

1. Veri Kaynakları ve Özellik Mühendisliği

1.1. Mevcut alternative_data.py'yi Zenginleştirme

Şu anki hali:

Google Trends (EURUSD, USDJPY, GBPUSD → her 15 dakikada güncelleniyor)

Twitter Sentiment ("forex" hashtag'ı üzerinden global İngilizce tweet'lerin compound skoru)

Geliştirme önerileri:

Çok dilli ve çok platformlu duygu analizi:

Sadece "forex" etiketi yerine "#forextrading", "#EURUSD" gibi spesifik etiketleri de takip et. Türkçe, İspanyolca, Arapça gibi farklı dillerdeki tweet'lere de bak. (Tweepy sorgularını lang:tr, lang:es, lang:ar şeklinde genişletebilirsin.)

Reddit, StockTwits, Investing.com gibi platformlardan da veri al.

Örneğin Reddit'in r/forex ya da r/algorithmictrading toplulukları, "pushshift.io" API'si üzerinden toplanabilir. Bu sayede tek platforma bağlı kalmamış olursun.

Konu Tabanlı Duygu Filtrelemesi (Topic Modeling):

Vader yerine, daha ince ayrıştırma yapan bir yöntem kullan. Örneğin Türkçe Reddit kirpiklerinden gelen veriye LSTM tabanlı bir "duygu sınıflandırıcı" eğiterek, "boğa haberleri mi, ayı mı" kategorisini net çıkar.

Geliştirdiğin model: TextCNN ya da DistilBERT (küçük bir transformer) olabilir. Training datası için geçmiş tweet/değerlendirmeleri elle etiketleyebilirsin. Böylece sadece compound skor değil, "boğa (bullish)", "ayı (bearish)", "nötr (neutral)" gibi üçlü bir çıktı alırsın.

Haber Kaynağı Çeşitliliği:

Şu an sadece NewsAPI üzerinden alıyorsun. Bu API'nin sınırlamaları (küçük haber başlıkları, gecikmeler) var. Bunun yanı sıra:

ForexFactory ve Investing.com Economic Calendar (REST / kebab) entegrasyonu ekle. Yüksek etkili haberleri (ör. NFP, Fed Kararı, CPI) "yakala".

"Panic keywords" listesini genişlet: Sadece "crash, dump, volatility" değil, "Fed açıklaması, faiz artırımını, varlık alımı, varlık satımı" gibi terimleri de ara.

Haber başlıklarına ek olarak RSS beslemeler (Bloomberg, Reuters) parçalayabilirsin.

On-Chain ve Akıllı Para İşaretçileri (Smart Money Signals):

Özellikle kripto paralar yerine forex odaklısın ama "kur paritelerindeki büyük banka pozisyonları"nı gösteren ücretsiz/ücretli API'ler var. Örneğin CFTC'ın haftalık "Commitment of Traders" raporunu (COT data) çek.

OTC bankalar veya hedge fonlarının "net long/net short" pozisyonlarını Google Sheets'ten çekip botun kararlarına ekle.

Derinlik Verisi (Order Book / DOM):

Mümkünse MT5'ten "Market Depth" (Level II) verisi al. Bir paritede anormal biriken emirler (büyük lot ask/bid duvarları) varsa, bu "manipülasyon" veya "yaklaşan volatilité" sinyali olabilir.

Eğer VPS/MT5 ile bağlantın bu veriyi veriyorsa, alt bantta bu order book verisini de normalize ederek ("bid ask ratio" gibi) state vektörüne ekleyebilirsin.

1.2. Çok Zamanlı ve Çok Ölçütlü Özellikler

Şu anki hali:

M1 (1 dakikalık), H1 (1 saatlik) ve H4 (4 saatlik) veri çekiliyor. LSTM H1 bazlı "bullish/bearish" rejim tespiti.

Geliştirme önerileri:

Daha İnce Zaman Aralıkları (Tick Data / 5S / 15S):

FTMO'da ultrakısa zaman dilimlerinde (örneğin 5 saniye) tarama yapmak, scalping fırsatlarını yakalamak için önemli. Mümkünse "MT5.copy_ticks_from" fonksiyonunu kullanarak son X saniyenin tick'lerini topla. Onlardan "son 5S'deki volatitiy spike" gibi ek filtreler oluştur.

Multi-Timeframe Özellik Birleştirme (MTF data fusion):

Sadece H1'den LSTM rejim tespiti değil, H4'ten trend açısı (örneğin H4'deki RSI, H4 MACD divergence) çıkar.

M1'de karar verirken:

M1'deki kısa vadeli momentum (RSI(7), CCI(14), Momentum(5)).

H1'deki orta vadeli trend (H1 RSI(14), H1 MACD histogram).

H4'teki uzun vadeli trend (örneğin H4 EMA(50) vs EMA(200) kesişmesi).

Bu üçünü normalize ederek (0-1 aralığına) state'e ekle. Böylece DQN, tek timeframe yerine "temel çerçeveyi" de bilir.

Teknik / Temel Kombinasyonu:

Paritenin işlem gördüğü ülke ekonomileriyle (örneğin EURUSD için Euro Bölgesi ve ABD CPI, İşsizlik, PMI) arasında regresyon analizi yap. Eğer fundamental veriler "uyumsuz" (yani fiyat hareketi sınırlı veya ters) ise DQN aşırı açılmaya teşebbüs etmesin.

Bunun için "pandas_datareader" yerine "yfinance" veya "fredapi" kullanabilirsin.

2. Model İyileştirmeleri ve Öğrenme Algoritmaları

2.1. DQN'den Rainbow'a: Deneyim Yeniden Oynama ve Gelişmiş RL

Şu anki hali:

Double DQN + Dueling yapısı var. Replay buffer (öncelikli değil), ϵ -greedy keşif.

Geliştirme önerileri:

Öncelikli Deneyim Yeniden Oynama (Prioritized Experience Replay):

ReplayBuffer'ı sadece FIFO queue olarak kullanmak yerine, önemli anları (yüksek TD hatası) daha sık örnekle. Böylece öğrenme hızlanır.

[Schaul vd., 2015] makalesindeki "Proportional Prioritization" algoritmasını Python tarafında entegre edebilirsin.

Distribütöyönel RL (Categorical DQN veya QR-DQN):

Q değerlerini tek bir sayı yerine bir dağılım olarak modellemek, risk duyarlılığını arttırır. "Mükemmel bir kazanç" yerine "en yüksek %5'lik kayıp riski"ni minimize edebilirsin.

TensorFlow'da bir "categorical DQN" mimarisini (51 atomlu C51) kurarak, DQN'in yerine geçirebilirsin.

Multi-step Returns (n-step target):

Şu anki DQN, bir adım sonrası getiriye bakıyor. 3-5 adımlık $G_t = r_t + \gamma V_{t+1} + \dots$ şeklinde daha uzun horizon hedefler, sinyal ağacını stabil hale getirir.

ReplayBuffer içine "state, action, reward, next_state, next_next_state, ..." olarak ekleyip, n_step ödülleri hesaplanabilir.

Noisy Nets ve Parameter Noise:

Epsilon-greedy yerine, ağ yapısına "Noisy Networks" katmanı

ekleyerek (örneğin `tfp.layers.NoisyDense` veya kendi “NoisyLinear” implementasyonu) keşfi parametrik hale getir. Bu sayede, her epizotta epsilon ayarıyla uğraşmak yerine, ağa “hangi ağırlıklar rastgeleleştirildi” diyebilirsiniz.

Rainbow DQN = Dueling + Double + Prioritized + Multi-step + Noisy + Distribütöyönel:

Tüm bu bileşenleri tek bir “RainbowAgent” sınıfı altında toplamak, teoride “en iyi DQN” performansını getirir. Kütüphanelerde (örneğin “tensorflow_agents” veya “stable_baselines3”) örnek altyapılar var; kendi özel ayarlarına uyarlayarak daha hızlı devreye alabilirsiniz.

Alternatif: Proximal Policy Optimization (PPO) veya A2C/A3C:

DQN tabanlı değil, politika tabanlı (actor-critic) yaklaşımlara geçebilirsiniz. PPO, işlem ortamının sürekli değişken dinamiklerine daha hızlı adapte olabilir. Özellikle “continuous action space” yerine “al/boş/sat” üçlü discrete action olsa da yine de PPO’yu uygulamak mümkün. “Stable Baselines3” içinden PPO’yu çağırıp, env alt yüzeyini hazırla.

Örneğin her M1 mum kapanışında bir adım at, ödül olarak “profit - risk_penalty” ver. PPO ile daha keskin (stabil) öğrenme elde edebilirsiniz.

2.2. Rejim Tespiti: LSTM’den Üst Seviyeye

Şu anki hali:

H1 kapanış verileriyle 16 nöronlu basit bir LSTM. “Return > 0” olarak etiketleme.

Geliştirme önerileri:

Daha Geniş Girdi Dizisi ve Çoklu Özellik:

Sadece “close” yerine H1’de [‘open’, ‘high’, ‘low’, ‘close’, ‘volume’] kullan. Bunları MinMaxScaler ile normalize et. “Close Returns” modeline ek olarak “High-Low Range” (volatilite bilgisi), “RSI” vb. ekle. Bu şekilde LSTM, sadece fiyat yönünü değil, momentum ve volatilite değişkenlerini de öğrenir.

Transformer Tabanlı Zaman Serisi Modeli:

LSTM’nin yerine “Informers” veya “Time Series Transformer” (örn. “TFT – Temporal Fusion Transformer”) kullan. Bu modeller, uzun vadeli bağımlılıkları daha hızlı kavrar. H1’deki 5000 saat yerine 20000 saatlik veriyi Transformer’a yedir, token-length 100–200 aralığında kaydırmalı pencere (sliding window) kullan.

