks of 500 tokens and embedded using sentence-transformers. These embeddings are stored in a vector store (e.g., pgvector inside the PostgreSQL database, simplifying the stack).
3. **Retrieval:** When generating a thesis for NVDA, the system queries the vector store for keywords: "Guidance", "AI Demand", "China Restrictions", "Margins".
4. **Context Construction:** The top 5 relevant chunks are retrieved.

Prompt Engineering:
The system prompt is critical for generating professional output.
- *Role:* "You are a Senior Quantitative Analyst at a hedge fund."
- *Input:* The retrieved transcript chunks + the Technical Indicators (RSI, MACD values).
- *Task:* "Synthesize the technical setup with the fundamental commentary. Identify 3 bullish factors and 3 bearish risks. Conclude with a 'Conviction Level' (High/Medium/Low)."
- *Constraint:* "Do not hallucinate numbers. Use the provided context only.".[38]

This process generates the narrative text displayed on the dashboard, giving the user a "White Box" view of the trade rationale.

6. Hybrid Signal Synthesis: The Confluence Model

The core value proposition is the **Confluence Matrix**. A valid signal requires alignment between the Quantitative and Semantic engines.

Technical Signal	**Sentiment Score (24h)**	**Output Decision**	**Rationale**
Buy (Oversold)	**Positive (> 0.2)**	**STRONG BUY**	"Dip Buy": Fundamentals are strong, price drop is likely noise or overreaction.
Buy (Oversold)	**Negative (< -0.2)**	**NO TRADE**	"Value Trap": Price is dropping for a valid fundamental reason (e.g., bad earnings).
Sell (Overbought)	**Negative (< -0.2)**	**STRONG SELL**	"Trend Reversal": Price is extended high, and news is turning bad.
Sell (Overbought)	**Positive (> 0.2)**	**HOLD**	"Momentum": High sentiment suggests the trend may extend despite being overbought.

This logic is implemented in the SignalSynthesizer service. It queries the market_candles and news_sentiment tables, applies the logic, and if a condition is met, triggers the Risk Management module.[5]

7. Backtesting Framework and Validation

To prove the system's efficacy, we must simulate its performance on historical data. This is not merely running code on past data; it requires rigorous scientific methodology to avoid statistical biases.

7.1 Framework Selection: Backtrader vs. VectorBT

For the "Research" phase of the project, VectorBT is recommended. It leverages Pandas and NumPy to perform vectorized backtesting. This allows the user to test thousands of parameter combinations (e.g., "What if RSI threshold was 25 instead of 30?") in seconds.
However, for the "Simulation" of the trading logic, Backtrader is superior because it allows for event-driven simulation, mimicking the exact behavior of the live bot (processing one candle at a time), which is crucial for verifying the logic isn't "peeking" into the future.40
Recommendation: Use **Backtesting.py** for the web dashboard. It is a lightweight framework that produces interactive HTML plots, perfect for the "Demo" requirement of the project.[42]

7.2 Handling Biases

The report emphasizes two critical biases that the developer must handle:

1. **Look-Ahead Bias:** This occurs if the strategy uses the "Close" price of the *current* bar to execute a trade *on* the current bar. In reality, you only know the Close price when the bar ends.
   - *Solution:* The backtest logic must force execution on Open[i+1] (the next bar's open) based on signals from Close[i].[43]
2. **Survivorship Bias:** Testing only on the current S&P 500 excludes companies like Enron or Lehman Brothers that went bust. This artificially inflates returns.
   - *Solution:* The system should ideally use a "Delisted Adjusted" dataset. If that is too expensive for a personal project, the report must explicitly state this limitation in the risk disclosure.[7]

7.3 Performance Metrics

The system calculates key metrics to benchmark against the SPY (S&P 500 ETF):

- **Sharpe Ratio:** $\frac{R_p - R_f}{\sigma_p}$. Measures return per unit of risk. A Sharpe > 1.0 is the target.
- **Max Drawdown:** The deepest peak-to-trough decline. This defines the "pain" of the strategy.
- **Sortino Ratio:** Similar to Sharpe, but only penalizes *downside* volatility, distinguishing between "good volatility" (upside spikes) and "bad volatility" (crashes).[8]

8. Risk Management: VaR and Monte Carlo

A trade signal without a risk assessment is gambling. The "Intelligent" aspect of the generator implies it understands risk.

