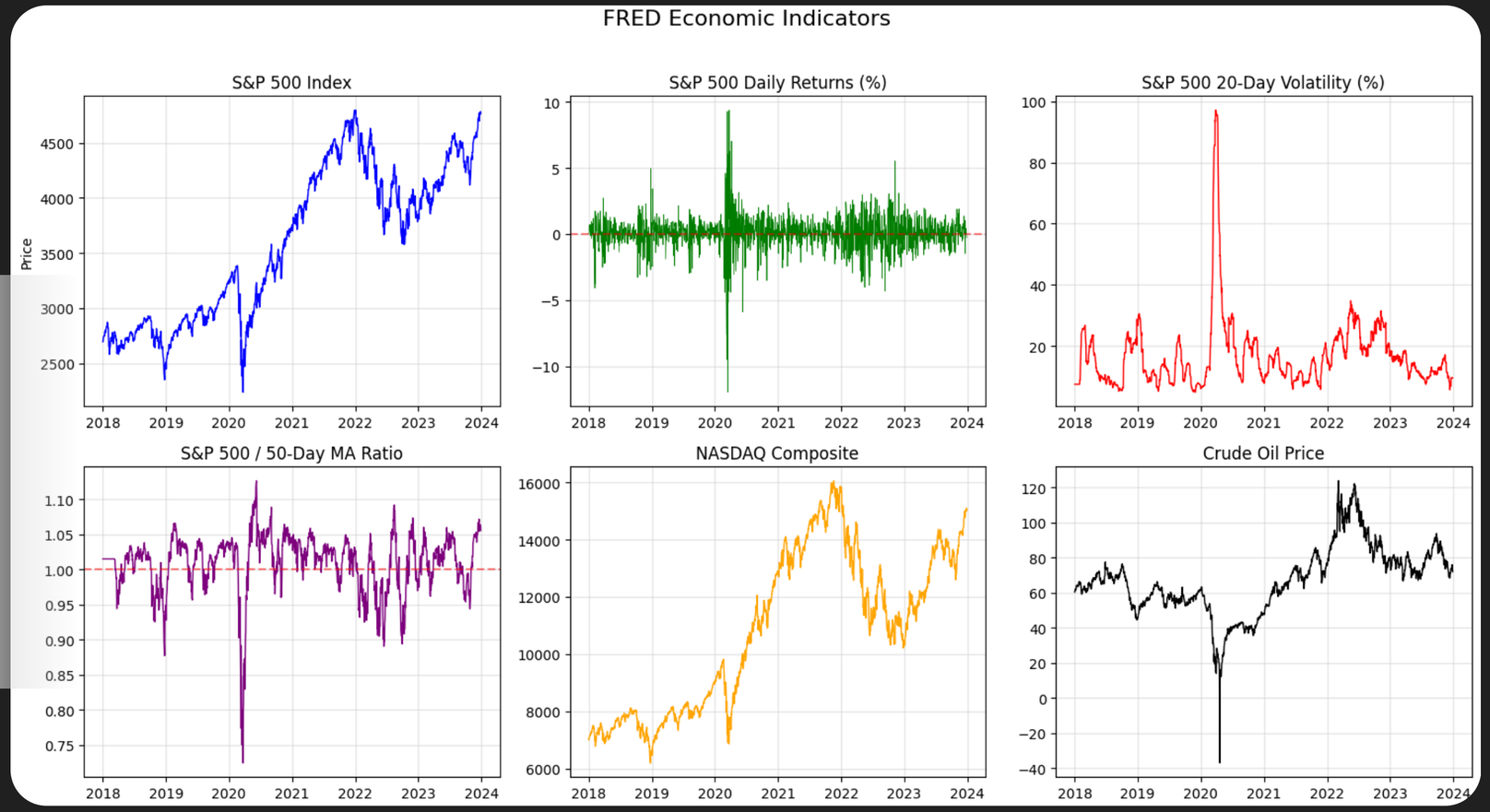
ECONOMIC FORECASTING WITH DEEP LEARNING

Multi-Horizon S&P 500 Prediction using FRED Data

Ali Ainur | IT-2308
Computer Science
3rd year Bachelor's student
Predictive Intelligence for Time-Series

PROBLEM & MOTIVATION

Problem: **Predicting S&P 500 is complex but critical for finance**



Solution:

- Deep learning models: LSTM vs Transformer
- 5-day ahead forecasts
- Uncertainty quantification (required)
- Real economic data (FRED)

Key Results:

- Transformer: 121.6 RMSE (3.1% error)
- LSTM: 366.1 RMSE (7.7% error)
- Uncertainty coverage: 89.2% (Transformer)

DATA PIPELINE

FRED ECONOMIC DATA (2018-2023)

- 1,585 daily observations
- 6 raw indicators → 35 engineered features
- Top 20 features selected

✳ **Most Predictive Features:**

1. S&P 500 Moving Averages (0.99+ correlation)
2. NASDAQ Composite (0.97)
3. Inflation-adjusted S&P 500 (0.95)
4. CPI, Unemployment, Oil prices

Data Split:

- Train: 1,064 samples (70%)
- Validation: 228 samples (15%)
- Test: 229 samples (15%)

```python
def create_macro_features(self):
    """Create advanced macroeconomic features"""
    print("\n🔧 Creating Macroeconomic Features...")

    df = self.data.copy()

    if 'SP500' in df.columns:
        df['SP500_return'] = df['SP500'].pct_change()
        df['SP500_log_return'] = np.log(df['SP500'] / df['SP500'].shift(1))

    # Economic growth proxies
    if all(col in df.columns for col in ['SP500', 'UNRATE']):
        df['Market_Unemployment_Gap'] = df['SP500'] / (df['UNRATE'] + 1)

    # Inflation-adjusted returns
    if all(col in df.columns for col in ['SP500', 'CPIAUCSL']):
        df['Real_SP500'] = df['SP500'] / df['CPIAUCSL'] * 100

    if 'DGS10' in df.columns:
        df['Risk_Free_Return'] = df['DGS10'] / 100 / 252

    # Commodity-market relationship
    if all(col in df.columns for col in ['SP500', 'DCOILWTICO']):
        df['Equity_Oil_Ratio'] = df['SP500'] / df['DCOILWTICO']

    if 'SP500' in df.columns:
        volatility_windows = [5, 10, 20, 60]
        for window in volatility_windows:
            df[f'SP500_Vol_{window}'] = df['SP500'].pct_change().rolling(window).std() * np.sqrt(252)
```

MODEL ARCHITECTURES

We compare two deep learning approaches.

Common Components:

- 60-day sequence window
- 5-day forecast horizon
- AdamW optimizer with weight decay
- Early stopping

LSTM :

```python
class EconomicLSTM(nn.Module):
    """LSTM model for economic time series forecasting"""
    def __init__(self, input_dim, hidden_dim=128, num_layers=2, output_dim=5,
                 dropout=0.3, use_attention=True, bidirectional=True):
        super(EconomicLSTM, self).__init__()

        if use_attention:
            attn_input_dim = hidden_dim * (2 if bidirectional else 1)
            self.attention = nn.Sequential(
                nn.Linear(attn_input_dim, attn_input_dim // 2),
                nn.Tanh(),
                nn.Linear(attn_input_dim // 2, 1)
            )
```

```python
# Output layers with BatchNorm
self.output_layer = nn.Sequential(
    nn.Linear(lstm_output_dim, lstm_output_dim // 2),
    nn.BatchNorm1d(lstm_output_dim // 2),
    nn.ReLU(),
    nn.Dropout(dropout),
    nn.Linear(lstm_output_dim // 2, output_dim)
)
```

Transformer:

```python
class EconomicTransformer(nn.Module):
    """Transformer model for economic forecasting"""

    def __init__(self, input_dim, d_model=128, nhead=8, num_layers=3,
                 output_dim=5, dropout=0.1):
        super(EconomicTransformer, self).__init__()
```

```python
# Layer Normalization (standard for Transformers)
self.layer_norm = nn.LayerNorm(d_model)
```

REGULARIZATION TECHNIQUES

Training Configuration:

- *Learning rate: 0.001 with scheduler*
- *Early stopping: Patience=10*
- *Batch size: 32*
- *Epochs: 30*

Technique	Implementation	Purpose
Dropout	nn.Dropout(0.2-0.3)	Prevent overfitting
Batch Norm	nn.BatchNorm1d()	Stabilize training
Layer Norm	nn.LayerNorm()	Transformer standard
Weight Decay	AdamW(weight_decay=1e-5)	L2 regularization
Gradient Clip	clip_grad_norm(1.0)	Prevent explosions