Regresyon/Eğilim (Trend) Sınıflandırmasından Daha Fazlası:

“Bullish/Bearish” yerine, üç sınıf kullan: “Güçlü boğa (strong bullish) / zayıf boğa / nötr / zayıf ayı / güçlü ayı”. Bu sayede daha incelikli karar mekanizmaları oluşturabilirsiniz.

Online Learning ve Periyodik Yeniden Eğitim:

Eğitilmiş H1 modeli her hafta/ay otomatik olarak yeniden eğitilsin (cron job veya bot başlatılırken).

Eski ağırlıkları “fine-tune” ederek yeni güncel veriye adapte ettir (“warm start”).

Sinyal Çatışması Açıklığı için Güven Skoru (Confidence Score):

LSTM/Transformer çıktısının sigmoid veya softmax skorunu al. Eğer “regime_confidence < 0.6” ise “nötr” de; aksi durumda tam “bullish/bearish” olarak tespit et. Bu güven skoru, DQN’e state olarak eklenebilir.

2.3. Enchilada: Ensemble (Topluluk) Modelleri

Neden?

Bir tek modelin yanılma riski vardır. Ensemble ile hata olasılığını

azaltır, daha genelleyici sonuç elde edersin.

Nasıl?

Farklı RL Algoritmalarının Odağı:

Bir tarafta Rainbow DQN, diğer tarafta PPO, bir diğer tarafta A2C. Her biri ayrı agent olarak paralel eğitilsin (aynı offline veride backtest).

Eğitim sonunda “meta-learner” (örneğin küçük bir MLP) hangi agent’ın o anki state’de (özellik vektörüne bakarak) daha güvenilir olduğunu tahmin etsin. Sonra o agent’ın eylemi (buy/hold/sell) dominansın olsun.

Majority Voting / Weighted Voting:

Her agent’ın Q veya policy çıktısı bir skor olarak alınıp, en yüksek ortalama skor kararına göre trade açılsın.

Uzun vadede hangi agent hata yapıyorsa, performansına göre ağırlığı azaltılsın.

3. Risk ve Portföy Yönetimi

3.1. FTM0 Kurallarını Kesin Sağlama

Şu anki hali:

Günlük %5, toplam %10 limit var, aynı anda 1 pozisyon (hedge yasak).

Geliştirme önerileri:

Dinamik Maksimum Pozisyon Sayısı:

FTM0’da “aynı anda en fazla 1 pozisyon” kuralı var ama “farklı semboller” için bazen loophole bulunabiliyor. Botu, “hesap sabit + floating PnL” takibini yapacak şekilde güncelle. Yani canlıda `mt5.account_info().equity + mt5.positions_total()` verilerini okuyan periyodik fonksiyon getir.

Gerçek Zamanlı Equity/Drawdown Takibi:

Şu an risk manager’e sadece backtest sonuçlarıyla geliyor bilgi. Canlıda, `mt5.account_info().balance`, `mt5.account_info().equity`’den anlık çekerek, “floating drawdown” (% olarak) kontrol et. Eğer yıllık “max daily drawdown” %5’i uzaktan aşacak gibi görünüyorsa (örneğin floating zarar > x), pozisyonları otomatik kapat.

VaR (Value at Risk) Hesaplama:

Her gün M1 kapanışında, portföyün “1 günlük %95 güven aralığındaki potansiyel en kötü kaybını” (Parametrik VaR veya Monte Carlo) hesapla. Eğer VaR %2.5’i aşıyorsa, o gün yeni pozisyon açma.

Basitçe `portfolio_return_mean` ve `portfolio_return_std` bulup $VaR = mean + 1.65 * std$ formülüyle bir eşik koy.

Pozisyon Büyüklüğünü Tasfiye Olasılığına (Probability of Ruin) Göre Ayarla:

Basit risk yönetimi: her işlem maksimum “hesap büyüklüğünün %1’i” kadar riske izin verse de, PoR (Probability of Ruin) teorisine göre puanı hesapla. Eğer artan volatilité, ardışık küçük kayıplar PoR’u “≥%5” eşikine çıkarırsa, lotu aşağı çek veya gün ortasında otomatik “hedge” yap.

Paralel Portföy Yönetimi (Asset Allocation):

Şu an üç ayrı sembol, ayrı ayrı risk hesaplaması var. Bunun yerine “global stop loss” da tanımla:

Hesap büyüklüğünün %3’ü kadar toplam floating zarar varsa, tüm pozisyonları kapat.

Her sembol için risk katsayısını dinamik ayarla: Örneğin EURUSD’e %40, USDJPY’ye %30, GBPUSD’ye %30 tahsis et. Böylece “bir sembolde zarar diğerini etkilemesin” diye.

Otomatik Sosyal Lokavt (Social Lockout):

Demo/gerçek geçişinde, FTM0 mücadelesi sırasında, insan beklenen bir volatilité hatasında müdahale edebilsin. Bot, kritik saatlerde (örneğin Fed toplantısı anında, BBC’de flaş haber çıktığında) hiçbir pozisyon açmasın (Zaman Çerçevesi: haber saatinin ±15 dakika). Bunu “economic calendar” entegrasyonu ile yapabilirsin.

3.2. Dinamik Lot ve Risk Katmanlama

Şu anki hali:

ATR’e göre lot mult. (yüksek vol → lot×0.5, düşük vol → lot×1.5)

Geliştirme önerileri:

Kelly Kriteri Temelli Lot Hesabı:

“Her işlemde beklenen başarı oranını” LSTM rejim modeli + backtest sonuçlarına göre hesapla. Diyelim “şu anda bullish rejim, başarısı %60”; Kelly formülünü uygula:

f

*

=

W

R

(W = speech of winning, R = risk/reward)

f

*

=

R

W

(W = speech of winning, R = risk/reward)

Örneğin kazanma olasılığı %60, risk/ödül 1:2 ise,

f

*

=

0.6

-

0.4

/

2

=

0.4

f

*

=0.6-0.4/2=0.4. Bu da “hesap sermayesinin %40’ı” anlamına gelmez, çünkü forex pip olarak hesaplanır. Ama “lot size = account_balance × f^* / fiyat” mantığıyla dinamik ayar yap.

Pozisyon Merdivenleme (Scaling In/Out):

İlk sinyalde yarım lot, fiyat senin lehine hareket etmeye devam ederse kalan yarım lot ile tekrar girecek şekilde bir “ladder entry” yap.

TP’ye yaklaşıncı kademeli kâr al (örneğin %50 lot kapanınca, kalan %50 için trailing stop geç).

Kar Kilitleme (Profit Lock-in):

Pozisyon açıldıktan sonra +X pip kar elde edildiyse (örneğin +10 pip), otomatik olarak SL’i entry price’a çek. Yani kayıp riskini sıfırla. Sonra +20 pip olunca SL’i “+5 pip” e taşı. Bu mantığı “sl_multiplier” ve “tp_multiplier” ile güncelleyebilirsin.

Kayıbı Hızla Azaltma (Loss Guard Geliştirmeleri):

3 zararlı işlem → 1 saat “sleep” yerine, “zamanla azalan bekleme” (adaptive dormancy): Eğer art arda zararlı işlemler artıyorsa, bir sonraki işlem için bekleme → 30 dk, 15 dk, 5 dk şeklinde düşer. Eğer “toplam floating drawdown > hesap bakiyesinin %2’si” ise, otomatik “hedge” sinyali çıkar. Yani “aynı paritede ters pozisyon aç” (kısmen veya tam).

Çok Katmanlı Risk Puanı (Composite Risk Score):

ATR spike, yüksek korelasyon, LSTM güven skoru, sosyal medya paniği, haber volatilité endeksi gibi unsurları 0–1 aralığında topladığın alt risk skorlarıyla “composite risk score” oluştur.

Eğer bu skor “>0.7” ise, pozisyon açma. (Benim anlamımda 0 = risksiz, 1 = aşırı riskli.)

4. Canlı İşlem Altyapısı ve Mimarisi

4.1. Gerçek Zamanlı Veri Hattı (Data Pipeline)

Şu anki hali:

“run_backtest” sırasında paralel olarak M1/H1/H4 verileri alınıyor. Canlıda her dakika “copy_rates_from_pos” çağrısı yapılıyor.

Geliştirme önerileri:

Webhook + Event-Driven Veri Alımı:

Sadece “her 60 saniyede bir” diye poll etmek yerine, MT5’nin “OnTick” olaylarına abone ol. Python tarafında mt5.copy_ticks_from ile tetiklenen bir “callback” fonksiyonu yaz. Bu sayede “anlık olarak tick verisini” işleyip, M1 mum kapanışını biraz daha gerçek zamanlı alırsın (60s yerine ~1–2s gecikmeyle).

Kafka / Redis Yayın (Pub/Sub) Katmanı:

Botunu ölçeklendirip, farklı modüllerin (veri toplama, sinyal üretme, risk yönetimi, order execution) birbirinden bağımsız çalışmasını istiyorsan, “message broker” kullan. Örneğin:

Veri Toplama Servisi → M1 tick’i geldiğinde “kuyruğa” push.

Özellik Hesaplama Servisi → Kuyruktan al → teknik göstergeleri hesapla → “state queue”a gönder.

Agent (RL) Servisi → “state queue”dan al → action belirle → “order queue”a gönder.

Order Execution Servisi → “order queue”dan al → MT5 order_send ile işlemi gerçekleştir.

Risk Manager Servisi → Her kapanan pozisyonda tetiklenir, drawdown, VaR hesaplar.