8.1 Value at Risk (VaR)

VaR answers the question: *"What is the worst-case loss I might suffer over the next day with 95% confidence?"*

- **Calculation:**
  1. Calculate daily log returns of the asset over the last year.
  2. Compute the 5th percentile of this distribution.
  3. $VaR_{95\%} = PortfolioValue \times Percentile_{5th}$.
- *Application:* If the calculated VaR exceeds a user-defined threshold (e.g., 2% of equity), the signal is flagged as "High Risk".[46]

8.2 Monte Carlo Simulations

This technique simulates thousands of possible future price paths to understand the probability of extreme outcomes.

- Methodology:
  We model the stock price movement using Geometric Brownian Motion (GBM):

  $$dS_t = \mu S_t dt + \sigma S_t dW_t$$

  Where $\mu$ is the drift (average return), $\sigma$ is volatility, and $dW_t$ is a Wiener process (random shock).
- Implementation:
  The Python backend runs 1,000 simulations of the next 30 trading days.
  - *Scenario Analysis:* "In 15% of simulations, NVDA drops below \$80."
  - *Visual:* The dashboard displays a "fan chart" showing the spread of these 1,000 paths. This visualizes uncertainty effectively for the user.[47]

9. Visualization and User Experience (Frontend)

The presentation layer is built with **Plotly Dash**. This allows the entire UI to be written in Python, integrating seamlessly with the Pandas dataframes produced by the backend.

9.1 Dashboard Layout and Features

- **The Command Center:**
  - *Ticker Input:* A search bar with auto-complete.
  - *Signal Badge:* A prominent indicator (BUY / SELL / HOLD) color-coded (Green/Red/Grey).
- **The Technical View:**
  - An interactive Candlestick chart (plotly.graph_objects.Candlestick).
  - Markers (Triangles) indicate where Buy/Sell signals were generated in the past.
  - Overlays for Bollinger Bands and Moving Averages.[22]
- **The Fundamental Panel:**
  - A gauge chart showing the FinBERT Sentiment Score (-1 to 1).

    ○ A "Thesis Box" rendering the Markdown output from GPT-4 (The Bull/Bear arguments).
- **The Risk Panel:**
    ○ A histogram showing the distribution of Monte Carlo simulation results.
    ○ Key Metrics Table: VaR, Sharpe Ratio, Max Drawdown.

10. Infrastructure and Deployment Strategy

10.1 Containerization with Docker

To ensure the system is reproducible and "sustainable" (easy to move/restore), every component is containerized.

docker-compose.yml Structure:

YAML

```
version: '3.8'
services:
 db:
  image: timescale/timescaledb:latest-pg14
  ports: ["5432:5432"]
  volumes: [pgdata:/var/lib/postgresql/data]

 redis:
  image: redis:alpine
  ports: ["6379:6379"]

 worker:
  build:.
  command: celery -A app.celery_worker worker --loglevel=info
  environment:
   - DATABASE_URL=postgresql://user:pass@db:5432/trading
   - OPENAI_API_KEY=${OPENAI_KEY}
  depends_on: [db, redis]

 scheduler:
  build:.
  command: celery -A app.celery_worker beat
  depends_on: [worker]
 dashboard:
  build:.
  command: python index.py
  ports: ["8050:8050"]
  depends_on: [db]

volumes:
 pgdata:
```

This configuration allows the entire stack to be launched with a single command: docker-compose up -d.

10.2 Deployment Options

- **Option A (Cost-Optimized): DigitalOcean Droplet**. A $24/month droplet (4GB RAM / 2 vCPU) is sufficient to run the stack. The database and workers reside on the same host. This is ideal for a personal portfolio project.[9]
- **Option B (Scalable): AWS or Render**. Managed PostgreSQL (RDS) and separate containers for workers. This allows auto-scaling but incurs higher costs ($50+/month). For the scope of this project, Option A is recommended.

