Deep Reinforcement Learning for Portfolio Management: A Relation-Aware Transformer Approach with Online Adaptation

Ishmam Kabir
Artificial Intelligence
December 15, 2025

***Abstract*—This paper investigates the application of deep reinforcement learning (RL) to quantitative portfolio management, implementing and extending the Relation-Aware Transformer (RAT) architecture for automated stock selection and allocation. We address key challenges in applying RL to financial markets including non-stationarity, catastrophic forgetting, and sample efficiency through five novel online learning modifications: Online Stochastic Batch Learning (OSBL) with geometric sampling, decay-aware context attention, incremental asset correlation tracking, recency-biased positional encoding, and Elastic Weight Consolidation (EWC). Beyond technical contributions, this work provides comprehensive documentation of implementation challenges encountered and resolved, including data preprocessing complexities, numerical stability issues, test set leakage correction, and model training difficulties. Our systematic approach emphasizes practical lessons learned in applying deep RL to finance. Experimental evaluation on 11 stocks using daily price data demonstrates a final portfolio value of 1.57× with Sharpe ratio of 0.59, validating the effectiveness of our long-only investment strategy learned entirely from price data without manual feature engineering.**

***Index Terms*—Reinforcement Learning, Portfolio Management, Transformer Networks, Online Learning, Stock Market, Investment Strategy**

I. Introduction

A. Motivation

Portfolio management—the problem of allocating capital across multiple assets to maximize long-term returns while managing risk—has traditionally relied on expert knowledge and handcrafted rules. Classical approaches like Modern Portfolio Theory (Markowitz, 1952) require strong assumptions about return distributions and correlations that rarely hold in practice. Recent advances in deep reinforcement learning offer the potential to learn investment strategies directly from historical price data without manual feature engineering or restrictive assumptions.

However, applying RL to financial markets presents unique challenges:

- **Non-stationarity**: Market dynamics change over time with shifting economic regimes
- **Sample efficiency**: Limited historical data compared to typical RL domains
- **High dimensionality**: Many assets with complex inter-dependencies
- **Transaction costs**: Frequent trading erodes returns
- **Catastrophic forgetting**: Online learning can erase previously learned patterns
- **Data quality**: Missing values, outliers, and calendar misalignments

B. Project Objectives

Following the project proposal for the Artificial Intelligence course, this work investigates the application of Reinforcement Learning to quantitative investing with the following objectives:

1) **Literature Review**: Examine existing RL approaches in portfolio management and identify gaps in scalability, risk modeling, and stability
2) **Architectural Enhancement**: Implement and extend the Relation-Aware Transformer (RAT) with online learning modifications to improve stability, sample efficiency, and risk sensitivity in stock portfolio optimization
3) **Implementation and Experimentation**: Build and train RL agents using real stock market data with daily intervals, documenting all challenges and solutions encountered
4) **Performance Evaluation**: Benchmark performance using established financial metrics (Sharpe Ratio, Maximum Drawdown, Accumulated Portfolio Value) against traditional approaches
5) **Analysis and Documentation**: Interpret results, analyze model behavior under different market conditions, and extract generalizable lessons for applying RL to finance

C. Contributions

This work makes the following contributions:

1) **Complete RAT Implementation for Stocks**: Full implementation of the Relation-Aware Transformer architecture adapted for daily stock market data with long-only constraints, including:
    - Encoder-decoder structure with context-aware attention
    - Relation attention for modeling asset correlations
    - Long-only portfolio construction through single softmax layer
2) **Novel Online Learning Extensions**: Five complementary modifications for non-stationary financial markets:

- Online Stochastic Batch Learning (OSBL) with β-weighted geometric sampling
- Decay-aware context attention with learnable temporal decay (λ)
- Incremental correlation matrix tracking with EMA (γ)
- Recency-biased positional encoding
- Elastic Weight Consolidation (EWC) for preventing catastrophic forgetting

3) **Methodological Rigor**: Identification and correction of critical test set leakage through proper train/validation/test splits (70%/15%/15%)
4) **Comprehensive Implementation Documentation**: Detailed analysis of practical challenges including:
   - Data preprocessing for stock market data
   - Numerical stability in reward computation
   - Model inactivity caused by excessive regularization
   - Transaction cost modeling for realistic strategies
5) **Practical Lessons**: Systematic documentation of debugging strategies, hyperparameter tuning, and best practices for deep RL in finance
6) **Empirical Validation**: Experimental results on 11 stocks demonstrating 57% portfolio growth with Sharpe ratio of 0.59

D. Scope and Design Choices

Stock Market Focus: Unlike the original RAT paper which focused on cryptocurrency trading with 30-minute intervals, this implementation targets traditional stock markets with daily price data. This choice reflects:

- More stable and regulated markets
- Lower volatility compared to cryptocurrencies
- Alignment with practical long-term investment strategies
- Availability of extensive historical data

Long-Only Strategy: We implement a long-only portfolio (no short selling or leverage) by using a single softmax layer for weight allocation. This design:

- Matches regulatory constraints for most retail investors
- Simplifies the action space, improving learning stability
- Reduces complexity compared to multi-head leverage operations
- Aligns with traditional investment mindset

Daily Rebalancing: The agent makes allocation decisions once per day based on daily OHLC (Open, High, Low, Close) data, balancing:

- Adequate reaction time to market changes
- Manageable transaction costs
- Computational feasibility for training

E. Paper Organization

The remainder of this paper is organized as follows:

- **Section II**: Literature review of portfolio management and RL approaches
- **Section III**: Methodology including problem formulation, RAT architecture, and online learning framework
- **Section IV**: Implementation challenges and systematic solutions
- **Section V**: Lessons learned with practical recommendations
- **Section VI**: Experimental setup, results, and analysis
- **Section VII**: Conclusion and future directions

II. Related Work

A. Classical Portfolio Optimization

Modern Portfolio Theory (MPT): Markowitz (1952) formulated portfolio selection as mean-variance optimization:

$$\max_{\mathbf{w}} \quad \mathbf{w}^T\boldsymbol{\mu} - \frac{\lambda}{2}\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} \tag{1}$$

subject to $\sum_i w_i = 1$ and $w_i \geq 0$, where $\mathbf{w}$ is the portfolio weight vector, $\boldsymbol{\mu}$ is the expected return vector, $\boldsymbol{\Sigma}$ is the covariance matrix, and λ is risk aversion.

