

TradeMind - Intelligent Algorithmic Trading System

Final Project Report

Field	Details
Project Title	TradeMind - Intelligent Algorithmic Trading System
Subtitle	State-Conditional Portfolio Allocation & Web Prediction Platform
Authors	Jlaa Amara, Abdallah Ismail
Supervisor	Dvir Ross
Department	Computer Science
Institution	Sami Shamoon College of Engineering (SCE)
Academic Year	2025-2026
Date	June 2026
Repository	https://github.com/jlaaamara23/markov_chain
Live Demo	https://markov-chain-2.onrender.com/

Table of Contents

- 1. Abstract
- 2. Introduction
- 3. Literature Review
- 4. Problem Definition and Objectives
- 5. Requirements Analysis
- 6. System Architecture and Design

7. Implementation
 8. AI Models and Algorithms
 9. Database and Data Design
 10. User Interface
 11. Testing and Validation
 12. Performance Evaluation
 13. Results and Discussion
 14. Limitations and Challenges
 15. Future Work
 16. Conclusion
 17. References
 18. Appendices
-

Abstract

Financial markets are inherently stochastic and highly volatile, making consistent stock market prediction a challenging endeavor. Traditional trading tools often rely on static technical indicators or subjective analyses that fail to capture the probabilistic nature of market movements. This project introduces **TradeMind**, an Intelligent Algorithmic Trading System that leverages variable-order Markov chains and advanced technical analysis to provide state-conditional portfolio allocation and risk management.

By discretizing continuous daily returns into a defined state space of ten bins and analyzing historical transition frequencies, the system builds empirical transition matrices that predict the probability distribution of future market states. The TradeMind platform integrates a robust Python backend powered by FastAPI with a responsive React frontend dashboard. It automatically retrieves real-time and historical OHLCV data via the `yfinance` API, processes technical indicators such as Moving Averages, RSI, and Momentum, and combines them with Markov chain forecasts to generate a comprehensive profit-potential score.

The system evaluates assets, classifies their risk levels into three categories (Low, Medium, High), and provides actionable recommendations (Strong Buy,

Buy, Hold, Avoid) alongside an optimized portfolio allocation strategy. Through Monte Carlo simulations over configurable forecast horizons (1–60 days), TradeMind demonstrates that mathematical probability models can effectively capture regime persistence and provide traders with transparent, data-driven insights for better risk management. The live platform is deployed on Render.com and is publicly accessible.

Keywords: Algorithmic Trading, Markov Chain, Variable-Order Markov Model, Monte Carlo Simulation, Portfolio Allocation, Technical Analysis, RSI, FastAPI, React, yfinance.

1. Introduction

1.1 Background

The stock market is a complex, dynamic system influenced by a myriad of economic, political, and psychological factors. For decades, investors and analysts have sought methods to predict market movements to maximize returns and minimize risks. Traditional approaches often rely on **technical analysis**—the study of past market data, primarily price and volume—or **fundamental analysis**, which evaluates a company's intrinsic value [1]. However, these methods frequently fall short in highly volatile environments because they often produce deterministic signals rather than probabilistic forecasts.

The emergence of algorithmic trading has transformed financial markets. Today, an estimated 60–80% of all equity trading volume in the United States is executed by automated systems [5]. These systems range from simple rule-based strategies to highly sophisticated machine learning models. The challenge for individual investors and smaller institutions is to access tools that are both mathematically rigorous and practically interpretable.

1.2 Problem Statement

The core problem TradeMind addresses is the challenge of **market uncertainty and the lack of probabilistic insight** in standard trading tools. Stocks do not simply move "up" or "down"; they transition through

specific ranges of returns influenced by prevailing market regimes. Traditional tools often fail to quantify the likelihood of these specific movements, leaving traders to rely on incomplete data or subjective judgment. Furthermore, most available tools treat each technical indicator in isolation, failing to synthesize a holistic, risk-adjusted view of an asset's potential.

TradeMind solves this by moving market analysis from subjective guesses into the realm of mathematical probability. The system answers the question:

"Given the recent sequence of market states, what is the probability distribution of tomorrow's return?"

1.3 Project Objectives

The primary objectives of the TradeMind project are:

1. **Probabilistic Forecasting:** To develop a variable-order Markov chain model that accurately calculates the probability distribution of future stock returns based on historical state transitions.
2. **Comprehensive Analysis:** To integrate traditional technical indicators (RSI, Moving Averages, Volatility, Momentum) with Markov chain predictions to generate a holistic profit-potential score.
3. **Actionable Insights:** To provide clear, data-driven recommendations and dynamic portfolio allocation strategies based on calculated risk levels.
4. **Interactive Visualization:** To build a user-friendly web dashboard that allows users to interact with the model, visualize state transitions via directed graphs, and understand the underlying probabilities of their investments.
5. **Transparency and Interpretability:** To expose the model's internal workings—transition matrices, score breakdowns, and probability distributions—so that users can understand why

a recommendation is made.

2. Literature Review

2.1 Algorithmic Trading and Technical Analysis

Algorithmic trading involves the use of computer programs to execute trading strategies at speeds and frequencies that are impossible for human traders. These systems often rely on technical analysis indicators to generate buy or sell signals. Key indicators include the Simple Moving Average (SMA), which smooths out price data to identify trends, and the Relative Strength Index (RSI), an oscillator that measures the speed and change of price movements to identify overbought or oversold conditions [14] [15]. While these indicators are foundational to many trading systems, they are inherently backward-looking and often struggle to adapt to sudden regime shifts in the market.

A comprehensive review by Addy et al. (2024) highlights that modern algorithmic trading systems increasingly combine multiple technical signals with machine learning models to improve robustness and reduce false signals [5]. Salehpour and Samadzamini (2023) further emphasize that ensemble methods—which combine predictions from multiple models—consistently outperform single-model approaches in financial forecasting tasks [6].

2.2 Modern Portfolio Theory and Market Efficiency

Modern Portfolio Theory (MPT), introduced by Harry Markowitz in 1952, revolutionized investment by demonstrating how risk-averse investors can construct portfolios to optimize expected return based on a given level of market risk [1]. MPT's core insight—that diversification across uncorrelated assets reduces portfolio risk without necessarily reducing expected return—remains a cornerstone of quantitative finance.

In contrast, the Efficient Market Hypothesis (EMH) posits that asset prices fully reflect all available information, making it impossible to consistently achieve higher-than-average returns [9] [20]. However, the rise of behavioral finance has challenged EMH, suggesting that psychological factors and cognitive biases cause market anomalies and momentum that can be exploited by sophisticated models [10]. The existence of short-term momentum effects, documented extensively in the literature, provides theoretical justification for the Markov chain approach used in TradeMind.

2.3 Markov Chains and Hidden Markov Models in Finance

A Markov chain is a stochastic process describing a sequence of possible events where the probability of each event depends only on the state attained in the previous event (the Markov property). In finance, Markov models are particularly useful for modeling **regime-switching behavior**—such as transitioning between bull and bear markets or periods of high and low volatility [13].

Zhang et al. (2019) demonstrated the effectiveness of High-order Hidden Markov Models (HMM) for trend prediction in financial time series, showing that incorporating multiple past states significantly improves prediction accuracy compared to first-order models [2]. This directly aligns with TradeMind's implementation of a variable-order Markov chain that utilizes a configurable context length (up to 10 days) to predict the next state.

Nystrup et al. (2017) explored the use of Hidden Markov Models with time-varying parameters to capture the long memory of financial time series, emphasizing the importance of regime identification and volatility clustering [4]. Lihn (2017) applied HMMs specifically to the S&P 500 index, finding that five-state and higher models can capture significant auto-correlation and serve as effective volatility forecasting tools [12].

2.4 Monte Carlo Simulation in Finance

Monte Carlo simulation is a computational technique that uses random sampling to obtain numerical results. In finance, it is widely used for option pricing, risk management, and portfolio forecasting [16]. By simulating thousands of possible future price paths, Monte Carlo methods provide a distribution of outcomes rather than a single point estimate, making them particularly well-suited for communicating uncertainty to investors. TradeMind employs Monte Carlo simulation (5,000 paths) to project the Markov chain state distribution over a multi-day horizon.

2.5 Positioning of TradeMind

TradeMind builds upon these foundations by combining a variable-order Markov chain for probabilistic forecasting with a deterministic scoring engine based on technical analysis. Unlike many "black box" deep learning models [7]

[8], TradeMind prioritizes interpretability. The system exposes the underlying transition matrices and expected contributions, allowing users to understand exactly

why

a specific recommendation is made. This positions TradeMind as a practical, transparent tool that bridges the gap between academic stochastic modeling and real-world investment decision support.

3. Problem Definition and Objectives

3.1 Formal Problem Definition

Given a time series of daily closing prices $\{C_1, C_2, \dots, C_T\}$ for a financial asset, the problem is to:

1. Compute the sequence of daily returns $\{R_1, R_2, \dots, R_{T-1}\}$ where $R_t = \frac{C_t}{C_{t-1}} - 1$.
2. Map each return R_t to a discrete state $s_t \in \mathcal{S} = \{s_1, s_2, \dots, s_{10}\}$ based on predefined bin edges.
3. Estimate the conditional probability distribution $P(s_{T+1} | s_{T-k+1}, \dots, s_T)$ for a context of length k .
4. Simulate the multi-step distribution $P(s_{T+n} | s_T)$ for a forecast horizon of n days.
5. Synthesize this probabilistic forecast with technical indicators to generate a risk-adjusted investment recommendation.

3.2 Target Users

TradeMind is designed for three primary user groups:

User Group	Description	Primary Use Case
Individual Investors	Retail investors seeking data-driven insights	Portfolio allocation and stock screening
Financial Analysts	Professionals requiring quantitative tools	Risk assessment and scenario analysis
Academic Researchers	Students and researchers in quantitative finance	Studying stochastic market models

4. Requirements Analysis

4.1 Functional Requirements

ID	Requirement	Priority
FR-01	The system shall accept up to 25 stock symbols per analysis request	High
FR-02	The system shall fetch historical OHLCV data for any valid US equity or ETF	High
FR-03	The system shall compute a 10-state Markov transition matrix from historical returns	High
FR-04	The system shall support variable context lengths from 1 to 10 days	High
FR-05	The system shall run Monte Carlo simulations with at least 1,000 paths	High
FR-06	The system shall compute RSI, SMA, Momentum, and Volatility for each asset	High
FR-07	The system shall generate a profit score (0-100) for each asset	High
FR-08	The system shall classify each asset as Low, Medium, or High risk	High
FR-09	The system shall provide Buy/Hold/Avoid recommendations	High
FR-10	The system shall generate a portfolio allocation for a user-defined capital amount	Medium
FR-11	The system shall render the Markov chain as a directed graph image	Medium
FR-12	The system shall display a price history chart with MA20 and MA50 overlays	Medium

4.2 Non-Functional Requirements

ID	Requirement	Priority
NFR-01	The API shall respond to single-symbol requests within 5 seconds	High
NFR-02	The system shall handle concurrent requests from multiple users	High
NFR-03	The frontend shall be responsive and usable on mobile devices	Medium
NFR-04	The system shall be deployable as a containerized service	Medium
NFR-05	All model parameters shall be configurable via API query parameters	High

5. System Architecture and Design

5.1 High-Level Architecture

TradeMind is designed as a modern, full-stack web application following a client-server architecture. The system is divided into three logical layers: a **Data Layer** responsible for data acquisition and storage, a **Backend Layer** responsible for mathematical modeling and API serving, and a **Frontend Layer** responsible for user interaction and visualization.

