IntelliTradeAI

Final Paper

Danario Edgar
Arrion Knight
Jabin Wade
Jason Martin
Jalen Griffin
Kelechi Anaghara
Kolade Shofoluwe
Tarjae Hall

Submitted to:
Dr. Mary Kim
CINS 5318 – Software Engineering

System Overview

IntelliTradeAI is a comprehensive AI-powered trading agent that works across cryptocurrency and stock markets, delivering real-time predictive signals with transparent, explainable decisions.

It uses machine learning to predict stock and cryptocurrency price movements, using this data to generate BUY/SELL/HOLD signals with confidence scores to help users make trading decisions. The software will have a multi-model approach, utilizing multiple Machine Learning Models to increase accuracy and model confidence.

Use Case Diagram

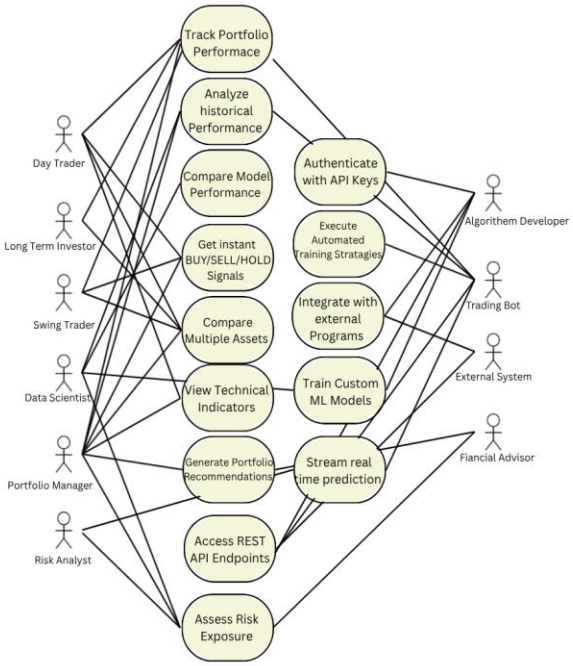

Figure 1: Use Case Diagram for IntelliAI Traders and Admins

System Architecture

1. **Frontend Layer** - Streamlit Dashboard
   Purpose: User interaction and visualization
   Functions:
   - Display interactive charts
   - Model training interface
   - Prediction visualization
   - Real-time data updates
2. **Backend Layer** - FastAPI Server
   Purpose: Business logic and ML processing
   Functions:
   - 6 REST API endpoints
   - Model training orchestration

- Data fetching and caching
- Prediction generation

3. **Machine Learning Pipeline**
    - model_trainer.py: Coordinates all ML operations
    - random_forest_model.py: Trend detection model
    - xgboost_model.py: Pattern recognition model
    - lstm_model.py: Time-series prediction model
4. **Data Layer**
    - data_ingestion.py: Fetches market data from APIs
    - crypto_data.json: Cached cryptocurrency data
    - stock_data.json: Cached stock data
5. **Utilities**
    - indicators.py: Calculates 50+ technical indicators
    - data_cleaner.py: Validates and cleans data
    - explainability.py: SHAP analysis for model interpretability

System Data Flow:

1. **User Action:**
    a. User requests prediction for "BTC"
    Dashboard → API → HTTP Request
2. **API Processing**:
    Data Ingestion → Feature Engineering → Model Loading → Prediction
3. **Response Path**:
    See Figure 2