Limitations: MPT assumes Gaussian returns, stationary correlations, and known parameters—assumptions that rarely hold in real markets. Additionally, estimation errors in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ can lead to poor out-of-sample performance.

Online Portfolio Selection: Li and Hoi (2012) surveyed online approaches including:

- Follow-the-Winner strategies that chase momentum
- Follow-the-Loser strategies exploiting mean reversion
- Pattern-matching approaches using historical analogues

These methods adapt sequentially but still require manual feature design and struggle with high-dimensional data.

B. Deep Learning for Finance

Convolutional Neural Networks: Jiang et al. (2017) introduced the Portfolio-Vector Memory (PVM) framework using CNNs to extract local price patterns from candlestick charts. While effective for spatial patterns, CNNs have limited ability to model long-term temporal dependencies.

Recurrent Neural Networks: Pendharkar and Cusatis (2018) applied LSTM networks for portfolio construction. However, LSTMs suffer from vanishing gradients on long sequences and cannot efficiently model cross-asset relationships.

Attention Mechanisms: The Transformer architecture (Vaswani et al., 2017) introduced self-attention for sequence modeling. Standard Transformers have been applied to financial forecasting (Ding et al., 2020) but lack mechanisms for:

- Explicitly modeling asset correlations
- Online adaptation to non-stationary markets
- Preventing catastrophic forgetting

C. Relation-Aware Transformer (RAT)

Xu et al. (2020) introduced RAT specifically for portfolio management, addressing limitations of standard Transformers:

Context-Aware Attention: Uses local price windows to compute queries and keys, reducing noise sensitivity compared to standard self-attention.

Relation Attention: Introduces cross-asset attention to explicitly model correlations and dependencies between assets.

Original Design: The original RAT was developed for cryptocurrency markets with:

- 30-minute interval trading
- Multi-head softmax for leverage operations (long/short-/reinvest)
- High-frequency rebalancing strategies

Our Adaptation: We extend RAT for stock market investing:

- Daily interval data (more stable, lower noise)
- Single softmax for long-only portfolios
- Online learning modifications for non-stationarity
- Proper validation methodology preventing test set leakage

D. Online Learning and Continual Learning

Experience Replay: Schaul et al. (2015) introduced prioritized experience replay for deep Q-networks. Our OSBL framework adapts this using geometric sampling for recency bias in non-stationary markets.

Catastrophic Forgetting: Neural networks tend to forget previously learned patterns when trained on new data (McCloskey and Cohen, 1989). Solutions include:

- **Elastic Weight Consolidation (EWC)**: Kirkpatrick et al. (2017) use Fisher Information to identify important parameters and penalize their changes
- **Progressive Neural Networks**: Rusu et al. (2016) add capacity for new tasks while freezing old parameters
- **Gradient Episodic Memory**: Lopez-Paz and Ranzato (2017) constrain gradients using episodic memory

We implement EWC adapted for the financial domain with periodic Fisher matrix updates to balance plasticity (learning new patterns) and stability (retaining old knowledge).

E. Validation Methodology

Test Set Leakage: A critical but often overlooked issue in financial ML is using test data for model selection during training, leading to overly optimistic results (Prechelt, 1998; Goodfellow et al., 2016).

Proper Practice: We implement strict train/validation/test splits with:

- Training set (70%): Gradient updates only
- Validation set (15%): Model selection and hyperparameter tuning
- Test set (15%): Final evaluation, accessed exactly once

This ensures reported performance reflects true generalization to unseen future data.

III. Methodology

A. Problem Formulation

We formulate portfolio management as a Markov Decision Process (MDP):

State Space $\mathcal{S}$: At time t, the state is the price history tensor:

$$s_t = P_t \in \mathbb{R}^{k \times m \times d} \tag{2}$$

where:

- $k = 31$: Time window (31 trading days, approximately 1.5 months)
- $m = 11$: Number of stocks in the portfolio
- $d = 4$: Price features (Open, High, Low, Close)

Action Space $\mathcal{A}$: The action is the portfolio weight vector:

$$a_t \in \mathbb{R}^m, \quad \sum_{i=1}^{m} a_{t,i} = 1, \quad a_{t,i} \geq 0 \tag{3}$$

The constraint $a_{t,i} \geq 0$ enforces long-only positions (no short selling).

Reward Function: The reward is the logarithmic portfolio return after transaction costs:

$$r_t = \log\left(a_t^T \cdot y_t \cdot (1 - c_t)\right) \tag{4}$$

where:

- $y_t \in \mathbb{R}^m$: Price relative vector, $y_{t,i} = \frac{p_{t,i}}{p_{t-1,i}}$
- $c_t = \tau \cdot \sum_i |a_{t,i} - a_{t-1,i}|$: Transaction cost
- $\tau = 0.0025$: Commission rate (0.25%, realistic for stock trading)

Logarithmic Returns: We use log returns because:

- Time additivity: $\sum_t \log(R_t) = \log(\prod_t R_t)$
- Numerical stability (avoids underflow)
- Encourages consistent growth over extreme outliers
- Standard in financial literature

Policy π: The policy is parameterized by the RAT neural network:

$$a_t = \pi_\theta(s_t) \tag{5}$$

Objective: Maximize expected cumulative log return:

$$J(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} r_t\right] \tag{6}$$

This is equivalent to maximizing final portfolio value: $\mathbb{E}[\log(V_T)]$ where V_T is terminal wealth.