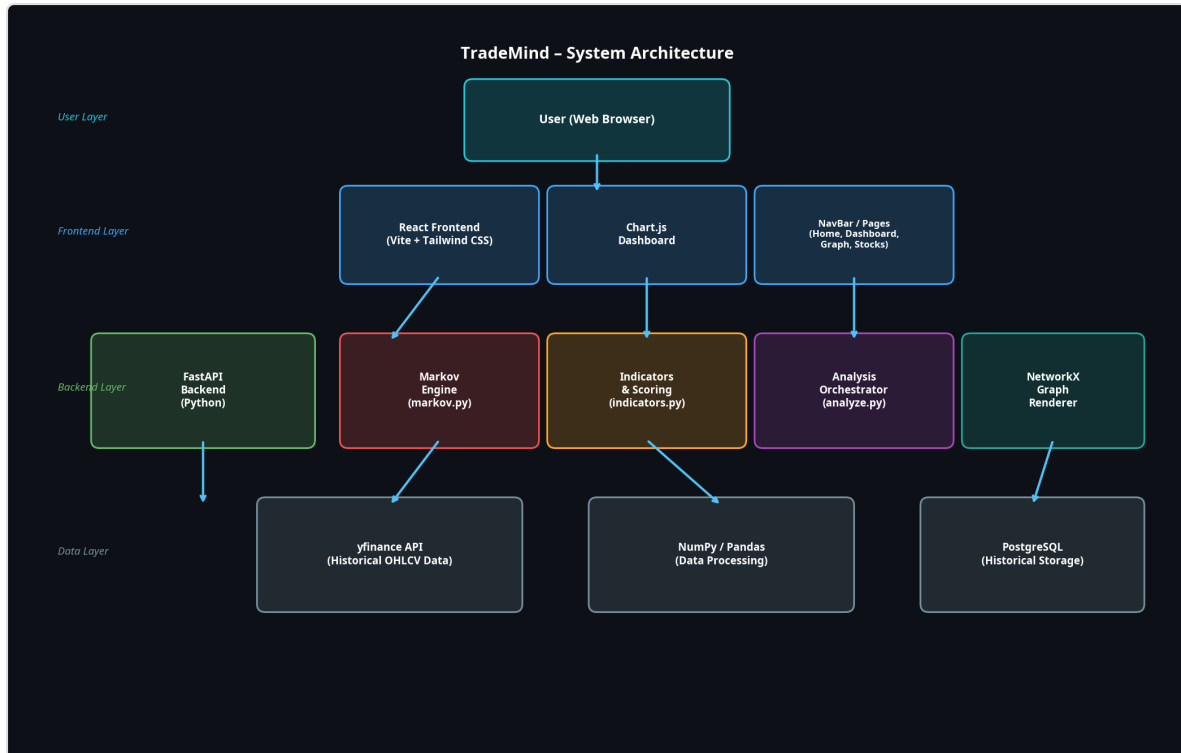


Figure 1: TradeMind High-Level System Architecture

The technology stack is summarized in the following table:

Layer	Technology	Purpose
Backend Framework	Python 3.11 + FastAPI	REST API serving and request routing
Data Acquisition	yfinance	Real-time and historical OHLCV data
Mathematical Core	NumPy, Pandas	Array operations and data manipulation
Graph Rendering	NetworkX + Matplotlib	Markov chain directed graph visualization
Frontend Framework	React 18 + Vite	Single-page application
UI Styling	Tailwind CSS	Responsive design
Data Visualization	Chart.js	Interactive price history charts
Deployment	Render.com	Cloud hosting (backend + frontend)

5.2 Backend API Design

The FastAPI backend exposes five REST endpoints:

Endpoint	Method	Description	Key Parameters
/	GET	Health check	—
/predict	GET	Single-symbol Markov prediction	symbol, period, steps, context_len
/analyze	GET	Multi-symbol full analysis	symbols, period, steps, context_len
/history/{symbol}	GET	Price history with MAs	symbol, period
/graph	GET	Markov chain graph as PNG	symbol, period, context_len

5.3 System Workflow

The end-to-end data processing pipeline is illustrated below:

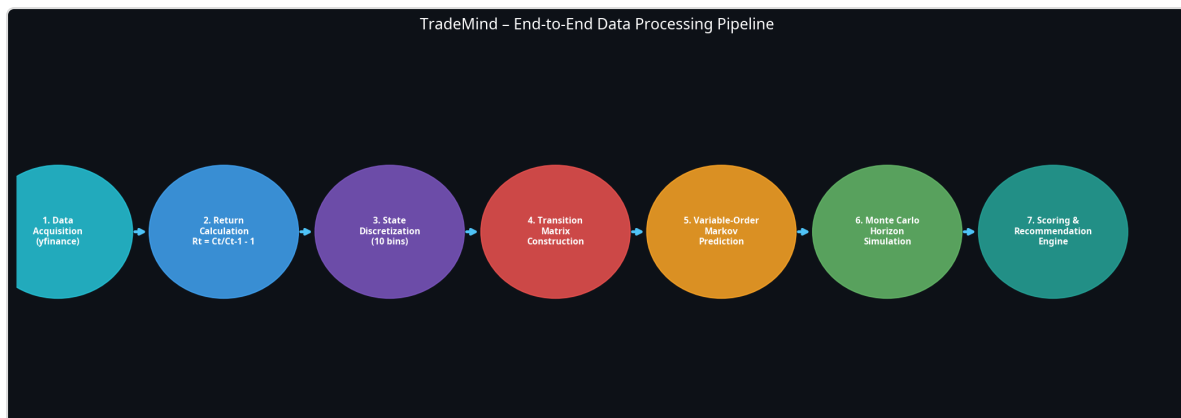


Figure 2: TradeMind End-to-End Data Processing Pipeline

The workflow proceeds through seven sequential stages: (1) Data Acquisition from yfinance, (2) Return Calculation, (3) State Discretization, (4) Transition Matrix Construction, (5) Variable-Order Markov Prediction, (6) Monte Carlo Horizon Simulation, and (7) Scoring and Recommendation Generation.

5.4 UML Class Diagram

The object-oriented design of the backend is captured in the following class diagram:

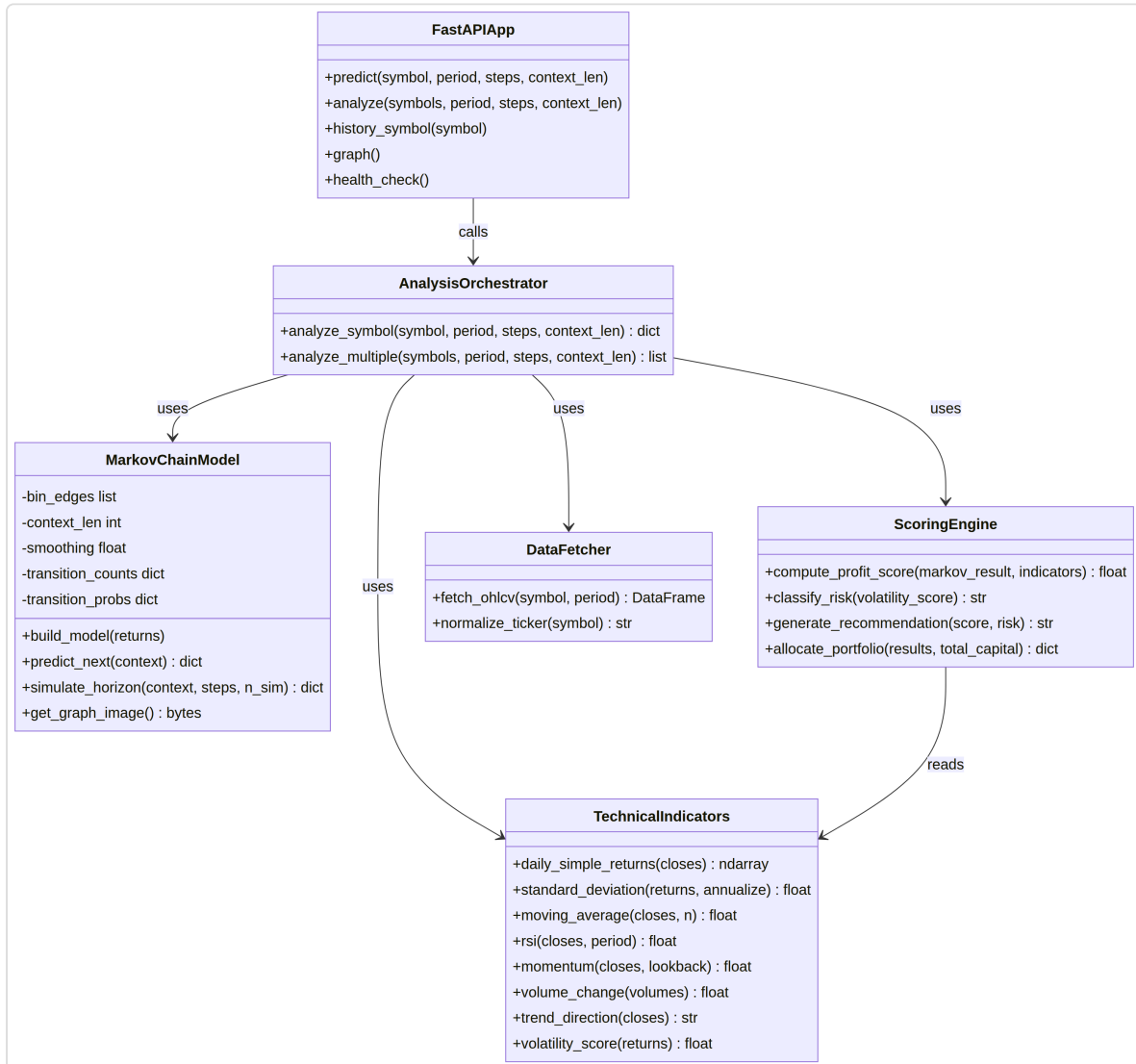


Figure 3: UML Class Diagram of the TradeMind Backend

5.5 UML Sequence Diagram

The interaction between system components for a typical analysis request is shown below:

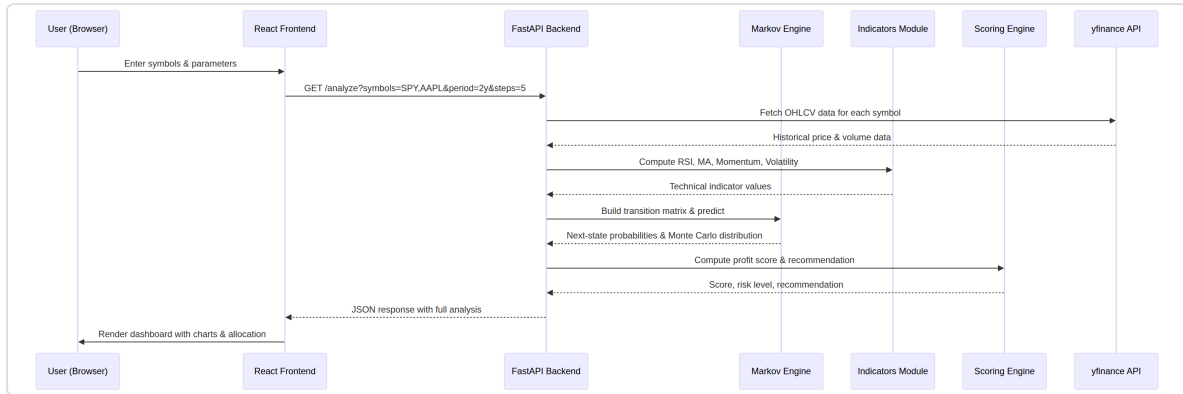


Figure 4: UML Sequence Diagram for the /analyze Endpoint

6. Implementation

6.1 Backend Implementation

6.1.1 Data Acquisition and Preprocessing

Data handling is implemented primarily using Pandas and NumPy for high-performance vector operations. The system fetches historical price data via the `yfinance` library, which provides access to Yahoo Finance's historical data API. The `ticker_util.py` module normalizes user-provided ticker symbols, handling common variants (e.g., converting "SPXL" to "SPXL" and ensuring correct exchange suffixes).