Böylece bir servis çökse bile diğerleri çalışmaya devam eder. Ayrıca ileride başka agent’lar eklemek isteyen birinin altyapıyı yeniden kurgulaması kolaylaşır.

Latensi Minimize Etmek için VPS:

FTMO challenge’da “0.01 saniye gecikme” bile kritik olabilir.

Botu mutlaka İstanbul veya Frankfurt lokasyonlu bir VPS’e taşı. MT5 Cloud VPN entegrasyonu kullanarak “en düşük ping” yoluyla broker sunucusuna bağlan.

Yüksek Erişilebilirlik (High Availability):

Botun ana süreci çöktüğünde hemen yedek bir instancia devreye girsin. Basitçe “systemd” ile Restart=on-failure ayarlayabilirsin. Bir de “ana logdosyası”nı (ftmo_trading_bot.log) haftalık olarak rotasyonlu (logrotate) tut. Böylece geçmişe dönük hata incelemesi kolaylaşsın.

4.2. İnsan-Makine Arayüzü (Human-in-the-Loop)

Neden?

“Tamamen otomatik” bir sistem her zaman yanılmaz demek değildir. Özellikle beklenmedik piyasa çöküşleri, kara swan olayları, veri kesintileri vs. için insan onayı gerekebilir.

Geliştirme önerileri:

Web Tabanlı Dashboard:

Grafana + Prometheus:

Bot metriklerini (loss, reward, epsilon, günlük PnL, güncel drawdown, open position sayısı) Prometheus’a push et. Grafana’da grafikleri canlı gözetle. İnsan piyasanın garip gittiğini görürse “manual override” yapabilir.

Flask/Django + Plotly Dash:

Eğer daha özelleştirilmiş bir şey istersen, Python tabanlı bir sonraki adım. Flask arkasında bir web sunucu, Plotly Dash ile:

Canlı eşik grafikleri: Günlük kâr/zarar eğrisi.

Open Position Listesi: Hangi sembolde, hangi lot, SL/TP, açılış fiyatı.

Trade History: Son 50 trade’in RSI, MACD, girdiği fiyat, çıktığı fiyat, GHN skorları vs.

“Manual Override” butonu: “Şu anda bot «GBPUSD» için alım yapmasın” diyebilirsin. Bu komut bir JSON içinde

custom_params['disable_GBPUSD'] = True şeklinde ana programa geçer.

Telegram Arayüzü (Command & Control):

Şu an sadece send_telegram_message var ama “/pause”, “/resume”, “/stats” komutlarını işleyen bir Telegram bot ekle.

/stats → “Bugün 5 işlem yapıldı, +120 USD kâr. En son işlem EURUSD, +10 pip.”

/pause → Botu geçici olarak durdur.

/resume → Botu kaldığı yerden devam ettir.

/override EURUSD sell → Bot şu anda EURUSD’de satma izni versin.

Bunu yapmak için “telegram.ext” içinden CommandHandler ve Updater kullanabilirsin.

E-mail ve SMS Uyarıları (Fallback Kanal):

Telegram’a erişim kesilirse, e-posta veya SMS (Twilio) üzerinden kritik uyarılar gönder:

“Günlük zarar limiti aşıldı”

“MT5 bağlantısı kayboldu”

“Yüzde 2 floating drawdown tetikledi”

Böylece her zaman erişim kanalı olur.

5. Test, Optimizasyon ve Dağıtım Süreçleri

5.1. Overfitting’i Engellemek için İleri Düzey Test Metodları

Şu anki hali:

Training: 2015–2023, Testing: 2024.

Geliştirme önerileri:

Walk-Forward Analizi (WFA):

Örneğin:

2015–2017 ile eğit → 2018 test

2015–2018 ile eğit → 2019 test

2015–2019 ile eğit → 2020 test

... → 2024 test

Her “eğitim–test” aşamasında en iyi hiperparametreleri seç, sonra ileriiki döneme geç. Bu sayede parametreler geçmişe “aşırı uyumlu” (overfit) olmaz.

K-Fold Zaman Serisi Çapraz Doğrulama:

Normal k-fold değil, “zaman pencereli” cross-validation: her fold’da test aralığını eğitim aralığından sonra konumlandır. Örneğin 5-Year Train, 1-Year Test olarak slayt.

Out-of-Sample (OOS) ve Out-of-Time (OOT) Dönemler:

Backtest’te “en yeni veriler”i asla eğitimde kullanma (örneğin 2024 yılının son çeyreği). Sadece final demo/verification için “live-replay” modunda o dönemi kullan.

Monte Carlo Simülasyonları & Sensitivite Analizi:

Son 1000 trade’in “şu pip dağılımına göre kümülatif PnL’si” adım adım varyasyonlu simülasyon yap. Hangi pip kayıplarına, hangi korelasyon değişimlerine bot nasıl tepki veriyor?

Python’da numpy.random.multivariate_normal ile pip hareketlerini varyasyonlu çek, backtest loop’unu 1000 kez çalıştır. Her seferinde drawdown, Sharpe, Sortino gibi metrikleri kaydet. Metriklerin varyansını incellersen, gerçek hayatta “bot ne kadar istikrarlı?” sorusunun yanıtını alırsın.

Takım Arkadaşlarıyla Kod İncelemesi (Code Review):

Bot mimarisini JTTWS projesi ekibinden en az bir meslektaşına gözden geçirt. Kodun “temiz kod” (clean code) standartlarına uygun olduğuna emin ol. Özellikle multithreaded/paralel yapıda “race condition” olmayacak şekilde dikkat et.

5.2. Hiperparametre Optimizasyonu

Şu anki hali:

custom_params içinde birkaç parametre el ile verilebiliyor, Genetik koda hazır bir “iskelet” var.

Geliştirme önerileri:

Optuna / Hyperopt Kütüphanesi Kullanarak Boost

Genetik yerine Optuna’nın TPE (Tree-structured Parzen Estimators) algoritmasını kullan.

objective(trial) fonksiyonunda:

```
rsi_low = trial.suggest_int('rsi_low', 10, 50)
rsi_high = trial.suggest_int('rsi_high', 50, 90)
atr_sl_mult = trial.suggest_float('atr_sl_mult', 1.0, 3.0)
atr_tp_mult = trial.suggest_float('atr_tp_mult', 2.0, 6.0)
learning_rate = trial.suggest_loguniform('learning_rate', 1e-4, 1e-2)
epsilon_decay = trial.suggest_uniform('epsilon_decay', 0.95, 0.999)
# vs...
```

```
return backtest_sharpe_ratio(...)
```

50–100 deneme sonunda en iyi parametreleri best_params =

study.best_params ile alıp, final testte kullan.

Multi-Objective Optimizasyon (Sharpe + Max Drawdown):

Sadece kâr değil, Sharpe’ı (veya Sortino) optimize et. Aynı anda “maksimum drawdown”’ı minimum tut. Optuna, study = optuna.create_study(directions=["maximize", "minimize"]) ile çok amaçlı arama yapabilir.

Bayesyen Optimizasyon (Gaussian Process):

Eğer Optuna istemezsen, skopt (scikit-optimize) kullanarak GP tabanlı arama yap. Özellikle parametrelerin sürekli ve dar bir aralıkta olduğu durumlarda GP daha hızlı yakınsama sağlar.

5.3. Sürekli Entegrasyon ve Dağıtım (CI/CD)

Neden?

Kod güncellemeleri sırasında “bir özellik bozuldu, canlıda takıldı”

gibi riskleri en aza indirmek için.

Geliştirme önerileri:

GitHub Actions ile Otomatik Testler:

Kod değiştiğinde:

Unit test: "calculate_indicators"'ın doğru çıktısı mı? "RiskManager" limitleri aşıyor mu? "generate_signal" edge-case'lerde None mu döndürüyor?

Linting/Formatting: flake8 ve black ile kod kalitesi.

Başarılı olduğunda main branch'e merge et.

Dockerize Etme:

Python versiyonu, bağımlılıkları (ta-lib, TensorFlow, vs) her makinede aynı olsun. Dockerfile örneği:

```
FROM python:3.9-slim
```

```
RUN apt-get update && \
```

```
    apt-get install -y libta-lib0 libta-lib0-dev build-essential
```

```
WORKDIR /app
```

```
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
COPY . .
```

```
CMD ["python", "ftmo_trading_bot.py"]
```

Böylece VPS'te docker-compose ile "indir → ayağa kaldır" basitliği sağlanır.

Versiyon Takibi + Model Kayıt (Model Registry):

Eğitilen her DQN/PP0 modeli ve LSTM/Transformer rejim modeli

"models/" klasörüne "datetime_model_adı.h5" gibi versiyonlu kaydedilsin.

Hangi versiyonun FTMO sınavında en iyi performans verdiğini geri izleyebilmek için bir küçük models/registry.json dosyası tut:

```
{
  "2025-06-01_18:00_ftmo_dqn": {
    "sharp_ratio": 1.25,
    "max_drawdown": 3.8,
    "notes": "En iyi test sonucu."
  }
}
```

6. İnsan ve Yapay Zekâ İşbirliği: Son Aşama

6.1. İnsan Karar Düğmeleri (Human Override)

Neden?

Her algoritma "büyük siyah kuğu" (black swan) olayını öngöremeyebilir. İnsan uzmanlığının kritik noktada devreye girmesi, zararları asgariye indirir.

Geliştirme önerileri:

"Manual Pause" ve "Manual Resume" Butonları:

Web arayüzde gerçek zamanlı "Pause" / "Resume" görebileceğin bir switch ekle. Buna basıldığında ana döngü bir "global flag" (örneğin is_paused=True) ile kontrol edilerek hiç işlem açılmasın.