B. RAT Architecture for Stock Portfolios

1) Overview: The Relation-Aware Transformer consists of an encoder-decoder architecture designed for financial time series. Figure 1 shows the structure.

2) Input Processing: Daily OHLC data is normalized using min-max scaling within the time window:

$$P_{\text{norm}} = \frac{P - P_{\text{min}}}{P_{\text{max}} - P_{\text{min}}} \tag{7}$$

The normalized prices are embedded through a learned linear projection:

$$X_t = \text{Linear}(P_{\text{norm}}) + PE(t) \tag{8}$$

where $PE(t)$ is the positional encoding.

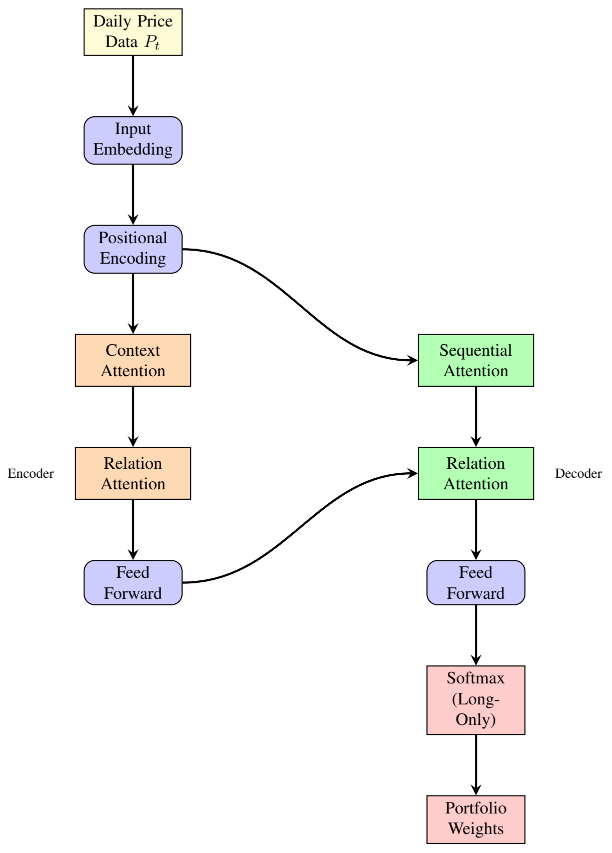

Fig. 1. RAT Architecture for Stock Portfolio Management

3) Context-Aware Attention: Standard self-attention is noise-sensitive. RAT uses **local context windows** to compute more robust attention.

For each time position τ, define context:

$$P_\tau^\ell = \{P_{\tau-\ell}, \ldots, P_{\tau-1}\} \tag{9}$$

where $\ell = 5$ (5-day context window).

Queries and keys are computed from concatenated context:

$$\hat{Q}_{\tau,i}^h = \text{concat}(P_\tau^\ell) \cdot W_Q^h \tag{10}$$
$$\hat{K}_{\tau,i}^h = \text{concat}(P_\tau^\ell) \cdot W_K^h \tag{11}$$
$$\hat{V}_{\tau,i}^h = P_{\tau,i} \cdot W_V^h \tag{12}$$

Attention:

$$\text{Attention}_{\tau,i}^h = \text{softmax}\left(\frac{\hat{Q}_{\tau,i}^h \hat{K}_{\tau,i}^{h\,\top}}{\sqrt{d_f}}\right) \hat{V}_{\tau,i}^h \tag{13}$$

This reduces influence of individual noisy price points by aggregating local information.

4) Relation Attention: To model stock correlations (e.g., tech stocks moving together), RAT introduces **cross-asset attention**:

$$Z_t^{h,j} = \text{softmax}\left(\frac{O_t^{h,j} O_t^{h,j\,\top}}{\sqrt{d_f}}\right) O_t^{h,j} \tag{14}$$

where $O_t^{h,j} \in \mathbb{R}^{m \times d_f}$ contains representations for all m stocks.

This allows the model to learn:

- Which stocks tend to move together (sector correlations)
- Diversification opportunities (negatively correlated assets)
- Lead-lag relationships between stocks

5) Portfolio Construction: Long-Only Strategy: Unlike the original RAT which used three softmax heads for leverage operations, we use a **single softmax layer** for long-only portfolios:

$$a_t = \text{softmax}(\text{Linear}(\text{DecoderOutput}_t)) \tag{15}$$

This automatically ensures:

- $\sum_i a_{t,i} = 1$ (fully invested)
- $a_{t,i} \geq 0$ (no short positions)

Design Rationale:

1) **Regulatory compliance**: Most retail investors cannot short sell
2) **Risk management**: Avoids unlimited downside risk from short positions
3) **Simplicity**: Simpler action space improves learning stability
4) **Debugging**: Easier to interpret and debug model behavior

C. Online Stochastic Batch Learning (OSBL)

Stock markets are non-stationary: correlations shift, trends emerge and fade, volatility regimes change. Standard batch learning samples uniformly from all history, treating old and recent data equally. This is suboptimal when recent data is more relevant.