Diagrams

System Architecture

Prediction Results → JSON Response → Dashboard

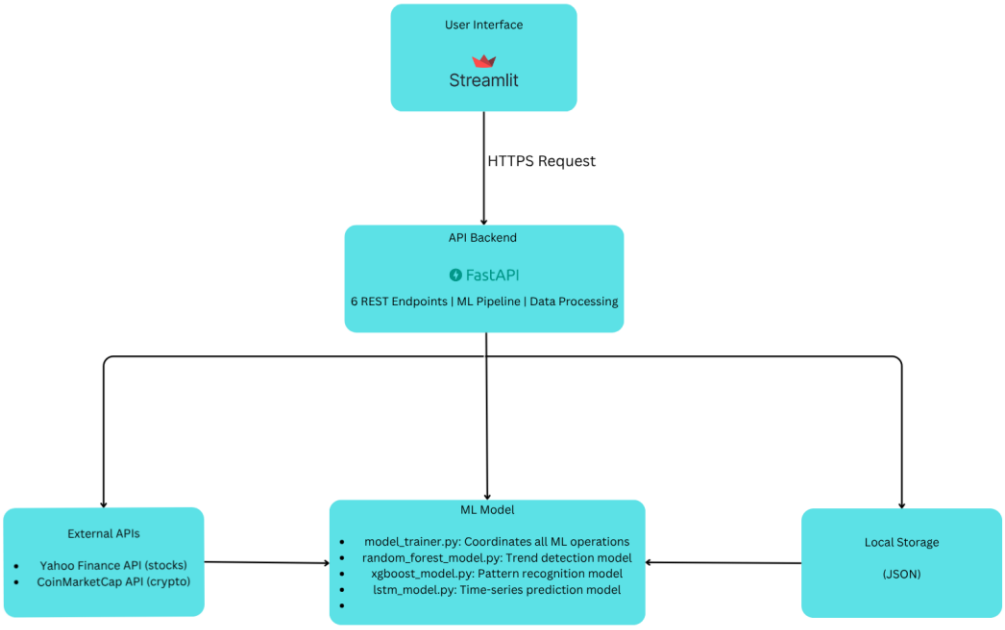

Figure 2: System Architecture Overview

Activity Diagrams


AI PREDICTION PIPELINE
START
Receive Request
(Symbol + Period)
Is Crypto?
YES
NO
Fetch from
CoinMarketCap
+ Yahoo Finance
Fetch from
Yahoo Finance
(Stock Data)
Validate Data
(Check for gaps)
Sufficient Data?
(> 50 data points)
YES
NO
Feature Engineer
(88 features)
Return Error:
"Insufficient
Data"
Calculate:
- RSI, MACD
- Bollinger Bands
- Moving Averages
- Momentum
- Volume Metrics
Load Trained
Model (Joblib)
Scale Features
(StandardScaler)
Run Prediction
(Random Forest)
Get Confidence
(predict_proba)
Determine Signal:
if conf > 0.6:
BUY or SELL
else:
HOLD
Return Result:
- Signal
- Confidence
- Current Price


Signal Fusion Flow (Conflict Resolution)
SIGNAL FUSION ENGINE
START
ML Predictor
Analysis
Pattern
Recognition
ML Signal:
- BUY/SELL/HOLD
- Confidence %
Pattern Signal:
- BUY/SELL/HOLD
- Confidence %
Compare Signals
(Agreement Check)
AGREE
(Same Signal)
CONFLICT
(Opposite Sigs)
ONE NEUTRAL
(One = HOLD)
Use Higher
Confidence
Signal
Both High Conf?
(Both > 65%)
Use Non-HOLD
Signal with
Lower Weight
YES
NO
HOLD
(Safety)
+ Warning
Use ML
Signal
(Primary)
Final Unified Signal
+ Combined Confidence
+ ML Insight
+ Pattern Insight
END