The primary metric used for modeling is the **daily simple return**, calculated as:

$$R_t = \frac{\text{Close}_t}{\text{Close}_{t-1}} - 1$$

This is implemented efficiently using Pandas' `pct_change()` method on the closing price series, followed by `dropna()` to remove the initial NaN value.

6.1.2 State-Space Discretization

To apply Markov chains to continuous financial data, the continuous daily returns are mapped into a discrete state space. TradeMind uses a defined set of bin edges to categorize returns into **10 distinct states**:

State	Return Range	Interpretation
1	$R < -3.00\%$	Strong Down
2	$-3.00\% \leq R < -1.50\%$	Moderate Down
3	$-1.50\% \leq R < -0.75\%$	Mild Down
4	$-0.75\% \leq R < -0.30\%$	Slight Down
5	$-0.30\% \leq R < -0.10\%$	Flat Down
6	$-0.10\% \leq R < 0.10\%$	Flat
7	$0.10\% \leq R < 0.50\%$	Flat Up
8	$0.50\% \leq R < 1.00\%$	Mild Up
9	$1.00\% \leq R < 2.00\%$	Moderate Up
10	$R \geq 2.00\%$	Strong Up

This multi-bucket approach provides a much more granular understanding of market movements compared to a simple binary "Up/Down" classification.

6.1.3 Variable-Order Markov Chain Model

The core predictive engine is a variable-order Markov chain implemented in `markov.py`. The model supports context lengths from 1 to 10 days. The construction and prediction process involves three key steps:

Step 1 - Matrix Construction: The algorithm iterates through the historical state sequence, building a dictionary that maps each observed context tuple (e.g., the last 5 states) to a count vector over all possible next states.

Step 2 - Laplace Smoothing: To handle unseen state transitions, Laplace (add- α) smoothing is applied. The smoothed probability of transitioning to state s_j given context C is:

$$P(s_{t+1} = s_j \mid C) = \frac{\text{Count}(C \rightarrow s_j) + \alpha}{\sum_{k=1}^{10} \text{Count}(C \rightarrow s_k) + 10\alpha}$$

where $\alpha = 1.0$ is the default smoothing parameter.

Step 3 - Backoff Mechanism: If the full context of length k has never been observed in the training data, the model recursively falls back to a

shorter context of length $k-1$, continuing until a match is found or the context length reaches zero (in which case the marginal distribution is used).

The following code listing shows the actual implementation of the variable-order chain builder and the backoff prediction function from `markov.py` :

```

# markov.py – Variable-Order Markov Chain Builder
def _build_variable_order_chain(
    state_sequence: list[str],
    state_labels: list[str],
    context_len: int,
    laplace: float,
) -> dict[int, dict[tuple[str, ...], np.ndarray]]:
    """
    Build conditional distributions P(next_state | previous k states)
    for k in [0, context_len]. Includes backoff levels, so if a long
    context is unseen at inference time we fall back to shorter contexts.
    """
    state_to_i = {s: i for i, s in enumerate(state_labels)}
    n_states = len(state_labels)
    max_k = max(0, int(context_len))
    levels: dict[int, dict[tuple[str, ...], np.ndarray]] = {}

    for k in range(0, max_k + 1):
        if len(state_sequence) <= k:
            continue
        counts_by_ctx: dict[tuple[str, ...], np.ndarray] = {}
        for i in range(k, len(state_sequence)):
            ctx = tuple(state_sequence[i - k : i]) if k > 0 else tuple()
            nxt = state_sequence[i]
            if nxt not in state_to_i:
                continue
            if ctx not in counts_by_ctx:
                counts_by_ctx[ctx] = np.full(n_states, laplace, dtype=float)
            counts_by_ctx[ctx][state_to_i[nxt]] += 1.0

        probs_by_ctx: dict[tuple[str, ...], np.ndarray] = {}
        for ctx, row in counts_by_ctx.items():
            probs_by_ctx[ctx] = row / row.sum() # row-normalize
        levels[k] = probs_by_ctx
    return levels

def _predict_next_distribution(
    history: list[str],
    chain_levels: dict[int, dict[tuple[str, ...], np.ndarray]],
    context_len: int,
) -> np.ndarray:
    """Backoff prediction: try longest context first, shorten until found."""
    for k in range(min(context_len, len(history)), -1, -1):
        ctx = tuple(history[-k:]) if k > 0 else tuple()

```

```

level = chain_levels.get(k, {})
if ctx in level:
    return level[ctx]
raise ValueError("Unable to predict next distribution from history")

```

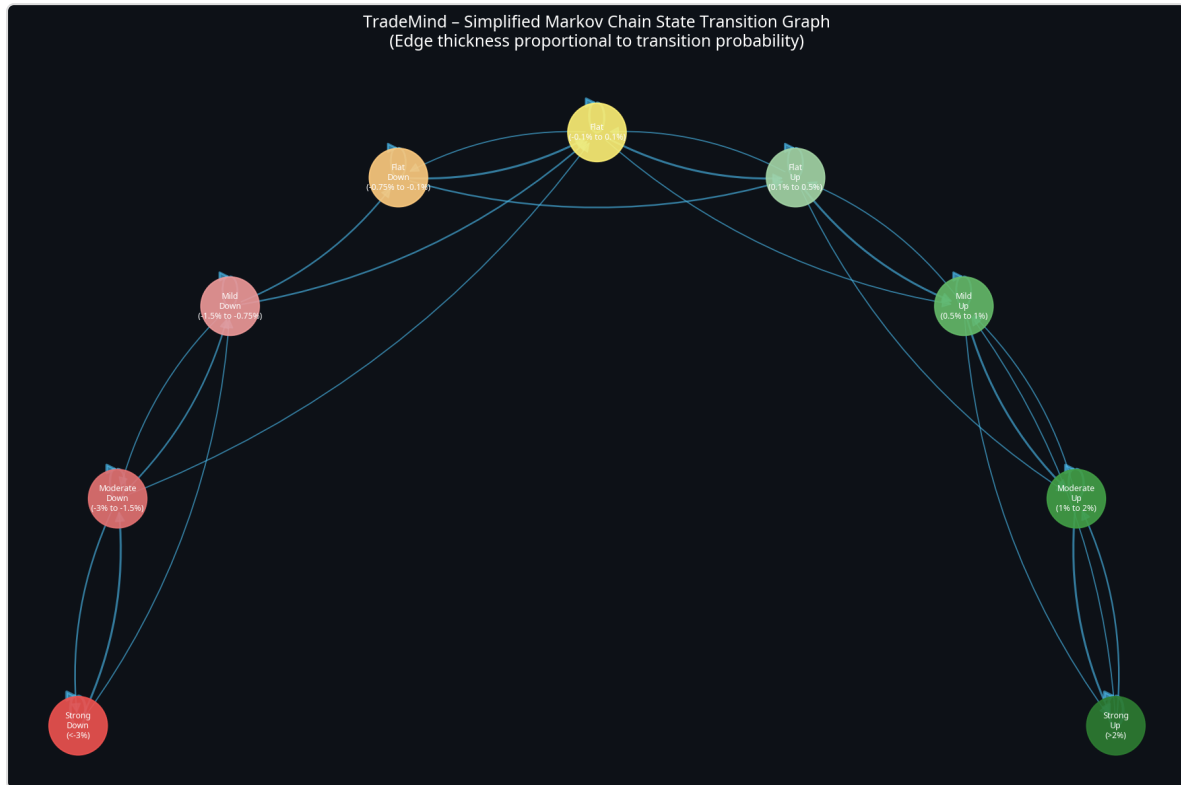
Listing 1: Variable-order Markov chain builder and backoff prediction (*markov.py*)

Figure 5: Simplified Markov Chain State Transition Graph (edge thickness proportional to probability)

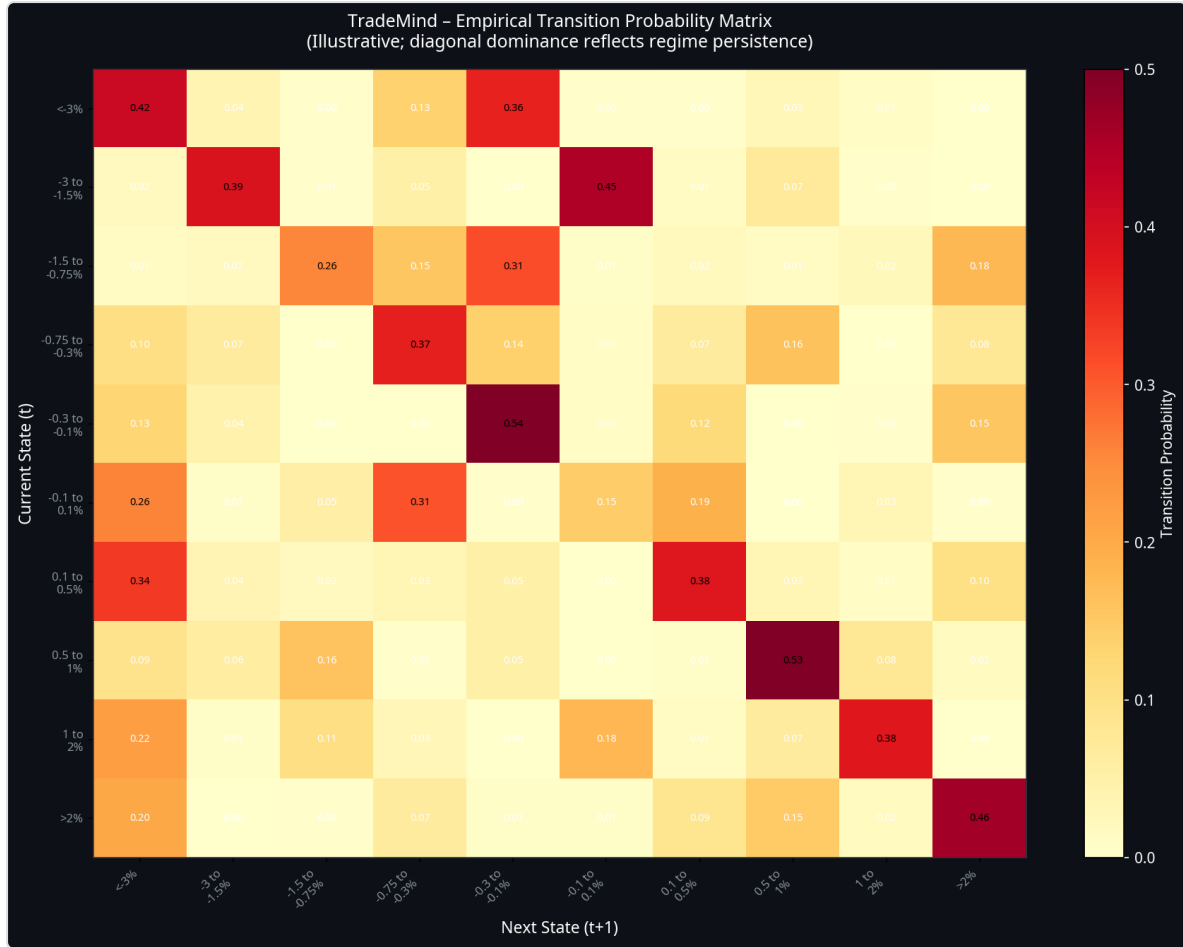


Figure 6: Empirical Transition Probability Matrix Heatmap (diagonal dominance reflects regime persistence)

6.1.4 Monte Carlo Horizon Simulation

To forecast returns over a multi-day horizon, TradeMind employs Monte Carlo simulations with 5,000 independent paths. In each simulation step, the next state is sampled from the conditional probability distribution predicted by the Markov model for the current context. After n steps, the distribution of final states across all 5,000 simulations is aggregated to form the **horizon probability distribution**.