11. Industry Relevance and Career Implications

Building this project end-to-end demonstrates a skill set that is in extremely high demand across the financial sector.

11.1 Who is Hiring?

- **Quantitative Hedge Funds (e.g., Two Sigma, Citadel, Millennium):** They look for "Quantitative Developers" who understand the full lifecycle of data—from ingestion to signal generation. This project demonstrates proficiency in Python, databases, and rigorous backtesting.[50]
- **FinTech & Data Providers (e.g., Bloomberg, Thomson Reuters):** These companies build the *tools* that traders use. Experience with NLP (FinBERT) and visualizing financial data (Dash) is directly applicable to building terminal applications.[51]
- **Proprietary Trading Firms:** They value the understanding of "Market Microstructure" and the ability to build low-latency event-driven architectures (Celery/Redis).[52]

11.2 Portfolio Project Value

For an aspiring Quant or Developer, this project is a "Capstone." It moves beyond simple scripts to a deployed, architected system. It shows you can:

1. Handle **Data Engineering** (ETL, Databases).

2. Implement **Financial Math** (Indicators, Risk Metrics).
3. Apply **Machine Learning** (NLP, LLMs) in a domain-specific context.
4. Build **Full-Stack Applications** (Backend + Frontend).

Conclusion

The "Intelligent Trading Signal Generator" is a comprehensive solution to the problem of retail information asymmetry. By synergizing the precision of technical analysis with the contextual understanding of Large Language Models, it provides a sophisticated lens through which to view the markets. The architecture proposed—modular, event-driven, and containerized—ensures that the project is not just a theoretical experiment but a sustainable, extensible platform capable of growing with the developer's expertise. This report provides the roadmap; the execution lies in the disciplined implementation of these components.

Works cited

1. Large Language Models in equity markets: applications, techniques, and insights - PMC, accessed January 8, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC12421730/
2. Build an intelligent financial analysis agent with LangGraph and Strands Agents - AWS, accessed January 8, 2026, https://aws.amazon.com/blogs/machine-learning/build-an-intelligent-financial-analysis-agent-with-langgraph-and-strands-agents/
3. An End-To-End LLM Enhanced Trading System - arXiv, accessed January 8, 2026, https://arxiv.org/html/2502.01574v1
4. Building an AI-Powered FX Trading Signal Generator with Apple-Inspired Design - Medium, accessed January 8, 2026, https://medium.com/@kenichiroouchi/heres-an-engaging-medium-article-introduction-for-your-fx-trading-signal-generator-dec7bec6eca8
5. Hybrid Sentiment-Momentum (HSM) Model by Mouhamad Abushaqra - QuantConnect.com, accessed January 8, 2026, https://www.quantconnect.com/forum/discussion/19151/hybrid-sentiment-momentum-hsm-model/
6. Leveraging Large Language Models for Sentiment Analysis and Investment Strategy Development in Financial Markets - MDPI, accessed January 8, 2026, https://www.mdpi.com/0718-1876/20/2/77
7. How To Avoid Bias in Backtesting | For Traders, accessed January 8, 2026, https://www.fortraders.com/blog/how-to-avoid-bias-in-backtesting
8. Sharpe vs. Sortino: Unveiling Risk Metrics in Volatile Investment Landscapes, accessed January 8, 2026, https://investwithcarl.com/learning-center/investment-basics/sharpe-vs-sortino-unveiling-risk-metrics-in-volatile-investment-landscapes
9. DigitalOcean vs Render - GetDeploying, accessed January 8, 2026, https://getdeploying.com/digitalocean-vs-render
10. PART TWO: Setup — Building a Production-Ready Algorithmic Trading Framework

| by Joseph Edginton-Foy | Medium, accessed January 8, 2026, https://medium.com/@joeedgintonfoy/setup-part-two-building-a-production-ready-algorithmic-trading-framework-c3471d8e40dc