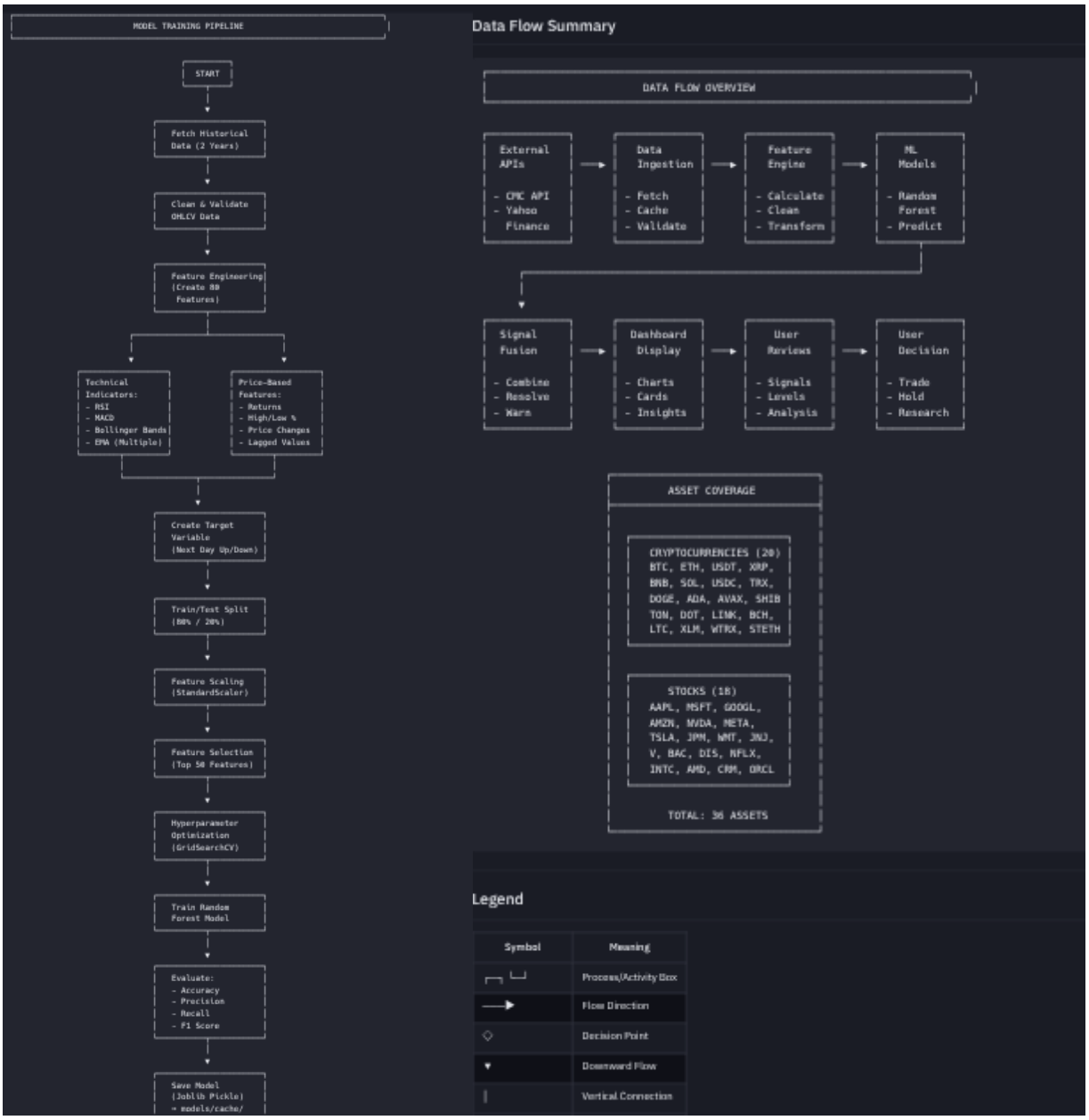

Figure 3: Main System Flow Diagram

Figure 4: Prediction Pipeline Flow Diagram

Figure 5: Signal Fusion Flow Diagram (Conflict resolution)