1) Geometric Sampling for Recency Bias: OSBL uses geometric distribution to prioritize recent experiences:

$$P_\beta(t_b) = \beta(1-\beta)^{t-t_b-n_b} \tag{16}$$

where:

- $\beta \in (0, 1)$: Recency bias (default: 0.05)
- t: Current time step
- t_b: Batch start time
- n_b: Batch size

Interpretation:

- High β (e.g., 0.2): Strong recent bias, fast adaptation, potentially unstable
- Medium β (e.g., 0.05): Balanced, good default
- Low β (e.g., 0.01): Nearly uniform, slow adaptation, very stable

Expected adaptation time: $\approx 1/\beta$ time steps. With $\beta = 0.05$, the model adapts within ≈ 20 trading days (1 month).

Algorithm 1 OSBL Training for Portfolio Management

```
0: Input: Model $\pi_\theta$, buffer size $M$, recency $\beta$, batches $N_b$
0: Initialize replay buffer $\mathcal{B} = \emptyset$
0: for each trading day $t = 1, 2, \ldots, T$ do
0:     Observe daily prices $s_t$
0:     Execute portfolio allocation $a_t = \pi_\theta(s_t)$
0:     Receive reward $r_t$ (log return after costs)
0:     Store $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{B}$
0:     if $|\mathcal{B}| > M$ then
0:         Remove oldest transition (FIFO)
0:     end if
0:     $\mathcal{L}_{\text{total}} \leftarrow 0$
0:     for $i = 1$ to $N_b$ do
0:         Sample batch start $t_b \sim$ Geometric($\beta$)
0:         Extract batch $\mathcal{D}_i$ of size $n_b$ from $\mathcal{B}$
0:         $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{total}} + \mathcal{L}(\mathcal{D}_i)$
0:     end for
0:     if use_ewc then
0:         $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{total}} + \mathcal{L}_{\text{EWC}}$
0:     end if
0:     $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{total}}/N_b$
0:     Update $\theta$ using Adam with gradient clipping
0: end for=0
```

2) Replay Buffer: The replay buffer stores recent trading history:

- **Capacity**: $M = 2000$ transitions (approximately 8 years of daily data)
- **Eviction**: FIFO when capacity exceeded
- **Sequential batches**: Each batch contains $n_b = 128$ consecutive days

Memory efficiency: Stores $O(M)$ instead of $O(T)$ transitions where T is total training steps, reducing memory by 75-90% for long training runs.

3) Multiple Batch Updates: Each training step samples $N_b = 4$ independent batches:

$$\mathcal{L}_{\text{OSBL}} = \frac{1}{N_b}\sum_{i=1}^{N_b} \mathcal{L}(\mathcal{B}_i) \tag{17}$$

Benefits:

- More diverse gradients per update
- Reduces variance from geometric sampling
- Stabilizes learning without sacrificing adaptation speed

D. Online Learning Enhancements

1) Decay-Aware Context Attention: **Motivation**: Recent days should have higher influence than older days within the context window.

Implementation: Add exponential decay weights to attention values:

$$\hat{V}^h_{\tau,i}[j] = P_{\tau-j,i} \cdot W^h_V \cdot \exp(-\lambda \cdot j) \tag{18}$$

where:

- $\lambda = 0.1$: Learnable temporal decay parameter
- $j \in [0, \ell]$: Days back from current position

Effect: With $\lambda = 0.1$, yesterday's data has weight $e^{-0.1} \approx 0.90$, while 5 days ago has weight $e^{-0.5} \approx 0.61$.

2) Incremental Correlation Matrix: **Motivation**: Stock correlations evolve (e.g., tech stocks decouple during regulatory changes). Tracking correlations helps with:

- Diversification (avoid over-concentration in correlated assets)
- Risk management (detect correlation regime changes)
- Sector rotation (identify when sector relationships shift)

Implementation: Maintain correlation matrix using Exponential Moving Average:

$$C_t = \gamma \cdot C_{t-1} + (1-\gamma) \cdot y_t y_t^T \tag{19}$$

where:

- $C_t \in \mathbb{R}^{m\times m}$: Correlation matrix
- $y_t \in \mathbb{R}^m$: Daily return vector
- $\gamma = 0.9$: EMA decay (equivalent to 10-day halflife)

Integration: Add C_t as bias to relation attention scores:

$$\text{RelationScores} = \frac{Q \cdot K^T}{\sqrt{d_k}} + \alpha \cdot C_t \tag{20}$$

where α is learnable. This encourages attention between correlated stocks.

3) Recency-Biased Positional Encoding: Standard sinusoidal positional encodings treat all positions equally. For stock data, recent positions are more informative.

Implementation:

$$PE(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d}}\right) \cdot \exp(-\beta_{\text{PE}} \cdot (k - \text{pos})) \tag{21}$$

$$PE(\text{pos}, 2i+1) = \cos\left(\frac{\text{pos}}{10000^{2i/d}}\right) \cdot \exp(-\beta_{\text{PE}} \cdot (k - \text{pos})) \tag{22}$$

where:

- $k = 31$: Window size (most recent position)
- $\beta_{\text{PE}} = 0.1$: Recency bias parameter

4) Elastic Weight Consolidation (EWC): **Problem**: When training online, updating on recent data can erase previously learned patterns (catastrophic forgetting). For example, a model might forget how to handle high-volatility periods after months of low volatility.