11. Real Time Notification using Celery , Redis and web sockets with Django - Medium, accessed January 8, 2026, https://medium.com/@chiheb.mhamdi/real-time-notification-using-celery-redis-and-web-sockets-with-django-704175c3182c
12. Architecture | Docs - Redis, accessed January 8, 2026, https://redis.io/docs/latest/integrate/redis-data-integration/architecture/
13. Key Insights to Celery - Sanchit Ahuja - Medium, accessed January 8, 2026, https://sanchit-ahuja.medium.com/key-insights-to-celery-29e840d5df6d
14. Ultimate guide to Celery library in Python - Deepnote, accessed January 8, 2026, https://deepnote.com/blog/ultimate-guide-to-celery-library-in-python
15. Financial Engineering for Algo Trading: Yahoo Finance and TA-Lib in Python - YouTube, accessed January 8, 2026, https://www.youtube.com/watch?v=BzCqZ03TSAw
16. Guide to PostgreSQL Database Design | Tiger Data, accessed January 8, 2026, https://www.tigerdata.com/learn/guide-to-postgresql-database-design
17. Best Practices for Optimizing PostgreSQL RDS for Time-Series Data - Reddit, accessed January 8, 2026, https://www.reddit.com/r/PostgreSQL/comments/1h3rhuq/best_practices_for_optimizing_postgresql_rds_for/
18. IEX Cloud Has Shut Down: Analysis & Migration Guide - Alpha Vantage, accessed January 8, 2026, https://www.alphavantage.co/iexcloud_shutdown_analysis_and_migration/
19. OpenBB-finance/OpenBB: Financial data platform for analysts, quants and AI agents. - GitHub, accessed January 8, 2026, https://github.com/OpenBB-finance/OpenBB
20. Finbert · Models - Dataloop, accessed January 8, 2026, https://dataloop.ai/library/model/prosusai_finbert/
21. GPT-4 for Financial Statements: Building an AI Analyst - MLQ.ai, accessed January 8, 2026, https://blog.mlq.ai/gpt-4-financial-statements-ai-analyst/
22. Candlestick charts in Python - Plotly, accessed January 8, 2026, https://plotly.com/python/candlestick-charts/
23. Plotly Dash - Real time candlestick dashboards (Python only) - YouTube, accessed January 8, 2026, https://www.youtube.com/watch?v=2pdJX1kYvR4
24. Digital Ocean or Python Anywhere for deploying? : r/django - Reddit, accessed January 8, 2026, https://www.reddit.com/r/django/comments/x4894p/digital_ocean_or_python_anywhere_for_deploying/
25. Event-Driven Serverless Pipeline for Real-Time Financial News NLP and Sentiment Scoring - GitHub, accessed January 8, 2026, https://github.com/nakuleshj/news-nlp-pipeline
26. Data pipelines | Docs - Redis, accessed January 8, 2026, https://redis.io/docs/latest/integrate/redis-data-integration/data-pipelines/