Önemli Haber Anında Bildirim + Onay:

Örneğin Fed kararının yayınlanmasına 10 dakika kala Telegram'dan "Fed toplantısı 10 dk sonra. Bot pozisyon kapatsın mı? (E/H)" sorusu gönder. İnsan "E" derse tüm pozisyonları kapatsın.

Bunu yapmak için, NewsAPI / ForexFactory takvimi parse eden ek bir coroutine yaz:

```
async def economic_event_watcher():
```

```
while True:
    now = datetime.utcnow()
    events = newsapi.get_everything(q="FOMC meeting",
from_param=now.strftime("%Y-%m-%dT%H:%M:%SZ"), to=...)
    # vs. ya da Investing.com calendar parse
    if büyük_etkinlik_varsa:
        asyncio.run(send_telegram_message("Fed toplantısı 10 dk
sonra. Onaylıyor musunuz?"))
        # Bekle 5 dakika insan cevabını
        await asyncio.sleep(300)
```

6.2. Sürekli Öğrenme ve İnsan Geri Bildirimi (Reinforcement Learning + Human Feedback, RLHF)

Neden?

Bot canlıda işlem yaptıkça “insan herhangi bir yerde hata gördü mü?” sorusunu sormadan öğrenmeye devam edemez. İnsan yorum veya “Bu sinyal hatalıydı” gibi doğrudan geri bildirimler, modelin self-correction (öz-düzeltilme) yeteneğini artırır.

Geliştirme önerileri:

“Feedback Loop” Arayüzü:

Web dashboard’da her kapanan işlemten sonra “Bu karar doğru muydu? (👍 / 👎)” sorusu sorulsun.

İnsan “👎” yaparsa, o trade “negative_example” olarak kaydolup, bir “HumanCritic” sınıfına eklenir. Belki “örneğin, RSI çok düşük, news negatifken hala almışsın” gibi kısa not alan bir alan da ekleyebilirsin.

RLHF (Reinforcement Learning from Human Feedback):

Her gün toplanan “olumlu” ve “olumsuz” örnekleri bir sınıflandırıcı (örn. küçük bir MLP) eğitir: “Yanlış karar” / “Doğru karar”.

Bir sonraki LSTM/Rejim modeline bu çıktıyı bir “advisor feature” olarak ekle: “İnsan bu durumu %80 hata olarak işaretledi”.

DQN’de, eğer “human_feedback < 0.5” (yani insan beğenmedi) ise, reward fonksiyonunu -X yap (ceza). Bu sayede RL ajanı, insanların hoşuna gitmeyen kararları daha az verecek şekilde kendini optimize eder.

Yarı-Atomatik (Semi-Auto) Mod:

Bot “Öneri” modunda çalışsın: Her sinyalde Telegram’a “EURUSD için AL sinyali. Onaylıyor musunuz? (E/H)” mesajı gitsin. İnsan onaya basarsa bot (“al” komutuyla) açsın. Bu şekilde insan, hem sinyal mantığını izler hem de canlıda müdahale eder.

Özellikle ampirik olarak “robot sinyali X sinyalini gönderdi; insan onayladı ve profit +5 pip” gibi günlük raporlar toplanır. Böylece “insan neden onayladı?” sorusunun cevabı veritabanına kaydedilmiş olur.

7. Özet ve Bir Sonraki Adımlar

Veri ve Özellik Setini Zenginleştir

Multiplatform duygu analizi, RSS beslemeleri, order-book, COT data, ek time frame’ler.

RL Algoritmasını Rainbow & Ensemble’a Yükselt

Prioritized Replay, Distribütöyönel RL, Noisy Nets, multi-agent ensemble.

Regim Tespiti (LSTM → Transformer)

Multi-feature girdi, daha fazla sınıf, ilave doğrulama, online fine-tune.

Risk Yönetimi ve Dinamik Lot

VaR, Kelly, çok katmanlı risk skoru, adaptif dormancy, otomatik hedge.

Canlı İşlem Altyapısını Geliştir (Event-Driven, Kafka, VPS)

“Websocket-like” on-demand veri alımı → latency düşürü, HA yapısına geç.

İnsan-Makine Arayüzünü Tesisi Et (Dashboard, Telegram Commands, Webhook)

Güven artar, kritik anlarda insan durdurabilir/devam ettirebilir.

Test ve Optimizasyon Katmanları Kuralım (Walk-Forward, Cross-Validation, Monte Carlo)

Overfitting’i en aza indir, hiperparametreleri Optuna aracılığıyla tara.

Sürekli Entegrasyon & Dağıtım (CI/CD, Docker, GitHub Actions)

Kod kalitesi, otomatik testler, versiyon takibi, reproducibility.

İnsan Geri Bildirimi Döngüsünü (RLHF) Ekleyerek Self-Correction Sağla

Web arayüzde anlık insan dönüşünü topla, modelin ödül fonksiyonunu güncelle.

Bu adımları hayata geçirmek, elindeki botu hem teknik hem de stratejik açıdan “insan kalibrasyonu + yapay zekâ optimizasyonu” ekseninde maksimum noktaya taşıyacak.

Uygulamaya Başlarken Önerilen Öncelik Sırası

1–2 Hafta İçinde:

alternative_data.py’ye Türkçe tweet, Reddit entegrasyonu ekle.

DQN sınıfına “Prioritized Experience Replay” kat.

LSTM rejim modelini birkaç ek özelliğe göre yeniden eğit.

RiskManager’a “VaR hesaplama” modülünü tak.

2–4 Hafta İçinde:

Rainbow DQN’ı vanilla DQN’in yerine geçir.

Optuna entegrasyonunu yap, geri test için hiperparametre arayışına başla.

Dashboard’u basitçe “Flask + Plotly Dash” ile ayağa kaldır; canlı veri gelmesini sağla.

1–2 Ay İçinde:

Transformer tabanlı rejim modelini hayata geçir, önce offline test et.

Ensemble ajan yapısını kur, meta-learner’ı basit bir MLP ile implement et.

Backtest’te Walk-Forward analizini devreye al, sonuçları raporla.

3–6 Ay İçinde:

Webhook + Kafka altyapısını kur, modüler mikroservis mimarisine geçişi tamamla.

RLHF modülünü entegre et; insan geri bildirimini hem sinyal kalitesini hem de risk yönetimini optimize edecek şekilde kullan.

Tam Dockerize edilmiş, GitHub Actions CI/CD pipeline’lı, production-ready bir yapı kur.

Sonuç:

Bu planı adım adım uyguladığında, elindeki DQN+LSTM tabanlı FTMO botu;

Saniyelik tick data ve çok kanallı (multiplatform) haber/duygu entegrasyonu,

Rainbow/Ensemble RL ile artırılmış öğrenme hızı ve kararlılık,
Gelişmiş risk yönetimi (VaR, Kelly, composite risk),
Otomatik "event-driven" altyapı (low-latency) ve
İnsan geri bildirimiyle sürekli adaptasyon (RLHF)
özelliklerine kavuşacak.

Genel Bakış: Botu Parçalara Ayırılım

1. Proje Ortamı & Altyapı Kurulumu

- * Python (3.9+), sanal ortam (venv ya da conda), Git repo oluşturma

- * Gerekli kütüphaneler: MetaTrader5, pandas, numpy, TA-Lib, tensorflow, torch (opsiyonel), newsapi-python, tweepy, vs.

- * Proje klasör yapısı (örneğin /data, /src, /models, /backtests, /reports)

2. Veri Kaynakları & Özellik Mühendisliği

- * a) alternative_data.py'i zenginleştirme

- * Google Trends verilerine ek olarak: #forextrading, #EURUSD etiketleri; diller: "en", "tr", "es", "ar"...

- * Twitter'dan birden fazla hashtag ve dilde duygu analizi (Tweepy + Vader/Light transformer modelleri)

- * Reddit (r/forex, r/algotrading) → Pushshift veya praw ile çekme

- * NewsAPI + ForexFactory/Investing.com Economic Calendar entegrasyonu

- * COT (Commitment of Traders) verisi çekimi (CFTC API ya da Google Sheets → gspread)

- * MT5 üzerinden Level II (Order Book) verisi alma (mt5.market_book_get)

- * b) Zaman ve Ölçüt Bazlı Özellikler

- * M1, M5, H1, H4 tick/tatap veri fusion

- * Çoklu timeframe'd e gösterge hesaplama (örn. M1 RSI(7), H1 RSI(14), H4 EMA crossover vb.)

- * Pandas ile normalize edilmiş ölçekler (MinMax) ve birleşik state vektörü

- * Temel verilerin (CPI, PMI, işsizlik) regresyon entegrasyonu (yfinance veya fredapi)

3. Model & Öğrenme Algoritmaları

- * a) DQN → Rainbow

- * Replay Buffer'ı FIFO'dan "Prioritized Experience Replay"'e çevirme

- * Multi-step returns, Noisy Nets, Categorical DQN (C51)

- * "RainbowAgent" sınıfı: Dueling + Double + Prioritized + Multi-step + Noisy + Distribütüsyonel

- * b) Regime Tespiti

- * H1'de LSTM yerine "Temporal Fusion Transformer" ya da "Informer" kullanımı

- * Çoklu giriş özelliği: [open, high, low, close, volume, RSI, MACD histogram, CCI...]