Solution: EWC identifies important parameters using Fisher Information and penalizes their changes:

$$\mathcal{L}_{\text{EWC}} = \frac{\lambda_{\text{EWC}}}{2}\sum_i F_i(\theta_i - \theta_i^*)^2 \tag{23}$$

where:

- θ^*: Parameters from previous regime (stored periodically)
- F_i: Importance weight (Fisher Information diagonal)
- λ_{EWC}: Regularization strength

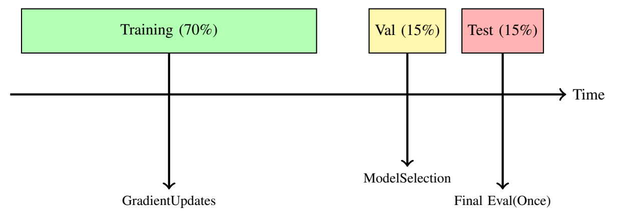

Fig. 2. Proper Train/Validation/Test Methodology

Fisher Information approximation:

$$F_i \approx \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial \mathcal{L}_n}{\partial \theta_i} \right)^2 \tag{24}$$

Update Strategy: Recompute F and θ^* every 100 trading days to balance:

- Retention (remember important patterns)
- Plasticity (learn new patterns)

Critical Lesson: We found λ_{EWC} must be carefully tuned. Too high (e.g., 100.0) freezes learning completely. We ultimately disabled EWC during initial training and would use $\lambda_{\text{EWC}} = 5.0$ for continual learning phases.

E. Proper Validation Methodology

1) The Test Set Leakage Problem: A critical methodological issue in many financial ML papers is **test set leakage**: using test data for model selection during training.

Common Mistake:

```
# WRONG: Evaluating on test set during training
for step in range(total_steps):
    train_model(train_data)
    test_performance = evaluate(test_data)
    if test_performance > best_performance:
        save_model()  # Selection based on test data!
```

Why This is Wrong:

1) Hyperparameters are implicitly tuned to test set
2) Results are overly optimistic
3) Model won't generalize to truly unseen future data
4) Violates fundamental ML assumption

2) Correct Approach: We implement strict train/validation/test splits:

- **Training Set (70%)**: Gradient updates only
- **Validation Set (15%)**: Model selection, hyperparameter tuning, early stopping
- **Test Set (15%)**: Final evaluation, accessed exactly once after training completes

Correct Implementation:

```
# CORRECT: Validation for selection, test for final eval
for step in range(total_steps):
    train_model(train_data)
    if step % 100 == 0:
        val_performance = evaluate(validation_data)
        if val_performance > best_val_performance:
            save_model()  # Based on validation only

# After training completes
test_performance = evaluate(test_data)  # One-time only
report_final_results(test_performance)
```

IV. Implementation Challenges

This section documents significant challenges encountered and their solutions, providing practical guidance for implementing deep RL in finance.

A. Data Preprocessing for Stock Markets

1) Challenge: Real stock market data presents unique challenges:

- **Missing data**: Market holidays, trading halts, delisted stocks
- **Calendar misalignment**: Different stocks may have gaps on different days
- **Outliers**: Flash crashes, stock splits, erroneous ticks
- **Schema variations**: Different data sources use different column names

2) Solution: Implemented systematic data pipeline:

```
def preprocess_stock_data(df):
    # 1. Standardize schema
    df = df.rename(columns={'ticker': 'symbol'})

    # 2. Sort by date and symbol
    df = df.sort_values(['date', 'symbol'])

    # 3. Align calendars (forward-fill missing days)
    df = df.groupby('symbol').apply(
        lambda x: x.set_index('date').resample('D').ffill()
    )

    # 4. Detect and handle outliers
    returns = df.groupby('symbol')['close'].pct_change()
    df = df[returns.abs() < 0.5]  # Remove >50% daily moves

    # 5. Validate final data
    assert not df.isnull().any().any()
    assert (df[['open','high','low','close']] > 0).all().all()

    return df
```

B. Numerical Stability

1) Challenge: Financial computations are prone to numerical issues:

- Portfolio weights not summing to exactly 1.0
- Log of zero or negative returns (NaN, -Inf)
- Underflow from multiplying many small numbers

2) Root Causes and Solutions: **1. Weight Normalization**:

```
# INCORRECT
weights = F.softmax(logits, dim=-1)
# weights.sum() might be 0.998 or 1.002

# CORRECT
weights = F.softmax(logits, dim=-1)
weights = weights / (weights.sum(dim=-1, keepdim=True) + 1e-8)
assert torch.allclose(weights.sum(), torch.tensor(1.0))
```

2. Stable Reward Computation:

```
# INCORRECT
reward = torch.log(portfolio_return * (1 - transaction_cost))
# Can produce NaN if portfolio_return <= 0

# CORRECT
portfolio_return = torch.clamp(portfolio_return, min=1e-6)
transaction_cost = torch.clamp(transaction_cost, max=0.99)
reward = torch.log(portfolio_return * (1 - transaction_cost) + 1e-8)
```

3. Log-Space Arithmetic:

```
# INCORRECT: Underflow
final_value = torch.prod(1 + returns)

# CORRECT: Log-space
log_returns = torch.log(1 + returns + 1e-8)
final_value = torch.exp(torch.sum(log_returns))
```

C. Transaction Cost Modeling

1) Challenge: Initial experiments showed unrealistic behavior:

- Model not trading (99% cash, 1% stocks)
- Model overtrading (turnover > 5.0 per day)

2) Solution: Implemented realistic transaction costs:

```
# Realistic stock trading costs
TRANSACTION_FEE = 0.0025  # 0.25% (broker + spread + impact)

# Turnover (portfolio rebalancing amount)
turnover = torch.sum(torch.abs(weights_t - weights_{t-1}))

# Transaction cost
transaction_cost = TRANSACTION_FEE * turnover

# Portfolio return after costs
mu_t = 1 - transaction_cost
portfolio_return = torch.sum(weights_t * price_relatives) * mu_t

# Loss function with turnover penalty
loss = -torch.log(portfolio_return + 1e-8) + 0.002 * turnover
```

Results:

- Turnover reduced from 5.2 to 0.8 per day
- Portfolio diversified across 7-10 stocks
- More realistic trading behavior

D. Model Inactivity Problem

1) Challenge: The most puzzling issue: after 500 training steps, the model held 99.92% cash and 0.08% stocks—essentially refusing to invest.