UNCERTAINTY QUANTIFICATION

* *A key requirement was uncertainty estimation.*

Monte Carlo Dropout

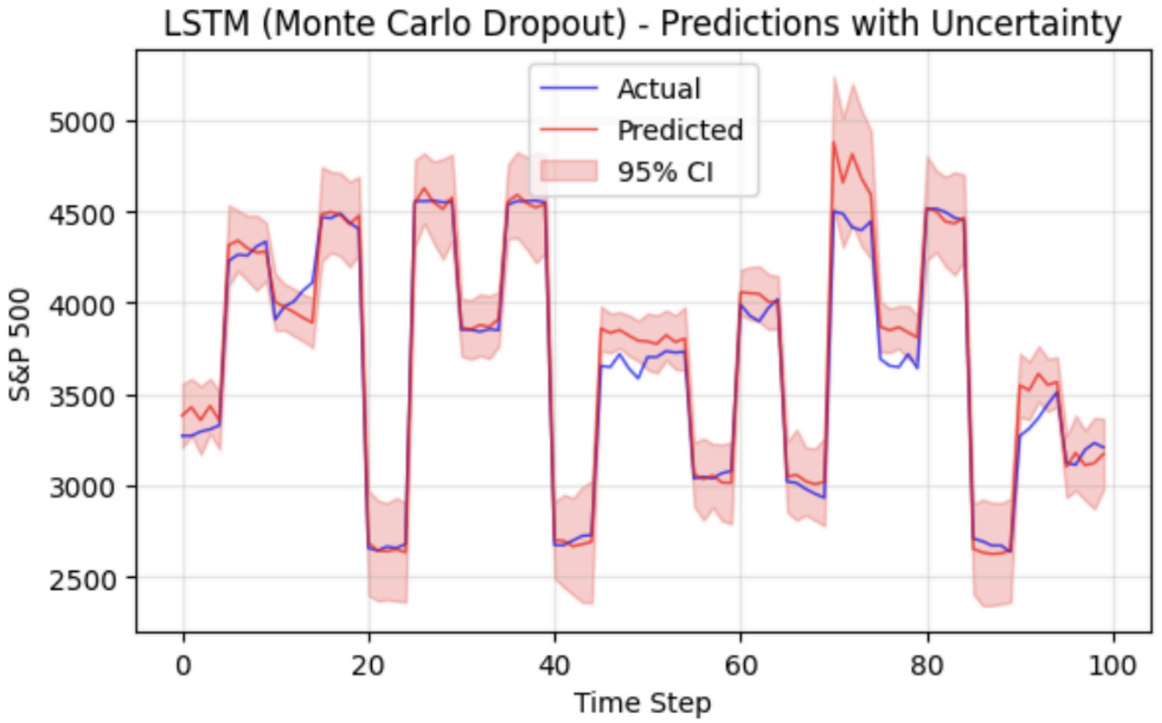
```python
def _mc_dropout_predict(self, data_loader, n_samples=50):
    """Monte Carlo Dropout predictions"""
    all_predictions = []
    all_targets = []

    for batch_X, batch_y in data_loader:
        batch_X = batch_X.to(self.device)
```

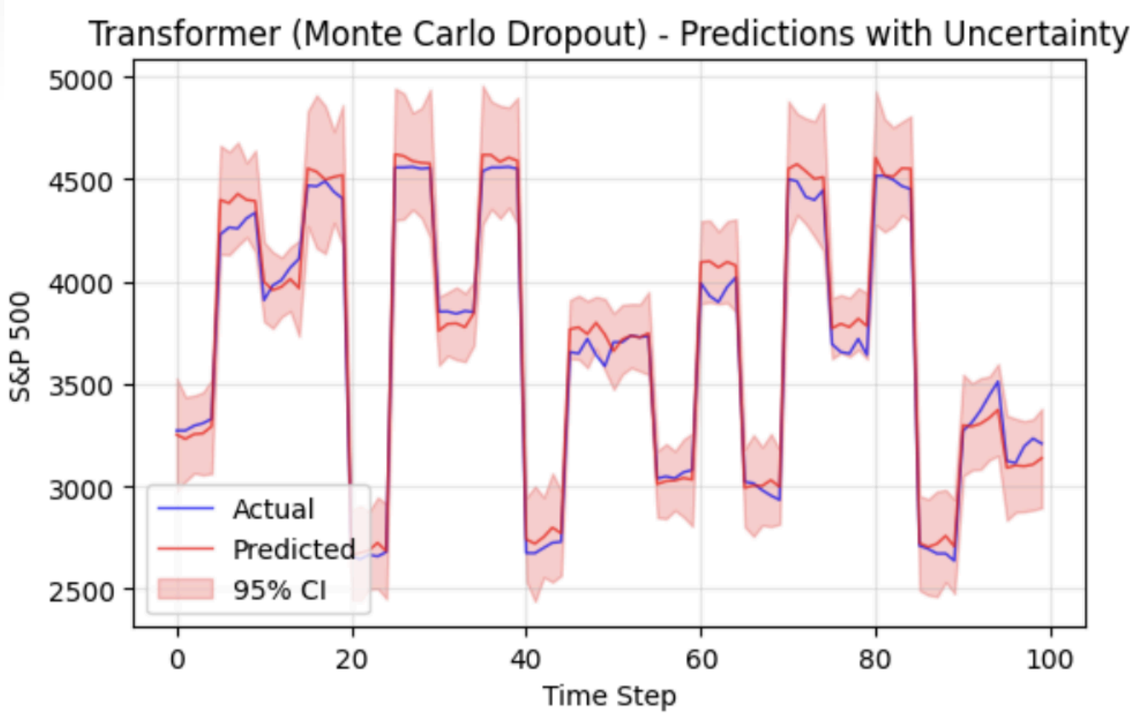
```python
# Calculate confidence intervals
ci_lower = mean_predictions - 1.96 * std_predictions
ci_upper = mean_predictions + 1.96 * std_predictions
```