- * "Strong Bullish / Weak Bullish / Neutral / Weak Bearish / Strong Bearish" beş sınıflı çıktı

- * Online fine-tune ve confidence threshold mekanizması

- * c) Ensemble (Çoklu Ajan)

- * Paralel olarak Rainbow DQN, PPO ve A2C ajanlarının offline

eğitimleri

- * Meta-learner (küçük bir MLP) ile “hangi ajan şu koşulda daha güvenilir?” kararı

- * Majority / Weighted voting ile nihai alım/satım/hold seçimi

4. Risk & Portföy Yönetimi (FTMO Uyumlu)

- * Dinamik Pozisyon Sayısı & Equity/Drawdown Kontrolü
 - * Günlük %5 (floating + realized) / Toplam %10 limit takibi
 - * Anlık `mt5.account_info().equity` ve `mt5.positions_total()` izleme; risk aşılrarsa otomatik pozisyon kapat

- * VaR (Value at Risk)
 - * Günlük M1 kapanışlarında portföyün 1% 1-günlük %95-ZARAR sınırı

- * VaR %2.5’i geçerse yeni pozisyon açma
- * Kelly Kriteri & Adaptive Lot
 - * LSTM/Transformer’ın öngördüğü kazanma olasılığına göre her işlem için ideal f^* hesaplama

- * Örneğin: $f^* = (W \cdot R - (1-W)) / R$
- * Pozisyon Merdivenleme & Kademeli Kâr Al
 - * İlk sinyalde %50 lot, fiyat lehine giderse kalan %50

ekleme

- * Kâr kilidi (örneğin +X pip sonra SL = entry veya entry + Y pip)

5. Canlı İşlem Altyapısı & Mimarisi

- * Event-Driven Veri Akışı
 - * MT5 OnTick benzeri callback: Python’da `mt5.copy_ticks_from` + `asyncio` ile altyapı
 - * Kafka/Redis (pub/sub) veya basit Queue mekanizması ile farklı servislerin birbirine bağlanması

- * Mikroservis Yapısı

1. Veri Toplama Servisi: Tick/M1/H1/H4 verilerini toplar, `state_queue`’ya yollar

2. Özellik Mühendisliği Servisi: Kuyruktan aldığı raw veriyi işler, MTF feature vektörü oluşturur

3. RL Agent Servisi: State vektörüne göre action (buy/sell/hold) üretir, `order_queue`ya yazar

4. Order Execution Servisi: `order_queue`dan komutları okur, `mt5.order_send` ile emirleri geçer

5. Risk Manager Servisi: Her açık pozisyonda equity/drawdown/vaR kontrolü yapar, gerekirse kapatma

- * High Availability & Latency Optimizasyonu
 - * İstanbul veya Frankfurt VPS, MT5 Cloud VPN entegrasyonu
 - * “systemd” ile Restart=on-failure, logrotate ile haftalık log arşivi

6. İnsan-Makine Arayüzü (HITL)

- * Web Dashboard (Flask + Plotly Dash veya Grafana + Prometheus)
 - * Gerçek zamanlı metrikler: günlük PnL, floating drawdown, open positions, reward eğrisi, epsilon değeri vs.

- * “Manual Override” butonları (ör. “Pause GBPUSD”, “Close ALL LONGLVES”)

- * Telegram Bot Komutları

- * `/stats` → “Bugün X işlem, +Y USD kâr, Equity = Z USD”

- * `/pause`, `/resume`, `/override SYMBOL BUY/SELL`

- * E-mail / SMS Fallback (Twilio veya SMTP)

- * “Günlük zarar limiti aşıldı”, “MT5 bağlantısı koptu”, “Yüzde X floating DD tetiklendi”

7. Test, Optimizasyon & Dağıtım

- * Walk-Forward Analizi (WFA) ve Zaman Serisi K-Fold CV
- * Monte Carlo Simülasyonları & Sensitivite Analizi (pip varyasyonları, korelasyon senaryoları)
- * Hiperparametre Optimizasyonu (Optuna veya Hyperopt)
 - * Tek amaçlı: Sharpe maks., Drawdown min.
 - * Çok amaçlı: “maximize Sharpe, minimize drawdown”
- * CI/CD & Docker
 - * GitHub Actions: Unit test (flask endpoint, gösterge hesaplama) + Linting (flake8, black)
 - * Dockerfile: Python 3.9, TA-Lib bağımlılıkları, TensorFlow, vs.
- * Model Registry: Her eğitim sonrası models/YYYYMMDD_model.h5 kaydı ve registry.json

8. İnsan Geri Bildirimi Döngüsü (RLHF)

- * Feedback UI: Her kapanan işlem sonrası “Bu karar doğru muydu? (👍/👎)” sorusu
 - * “👎” işaretlenenler human_feedback_dataset.json’a yazılır
- * HumanCritic Sınıfı: Olumlu/olumsuz örneklerden küçük bir sınıflandırıcı (MLP) eğitilir
 - * DQN reward fonksiyonuna human_feedback_score eklenir; “insan onaylamadıysa” ek ceza (-X)
- * Semi-Auto Mod: Telegram üzerinden “EURUSD AL sinyali; onaylıyor musun?”
 - * İnsan onaylarsa mt5.order_send, “hayır” derse bekle veya o sinyali iptal et

“En Baştan En Sona” Yol Haritası
Aşağıda sana tam bir yol haritası veriyorum. Her adımda kod parçacıkları, açıklamalar ve senin “ben hazırım, devam et” demeni beklediğim noktalar olacak. Böylece devamlı sormana gerek kalmadan adım adım botu tamamlayacağız.

1 Adım: Projeyi Baştan Kurmak

1.1. Sanal Ortam ve Git Repo

```
# Proje klasörünü oluştur
mkdir ftmo_bot && cd ftmo_bot
```

```
# Git repo başlat
git init
```

```
# Sanal ortam oluştur (tercihen venv)
python3 -m venv venv
source venv/bin/activate # macOS/Linux
# Windows için: venv\Scripts\activate
```

```
# Gerekli paketleri yükleyeceğimiz requirements.txt dosyası
oluşturacağız
touch requirements.txt
```

requirements.txt içine şimdilik şunları ekleyelim:

```
MetaTrader5
pandas
numpy
TA-Lib
tensorflow # ya da torch, tercih sana kalmış (DQN için TensorFlow
öneriyorum)
newsapi-python
tweepy
praw # Reddit API için
psycopg2-binary # Eğer Postgres kullanacaksan
redis # Redis client, eğer pub/sub kullanacaksan
kafka-python # Kafka client, eğer Kafka kullanacaksan
flask
plotly
dash
python-dotenv
```

Not: TA-Lib'ın derlenmiş kütüphanesini işletim sistemine göre ayrıca yüklemen gerekebilir. macOS'ta brew install ta-lib, Ubuntu'da sudo apt-get install libta-lib0 libta-lib-dev, vs.

```
# requirements.txt oluşturduktan sonra:
pip install -r requirements.txt
```

1.2. Klasör Yapısı

Projede daha sonra dosyaları düzenli tutmak için şöyle bir yapı öneriyorum:

```
ftmo_bot/
├── data/ # Ham veri (CSV, JSON, pushshift
dump, vs.)
├── src/
│   ├── alternative_data.py # Senin mevcut dosyan; burayı
zenginleştireceğiz
│   ├── feature_engineering.py # Multitimeframe, MTF fusion,
indicator hesaplamaları
│   ├── models/
│   │   ├── dqn_agent.py # Rainbow DQN vs.
│   │   ├── transformer_model.py # Regime tespiti için Transformer
│   │   └── ensemble.py # Çoklu ajan ve meta-learner
│   └── risk_manager.py # FTMO uyumlu drawdown, VaR, kayıp
limitleri
├── execution.py # MT5 order_send wrapper'ları
├── live_pipeline.py # Event-driven gerçek zamanlı veri
hattı
├── dashboard/ # Flask + Dash app kodları
├── telegram_bot.py # Bot komutları (/pause, /stats, vs.)
├── backtester.py # Geri test & WFA modülleri
├── utils.py # Ortak yardımcı fonksiyonlar (örn.
logger, date parser)
├── config.py # API anahtarları, sabitler, global
ayarlar
├── models/ # Eğitilmiş model ağırlıkları
└── (.h5, .pt)
```

```
|— logs/                # Log dosyaları (rotasyonlu)
|— reports/            # Backtest raporları, performans
|— tabloları
|— Dockerfile
|— docker-compose.yml
|— requirements.txt
|— README.md
```

– README.md: Projenin amacı, kurulum adımları, nasıl çalıştırılır, konfigürasyon örnekleri burada olsun.
– config.py:

```
import os
from dotenv import load_dotenv

load_dotenv() # .env dosyasından API anahtarlarını okur

# MetaTrader 5 ayarları
MT5_LOGIN = int(os.getenv("MT5_LOGIN", 0))
MT5_PASSWORD = os.getenv("MT5_PASSWORD", "")
MT5_SERVER = os.getenv("MT5_SERVER", "")
MT5_PATH = os.getenv("MT5_PATH", "") # MT5 terminal yolu
(özelletirilebilir)

# NewsAPI, Twitter, Reddit, CFTC API vs. anahtarları
NEWSAPI_KEY = os.getenv("NEWSAPI_KEY", "")
TWITTER_BEARER_TOKEN = os.getenv("TWITTER_BEARER_TOKEN", "")
REDDIT_CLIENT_ID = os.getenv("REDDIT_CLIENT_ID", "")
REDDIT_CLIENT_SECRET = os.getenv("REDDIT_CLIENT_SECRET", "")
REDDIT_USER_AGENT = os.getenv("REDDIT_USER_AGENT", "")

# Risk Yönetimi Ayarları
DAILY_LOSS_LIMIT_PERCENT =
float(os.getenv("DAILY_LOSS_LIMIT_PERCENT", 5)) # %5
TOTAL_LOSS_LIMIT_PERCENT =
float(os.getenv("TOTAL_LOSS_LIMIT_PERCENT", 10)) # %10
MAX_EQUITY_DRAWDOWN_PERCENT =
float(os.getenv("MAX_EQUITY_DRAWDOWN_PERCENT", 4)) # %4
```