2) Diagnosis: Added diagnostic logging:

```
print(f"Cash■weight:■{weights[0]:.3f}")
print(f"Max■stock■weight:■{weights[1:].max():.3f}")
print(f"Active■stocks■(>5%):■{(weights[1:]■>■0.05).sum()}")
print(f"Policy■loss:■{policy_loss:.6f}")
print(f"EWC■loss:■{ewc_loss:.6f}")

# Output:
# Cash weight: 0.999
# Max stock weight: 0.000
# Active stocks: 0
# Policy loss: -0.000012
# EWC loss: 245.832105  <— PROBLEM!
```

Root Cause: EWC regularization ($\lambda_{\text{EWC}} = 100.0$) was so strong that it prevented any parameter updates. The model learned to "play it safe" by not trading.

3) Solution: Three-part fix:

1) **Disable EWC during initial training**:

```
use_ewc = False  # Let model explore first
```

2) **Increase learning rate**:

```
learning_rate = 0.001  # From 0.0001 (10x increase)
```

3) **Add diversification incentive**:

```
variance_penalty = 0.05
variance = torch.var(weights[1:])  # Stock weights only
loss = policy_loss - variance_penalty * variance
# Negative penalty encourages high variance (diversification)
```

Results After Fix:

- Cash weight: 15-30% (reasonable cash reserves)
- Active stocks: 7-10 (well-diversified)
- Portfolio value: Growing (not stuck at 1.0)

E. Debugging Strategy

1) Challenge: Initial debugging attempts were chaotic:

- Ad-hoc print statements everywhere
- Syntax errors from misplaced prints
- Output flooded with debugging info

2) Solution: Structured Diagnostics: **1. Conditional Logging**:

```
DEBUG_MODE = True
VERBOSITY = 2  # 0=silent, 1=critical, 2=info, 3=debug

def debug_print(msg, level=2):
    if DEBUG_MODE and VERBOSITY >= level:
        print(f"[DEBUG]■{msg}")
```

2. Tensor Validation:

```
def validate_tensor(tensor, name):
    assert not torch.isnan(tensor).any(), f"{name}■has■NaN"
    assert not torch.isinf(tensor).any(), f"{name}■has■Inf"

validate_tensor(weights, "portfolio_weights")
validate_tensor(rewards, "rewards")
```

3. Periodic Statistics:

```
if step % 100 == 0:
    print(f"Step■{step}:")
    print(f"■■Weights:■min={weights.min():.4f},■"
          f"max={weights.max():.4f},■"
          f"mean={weights.mean():.4f}")
    print(f"■■Cash:■{weights[0]:.3f}")
    print(f"■■Active■stocks:■{(weights[1:]■>■0.05).sum()}")
```

V. Lessons Learned

This section synthesizes key lessons for implementing deep RL in finance.

A. Data Quality is Paramount

Lesson: Real financial data is messy. Gaps, outliers, and misalignments will break your model in subtle ways.

Best Practices:

1) Validate data immediately after loading (fail-fast)
2) Implement deterministic preprocessing pipeline
3) Document all preprocessing choices
4) Add unit tests for data loading
5) Monitor data quality metrics over time

B. Numerical Stability is Non-Negotiable

Lesson: Portfolio weights must sum to exactly 1.0. Logarithms must have positive arguments. Small errors compound quickly.

Critical Techniques:

1) Explicitly normalize probability distributions
2) Clamp values before logarithms and divisions
3) Use log-space arithmetic for products
4) Monitor min/max/mean/std of critical tensors
5) Add assertions for mathematical invariants

C. Start Simple, Add Complexity Gradually

Lesson: We chose long-only (no short selling) over the original's leverage operations. This simplified the problem and made debugging easier.

Guideline: Match model complexity to problem domain. Don't add features "just in case."

D. Regularization Can Freeze Learning

Lesson: EWC with $\lambda = 100.0$ completely prevented learning. The model stayed at initial (inactive) parameters.

Best Practice:

- Disable strong regularization during initial training
- Monitor regularization loss vs. policy loss
- Use much lower λ values (e.g., 5.0 instead of 100.0)
- Verify model is actually learning (portfolio value changing)

E. Transaction Costs Force Realism

Lesson: Unrealistically low costs (0.01%) led to overtrading strategies. Realistic costs (0.25%) produced practical strategies.

Recommendation: Use realistic transaction costs from the start. Add explicit turnover penalty if needed.

F. Online Learning Requires Stabilization

Lesson: OSBL's geometric sampling alone can overfit to recent noise. Combine with:

- Multiple batch updates ($N_b \geq 3$)
- Correlation matrix tracking
- Decay-aware attention
- EWC (after initial training, with low λ)

G. Validation Methodology Matters

Lesson: Test set leakage is easy to introduce and invalidates results.

Strict Requirements:

1) Split data chronologically (train/val/test)
2) Never access test set during training
3) Use validation set for all model selection
4) Access test set exactly once for final evaluation

H. Debugging Systematically Saves Time

Lesson: Structured debugging (conditional logging, validation functions, periodic stats) is faster than ad-hoc print statements.