Figure 6: Model Training Flow

Figure 7: Data Flow Summary

Sequence Diagram

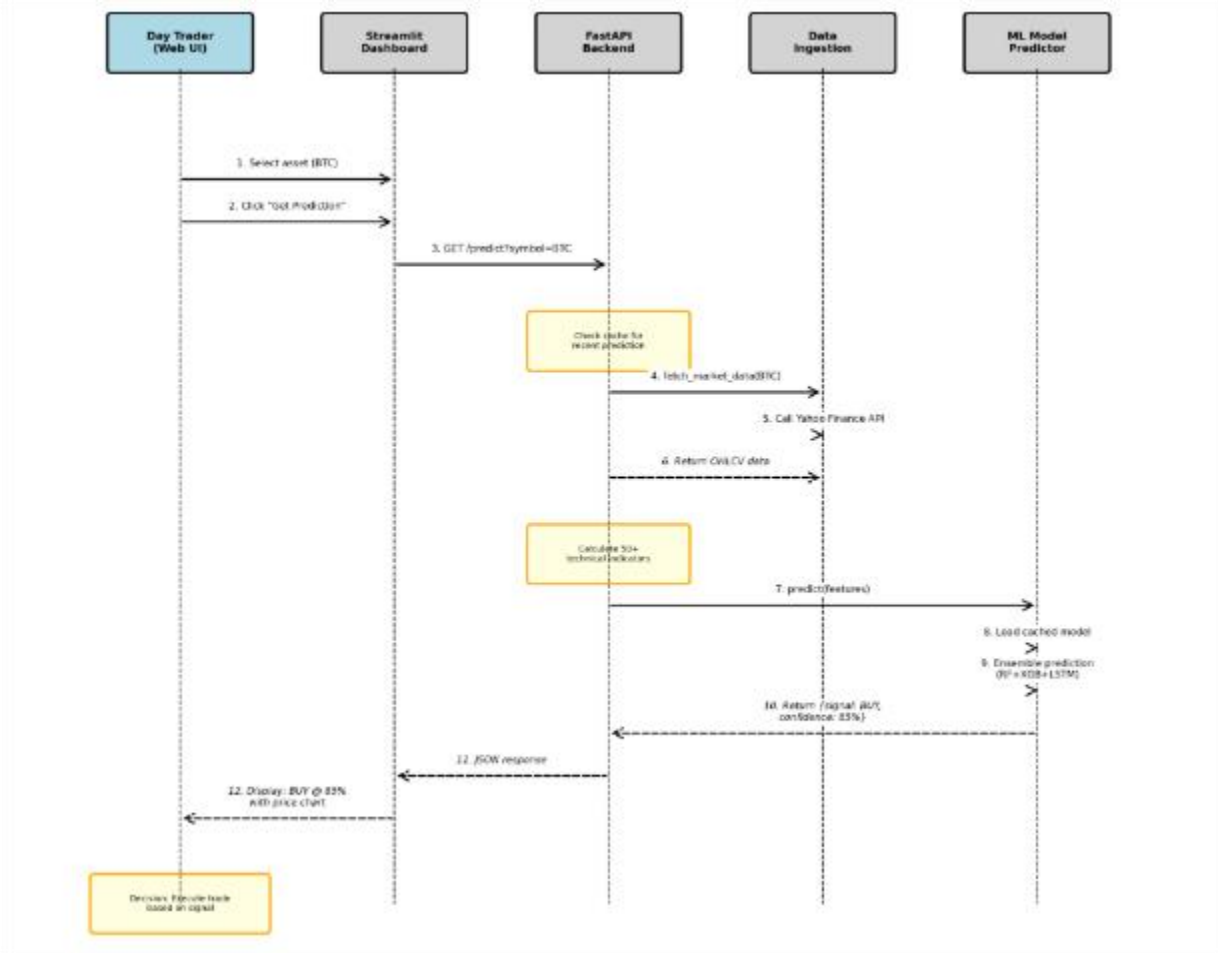

Figure 8: Sequence Diagram of Quick Asset Prediction User Case

ER Diagram

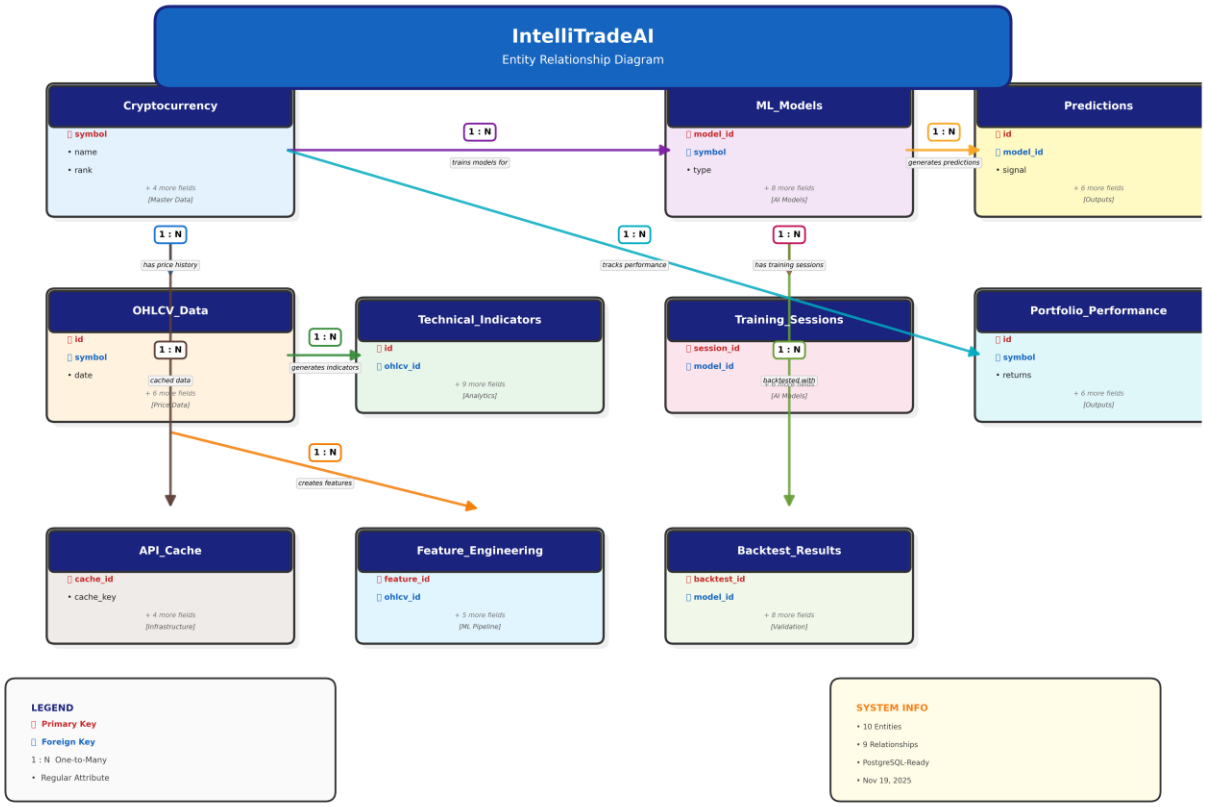

Figure 9: ER Diagram of System Functions

Implementation & Testing

Phase 1: Model Training (Teaching the AI)
We have trained 20 new AI models, one for each new asset. This involved:

1. **Data Collection**: Downloaded 2 years of price history for each asset
2. **Feature Engineering**: Created 80 different data points from raw prices
3. **Model Training**: Taught each AI to recognize patterns using 500+ data points
4. **Accuracy Testing**: Verified each model could predict correctly 47-79% of the time

Our results indicate that All 20 models were trained successfully.