2 Adım: alternative_data.py'yi Zenginleştirmek
Şu anki haliyle, sadece Google Trends ve tek dilli Twitter duygu analizi var. İlk olarak, o dosyayı biraz genişleteceğiz.
2.1. Twitter Duygu Analizini Çok Dilli & Çok Hashtag'li Yapmak
alternative_data.py'de Tweepy'de şu an muhtemelen şöyle bir satır vardır:

```
tweets = twitter_client.search_recent_tweets(q="#forex", lang="en",
max_results=100)
```

Bunu değiştirelim:

```
import tweepy
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

```

class MultiLingualTwitterSentiment:
    def __init__(self, bearer_token):
        self.client = tweepy.Client(bearer_token=bearer_token)
        self.analyzer = SentimentIntensityAnalyzer()
        self.hashtags = ["#forex", "#forextrading", "#EURUSD",
"#GBPUSD", "#USDJPY"]
        self.langs = ["en", "tr", "es", "ar"]

    def fetch_and_analyze(self):
        scores = []
        for tag in self.hashtags:
            for lang in self.langs:
                query = f"{tag} lang:{lang} -is:retweet"
                # son 100 tweet'i çek (miktarı ve zamanı ihtiyaca
göre ayarla)
                response =
self.client.search_recent_tweets(query=query, max_results=50)
                if response and response.data:
                    for tweet in response.data:
                        vs =
self.analyzer.polarity_scores(tweet.text)
                        scores.append(vs["compound"])
        if scores:
            return sum(scores) / len(scores)
        return 0.0

```

Not: Twitter API'sinin kısıtlamalarını unutma; bir rate limit planı kullanman gerekebilir.

2.2. Reddit ve Pushshift Entegrasyonu

Pushshift API'si üzerinden r/forex ve r/algotrading subreddit'lerinden veri çekebiliriz. Basit bir örnek:

```

import requests
import datetime

class RedditSentiment:
    def __init__(self):
        self.subreddits = ["forex", "algotrading"]
        self.base_url = "https://api.pushshift.io/reddit/search/
submission/"

    def fetch_recent_posts(self, subreddit, hours=1):
        now = int(datetime.datetime.utcnow().timestamp())
        after = now - 3600 * hours # son X saat
        params = {
            "subreddit": subreddit,
            "size": 100,
            "after": after,
            "sort": "desc"
        }
        r = requests.get(self.base_url, params=params)
        if r.status_code == 200:
            return [post["title"] + " " + post.get("selftext", "")]

```

```

for post in r.json().get("data", [])]
    return []

    def analyze_sentiment(self):
        all_scores = []
        for sub in self.subreddits:
            texts = self.fetch_recent_posts(sub, hours=2)
            for text in texts:
                vs =
SentimentIntensityAnalyzer().polarity_scores(text)
                all_scores.append(vs["compound"])
            return (sum(all_scores) / len(all_scores)) if all_scores
else 0.0

```

2.3. NewsAPI & Economic Calendar

```

from newsapi import NewsApiClient

class NewsSentiment:
    def __init__(self, api_key):
        self.client = NewsApiClient(api_key=api_key)
        self.keywords = ["forex", "USD", "EUR", "FED rate",
"interest rate"]

    def fetch_headlines(self):
        today = datetime.datetime.utcnow().date().isoformat()
        articles = self.client.get_everything(q=" OR
".join(self.keywords),
                                                from_param=today,
to=today,
                                                language="en",
                                                sort_by="relevancy",
                                                page_size=100)
        return [art["title"] + " " + art["description"] for art in
articles["articles"] if art["description"]]

    def analyze_sentiment(self):
        headlines = self.fetch_headlines()
        scores = []
        for text in headlines:
            vs = SentimentIntensityAnalyzer().polarity_scores(text)
            scores.append(vs["compound"])
        return (sum(scores)/len(scores)) if scores else 0.0

```

Ek olarak, ForexFactory/Investing.com takvimini parse etmek istersen:

- * Investing.com'un resmi API'si yok ama selenium veya requests+BeautifulSoup ile sayfayı scrape edebilirsiniz.
- * ForexFactory API'si için:

```

import requests
class EconomicCalendar:
    def __init__(self):
        self.url = "https://cdn-nfs.forexfactory.net/
ff_calendar_thisweek.json"

```

```

def fetch(self):
    r = requests.get(self.url)
    if r.status_code == 200:
        data = r.json()
        # "high impact" event'leri filtrele
        hi_events = [e for e in data["events"] if e["impact"] ==
"High"]
        return hi_events
    return []

```

* Sonra bu event'ler içinde "FED Rate Decision" veya "CPI Release" varsa bunu bir sinyal filtresi olarak kullanabilirsin.

2.4. COT (Commitment of Traders) Verisi

CFTC'in haftalık raporunu CSV olarak çekebilir veya web'den parse edebilirsin. Basit örnek:

```

import pandas as pd

class COTLoader:
    def __init__(self, url):
        # Örneğin "https://www.cftc.gov/files/dea/history/
        dea_cot_<MONTH><YEAR>.csv"
        self.url = url

    def fetch(self):
        try:
            df = pd.read_csv(self.url)
            # İlgili döviz pariteleri: EUR, JPY, GBP net
            pozisyonları
            return df
        except Exception as e:
            print("COT veri indirme hatası:", e)
            return None

```

Elde ettiğin net long/net short verilerini periyotluk (haftalık) feature olarak ekleyebilirsin.

2.5. MT5 Level II (Order Book) Verisi

```

import MetaTrader5 as mt5

class OrderBookFeatures:
    def __init__(self, symbol):
        self.symbol = symbol

    def fetch_order_book(self):
        book = mt5.market_book_get(self.symbol)
        if book:
            bids = sum([entry.volume for entry in book if entry.type
== mt5.BOOK_TYPE_SELL])
            asks = sum([entry.volume for entry in book if entry.type
== mt5.BOOK_TYPE_BUY])
            ratio = bids / asks if asks > 0 else 0
            return {"bid_volume": bids, "ask_volume": asks,

```

```
"bid_ask_ratio": ratio}
    return {"bid_volume":0, "ask_volume":0, "bid_ask_ratio":0}
```

– Bu bid_ask_ratio yüksekse (örneğin >1.5), “yaklaşan volatilité” veya “kur koruma” sinyali olarak state’e ekleyebilirsiniz.

3 Adım: Çok Zamanlı ve Çok Ölçütlü Özellik Seti

3.1. Tick / 5S / 15S Verisi

```
import MetaTrader5 as mt5

class TickFeatures:
    def __init__(self, symbol):
        self.symbol = symbol

    def fetch_last_seconds_ticks(self, seconds=5):
        utc_to = datetime.datetime.utcnow()
        utc_from = utc_to - datetime.timedelta(seconds=seconds)
        ticks = mt5.copy_ticks_range(self.symbol, utc_from, utc_to,
mt5.COPY_TICKS_ALL)
        return ticks # numpy structured array

    def compute_volatility_spike(self, ticks):
        # Örneğin son 5 saniyedeki fiyat farkı:
        if len(ticks) < 2:
            return 0
        prices = ticks["last"]
        return max(prices) - min(prices)
```

- Bu “5S volatility spike” özelliğini her state adımında hesaplayıp state vektörüne ekle.

3.2. Multi-Timeframe Feature Fusion

feature_engineering.py içinde:

```
import pandas as pd
import talib

class FeatureEngineer:
    def __init__(self, symbol):
        self.symbol = symbol

    def get_bars(self, timeframe, n):
        # Örneğin: timeframe = mt5.TIMEFRAME_H1, n = geriye dönük
bar sayısı
        rates = mt5.copy_rates_from_pos(self.symbol, timeframe, 0,
n)
        df = pd.DataFrame(rates)
        df["time"] = pd.to_datetime(df["time"], unit="s")
        return df

    def compute_indicators(self):
```

```

# M1, H1, H4 barlarını çek
df_m1 = self.get_bars(mt5.TIMEFRAME_M1, 100)
df_h1 = self.get_bars(mt5.TIMEFRAME_H1, 100)
df_h4 = self.get_bars(mt5.TIMEFRAME_H4, 100)

# M1 Momentum, RSI7, CCI14
df_m1["RSI7"] = talib.RSI(df_m1["close"], timeperiod=7)
df_m1["CCI14"] = talib.CCI(df_m1["high"], df_m1["low"],
df_m1["close"], timeperiod=14)
df_m1["Momentum5"] = talib.MOM(df_m1["close"], timeperiod=5)

# H1 RSI14, MACD histogram
df_h1["RSI14"] = talib.RSI(df_h1["close"], timeperiod=14)
macd_h1, macd_signal_h1, macd_hist_h1 =
talib.MACD(df_h1["close"], fastperiod=12, slowperiod=26,
signalperiod=9)
df_h1["MACD_hist"] = macd_hist_h1

# H4 EMA50, EMA200 crossing
df_h4["EMA50"] = talib.EMA(df_h4["close"], timeperiod=50)
df_h4["EMA200"] = talib.EMA(df_h4["close"], timeperiod=200)
df_h4["EMA_cross_up"] = (df_h4["EMA50"] >
df_h4["EMA200"]).astype(int)

# Normalize et (0-1 aralığına) - MinMaxScaler'ı elle uygula
def normalize(series):
    return (series - series.min()) / (series.max() -
series.min() + 1e-9)

features = {
    "M1_RSI7": normalize(df_m1["RSI7"].iloc[-1]),
    "M1_CCI14": normalize(df_m1["CCI14"].iloc[-1]),
    "M1_Mom5": normalize(df_m1["Momentum5"].iloc[-1]),
    "H1_RSI14": normalize(df_h1["RSI14"].iloc[-1]),
    "H1_MACD_hist": normalize(df_h1["MACD_hist"].iloc[-1]),
    "H4_EMA_cross_up": df_h4["EMA_cross_up"].iloc[-1],
}
return features

```

3.3. Teknik + Temel Kombinasyonu

