```python
from flask import Flask, render_template
from flask_socketio import SocketIO
import numpy as np
import pandas as pd
import yfinance as yf
import time
import threading
import requests
from statsmodels.tsa.arima.model import ARIMA
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from fbprophet import Prophet
import xgboost as xgb

app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins="*")

SYMBOLS = ["AAPL", "TSLA", "MSFT"]

def fetch_realtime_data(symbol):
    df = yf.download(symbol, period="7d", interval="1h")
    return df[['Close']].dropna()

def train_arima(symbol):
    df = fetch_realtime_data(symbol)
    model = ARIMA(df['Close'], order=(5,1,0))
    model_fit = model.fit()
    forecast = model_fit.forecast(steps=5)
    return forecast.values[-1]

def train_lstm(symbol):
    df = fetch_realtime_data(symbol)
    data = df['Close'].values.reshape(-1,1)

    model = Sequential([
        LSTM(50, activation='relu', return_sequences=True, input_shape=(10, 1)),
        LSTM(50, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')

    x_train = np.random.rand(100, 10, 1)
    y_train = np.random.rand(100, 1)
    model.fit(x_train, y_train, epochs=5, verbose=0)

    prediction = model.predict(data[-10:].reshape(1,10,1))
    return prediction[0][0]

def train_prophet(symbol):
    df = fetch_realtime_data(symbol)
    df.reset_index(inplace=True)
    df = df.rename(columns={"Date": "ds", "Close": "y"})
    model = Prophet()
    model.fit(df)
    future = model.make_future_dataframe(periods=5)
    forecast = model.predict(future)
    return forecast['yhat'].iloc[-1]

def train_xgboost(symbol):
    df = fetch_realtime_data(symbol)
    X = np.arange(len(df)).reshape(-1,1)
    y = df['Close'].values
    model = xgb.XGBRegressor(objective='reg:squarederror')
    model.fit(X, y)
    future = np.array([[len(df) + 5]])
    return model.predict(future)[0]
```

```python
def analyze_news(symbol):
    # Örnek olarak sabit haber etkisi
    news_sentiment = {"AAPL": 0.8, "TSLA": -0.5, "MSFT": 0.4}
    return news_sentiment.get(symbol, 0)

def calculate_volatility(symbol):
    df = fetch_realtime_data(symbol)
    return df['Close'].pct_change().std()

def generate_trade_signal(symbol):
    arima_pred = train_arima(symbol)
    lstm_pred = train_lstm(symbol)
    prophet_pred = train_prophet(symbol)
    xgb_pred = train_xgboost(symbol)
    news_score = analyze_news(symbol)
    volatility = calculate_volatility(symbol)

    current_price = fetch_realtime_data(symbol).iloc[-1, 0]

    combined_pred = (arima_pred + lstm_pred + prophet_pred + xgb_pred) / 4 + (news_score * 0.05)

    signal = "AL" if combined_pred > current_price else "SAT"

    return {
        "symbol": symbol,
        "current_price": current_price,
        "arima_pred": arima_pred,
        "lstm_pred": lstm_pred,
        "prophet_pred": prophet_pred,
        "xgb_pred": xgb_pred,
        "combined_pred": combined_pred,
        "news_score": news_score,
        "volatility": volatility,
        "signal": signal
    }

def send_live_data():
    while True:
        trade_data = [generate_trade_signal(symbol) for symbol in SYMBOLS]
        socketio.emit('update_data', trade_data)
        time.sleep(10)

@app.route('/')
def index():
    return render_template("index.html")

if __name__ == "__main__":
    threading.Thread(target=send_live_data, daemon=True).start()
    socketio.run(app, debug=True, host='0.0.0.0', port=5000)
```