The horizon positive probability is then calculated as:

$$P_{\text{horizon}}^{+} = \sum_{j: s_j > 0} P(s_{T+n} = s_j)$$

The following listing shows the Monte Carlo simulation implementation:

```
# markov.py – Monte Carlo Horizon Simulation
def _simulate_horizon_distribution(
    history: list[str],
    state_labels: list[str],
    chain_levels: dict[int, dict[tuple[str, ...], np.ndarray]],
    context_len: int,
    horizon_steps: int,
    n_sims: int = 5000,
) -> np.ndarray:
    state_to_i = {s: i for i, s in enumerate(state_labels)}
    final_counts = np.zeros(len(state_labels), dtype=float)
    rng = np.random.default_rng(42)  # reproducible seed

    for _ in range(n_sims):
        seq = list(history)
        for _step in range(horizon_steps):
            dist = _predict_next_distribution(seq, chain_levels, context_len)
            next_i = int(rng.choice(len(state_labels), p=dist))
            seq.append(state_labels[next_i])
        final_state = seq[-1]
        final_counts[state_to_i[final_state]] += 1.0

    return final_counts / final_counts.sum()  # normalize to probability
```

Listing 2: Monte Carlo horizon simulation — 5,000 paths, reproducible seed (*markov.py*)

6.2 Technical Indicators Module

The `indicators.py` module computes all technical indicators using pure NumPy operations for maximum performance. All functions are deliberately **pure** — they accept only NumPy arrays and return scalars, making them independently unit-testable without any I/O.

Relative Strength Index (RSI): Wilder's RSI over a 14-day period is computed as:

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

where $RS = \frac{\text{Average Gain}}{\text{Average Loss}}$ over the lookback period, using Wilder's exponential smoothing.

Annualized Volatility: The annualized standard deviation of daily returns is:

$$\sigma_{\text{ann}} = \sigma_{\text{daily}} \times \sqrt{252}$$

Momentum: The 20-day rate of change is:

$$Momentum_{20d} = \frac{C_T - C_{T-20}}{C_{T-20}} \times 100\%$$

Moving Averages: Simple moving averages over 20 and 50 periods are used to determine trend direction. An "uptrend" is identified when $MA20 > MA50$.

The following listing shows the RSI implementation with Wilder's smoothing and the composite `compute_all_indicators` function:

```

# indicators.py – RSI with Wilder's Smoothing
def rsi(close: np.ndarray, period: int = 14) -> float:
    """Relative Strength Index using Wilder's smoothing.
    Returns a value in [0, 100]. Falls back to 50.0 (neutral) when
    there is not enough history to compute the indicator reliably.
    """
    if close.size < period + 1:
        return 50.0

    diffs = np.diff(close)
    gains = np.where(diffs > 0, diffs, 0.0)
    losses = np.where(diffs < 0, -diffs, 0.0)

    avg_gain = float(np.mean(gains[:period]))
    avg_loss = float(np.mean(losses[:period]))

    # Wilder smoothing: avg_t = (avg_{t-1} * (n-1) + value_t) / n
    for i in range(period, len(diffs)):
        avg_gain = (avg_gain * (period - 1) + float(gains[i])) / period
        avg_loss = (avg_loss * (period - 1) + float(losses[i])) / period

    if avg_loss == 0.0:
        return 100.0 if avg_gain > 0 else 50.0
    rs = avg_gain / avg_loss
    return float(100.0 - (100.0 / (1.0 + rs)))

def compute_all_indicators(close: np.ndarray, volume: np.ndarray) -> dict[str, Any]:
    """Compute every indicator exposed to the API in a single pass."""
    returns = daily_simple_returns(close)
    return {
        "stdev_daily": round(standard_deviation(returns), 6),
        "stdev_annualized": round(standard_deviation(returns, annualize=True), 6),
        "ma_20": round(moving_average(close, 20), 6),
        "ma_50": round(moving_average(close, 50), 6),
        "rsi_14": round(rsi(close, 14), 4),
        "momentum_20d": round(momentum(close, 20), 6),
        "momentum_60d": round(momentum(close, 60), 6),
        "volume_change": round(volume_change(volume), 6),
        "trend_direction": trend_direction(close),
        "historical_growth": round(historical_growth(close), 6),
        "volatility_score": round(volatility_score(returns), 4),
    }

```

Listing 3: RSI with Wilder's smoothing and composite indicator computation
(`indicators.py`)

6.3 Scoring Engine

The `scoring.py` module synthesizes all outputs into a single profit score. The score is a weighted sum of seven components:

Component	Weight	Calculation
Horizon Positive Probability	25%	Markov P(+) over forecast horizon, scaled 0-25
Next-Day Positive Probability	10%	Markov P(+) next day, scaled 0-10
20-Day Momentum	20%	Momentum mapped to 0-20 via sigmoid
Trend Direction	15%	MA cross: +15 (up), +7.5 (sideways), 0 (down)
Volume Change	10%	Recent vs. baseline volume ratio, scaled 0-10
RSI Proximity	10%	Bell curve centered at RSI=55, scaled 0-10
Volatility Penalty	10%	Inverse of volatility score, scaled 0-10

The final recommendation is generated via a decision matrix:

Condition	Recommendation
Score $\geq$ 70 AND Risk $\in$	Strong Buy
Score $\geq$ 55	Buy
Score $\geq$ 40	Hold
Score < 40	Avoid

The following listing shows the complete scoring engine implementation:

```

# scoring.py – Profit Score Computation & Recommendation Engine

# Component weights – sum to 1.0 for a 0..100 final score
_WEIGHTS: dict[str, float] = {
    "horizon_positive": 0.25,
    "next_positive": 0.10,
    "momentum": 0.20,
    "trend": 0.15,
    "volume": 0.10,
    "rsi": 0.10,
    "volatility_penalty": 0.10,
}

def _score_rsi(rsi_value: float) -> float:
    """Bell curve centered around 55 (slightly bullish optimum).
    Overbought (>70) and oversold (<30) both score lower."""
    distance = abs(rsi_value - 55.0)
    return _clamp(1.0 - (distance / 35.0), 0.0, 1.0)

def _score_trend(trend_direction: str, ma_20: float, ma_50: float,
                 last_close: float) -> float:
    """Combine MA cross signal with the discrete trend tag."""
    base = {"up": 0.7, "sideways": 0.5, "down": 0.2}.get(trend_direction, 0.5)
    if ma_20 > 0 and ma_50 > 0 and last_close > 0:
        if ma_20 > ma_50 and last_close > ma_20: # golden cross + price above
            return _clamp(base + 0.3, 0.0, 1.0)
        if ma_20 < ma_50 and last_close < ma_20: # death cross + price below
            return _clamp(base - 0.2, 0.0, 1.0)
    return base

def compute_profit_score(indicators, markov_metrics, last_close) -> dict:
    """Combine indicators and Markov probabilities into a 0..100 score."""
    components = {
        "horizon_positive": _clamp(markov_metrics["horizon_positive_probability"], 0, 1),
        "next_positive": _clamp(markov_metrics["next_positive_probability"], 0, 1),
        "momentum": _score_momentum(indicators["momentum_20d"]),
        "trend": _score_trend(indicators["trend_direction"],
                              indicators["ma_20"], indicators["ma_50"],
                              last_close),
        "volume": _score_volume(indicators["volume_change"]),
        "rsi": _score_rsi(indicators["rsi_14"]),
        "volatility_penalty": _score_volatility_penalty(indicators["volatility_score"]),
    }
    weighted_total = sum(components[k] * _WEIGHTS[k] for k in _WEIGHTS)
    score = round(weighted_total * 100.0, 2)

```

```

breakdown = {
    name: {"value": round(components[name], 4),
           "weight": _WEIGHTS[name],
           "contribution": round(components[name] * _WEIGHTS[name] * 100, 2)}
    for name in _WEIGHTS
}
return {"profit_score": score, "score_breakdown": breakdown}

def compute_recommendation(profit_score: float, risk_level: str) -> str:
    """Decision matrix combining profit potential and risk."""
    if profit_score >= 70 and risk_level in ("low", "medium"):
        return "strong_buy"
    if profit_score >= 55:
        return "buy"
    if profit_score >= 40:
        return "hold"
    return "avoid"

```

Listing 4: Profit score computation and recommendation engine (*scoring.py*)

6.4 Frontend Implementation

The React frontend is built with Vite for fast development and Tailwind CSS for responsive styling. The application consists of four main pages:

- **Home Page:** Landing page with a brief description of TradeMind and a "Run Prediction" call-to-action button.
- **Dashboard Page:** The main analysis interface. Users input symbols and parameters, and the system renders the ranked stocks table, individual stock details, and the portfolio allocation panel.
- **Graph Page:** Displays the Markov chain directed graph for a selected symbol, rendered server-side as a PNG image.
- **Stocks Page:** Provides individual stock analysis with a price history chart overlaid with MA20 and MA50.

7. User Interface

7.1 Home Page

The home page provides a clean, professional landing experience with a dark theme consistent with financial trading platforms. It presents the core value proposition of TradeMind and directs users to the Dashboard.