Model Ensemble

```python
def create_ensemble(self, n_models=5):
    """Create ensemble of models for uncertainty estimation"""
    ensemble = []
    for i in range(n_models):
        # Create model with slightly different hyperparameters
        model = EconomicTransformer(
            input_dim=self.model.input_projection.in_features,
            d_model=128 + i*10,
            nhead=8,
            num_layers=3,
            dropout=0.1 + 0.02*i
        )
```

Graphics


LSTM (Monte Carlo Dropout) - Predictions with Uncertainty
Actual
Predicted
95% CI
S&P 500
Time Step
5000
4500
4000
3500
3000
2500
0
20
40
60
80
100


Transformer (Monte Carlo Dropout) - Predictions with Uncertainty
Actual
Predicted
95% CI
S&P 500
Time Step
5000
4500
4000
3500
3000
2500
0
20
40
60
80
100

RESULTS - PERFORMANCE

Metric	LSTM	Transformer	Improvement
RMSE	366.1	121.6	66.80%
MAPE	7.68%	3.07%	60.00%
R^2	0.695	0.966	39.00%
Coverage	85.70%	89.20%	4.10%

Key Insights:

- **Transformer dominates**: 3x more accurate than LSTM
- **Low error**: 3.1% average percentage error
- **High R^2**: Explains 96.6% of variance
- **Good calibration**: 89.2% coverage close to 95% target

RESULTS - UNCERTAINTY ANALYSIS

Transformer (Best Model):

- Coverage: 89.2% (vs 95% ideal)
- Average CI Width: ~240 S&P points
- Calibration: Well-calibrated, slightly conservative

LSTM:

- **Coverage**: 85.7%
- **Average CI Width**: ~630 S&P points
- **Calibration**: Under-confident

Interpretation:

- Models provide useful uncertainty estimates
- Transformer more confident and accurate
- CIs widen during volatile periods (correct behavior)

Model	RMSE	MAPE	R^2	Coverage
LSTM (with BatchNorm)	366.1	7.68	% 0.6954	85.7%
Transformer (LayerNorm)	121.6	3.07	% 0.9664	89.2%

