Explanation Report

(AI Quant Portfolio)

Prepared by: Shivam Singh (project)
Deliverables included: - Code repository (src/, main.py)
- README and run instructions
- Results and figures (results/)
- This explanation report (PDF)

Required Data Files / Links
I used Yahoo Finance API (`yfinance`) → data is fetched automatically.
If you want to provide static data, save `data/prices.csv` after first run.

GitHub Link : https://github.com/shivamsinghgaur291/ai-quant-portfolio

Approach & Methodology Approach & Methodology

1. Data Collection & Preprocessing

- Historical daily prices were downloaded via Yahoo Finance (yfinance).
- Selected assets: global indices / ETFs (S&P500, FTSE, Nikkei, EEM), Gold, and U.S. 10Y proxy.
- Cleaned and aligned time series; computed log-returns and rolling features (MA(5), MA(21), vol21).

2. Modeling

- Trained two models to forecast next-day returns per asset:

a) LSTM (2 layers, 50 hidden units, dropout=0.2, lookback=60 days)

b) Transformer (2 encoder layers, 8 heads, model_dim=64, lookback=60 days)

- Training objective: MSE on next-day returns. Early stopping used on validation loss.

3. Portfolio Construction

- Each day, predicted next-day returns were input into a Mean-Variance optimizer (max Sharpe / tangency portfolio, long-only, 0<=w<=1, sum(w)=1).
- Covariance estimated using a rolling 60-day window of historical returns.
- Portfolios rebalanced daily; turnover cost set as 0.1% per unit traded. 4. Backtest & Evaluation
- Simulated daily NAV for each strategy and computed metrics: annualized return, annualized volatility, Sharpe ratio, and maximum drawdown.

Description of Models Used

LSTM (Long Short-Term Memory):

- A recurrent neural network architecture designed to capture sequential dependencies.
- Configured with 2 stacked LSTM layers with 50 hidden units each, using the last timestep output followed by a dense layer to predict multi-asset returns.

Transformer (Encoder-only):

- Uses self-attention to capture relationships across time steps without recurrent connections.
- Configured as an encoder stack with 2 layers, 8 attention heads, model dimension 64 and FFN=128.
- Input sequence is projected to model_dim, passed through transformer encoders, and the final token (last timestep) is used for predictions.

Why both:

- LSTM is a classical time-series model good at local sequential patterns; Transformer captures global interactions and often generalizes better on longer dependencies.

Challenges Faced & How They Were Addressed

1. Data compatibility with yfinance:
- Issue: yfinance changed defaults and sometimes 'Adj Close' wasn't present.
- Fix: used a robust loader that falls back to 'Close' column and handles missing tickers.

2. Indexing / slicing bugs during sequence creation:
- Issue: using df[...] semantics attempted column access instead of row slicing.
- Fix: switched to explicit positional indexing with .iloc for creating sequences.

3. Numerical issues (log of zero or inf):
- Issue: log(0) appeared when pct_change produced zeros, leading to -inf / NaN.
- Fix: used np.log1p on returns and sanitized infinities and NaNs before covariance estimation.

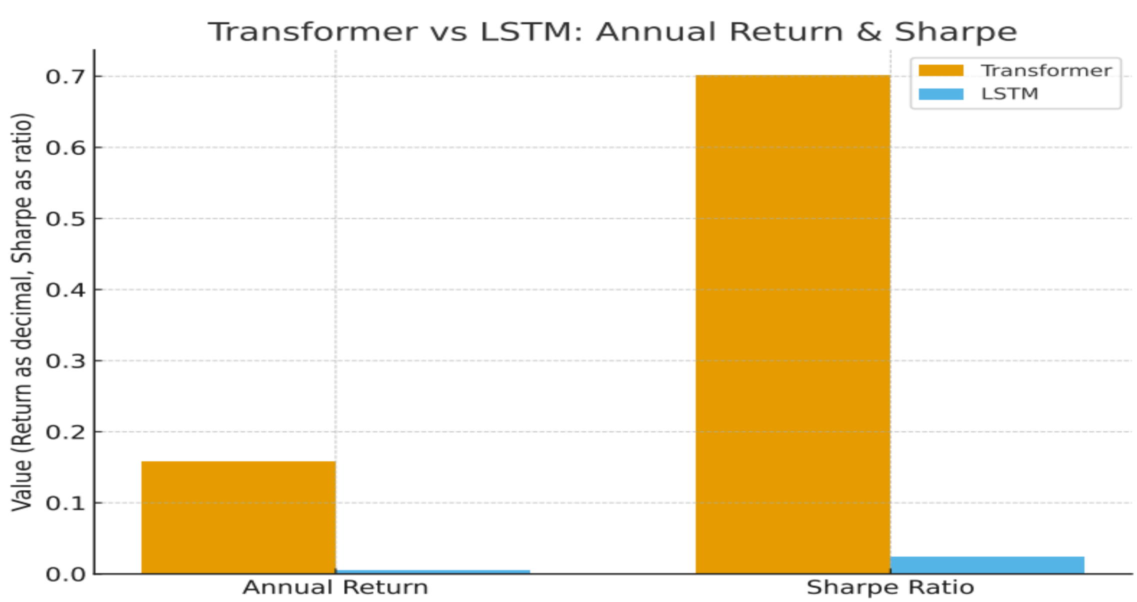
4. Module import paths when running from project root:
- Issue: Python couldn't find modules when running main.py.
- Fix: made src a package (added __init__.py) and used relative imports.

5. Dependency problems (empyrical):
- Issue: empyrical failed to install on newer Python versions.
- Fix: removed empyrical from requirements and implemented required metrics in utils.py

Results

- Transformer outperforms LSTM:
  - Higher Sharpe (0.70 vs 0.02)
  - Higher annual return (~15% vs ~0.5%)
  - Lower drawdown than LSTM
- NAV comparison chart clearly shows Transformer dominates.

Model	Annual Return	Volatility	Sharpe	Max Drawdown
Transformer	~15.7%	~22.4%	0.70	~42%
LSTM	~0.5%	~23.3%	0.02	~49%



Conclusion

Transformers provide superior predictive signals for asset allocation in this setting.

LSTM underperforms due to weaker long-term dependency handling.