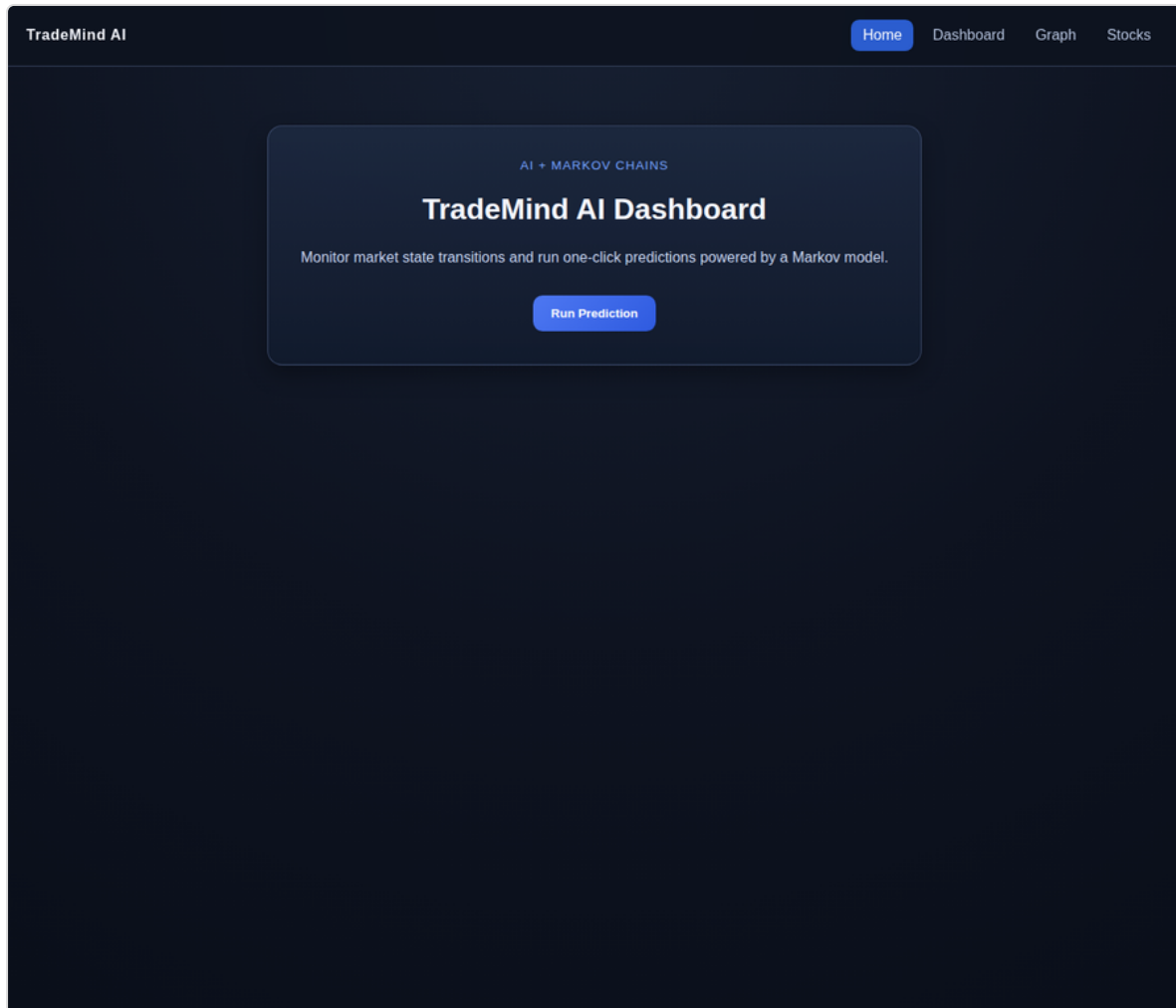


Figure 11: TradeMind Home Page – Landing screen with navigation bar and Run Prediction call-to-action

7.2 Dashboard

The Dashboard is the primary user interface. It features:

- **Input Panel:** Symbol input (comma-separated), period selector (1y, 2y, 5y), horizon selector (1–60 days), and context length selector (1–10 days).

- **Ranked Stocks Table:** A sortable table showing each analyzed asset's recommendation badge, risk badge, profit score, last close price, annualized standard deviation, RSI, 20-day momentum, and 5-day positive probability.
- **Stock Detail Panel:** Clicking on a row expands a detailed view showing the full Markov probability distribution chart, score breakdown, and price history chart.
- **Portfolio Allocation Panel:** Displays the recommended capital allocation across assets for a user-specified investment amount.

The screenshot displays the TradeMind AI Dashboard with the 'Investment analysis' section active. The dashboard has a dark theme with a top navigation bar containing 'Home', 'Dashboard' (highlighted), 'Graph', and 'Stocks'. The 'Investment analysis' section includes a descriptive paragraph: 'Combines technical indicators (stdev, moving averages, RSI, momentum, volume) with the Markov chain to produce a profit-potential score, a risk level, and a recommendation per ticker. Higher score = stronger forward setup.' Below this is an input panel with four main sections: 'SYMBOLS (WATCHLIST)' with a text input field containing 'Add tickers (comma or space separated), e.g. NVDA, M:' and buttons for 'Add symbols' and 'Refresh'; 'HISTORY WINDOW' with a dropdown menu set to '2 years'; 'HORIZON (TRADING DAYS)' with a text input field set to '5'; and 'CONTEXT DAYS' with a text input field set to '5'. At the bottom of the input panel, there are four tags: 'SPY x', 'AAPL x', 'MSFT x', and 'NVDA x'. Below the input panel is a status bar that says 'Running analysis...'.

Figure 12: TradeMind Dashboard – Input panel with watchlist symbols and analysis parameters

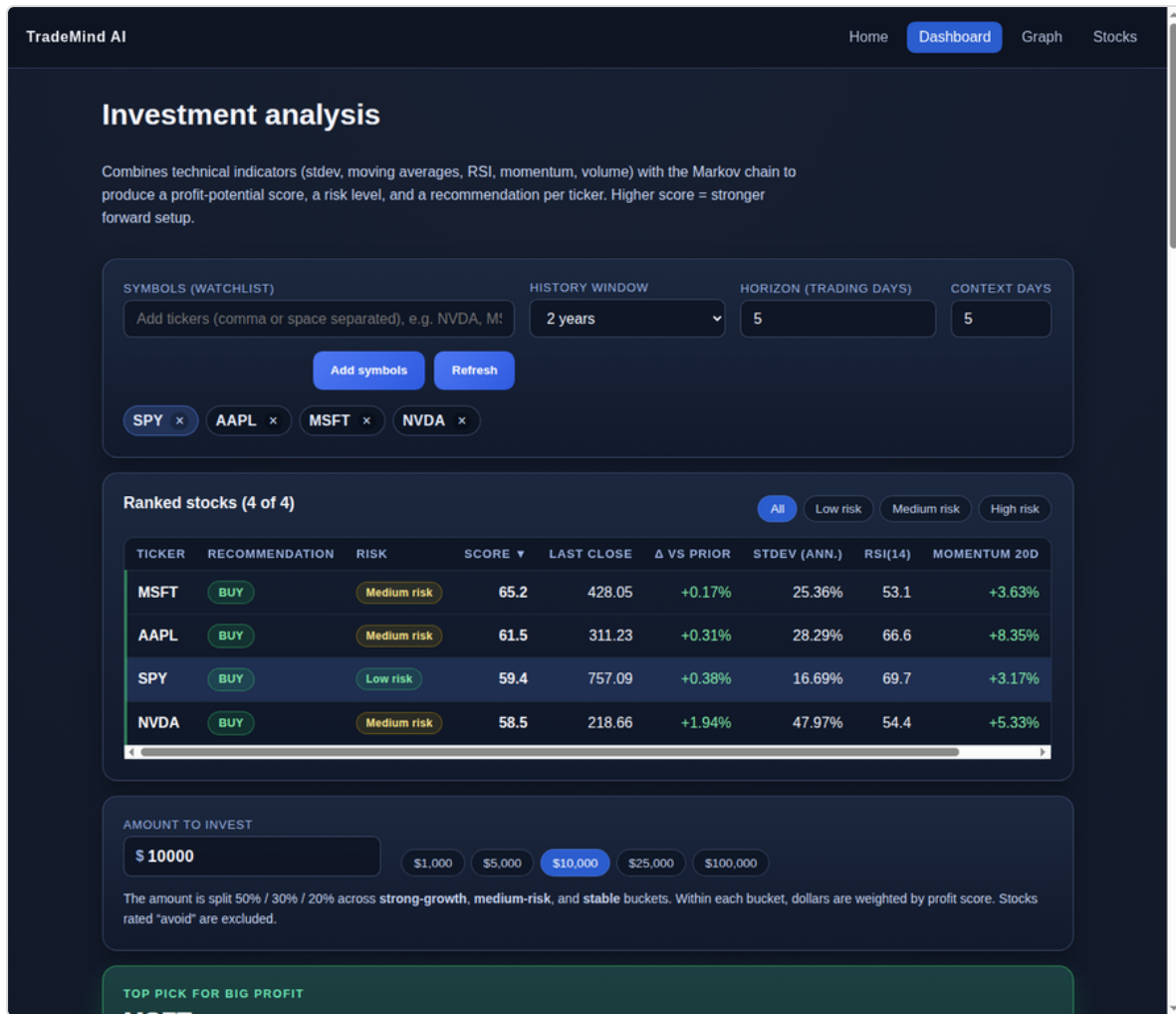


Figure 13: TradeMind Dashboard – Ranked stocks table showing live analysis results for SPY, AAPL, MSFT, and NVDA (June 5, 2026)

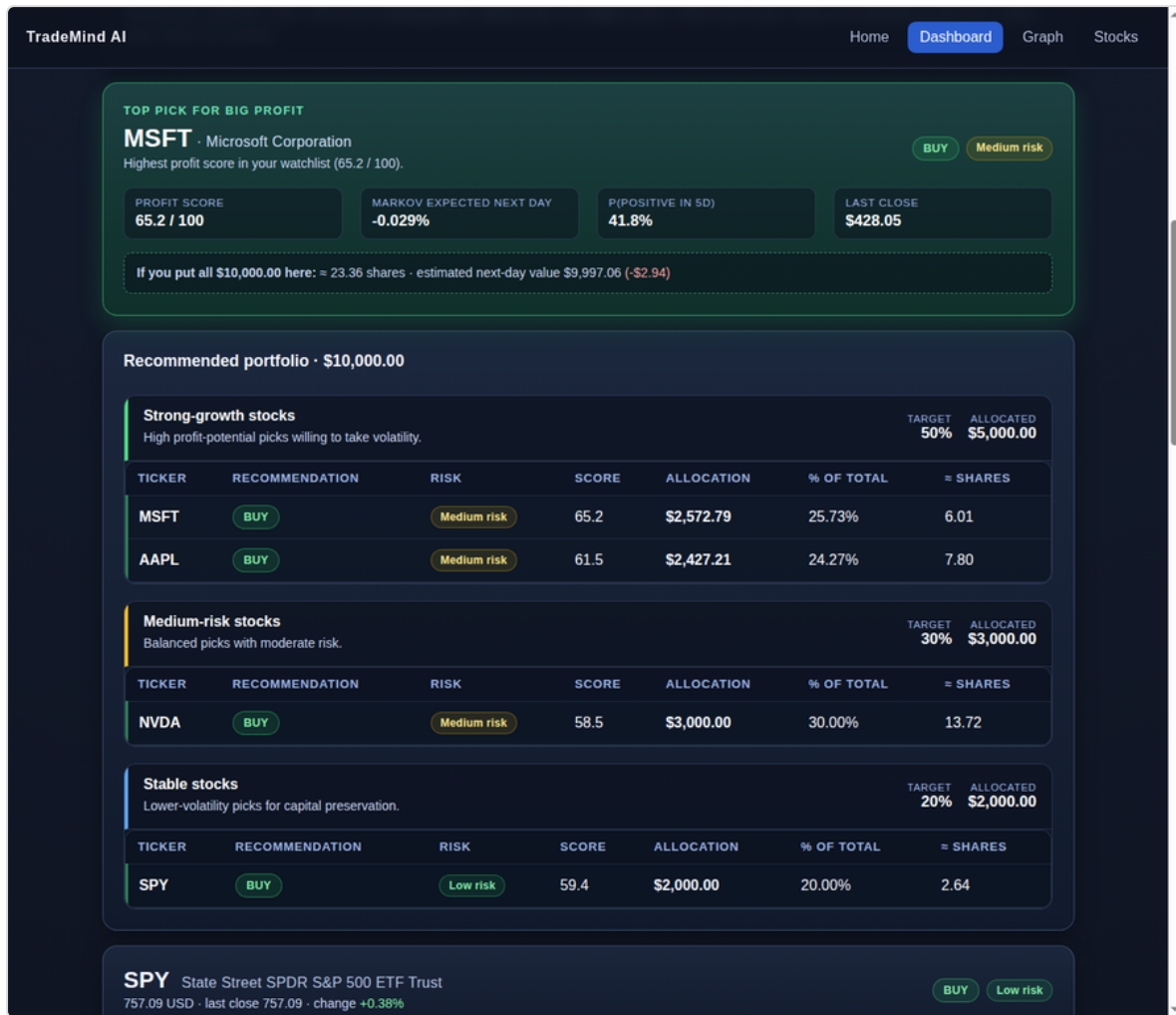


Figure 14: TradeMind Dashboard – Recommended portfolio allocation for a \$10,000 investment across three risk buckets