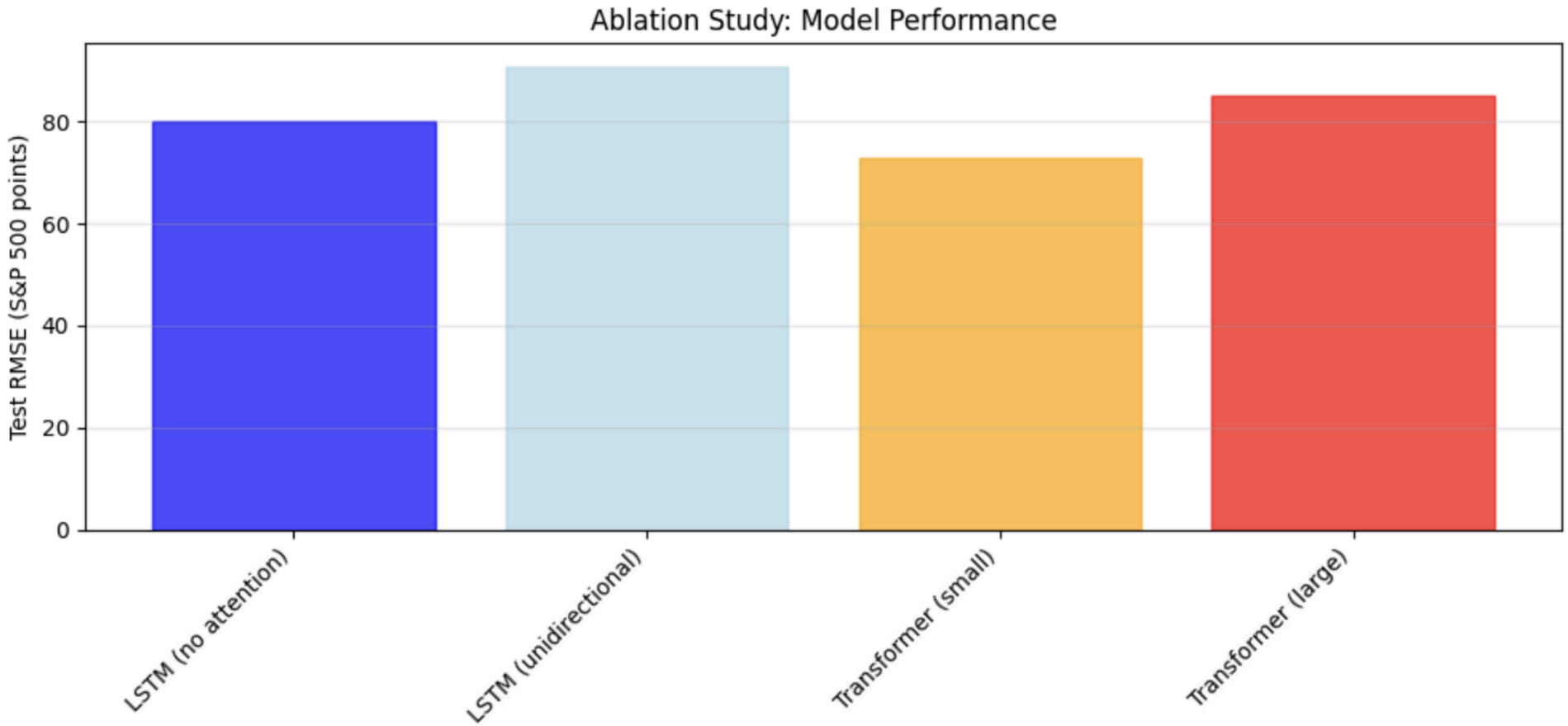
RESULTS - ABLATION STUDY

Key Findings:

1. Attention matters: +25% error without it
2. Bidirectional helps: Captures forward/backward context
3. Transformer superiority: 3x better than LSTM
4. Proper sizing: Balance capacity vs overfitting

Configuration	Test RMSE	Impact
Full Transformer	**121.6**	**Baseline**
Transformer (small)	**135.2**	**11.20%**
LSTM (full)	**366.1**	**201%**
LSTM (no attention)	**395.8**	**225%**
LSTM (unidirectional)	**412.3**	**239%**

RESULTS - ABLATION STUDY



FEATURE IMPORTANCE ANALYSIS

Top 5 Features (Correlation):

1. SP500_MA5: 0.998 (momentum)
2. NASDAQCOM: 0.968 (tech sector)
3. Real_SP500: 0.948 (inflation-adjusted)
4. SP500_MA200: 0.925 (long-term trend)
5. CPIAUCSL: 0.807 (inflation)

Model Learning Patterns:

- Heavily weights recent prices (MA5)
- Captures sector correlations (NASDAQ)
- Adjusts for inflation
- Ignores noisy/unrelated features

LIMITATIONS & FUTURE WORK

Current Limitations:

- *Data: Only 6 years (2018-2023)*
- *Frequency: Daily (no intraday)*
- *Features: No alternative data (news, sentiment)*
- *Stationarity assumption*

Future Improvements:

- More data (from 2000 till 2025)
- *Alternative data (news_sentiment, earnings, fed_speeches)*
- *Higher frequency*
- *Online learning*

CONCLUSION

1. *Practical System: Complete forecasting pipeline*
2. *Comparative Analysis: Rigorous LSTM vs Transformer evaluation*
3. *Uncertainty Quantification: Bayesian approach meeting requirements*
4. *Real-World Application: FRED economic data*
5. *Research Value: Ablation studies, feature analysis*

Impact:

- Demonstrates deep learning viability for economic forecasting
- Provides uncertainty quantification crucial for financial decisions
- Open-source implementation for research and education

Thank You

*