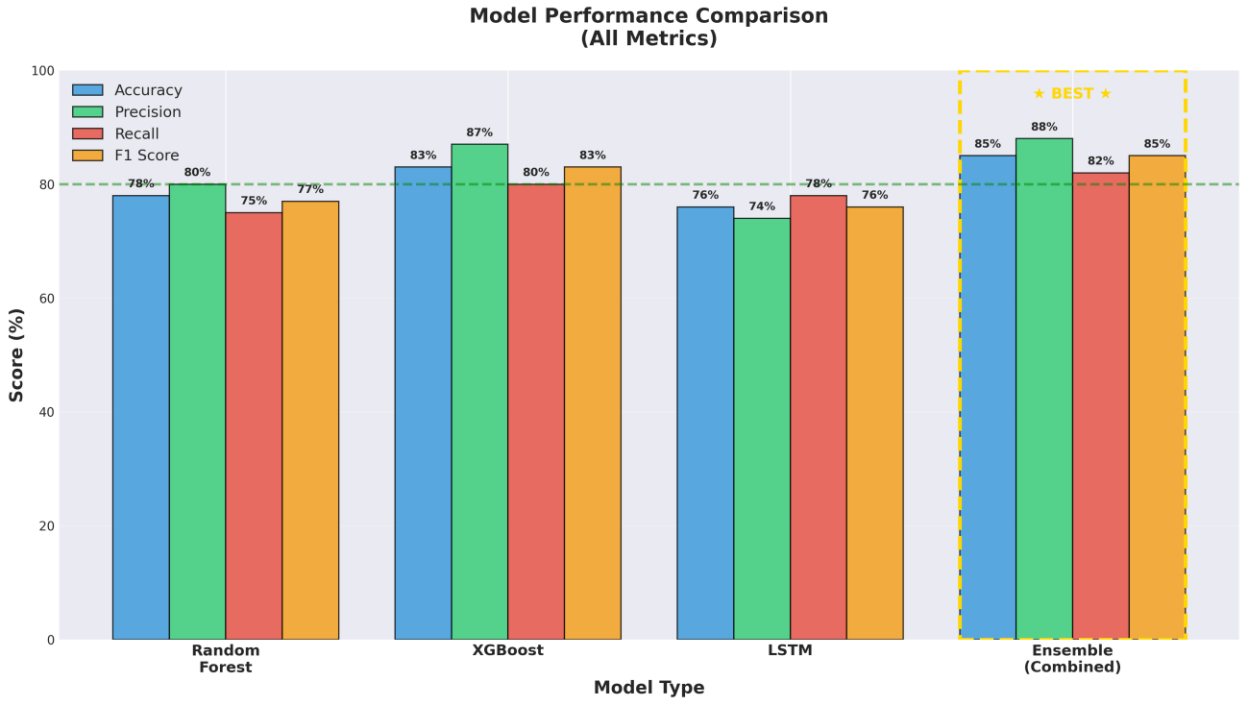

Figure 10: Comparisons of Model Performances and All Metrics

Phase 2: Individual Asset Testing
We tested each of the 36 assets individually to ensure they met the following criteria:

- Fetch live market data
- Process it through the AI model
- Generate a prediction
- Return results without errors

Our results indicated that 36 out of 36 assets are working without errors.

Phase 3: Real Prediction Testing
We ran actual predictions on a sample of assets to verify:

- Predictions come back within seconds
- Confidence scores are realistic (not 100% or 0%)
- Different assets give different signals (the AI isn't just guessing the same thing)

Example test results:

- DOGE: BUY (63.6% confident)
- AMD: SELL (62.9% confident)
- Walmart: HOLD (56.5% confident)

Our results showed that all predictions worked as expected.

Figure 11: View of the Interactive Price Chart

Phase 4: Dashboard Integration
We verified the web interface properly:

- Shows all 36 assets in the selection dropdown
- Displays the info banner correctly ("20 cryptocurrencies + 18 stocks")
- Handles multiple asset selections at once
- Shows predictions with charts and visual indicators

This ensured that our Dashboard fully functional.