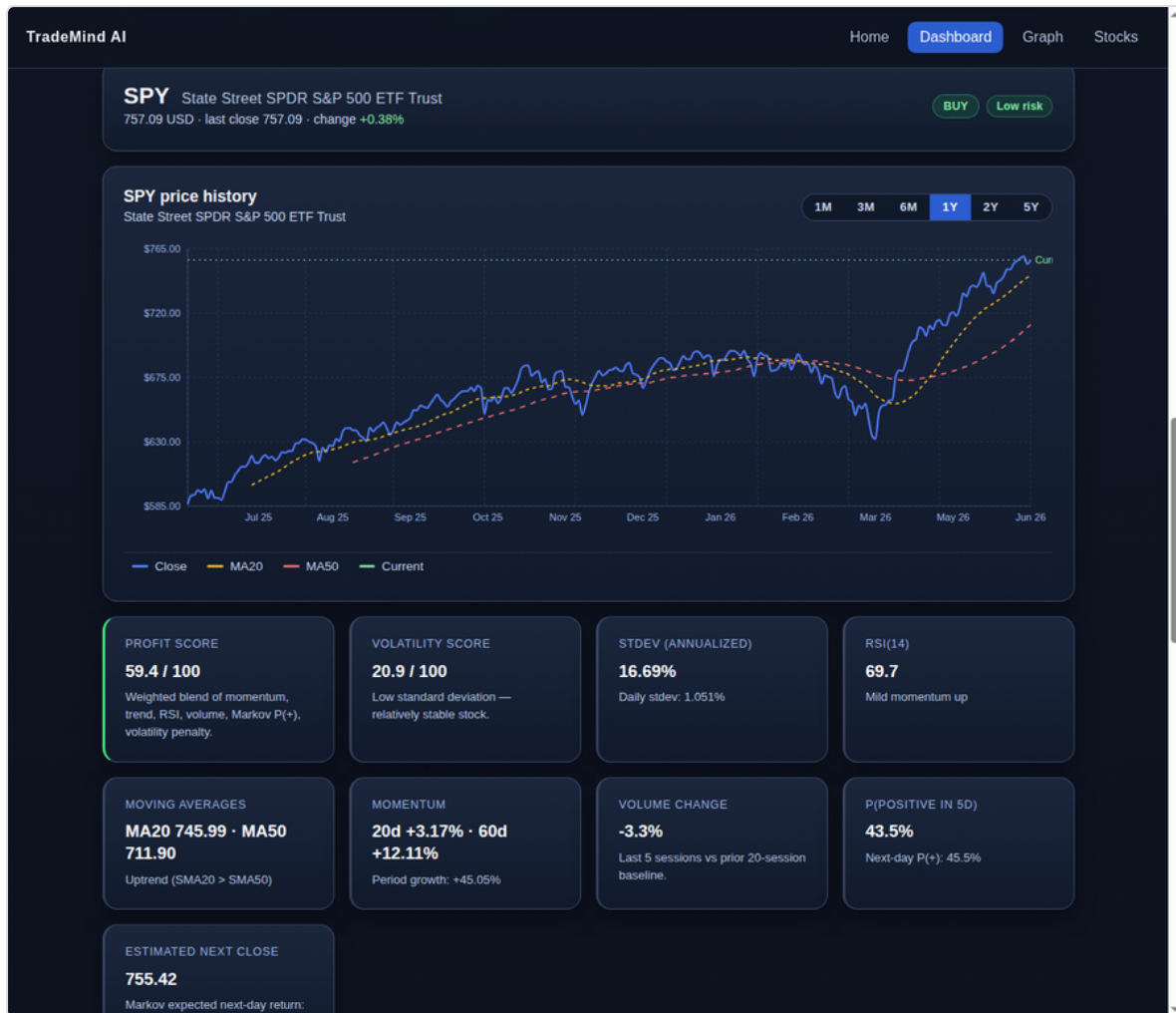


Figure 15: TradeMind Dashboard – SPY price history chart with MA20 and MA50 overlays, score breakdown, and Markov probability details

7.3 Graph Page

The Graph page renders the Markov chain state transition graph as a directed network diagram. Nodes represent market states (return bins), and edges represent transitions with probabilities above a configurable threshold (default ≥ 0.05). This visualization makes the stochastic model directly interpretable to users, allowing them to see which market states are most likely to follow from the current state.

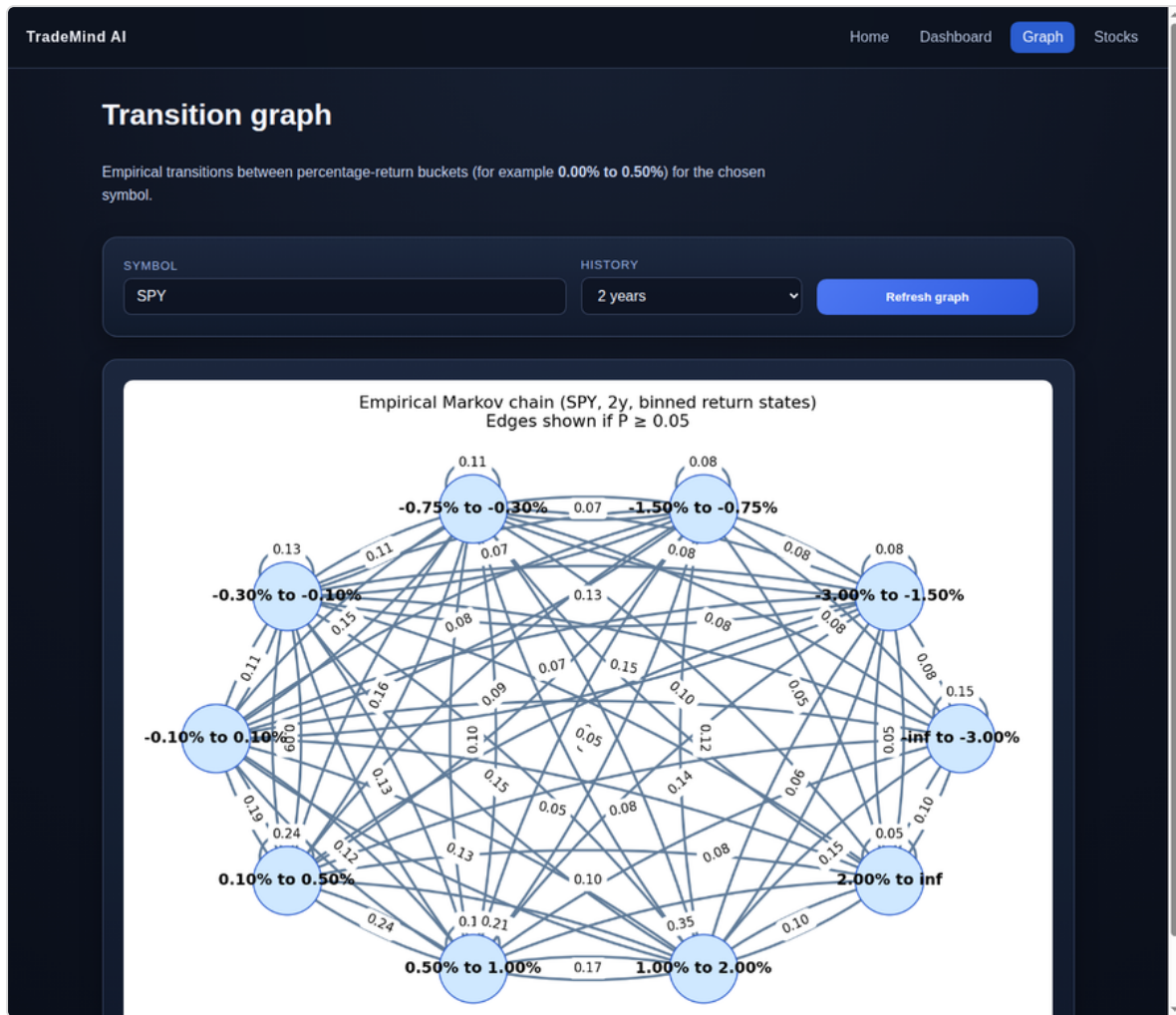


Figure 16: TradeMind Graph Page – Empirical Markov chain transition graph for SPY (2-year history). Each node is a return-percentage bucket; edge weights show transition probabilities ≥ 0.05

7.4 Stocks Page

The Stocks page provides a quick-lookup tool for individual stock quotes. Users can search by name or symbol from a curated list, or enter any valid ticker for a live Yahoo Finance quote. The page displays the last close, prior close, change, and percentage change for the selected asset.

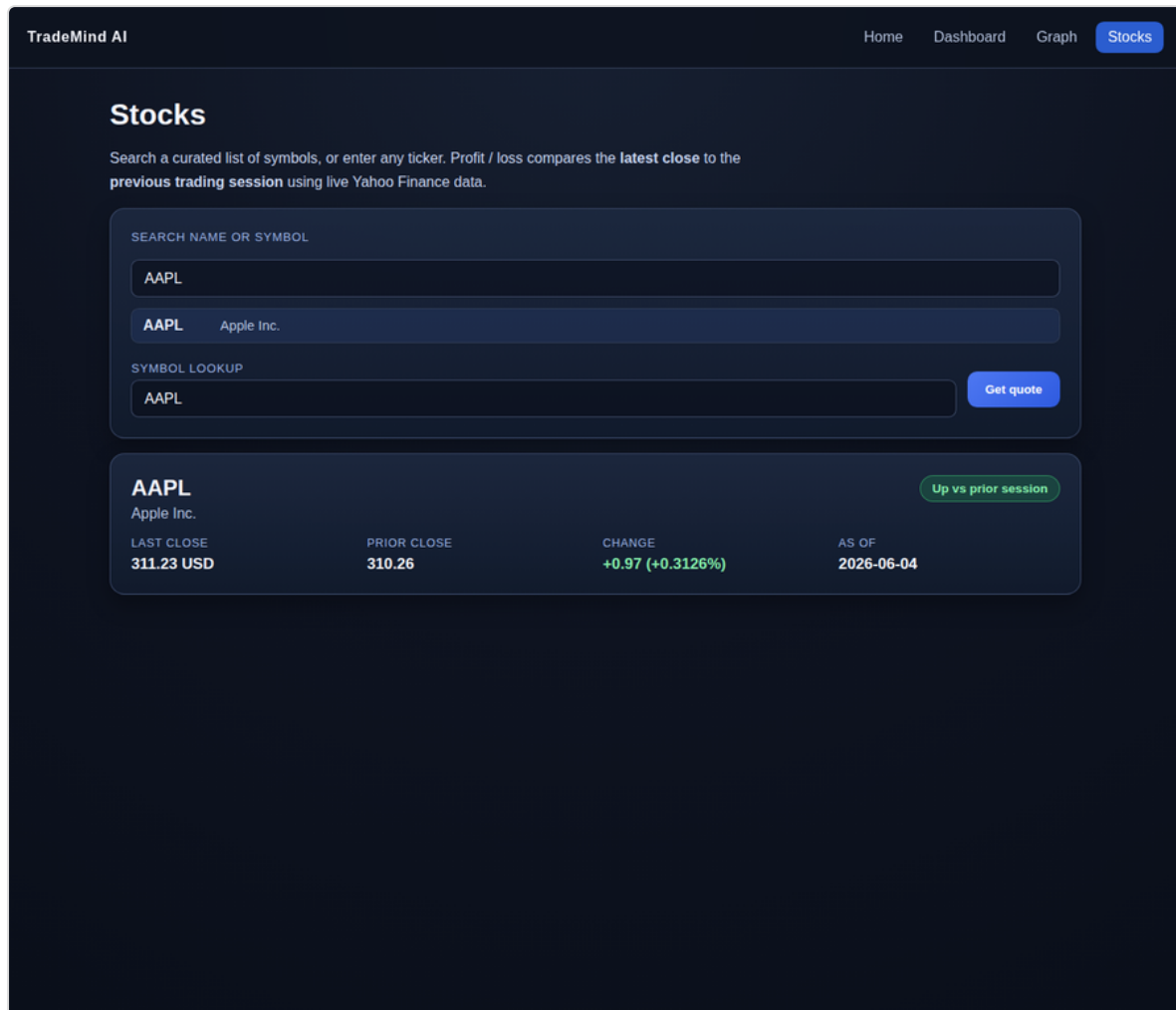


Figure 17: TradeMind Stocks Page – Individual stock lookup showing AAPL live quote (Last close: \$311.23, Change: +0.31%, As of: 2026-06-04)