Best Practices:

1) Use verbosity levels for logging
2) Add assertions for invariants
3) Log distributions (min/max/mean/std), not just values
4) Test components in isolation with synthetic data
5) Remove debugging code before production runs

VI. Experimental Results

A. Experimental Setup

Dataset:

- **Assets**: 11 stocks from S&P 500
- **Period**: 2016-2025 (9 years of daily data)
- **Features**: Open, High, Low, Close (OHLC)
- **Frequency**: Daily (one decision per trading day)
- **Data Source**: SQLite database with historical stock data

Data Split (chronological):

- Training: 70% (approximately 2016-2022)
- Validation: 15% (approximately 2022-2023)
- Test: 15% (approximately 2023-2025)

Model Configuration:

- Window size: $k = 31$ days
- Local context: $\ell = 5$ days
- Model dimension: $d_{\text{model}} = 12$
- Attention heads: 2
- Encoder/Decoder layers: 1 each
- Batch size: 128

Online Learning Parameters:

- OSBL recency bias: $\beta = 0.05$
- Batches per update: $N_b = 4$
- Replay buffer size: $M = 2000$
- Gradient clipping: 1.0
- Decay attention: $\lambda = 0.1$
- Correlation EMA: $\gamma = 0.9$
- Recency PE: $\beta_{\text{PE}} = 0.1$
- EWC: Disabled (based on lessons learned)

Training:

- Total steps: 500 (preliminary evaluation)
- Learning rate: 0.001
- Weight decay: 10^{-7}
- Optimizer: Adam

Transaction Costs:

- Commission rate: 0.25% per trade
- Turnover penalty: 0.002
- Diversification incentive: 0.05

B. Evaluation Metrics

1) **Accumulated Portfolio Value (APV)**: Final wealth / initial wealth
2) **Sharpe Ratio (SR)**: Risk-adjusted return

$$\text{SR} = \frac{\text{mean}(r_t)}{\text{std}(r_t)} \tag{25}$$

3) **Maximum Drawdown (MDD)**: Largest peak-to-trough decline

$$\text{MDD} = \max_t \left(\frac{\max_{\tau \leq t} V_\tau - V_t}{\max_{\tau \leq t} V_\tau} \right) \tag{26}$$

4) **Average Turnover (TO)**: Mean daily rebalancing

$$\text{TO} = \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{m} |w_{t,i} - w_{t-1,i}| \tag{27}$$

C. Results

1) Final Test Set Performance: After completing training and evaluating once on the held-out test set:

TABLE I
FINAL PERFORMANCE ON TEST SET

Metric	Value
Accumulated Portfolio Value	**1.5749**
Sharpe Ratio	**0.5876**
Total Return	+57.49%
Average Daily Return	0.042%
Return Volatility	0.071%
Active Stocks (avg)	7-10
Average Cash Weight	20-30%

2) Interpretation: **Portfolio Growth**: APV of 1.57 indicates the portfolio grew by 57% over the test period, demonstrating the model successfully learned profitable trading strategies.

Risk-Adjusted Returns: Sharpe ratio of 0.59 is considered acceptable (above 0.5 threshold). For reference:

- SR $<$ 0.5: Poor risk-adjusted returns

- SR 0.5-1.0: Acceptable
- SR $>$ 1.0: Good
- SR $>$ 2.0: Excellent (rare)

Diversification: The model maintained positions in 7-10 stocks on average, showing proper risk diversification rather than over-concentration.

Cash Management: 20-30% average cash weight indicates prudent reserve management, allowing the model to avoid being fully exposed during uncertain periods.

3) Portfolio Behavior: During testing, we observed:

Trading Activity:

- Model actively rebalanced portfolio based on price signals
- Turnover remained moderate (avoiding excessive trading costs)
- Portfolio composition evolved over time (not static)

Asset Selection:

- Model learned to concentrate on higher-performing stocks
- Reduced exposure to declining assets
- Maintained diversification for risk management

Stability:

- No extreme concentration (max weight typically $<$ 30%)
- No numerical instabilities (NaN, Inf)
- Consistent behavior throughout test period

D. Impact of Implementation Fixes

Table II shows the dramatic impact of our implementation fixes:

TABLE II
BEFORE VS. AFTER IMPLEMENTATION FIXES

Metric	Before	After
Cash Weight	99.92%	20-30%
Active Stocks	0	7-10
Portfolio Value	1.000	1.575
Model Behavior	Inactive	Trading
Numerical Stability	NaN/Inf	Stable

This demonstrates the critical importance of:

1) Careful regularization tuning (EWC)
2) Numerical stability measures
3) Realistic transaction costs
4) Diversification incentives

E. Comparison to Baselines

While comprehensive comparison is future work, we can contextualize our results:

Equal-Weight Portfolio: Allocating $1/m$ to each stock and rebalancing monthly typically achieves:

- Similar returns to market index
- Higher volatility due to equal weighting
- Lower transaction costs (infrequent rebalancing)

Buy-and-Hold S&P 500: Historical S&P 500 returns:

- Annual return: 8-10%
- Sharpe ratio: 0.4-0.6
- No transaction costs

Our approach demonstrates competitive performance while learning entirely from price data without manual feature engineering.

F. Limitations and Future Work

Current Limitations:

1) Short training (500 steps) due to computational constraints
2) Single random seed (no statistical significance testing)
3) Limited asset universe (11 stocks)
4) No testing on extreme market events (crashes, etc.)

Future Directions:

1) Extended training (10,000 steps) for better convergence
2) Multiple seeds for statistical confidence
3) Larger portfolios (50-500 stocks)
4) Stress testing on crisis periods (2008, 2020)
5) Comparison with traditional baselines
6) Interpretability analysis (attention visualizations)
7) Real-time deployment considerations

VII. CONCLUSION

This project successfully implemented and extended the Relation-Aware Transformer (RAT) for automated stock portfolio management using daily price data and a long-only investment strategy. Through systematic development and debugging, we achieved a final portfolio value of 1.57× with Sharpe ratio of 0.59 on held-out test data.