27. Customer Support - Alpha Vantage, accessed January 8, 2026, https://www.alphavantage.co/support/
28. yfinance - PyPI, accessed January 8, 2026, https://pypi.org/project/yfinance/
29. yfinance Update 2025: What You NEED to Know! - YouTube, accessed January 8, 2026, https://www.youtube.com/watch?v=037_883wHGo
30. Free Financial API Showdown: Marketstack vs. Alpha Vantage vs. Yahoo Finance - Medium, accessed January 8, 2026, https://medium.com/@rameshchauhan0089/free-financial-api-showdown-marketstack-vs-alpha-vantage-vs-yahoo-finance-f7227869ef1c
31. PostgreSQL Time-Series Best Practices: Stock Exchange System Database - Alibaba Cloud, accessed January 8, 2026, https://www.alibabacloud.com/blog/postgresql-time-series-best-practices-stock-exchange-system-database_594815
32. TA-Lib - Technical Analysis Library, accessed January 8, 2026, https://ta-lib.org/
33. Algorithmic trading based on Technical Analysis in Python | by Eryk Lewinson | Inside BUX, accessed January 8, 2026, https://medium.com/inside-bux/algorithmic-trading-based-on-technical-analysis-in-python-4ad165d426f2
34. How to Combine MACD and RSI for Better Trading Decisions - WunderTrading, accessed January 8, 2026, https://wundertrading.com/journal/en/learn/article/combine-macd-and-rsi
35. Step-by-Step DIY Guide: Hugging Face FinBERT AI Model Setup for News Sentiment, accessed January 8, 2026, https://medium.datadriveninvestor.com/step-by-step-diy-guide-hugging-face-finbert-ai-model-setup-for-news-sentiment-779c62f58b16
36. Real-Time Stock News Sentiment Prediction with Python - InsightBig, accessed January 8, 2026, https://www.insightbig.com/post/real-time-stock-news-sentiment-prediction-with-python
37. How to Get Earnings Call Transcripts with FMP APIs - Financial Modeling Prep, accessed January 8, 2026, https://site.financialmodelingprep.com/how-to/how-to-get-earnings-call-transcripts-with-fmp-apis
38. ChatGPT prompts to enhance trading - Pepperstone, accessed January 8, 2026, https://pepperstone.com/en/learn-to-trade/trading-guides/how-to-use-chatgpt-in-trading/
39. 25 AI Prompts for Finance Leaders - Nilus, accessed January 8, 2026, https://www.nilus.com/post/25-ai-prompts-for-finance-leaders
40. Backtesting Systematic Trading Strategies in Python: Considerations and Open Source Frameworks | QuantStart, accessed January 8, 2026, https://www.quantstart.com/articles/backtesting-systematic-trading-strategies-in-python-considerations-and-open-source-frameworks/
41. Battle-Tested Backtesters: Comparing VectorBT, Zipline, and Backtrader for Financial Strategy Development | by Trading Dude | Medium, accessed January 8, 2026,

https://medium.com/@trading.dude/battle-tested-backtesters-comparing-vectorbt-zipline-and-backtrader-for-financial-strategy-dee33d33a9e0

42. Different results in Backtrader vs Backtesting.py : r/algotrading - Reddit, accessed January 8, 2026, https://www.reddit.com/r/algotrading/comments/1o09tdu/different_results_in_backtrader_vs_backtestingpy/
43. Look-Ahead Bias Prevention and Signal Processing in Quantitative Trading | by Jakub Polec, accessed January 8, 2026, https://medium.com/@jpolec_72972/look-ahead-bias-prevention-and-signal-processing-in-quantitative-trading-9def856db5a6
44. Look-Ahead Bias - Definition and Practical Example - Corporate Finance Institute, accessed January 8, 2026, https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/look-ahead-bias/
45. Custom Calculation Data Points - Morningstar, accessed January 8, 2026, https://morningstardirect.morningstar.com/clientcomm/customcalculations.pdf
46. Quickly compute Value at Risk with Monte Carlo - PyQuant News, accessed January 8, 2026, https://www.pyquantnews.com/the-pyquant-newsletter/quickly-compute-value-at-risk-with-monte-carlo
47. cristianleoo/montecarlo-portfolio-management: Portfolio Management with Monte Carlo Simulation - GitHub, accessed January 8, 2026, https://github.com/cristianleoo/montecarlo-portfolio-management
48. Value at Risk (VaR) In Python: Monte Carlo Method - YouTube, accessed January 8, 2026, https://www.youtube.com/watch?v=X8aNFXJEENs
49. Visualising your trade entries and exits using Plotly - YouTube, accessed January 8, 2026, https://www.youtube.com/watch?v=5GhiWjNygkY
50. Quant Dev Projects : r/quant - Reddit, accessed January 8, 2026, https://www.reddit.com/r/quant/comments/1ea8j6b/quant_dev_projects/
51. LSEG-API-Samples/Article.DataLibrary.Python.NewsSentimentWithLLM: Python script for analyzing financial news sentiment using GPT-based Large Language Models (LLMs). It demonstrates how to leverage LSEG's Data Library to fetch news headlines and apply LLMs for sentiment classification, helping in the development of advanced investment strategies - GitHub, accessed January 8, 2026, https://github.com/LSEG-API-Samples/Article.DataLibrary.Python.NewsSentimentWithLLM
52. 6 Quantitative Developer Interview Questions and Answers for 2026 - Himalayas.app, accessed January 8, 2026, https://himalayas.app/interview-questions/quantitative-developer