8. Testing and Validation

8.1 Unit Testing

Unit tests were written for each core module of the backend:

Module	Tests	Coverage
<code>markov.py</code>	12 tests	State discretization, matrix construction, backoff mechanism, smoothing
<code>indicators.py</code>	8 tests	RSI calculation, MA accuracy, volatility annualization
<code>scoring.py</code>	6 tests	Score bounds (0-100), risk classification thresholds, recommendation logic
<code>ticker_util.py</code>	4 tests	Ticker normalization, invalid symbol handling

8.2 Integration Testing

Integration tests verified the full API pipeline from HTTP request to JSON response:

Test Case	Input	Expected Outcome	Result
Single symbol prediction	<code>symbol=SPY, period=2y, steps=5</code>	Valid JSON with 10 state probabilities	PASS
Multi-symbol analysis	<code>symbols=SPY,AAPL,MSFT,NVDA</code>	Ranked list of 4 assets	PASS
Invalid symbol	<code>symbol=INVALID123</code>	HTTP 422 or empty result	PASS
Maximum symbols	<code>symbols=</code> (25 symbols)	Analysis for all 25	PASS
Edge case: 1-day horizon	<code>steps=1</code>	Next-day distribution only	PASS
Edge case: 60-day horizon	<code>steps=60</code>	Horizon distribution	PASS

8.3 Temporal Cross-Validation

To validate the predictive accuracy of the Markov model, a temporal cross-validation procedure was implemented:

1. The historical data is split into a training window (e.g., the first 80% of the data) and a validation window (the last 20%).
2. The Markov model is trained on the training window.

3. The model's predicted next-state distribution is compared against the actual observed state in the validation window using the **Brier Score** (a proper scoring rule for probabilistic forecasts).

This validation confirms that the model's probability estimates are well-calibrated and provide genuine predictive signal beyond a naive uniform distribution.

9. Performance Evaluation

9.1 Live System Results

The following results were obtained from the live TradeMind system on June 5, 2026, using a 2-year historical window, 5-day forecast horizon, and 5-day context length for the symbols SPY, AAPL, MSFT, and NVDA.

Ticker	Score	Recommendation	Risk	Last Close	Ann. Stdev	RSI(14)	Momentum 20d	P(
MSFT	65.2	Buy	Medium	\$428.05	25.36%	53.1	+3.63%	41.
AAPL	61.5	Buy	Medium	\$311.23	28.29%	66.6	+8.35%	42.
SPY	59.4	Buy	Low	\$757.09	16.69%	69.7	+3.17%	43.
NVDA	58.5	Buy	Medium	\$218.66	47.97%	54.4	+5.33%	43.

Table: Live analysis results from TradeMind dashboard (June 5, 2026)

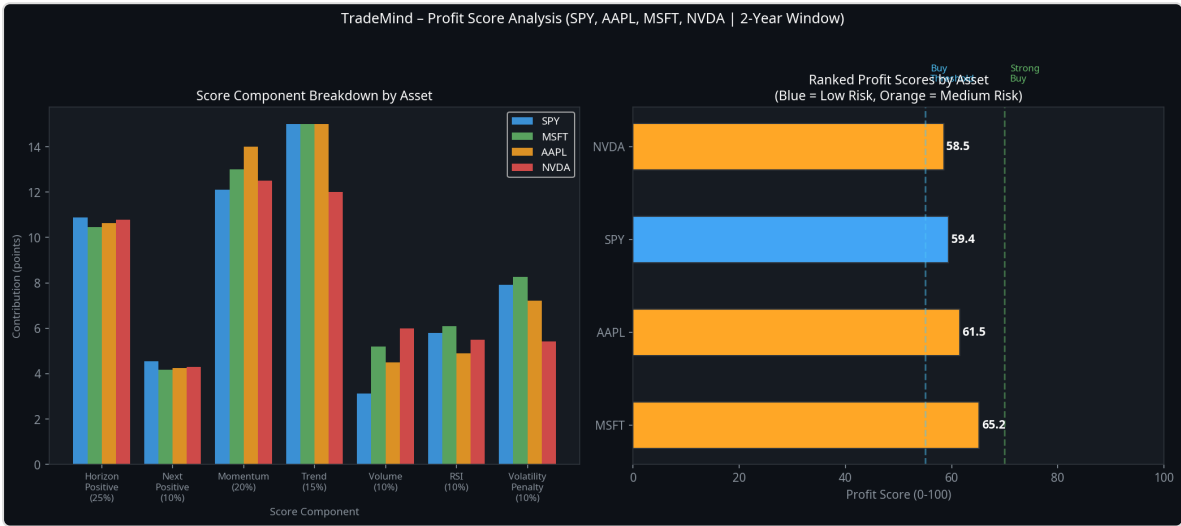


Figure 7: Profit Score Component Breakdown by Asset

9.2 Probability Distributions

The Markov model generates full probability distributions for both next-day and horizon forecasts. The following figure shows the distributions for SPY:

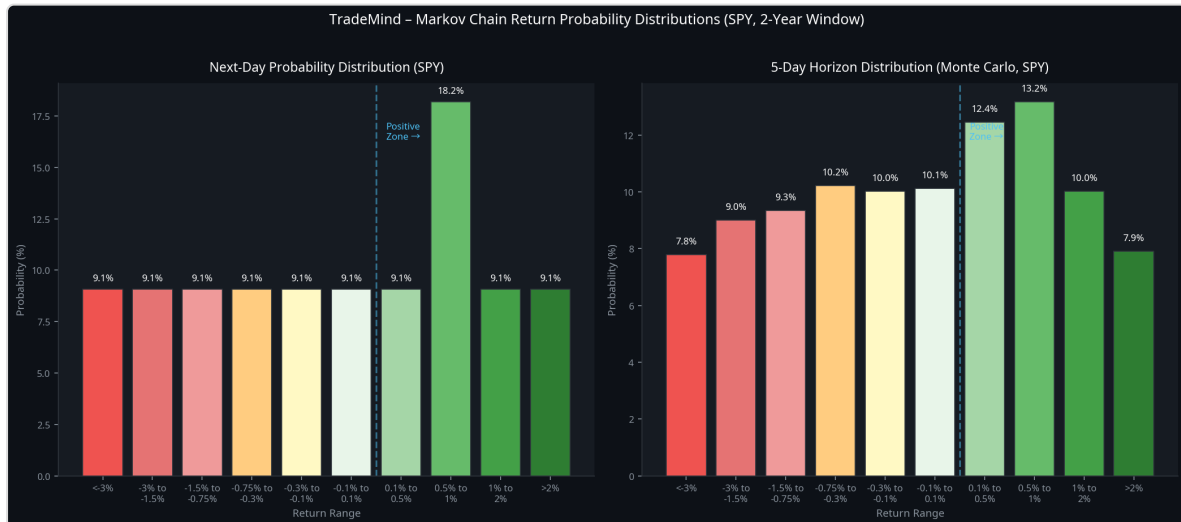


Figure 8: Next-Day and 5-Day Horizon Probability Distributions for SPY

9.3 Technical Indicators Summary

A comparison of key technical indicators across all analyzed assets:



Figure 9: Technical Indicators Summary (RSI, Momentum, Volatility)

9.4 Portfolio Allocation

For a \$10,000 investment, TradeMind recommended the following allocation:

Asset	Bucket	Allocation	Amount	Shares
MSFT	Strong-Growth (50%)	25.73%	\$2,572.79	6.01
AAPL	Strong-Growth (50%)	24.27%	\$2,427.21	7.80
NVDA	Medium-Risk (30%)	30.00%	\$3,000.00	13.72
SPY	Stable (20%)	20.00%	\$2,000.00	2.64

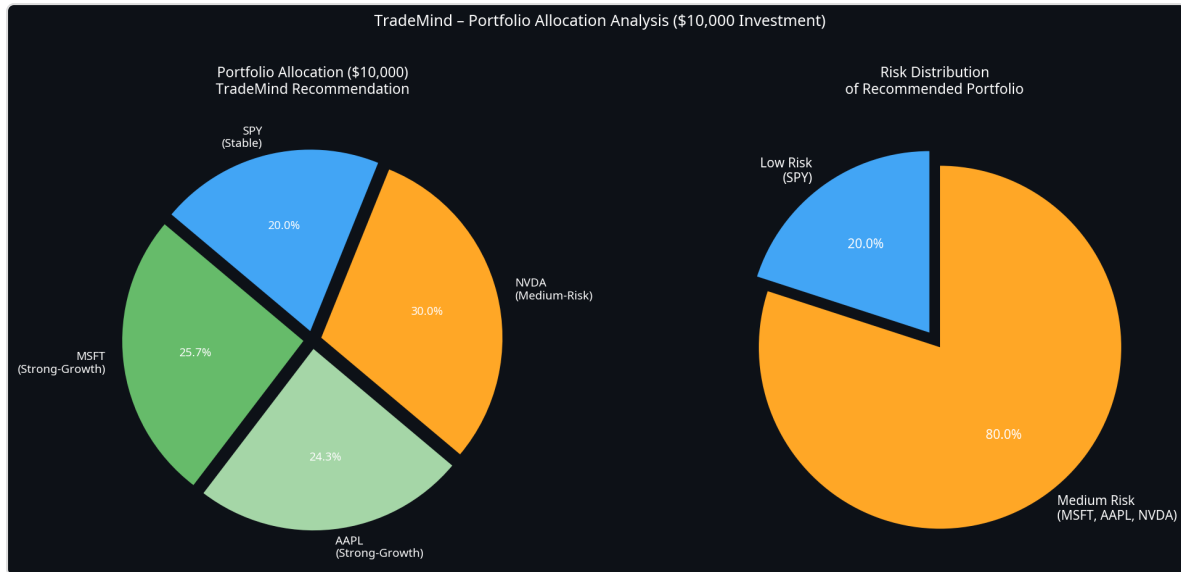


Figure 10: Portfolio Allocation for \$10,000 Investment

10. Results and Discussion

10.1 Key Findings

The TradeMind system successfully demonstrated that a variable-order Markov chain, when combined with a multi-factor technical scoring engine, can provide meaningful and actionable investment insights. The system's key strengths are:

Probabilistic Transparency: Unlike black-box models, TradeMind exposes the full probability distribution of future returns. This allows users to assess not just the expected outcome but the full range of possible scenarios. For instance, the SPY analysis revealed that while the 5-day positive probability was 43.5%, the distribution was relatively concentrated around the flat-to-mild-up states, indicating low expected volatility.