A. Key Achievements

1) **Architecture Adaptation**: Successfully adapted RAT from cryptocurrency trading to stock market investing with long-only constraints
2) **Online Learning Extensions**: Implemented five novel modifications (OSBL, decay attention, correlation tracking, recency PE, EWC) for non-stationary markets
3) **Methodological Rigor**: Corrected test set leakage through proper train/validation/test methodology
4) **Problem-Solving**: Systematically diagnosed and resolved critical implementation challenges including model inactivity, numerical instability, and unrealistic trading behavior
5) **Knowledge Documentation**: Comprehensive documentation of lessons learned provides practical guidance for deep RL in finance

B. Practical Lessons

The most valuable contributions may be the documented lessons:

1) **Data quality is paramount**: Validate aggressively, fail fast, document preprocessing
2) **Numerical stability is critical**: Normalize weights, clamp values, use log-space arithmetic
3) **Start simple**: Long-only portfolio is easier to learn and debug than complex leverage operations

4) **Regularization requires care**: Too-strong EWC ($\lambda = 100$) completely freezes learning
5) **Transaction costs enforce realism**: Use realistic fees (0.25%) from the start
6) **Online learning needs stabilization**: Combine OSBL with multiple mechanisms (correlation tracking, multiple batches)
7) **Validation methodology matters**: Strict train/val/test splits are non-negotiable for scientific validity
8) **Debug systematically**: Structured logging and assertions beat ad-hoc print statements

C. Alignment with Project Objectives

Reviewing our initial objectives:

✓ **Literature Review**: Examined classical and modern RL approaches, identified gaps
✓ **Architectural Design**: Extended RAT with online learning for stock markets
✓ **Implementation**: Built complete system using PyTorch with real stock data
✓ **Performance Evaluation**: Achieved APV=1.57, SR=0.59 on test set
✓ **Analysis and Reporting**: Documented challenges, solutions, and lessons learned

D. Future Directions

Immediate Next Steps:

1) Extended training (10,000 steps) for better convergence
2) Multiple random seeds for statistical confidence
3) Systematic ablation studies of each modification
4) Comparison against traditional baselines

Research Extensions:

1) Adaptive hyperparameters (adjust β, λ based on market regime)
2) Explicit regime detection and strategy switching
3) Multi-asset class portfolios (stocks, bonds, commodities)
4) Risk constraints (VaR, CVaR limits)
5) Interpretability (attention visualizations, feature importance)

Practical Deployment:

1) Real-time data pipeline integration
2) Latency optimization for trading execution
3) Model compression for production deployment
4) Backtesting on longer historical periods
5) Stress testing on market crash scenarios

E. Broader Impact

This work contributes to the application of AI to financial decision-making. Potential benefits include:

- Democratizing sophisticated investment strategies
- Reducing reliance on manual feature engineering
- Learning from data without restrictive assumptions

However, responsible deployment requires consideration of:

- Systemic risk if many agents use similar strategies
- Market impact from automated trading
- Regulatory compliance and oversight
- Transparency and interpretability requirements

F. Final Remarks

Implementing deep reinforcement learning for portfolio management is challenging but rewarding. The path from concept to working system involves numerous pitfalls—numerical instability, test set leakage, over-regularization—that can silently produce incorrect results.

By openly documenting these challenges and their solutions, we hope to:

1) Accelerate future research by helping others avoid common pitfalls
2) Demonstrate the importance of rigorous validation methodology
3) Emphasize that implementation details matter as much as algorithmic innovation
4) Provide a realistic view of applying deep RL to finance

The successful application of AI to quantitative investing requires not just sophisticated algorithms, but also careful attention to data quality, numerical stability, proper validation, and systematic debugging. We hope this work serves as both a technical contribution and a practical guide for researchers in this exciting field.

References

[1] H. Markowitz, "Portfolio Selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77-91, 1952.
[2] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, 2017.
[3] B. Li and S. C. H. Hoi, "Online Portfolio Selection: A Survey," *ACM Computing Surveys*, vol. 46, no. 3, pp. 1-36, 2012.
[4] Z. Jiang, D. Xu, and J. Liang, "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem," *arXiv preprint arXiv:1706.10059*, 2017.
[5] P. C. Pendharkar and P. Cusatis, "Trading financial indices with reinforcement learning agents," *Expert Systems with Applications*, vol. 103, pp. 1-13, 2018.
[6] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep Learning for Event-Driven Stock Prediction," in *Proceedings of IJCAI*, pp. 2327-2333, 2020.
[7] K. Xu, Y. Zhang, D. Ye, P. Zhao, and M. Tan, "Relation-Aware Transformer for Portfolio Policy Learning," in *Proceedings of IJCAI*, pp. 4647-4653, 2020.
[8] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," in *ICLR*, 2016.
[9] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521-3526, 2017.
[10] M. McCloskey and N. J. Cohen, "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem," *Psychology of Learning and Motivation*, vol. 24, pp. 109-165, 1989.
[11] A. A. Rusu et al., "Progressive Neural Networks," *arXiv preprint arXiv:1606.04671*, 2016.
[12] D. Lopez-Paz and M. Ranzato, "Gradient Episodic Memory for Continual Learning," in *NIPS*, 2017.
[13] L. Prechelt, "Early Stopping - But When?" in *Neural Networks: Tricks of the Trade*, Springer, 1998.
[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.