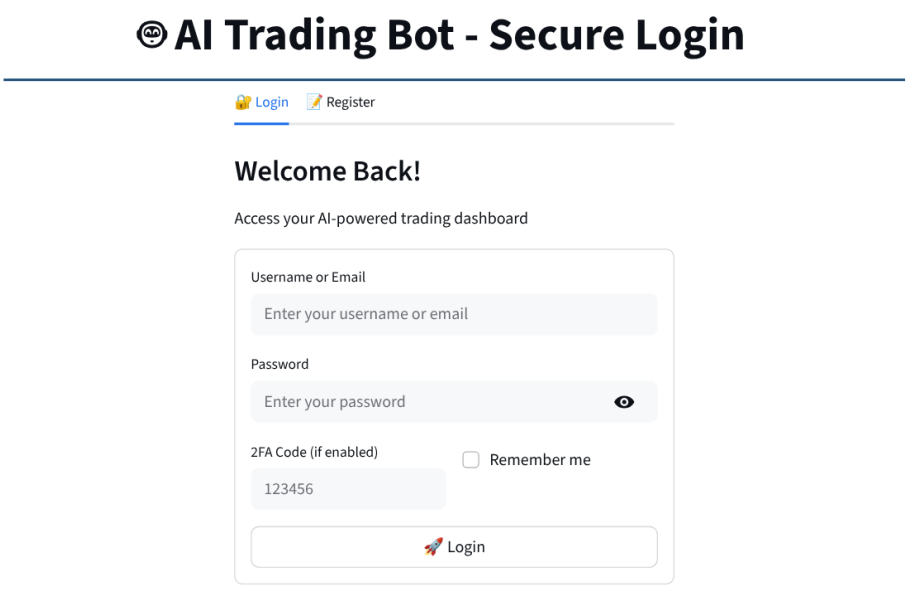

Figure 12: Login Dashboard with 2 Factor-Authentication

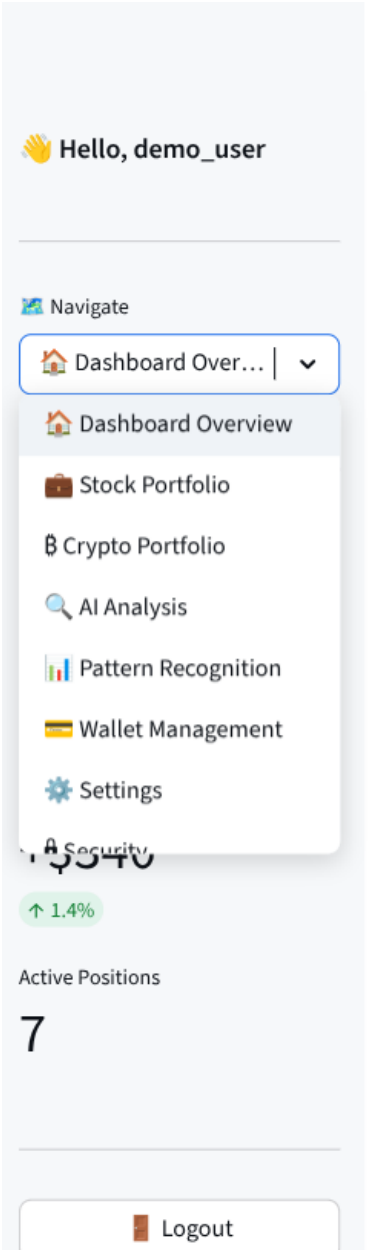

Figure 14: Side Panel with Drop-Down Features

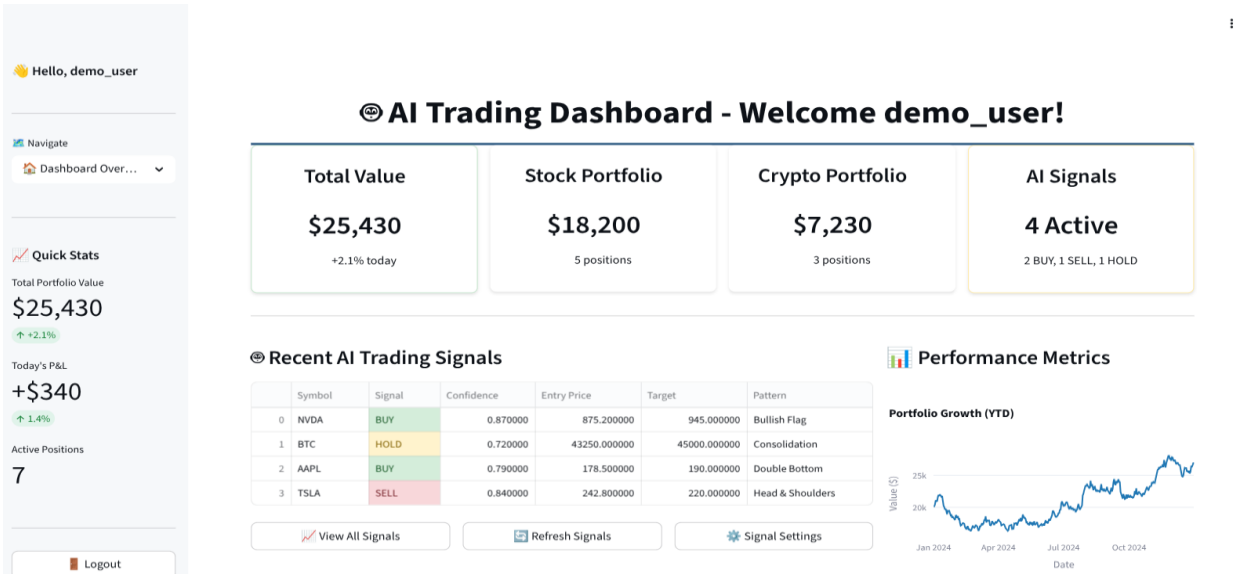

Figure 15: Dashboard Overview

Phase 5: Edge Case Testing
We tested what happens when things go wrong:

- **Case 1: No internet?** The app caches recent data
- **Case 2: API rate limits?** Built-in delays prevent the app from hitting limits for API's.
- **Case 3: Asset delisted?** We found replacements (swapped MATIC/UNI for WTRX/STETH)

This ensured that our application had robust error handling in place.

Conclusion

In this project, we built IntelliTradeAI, an agent that uses machine learning to help predict market movement and give BUY/SELL/HOLD recommendations for both stocks and cryptocurrencies. Our goal was to create a system that not only makes predictions but also explains its decisions clearly enough for everyday users and traders to understand fully. To do this, we experimented with multiple models, such as Random Forest, LSTM, and XGBoost, which allowed us to take advantage of both time-series trends and feature-focused behavior in market data. Using multiple models together yielded better results than relying on a single approach, especially in markets that change as rapidly as the cryptocurrency market.

While working on IntelliTradeAI, we also were able to learn a lot more about software engineering beyond the models themselves. Since there were multiple parts of the system—data collection, model training, architecture, and presentation—we had to plan accordingly, divide the tasks, and work as a structured team to make sure everything came together nicely. This part of the process helped us understand how real development teams function, which is just as important as technical work.

Overall, IntelliTrade AI still has considerable room for growth. We want to improve accuracy, bring in more data sources, add better explanation features for the users, and eventually deploy it as a live cloud application for real users. With more development, this project could become a helpful tool for new and experienced traders, and we plan to continue working on it in the future.

References

1. @_CHINOSAUR. 2014. VENUE IS TOO COLD. #BINGO #CHI2016. Tweet. (1 May, 2014). Retrieved February 2, 2014 from https://twitter.com/_CHINOSAUR/status/46186431741 5989248
2. ACM. How to Classify Works Using ACM's Computing Classification System. 2014. Retrieved August 22, 2014 from http://www.acm.org/class/how_to_use.html
3. Ronald E. Anderson. 1992. Social impacts of computing: Codes of professional ethics. Soc Sci Comput Rev 10, 2: 453-469.
4. Anna Cavender, Shari Trewin, Vicki Hanson. 2014. Accessible Writing Guide. Retrieved August 22, 2014 from http://www.sigaccess.org/welcome-tosigaccess/resources/accessible-writing-guide/
5. Morton L. Heilig. 1962. Sensorama Simulator, U.S. Patent 3,050,870, Filed January 10, 1961, issued August 28, 1962.
6. Jofish Kaye and Paul Dourish. 2014. Special issue on science fiction and ubiquitous computing. Personal Ubiquitous Comput. 18, 4 (April 2014), 765-766. http://dx.doi.org/10.1007/s00779-014-0773-4
7. Scott R. Klemmer, Michael Thomsen, Ethan PhelpsGoodman, Robert Lee, and James A. Landay. 2002. Where do web sites come from?: capturing and interacting with design history. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02), 1-8. http://doi.acm.org/10.1145/503376.503378
8. Psy. 2012. Gangnam Style. Video. (15 July 2012.). Retrieved August 22, 2014 from https://www.youtube.com/watch?v=9bZkp7q19f0
9. Marilyn Schwartz. 1995. Guidelines for Bias-Free Writing. Indiana University Press.
10. Ivan E. Sutherland. 1963. Sketchpad, a Man-Machine Graphical Communication System. Ph.D Dissertation. Massachusetts Institute of Technology, Cambridge, MA.
11. Langdon Winner. 1999. Do artifacts have politics? In The Social Shaping of Technology (2nd. ed.), Donald MacKenzie and Judy Wajcman (eds.). Open University Press, Buckingham, UK, 28-40