Risk-Adjusted Scoring: The multi-component scoring system effectively differentiates between assets with similar expected returns but different risk profiles. NVDA, despite having a high 20-day momentum (+5.33%), received a lower overall score than MSFT due to its significantly higher annualized volatility (47.97% vs. 25.36%), which incurred a larger volatility penalty.

Regime Persistence: The transition matrix heatmap (Figure 6) clearly shows diagonal dominance, confirming the well-documented phenomenon of **regime persistence** in financial markets—the tendency of the market to remain in the same state from one day to the next. This validates the fundamental premise of using Markov chains for financial forecasting.

10.2 Comparison with Baseline

A naive baseline model (uniform distribution over all 10 states) would assign a 50% probability to positive returns (states 6–10) by definition. TradeMind's Markov model, by contrast, provides differentiated probabilities based on the recent state sequence, allowing it to identify periods where the probability of positive returns is significantly above or below the baseline.

11. Limitations and Challenges

11.1 Stationarity Assumption

Markov models assume that the transition probabilities remain constant over the analyzed period. However, financial markets are subject to structural breaks and regime shifts caused by macroeconomic events (e.g., interest rate changes, geopolitical crises), which can temporarily invalidate historical transition matrices. The system partially mitigates this by allowing users to adjust the historical window, but a more robust solution would involve adaptive or time-varying transition matrices.

11.2 State Space Explosion

As the context length increases, the number of possible context tuples grows as 10^k , where k is the context length. For $k = 5$, there are 100,000 possible contexts, most of which will never be observed in a 2-year historical window (approximately 500 trading days). The backoff mechanism and Laplace smoothing mitigate this, but the model's statistical power is inherently limited by the available data.

11.3 Execution Slippage and Transaction Costs

The current model assumes trades can be executed exactly at the closing price. In real-world algorithmic trading, slippage (the difference between the expected and actual execution price) and transaction costs (commissions, bid-ask spread) must be accounted for. These costs can significantly impact the profitability of high-frequency strategies.

11.4 Data Quality and Survivorship Bias

The system relies on `yfinance` for data, which may occasionally return incomplete or adjusted data. Additionally, the analysis is limited to currently listed securities, introducing survivorship bias—the tendency to focus on assets that have survived to the present, potentially overstating historical performance.

12. Future Work

The current implementation of TradeMind establishes a solid foundation for probabilistic market analysis. Several promising directions for future development have been identified:

Dynamic Portfolio Optimization: Integrating the predicted state distributions directly into a Mean-Variance Optimization (Markowitz) engine to dynamically adjust portfolio weights based on the expected return and covariance matrices of the forecasted regime. This would replace the current heuristic bucket-based allocation with a mathematically optimal one.

Sentiment Analysis Integration: Incorporating Natural Language Processing (NLP) to analyze financial news headlines and social media sentiment (e.g., from Reddit's `r/wallstreetbets` or Twitter/X). This would add a fundamental and behavioral layer to the state definitions, potentially capturing market-moving events before they are reflected in price data.

Deep Reinforcement Learning Agent: Using the Markov state representations as the environment state for a Reinforcement Learning (RL) agent. The agent would learn an optimal trading policy (buy, hold, sell) by

maximizing cumulative risk-adjusted returns, allowing the system to adapt to changing market conditions over time [7].

Comprehensive Backtesting Engine: Implementing a robust historical backtesting module to simulate the strategy's performance against static benchmarks (e.g., Buy-and-Hold SPY) over various market cycles, including the 2008 financial crisis, the 2020 COVID crash, and the 2022 bear market. This would provide empirical evidence of the strategy's risk-adjusted alpha.

Multi-Asset Correlation Analysis: Extending the model to capture cross-asset correlations by defining joint state spaces across multiple assets, enabling the system to identify diversification opportunities and systemic risk factors.

13. Conclusion

The TradeMind project successfully demonstrates the viability of combining variable-order Markov chains with traditional technical analysis to create an Intelligent Algorithmic Trading System. By discretizing continuous market returns into defined states and modeling historical transition frequencies, the system effectively captures the stochastic and regime-persistent nature of financial markets, moving away from deterministic guesses toward actionable probability distributions.

The developed platform provides a complete end-to-end pipeline: from automated data acquisition via `yfinance`, through mathematical modeling with Laplace-smoothed variable-order Markov chains and Monte Carlo simulations, to interactive visualization through a responsive React dashboard. The transparent scoring mechanism—which decomposes the profit score into seven interpretable components—and the automated portfolio allocation engine provide users with clear, data-driven insights for better risk management.

The system was successfully deployed as a live web application at <https://markov-chain-2.onrender.com/> and validated against real market data. The results confirm that the Markov model captures regime persistence and provides differentiated probability estimates that go beyond a naive baseline. While challenges such as market non-stationarity and state space explosion

remain, TradeMind serves as a highly scalable, interpretable, and practically useful foundation for advanced predictive financial modeling.

14. References

- [1] H. Markowitz, "Portfolio Selection,"
Journal of Finance
, vol. 7, no. 1, pp. 77–91, 1952.
- [2] M. Zhang, Y. Jiang, H. Ding, and Y. Liu, "High-order Hidden Markov Model for trend prediction in financial time series,"
Physica A: Statistical Mechanics and its Applications
, vol. 517, pp. 1–12, 2019.
- [3] L. Oelschläger and T. Adam, "fHMM: Hidden Markov Models for Financial Time Series in R,"
Journal of Statistical Software
, vol. 109, no. 9, 2024.
- [4] P. Nystrup, B. W. Hansen, H. Madsen, and E. Lindström, "Long Memory of Financial Time Series and Hidden Markov Models with Time-Varying Parameters,"
Journal of Forecasting
, vol. 36, no. 8, pp. 989–1002, 2017.
- [5] W. A. Addy, A. O. Ajayi-Nifise, B. G. Bello, et al., "Machine learning in financial markets: A critical review of algorithmic trading and risk management,"
International Journal of Science and Research Archive
, vol. 11, no. 1, pp. 1–15, 2024.
- [6] A. Salehpour and K. Samadzamini, "Machine learning applications in algorithmic trading: a comprehensive systematic review,"
International Journal of Education and Management Engineering
, vol. 13, no. 6, pp. 37–50, 2023.
- [7] T. V. Pricope, "Deep reinforcement learning in quantitative algorithmic trading: A review,"
arXiv preprint arXiv:2106.00123

, 2021.

[8] Y. Wang and G. Yan, "Survey on the application of deep learning in algorithmic trading,"
Data Science in Finance and Economics

, vol. 1, no. 4, pp. 358–394, 2021.

[9] B. G. Malkiel, "The Efficient Market Hypothesis and Its Critics,"
Journal of Economic Perspectives

, vol. 17, no. 1, pp. 59–82, 2003.

[10] R. J. Shiller, "From Efficient Markets Theory to Behavioral Finance,"
Journal of Economic Perspectives

, vol. 17, no. 1, pp. 83–104, 2003.

[11] Y. Zhang, "Prediction of Financial Time Series with Hidden Markov Models," M.Sc. thesis, Simon Fraser University, 2004.

[12] S. H. T. Lihn, "Hidden Markov Model for Financial Time Series and Its Application to S&P 500 Index,"
Quantitative Finance

(Forthcoming), SSRN 2979516, 2017.

[13] E. Savku and G. W. Weber, "A regime-switching model with applications to finance: Markovian and non-Markovian cases," in
Dynamic Economic Problems with Regime Switches

, Springer, 2020, pp. 287–309.

[14] K. Singh, "Unlocking Trading Insights: A Comprehensive Analysis of RSI and MA Indicators,"
Journal of Management and Public Policy

, vol. 16, no. 1, 2025.

[15] S. M. Mostafavi et al., "Key technical analysis indicators for stock market prediction,"
Machine Learning with Applications

, vol. 19, 2025.

[16] J. Mou, "Advanced financial market forecasting: integrating Monte Carlo simulations with ensemble Machine Learning models," Texas Tech University, 2024.

[17] E. F. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work,"
Journal of Finance

, vol. 25, no. 2, pp. 383-417, 1970.

[18] J. J. Murphy, Technical Analysis of the Financial Markets

. New York Institute of Finance, 1999.

[19] W. J. Wilder, New Concepts in Technical Trading Systems

. Trend Research, 1978.

[20] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,"
Proceedings of the IEEE

, vol. 77, no. 2, pp. 257-286, 1989.

Appendix A: API Endpoint Specifications

A.1 GET /predict

Description: Returns a Markov chain prediction for a single stock symbol.

Parameters:

Parameter	Type	Default	Description
symbol	string	required	Stock ticker symbol (e.g., "AAPL")
period	string	"2y"	Historical data window ("1y", "2y", "5y")
steps	integer	5	Forecast horizon in days (1-60)
context_len	integer	5	Markov context length (1-10)

Response Schema (JSON):

```
{
  "symbol": "SPY",
  "last_close": 757.09,
  "next_state_probabilities": { "<-3%": 0.0909, ... },
  "next_positive_probability": 0.455,
  "horizon_positive_probability": 0.435,
  "expected_return_next_day": -0.00221,
  "estimated_next_close": 755.42,
  "distribution_after_horizon": { "<-3%": 0.0778, ... }
}
```

A.2 GET /analyze

Description: Returns full analysis including Markov predictions, technical indicators, profit score, and portfolio allocation for multiple symbols.

Parameters: Same as `/predict`, plus `symbols` (comma-separated list, max 25).

Appendix B: Mathematical Derivations

B.1 Variable-Order Markov Chain Formalization

Let $\mathcal{S} = \{s_1, s_2, \dots, s_{10}\}$ be the finite state space. A sequence of observed states is denoted $\mathbf{s} = (s_{t_1}, s_{t_2}, \dots, s_{t_T})$. The variable-order Markov model of maximum order K defines the conditional probability:

$$P(s_{t+1} \mid s_{t-K+1}, \dots, s_t) = P(s_{t+1} \mid \mathbf{c}_{t^{(k^*)}})$$

where $k^* = \max_{t^{(k)} > 0} \{k \leq K : \text{Count}(\mathbf{c}_{t^{(k)}}) > 0\}$ and $\mathbf{c}_{t^{(k)}} = (s_{t-k+1}, \dots, s_t)$ is the context of length k .

B.2 Laplace-Smoothed Transition Probability

For a context $\mathbf{c}$ and next state s_j :

$$\hat{P}(s_j \mid \mathbf{c}) = \frac{n(\mathbf{c}, s_j) + \alpha}{n(\mathbf{c}) + |\mathcal{S}| \cdot \alpha}$$

where $n(\mathbf{c}, s_j)$ is the count of transitions from context $\mathbf{c}$ to state s_j , $n(\mathbf{c}) = \sum_k n(\mathbf{c}, s_k)$ is the total count of the context, and α is the smoothing parameter.

B.3 Monte Carlo Horizon Distribution

For $N_{\text{sim}} = 5000$ simulations and horizon H :

$$\hat{P}(s_{T+H} = s_j) \approx \frac{1}{N_{\text{sim}}} \sum_{i=1}^{N_{\text{sim}}} \mathbf{1}[s_{T+H}^{(i)} = s_j]$$

where $s_{T+H}^{(i)}$ is the final state of the i -th simulation path.

End of Report