```

import yfinance as yf
import pandas as pd

class FundamentalFeatures:
    def __init__(self, symbol):
        self.symbol = symbol #örn. "EURUSD=X" gibi Yahoo Finance
formatı

    def fetch_macro(self):
        # Örnek: EURUSD için Euro Bölgesi CPI ve ABD CPI
karşılaştırması
        eur_cpi = yf.download("^EUECPI", period="1mo",
interval="1d") # demo amaçlı

```

```

    usd_cpi = yf.download("^USCPI", period="1mo", interval="1d")

    # Son değerleri al
    try:
        ratio = eur_cpi["Close"].iloc[-1] /
usd_cpi["Close"].iloc[-1]
    except:
        ratio = 1.0
    return {"EUR_US_CPI_ratio": ratio}

```

4 Adım: “Rainbow DQN” ve RL Alt Yapısı

Bu kısım en çok zaman alacak, ama adım adım inşa ederiz. İlk olarak basit bir “Double DQN + Dueling” yapısına sahip ajanın iskeletini kuralım. Sonra:

- * Prioritized Experience Replay: replay buffer sınıfı
- * Multi-Step Returns: basitleştirilmiş 3-adım hedef
- * NoisyNet Katmanları: Tensorflow içinde NoisyDense’i ekleyelim
- * Distribütöyönel DQN (C51): $Q(s,a)$ yerine dağılım tahmini

4.1. Basit “Dueling + Double DQN” İskeleti

src/models/dqn_agent.py:

```

import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import random
from collections import deque, namedtuple

# Deneyimleri saklamak için tuple
Experience = namedtuple("Experience", field_names=["state",
"action", "reward", "next_state", "done"])

class PrioritizedReplayBuffer:
    def __init__(self, capacity, alpha=0.6):
        self.capacity = capacity
        self.memory = []
        self.pos = 0
        self.priorities = np.zeros((capacity,), dtype=np.float32)
        self.alpha = alpha

    def add(self, state, action, reward, next_state, done):
        max_prio = self.priorities.max() if self.memory else 1.0
        if len(self.memory) < self.capacity:
            self.memory.append(Experience(state, action, reward,
next_state, done))
        else:
            self.memory[self.pos] = Experience(state, action,
reward, next_state, done)
            self.priorities[self.pos] = max_prio
            self.pos = (self.pos + 1) % self.capacity

    def sample(self, batch_size, beta=0.4):
        if len(self.memory) == self.capacity:
            prios = self.priorities
        else:

```

```

        prios = self.priorities[:self.pos]
        # Prioritelerden örnek seçimi
        probs = prios ** self.alpha
        probs /= probs.sum()

        indices = np.random.choice(len(self.memory), batch_size,
p=probs)
        experiences = [self.memory[idx] for idx in indices]

        # Importance-Sampling weight
        total = len(self.memory)
        weights = (total * probs[indices]) ** (-beta)
        weights /= weights.max()
        weights = np.array(weights, dtype=np.float32)

        batch = Experience(*zip(*experiences))
        return batch, indices, weights

    def update_priorities(self, batch_indices, batch_priorities):
        for idx, prio in zip(batch_indices, batch_priorities):
            self.priorities[idx] = prio

    def __len__(self):
        return len(self.memory)

class DuelingDQN(tf.keras.Model):
    def __init__(self, state_size, action_size):
        super(DuelingDQN, self).__init__()
        self.fc1 = layers.Dense(128, activation="relu")
        self.fc2 = layers.Dense(128, activation="relu")
        # Value stream
        self.value_fc = layers.Dense(64, activation="relu")
        self.value = layers.Dense(1, activation=None)
        # Advantage stream
        self.adv_fc = layers.Dense(64, activation="relu")
        self.adv = layers.Dense(action_size, activation=None)

    def call(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        val = self.value_fc(x)
        val = self.value(val)
        adv = self.adv_fc(x)
        adv = self.adv(adv)
        q = val + (adv - tf.reduce_mean(adv, axis=1, keepdims=True))
        return q

class RainbowAgent:
    def __init__(self, state_size, action_size, buffer_size=100000,
batch_size=64, gamma=0.99, lr=1e-4):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = PrioritizedReplayBuffer(buffer_size)
        self.batch_size = batch_size

```

```

        self.gamma = gamma
        self.beta = 0.4 # başlangıçta düşük, eğitim ilerledikçe
1.0'a çıkacak
        self.lr = lr

        # Online ve hedef ağ
        self.q_network = DuelingDQN(state_size, action_size)
        self.target_network = DuelingDQN(state_size, action_size)
        self.q_network.build(input_shape=(None, state_size))
        self.target_network.build(input_shape=(None, state_size))
        self.optimizer =
tf.keras.optimizers.Adam(learning_rate=self.lr)

        # Hedef ağ kopyalanması
        self.update_target_network()

    def update_target_network(self):

self.target_network.set_weights(self.q_network.get_weights())

    def act(self, state, epsilon=0.1):
        if np.random.rand() < epsilon:
            return random.randrange(self.action_size)
        state = np.expand_dims(state, axis=0).astype(np.float32)
        q_values = self.q_network(state)
        return int(tf.argmax(q_values[0]).numpy())

    def learn(self):
        if len(self.memory) < self.batch_size:
            return

        experiences, indices, weights =
self.memory.sample(self.batch_size, self.beta)
        states = np.vstack(experiences.state)
        actions = np.array(experiences.action)
        rewards = np.array(experiences.reward)
        next_states = np.vstack(experiences.next_state)
        dones = np.array(experiences.done).astype(np.float32)
        weights = np.array(weights)

        # Q hedeflerine Double DQN:
        next_q_values = self.q_network(next_states)
        next_actions = tf.argmax(next_q_values, axis=1)
        next_target_q = self.target_network(next_states)
        target_q = rewards + (1 - dones) * self.gamma *
tf.gather(next_target_q, next_actions, axis=1, batch_dims=1)

        with tf.GradientTape() as tape:
            q_vals = self.q_network(states)
            q_vals = tf.gather(q_vals, actions, axis=1,
batch_dims=1)
            td_errors = target_q - q_vals
            loss = tf.reduce_mean(weights * tf.square(td_errors))

```

```

        grads = tape.gradient(loss,
self.q_network.trainable_variables)
        self.optimizer.apply_gradients(zip(grads,
self.q_network.trainable_variables))

        # Prioritized replay için öncelikleri güncelle
new_priorities = np.abs(td_errors.numpy()) + 1e-6
self.memory.update_priorities(indices, new_priorities)

        # Beta'yı yavaş yavaş 1.0'a çıkar
self.beta = min(1.0, self.beta + 1e-6)

def remember(self, state, action, reward, next_state, done):
    self.memory.add(state, action, reward, next_state, done)

```

Not: Yukarıdakiler sadece temel bileşenler-Noisy Nets, multi-step returns, distribütöyönel DQN, C51, kâr kilitleme mekanizması, vb. daha sonra eklenecek. Bu iskelet, kod akışını anlamak için başlangıç.

5 Adım: FTM0 Uyumlu Risk Yöneticisi
src/risk_manager.py:

```

import MetaTrader5 as mt5

class RiskManager:
    def __init__(self, starting_balance, daily_limit_pct,
total_limit_pct, max_equity_dd_pct):
        self.starting_balance = starting_balance
        self.daily_limit = starting_balance * (daily_limit_pct /
100)
        self.total_limit = starting_balance * (total_limit_pct /
100)
        self.max_equity_dd_pct = max_equity_dd_pct
        self.max_equity = starting_balance
        self.daily_start_balance = starting_balance
        self.current_day = None

    def update_daily_balance(self):
        info = mt5.account_info()
        now_day = pd.to_datetime(info.time).date()
        if self.current_day != now_day:
            self.daily_start_balance = info.balance
            self.current_day = now_day

    def check_daily_loss(self):
        info = mt5.account_info()
        loss = self.daily_start_balance - info.balance
        return loss >= self.daily_limit

    def check_total_loss(self):
        info = mt5.account_info()
        loss = self.starting_balance - info.equity
        return loss >= self.total_limit

```

```

def update_max_equity(self):
    info = mt5.account_info()
    self.max_equity = max(self.max_equity, info.equity)

def check_equity_drawdown(self):
    info = mt5.account_info()
    dd_pct = (self.max_equity - info.equity) / self.max_equity *
100
    return dd_pct >= self.max_equity_dd_pct

def enforce_limits(self):
    if self.check_daily_loss():
        # Tüm zarar eden pozisyonları kapat
        for pos in mt5.positions_get():
            if pos.profit < 0:
                mt5.order_send(request={...}) # Kapama isteği
        return False # Bot işlem açmamalı
    if self.check_total_loss() or self.check_equity_drawdown():
        # Tüm pozisyonları kapat ve botu durdur (ana döngüde
kontrol edilecek)
        for pos in mt5.positions_get():
            mt5.order_send(request={...})
        return False
    return True

```

– Ana döngü içinde her yeni bar’da önce `update_daily_balance()`, sonra `enforce_limits()` çağıracağız. Eğer `Falsedönerse`, yeni pozisyon açılmamasını sağlayacağız.

6 Adım: Gerçek Zamanlı Veri Hattı ve Mikroservis
src/live_pipeline.py:

```

import asyncio
import MetaTrader5 as mt5
from feature_engineering import FeatureEngineer
from alternative_data import MultiLingualTwitterSentiment,
RedditSentiment, NewsSentiment, OrderBookFeatures
from execution import OrderExecutor
from risk_manager import RiskManager
from models.dqn_agent import RainbowAgent

async def on_tick(symbol, agent, risk_manager):
    # 1) Veri toplama
    fe = FeatureEngineer(symbol)
    tf_features = fe.compute_indicators()

    tw_sent =
MultiLingualTwitterSentiment(bearer_token=TWITTER_BEARER_TOKEN).fetc
h_and_analyze()
    rd_sent = RedditSentiment().analyze_sentiment()
    nw_sent = NewsSentiment(NEWSAPI_KEY).analyze_sentiment()

```

```

ob = OrderBookFeatures(symbol).fetch_order_book()

# 2) State oluşturma
state = []
state.extend(list(tf_features.values()))
state += [tw_sent, rd_sent, nw_sent, ob["bid_ask_ratio"]]
state = np.array(state, dtype=np.float32)

# 3) Risk kontrolleri
risk_manager.update_daily_balance()
if not risk_manager.enforce_limits():
    return

# 4) Ajan eylemi
action = agent.act(state, epsilon=0.05)
if action == 0: # örneğin 0 = Hold, 1 = Buy, 2 = Sell
    pass
elif action == 1:
    OrderExecutor.open_long(symbol, lot=0.01)
elif action == 2:
    OrderExecutor.open_short(symbol, lot=0.01)

# 5) Öğrenme (özellikle backtest modunda)
# reward hesaplaması, next_state, done flag'leri inşa edip
agent.learn()'a besle.

async def main_loop(symbol):
    if not mt5.initialize(login=MT5_LOGIN, password=MT5_PASSWORD,
server=MT5_SERVER, path=MT5_PATH):
        print("MT5 initialize hatası:", mt5.last_error())
        return

    agent = RainbowAgent(state_size=10, action_size=3) # state_size
ve action_size gerçek feature sayısına göre
    risk_manager = RiskManager(starting_balance=25000,
daily_limit_pct=5, total_limit_pct=10, max_equity_dd_pct=4)

    while True:
        ticks = mt5.copy_ticks_from(symbol,
datetime.datetime.utcnow() - datetime.timedelta(minutes=1), 1000,
mt5.COPY_TICKS_ALL)
        # Burada 1 dakikalık kapanış barını simüle edebiliriz, ya da
OnTick mantığı
        await on_tick(symbol, agent, risk_manager)
        await asyncio.sleep(1) # her saniye tetiklenme

if __name__ == "__main__":
    asyncio.run(main_loop("EURUSD"))

```

Bu iskelet, gerçek-time veri hattının nasıl çalışacağına dair ana hatları gösterir. Canary test'i olarak önce demo hesapta "while True" döngüsünü bir süre çalıştırıp "state"leri doğru hesapladığından emin olmalısın.

1 Adım: Proje Ortamı ve Klasör Yapısı

1.1. Proje Klasörünü Oluştur ve Git Başlat

1. Terminal (Mac/Linux) ya da PowerShell (Windows) aç.
2. Aşağıdaki komutları çalıştır:

```
# 1. Proje ana klasörünü oluştur ve içine gir
mkdir ftmo_bot
cd ftmo_bot
```

```
# 2. Git deposu başlat
git init
```

Bu iki satır, projenin ana dizinini (ftmo_bot) oluşturacak ve o dizinde bir Git deposu başlatacaktır.

1.2. Sanal Ortam (Virtual Environment) Oluştur

Python'un paket çakışmalarını önlemek için projeye özel bir sanal ortam yapısı kuralım.

Windows (PowerShell):

```
python -m venv venv
.\venv\Scripts\activate
```

Komutu çalıştırdıktan sonra terminalinizde (venv) ön eki görünecek; bu, sanal ortamın aktif olduğunu gösterir.

1.3. requirements.txt Hazırlama ve Gerekli Kütüphaneleri Yükleme
Projemizde ilerleyen aşamalarda ihtiyaç duyacağımız temel Python paketlerini bir requirements.txt dosyasına yazalım. Root (ana) dizinde şu adımları takip et:

1. requirements.txt dosyasını oluştur:

```
touch requirements.txt
```

2. Bir metin editörü (VSCode, Sublime, nano, vs.) ile requirements.txt aç ve içine aşağıdaki satırları yapıştır:

```
MetaTrader5
pandas
numpy
TA-Lib
tensorflow
newsapi-python
tweepy
praw
redis
kafka-python
flask
plotly
dash
python-dotenv
```

- Not:

- * Eğer ileride PyTorch kullanmak istersen, tensorflow yerine torch ekleyebilirsin. Şimdilik TensorFlow ile devam edeceğiz.

- * TA-Lib bağımlılığı işletim sistemine göre ek paketler gerektirebilir (örneğin macOS'ta brew install ta-lib, Ubuntu'da sudo apt-get install libta-lib0 libta-lib-dev).

Paketleri yüklemek için terminalde (sanal ortam aktifken) şu komutu çalıştır:

```
pip install -r requirements.txt
```

Bu komut, listedeki tüm kütüphaneleri (MetaTrader5, pandas, numpy, vb.) sanal ortama indirecek.

Yükleme tamamlandığında, versiyon kontrolü için requirements.txt'i Git'e ekle:

```
git add requirements.txt
```

```
git commit -m "İlk: requirements.txt eklendi"
```

1.4. Klasör Yapısını Oluşturma

Projenin ilerleyen aşamalarını düzenli tutmak için şu klasörleri oluşturalım (terminalde hâlâ ftmo_bot dizinindeyken):

```
mkdir data
mkdir src
mkdir src/models
mkdir src/dashboard
mkdir logs
mkdir reports
```

Bu yapı, ileride kodu ve dosyaları organize etmemizi kolaylaştıracak:

```
ftmo_bot/
├── data/           # Ham veri dosyaları (CSV, JSON, vb.)
├── src/            # Tüm Python kodları
│   ├── models/    # DQN, Transformer, ensemble modelleri
│   └── dashboard/ # Flask + Dash uygulaması
├── logs/          # Log dosyaları (çalışma zamanı kayıtları)
├── reports/       # Backtest raporları, grafikleri
└── barındıracak
    ├── requirements.txt # İhtiyaç duyulan Python paketleri
    └── README.md        # Proje ile ilgili kısa açıklamalar (henüz boş)
```

Şimdi bu adımı da Git'e kaydedelim:

```
git add data src logs reports
```

```
git commit -m "Proje klasör yapısı oluşturuldu"
```

1.5. README.md Dosyasını Başlatma

Projeyi sonraki kullanıcılara (ve kendimize) daha anlaşılır kılmak

için kök dizine bir README.md ekleyelim:

```
touch README.md
```

Ardından README.md'i açıp başlık satırıyla kısa bir giriş yazabilirsin. Örneğin:

```
# FTM0 Uyumlu Forex Ticaret Botu
```

Bu proje, FTM0 kurallarına tam uyumlu, AI destekli ve çok katmanlı risk yönetimi içeren bir otomatik forex ticaret botu geliştirmeyi amaçlamaktadır. A'dan Z'ye adım adım hem teknik hem de stratejik altyapıyı sağlayacak modüller içerir.

Kaydedip kapat. Son olarak:

```
git add README.md
git commit -m "README.md eklendi"
```

```
import os
import datetime
import requests
import pandas as pd
import MetaTrader5 as mt5
import tweepy
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from newsapi import NewsApiClient
from dotenv import load_dotenv
```

```
load_dotenv()
```

```
# -----
# Çok Dilli Twitter Duygu Analizi
# -----
class MultiLingualTwitterSentiment:
    def __init__(self, bearer_token=None):
        token = bearer_token or os.getenv("TWITTER_BEARER_TOKEN")
        if not token:
            raise ValueError("TWITTER_BEARER_TOKEN çevresel
değişkeni ayarlanmalı.")
        self.client = tweepy.Client(bearer_token=token)
        self.analyzer = SentimentIntensityAnalyzer()
        # İzlenecek hashtagler
        self.hashtags = ["#forex", "#forextrading", "#EURUSD",
"#GBPUSD", "#USDJPY"]
        # Desteklenen diller
        self.langs = ["en", "tr", "es", "ar"]

    def fetch_and_analyze(self):
        scores = []
        for tag in self.hashtags:
            for lang in self.langs:
                query = f"{tag} lang:{lang} -is:retweet"
```

```

        try:
            response =
self.client.search_recent_tweets(query=query, max_results=50)
        except Exception:
            continue
        if not response or not response.data:
            continue
        for tweet in response.data:
            vs = self.analyzer.polarity_scores(tweet.text)
            scores.append(vs["compound"])
    if scores:
        return sum(scores) / len(scores)
    return 0.0

```

```

# -----
# Reddit Duygu Analizi (Pushshift API)
# -----
class RedditSentiment:
    def __init__(self):
        self.subreddits = ["forex", "algotrading"]
        self.base_url = "https://api.pushshift.io/reddit/search/
submission/"
        self.analyzer = SentimentIntensityAnalyzer()

    def fetch_recent_posts(self, subreddit, hours=2):
        now = int(datetime.datetime.utcnow().timestamp())
        after = now - 3600 * hours
        params = {
            "subreddit": subreddit,
            "size": 100,
            "after": after,
            "sort": "desc"
        }
        try:
            r = requests.get(self.base_url, params=params,
timeout=10)
            r.raise_for_status()
            posts = r.json().get("data", [])
            return [post.get("title", "") + " " +
post.get("selftext", "") for post in posts]
        except Exception:
            return []

    def analyze_sentiment(self):
        all_scores = []
        for sub in self.subreddits:
            texts = self.fetch_recent_posts(sub, hours=2)
            for text in texts:
                vs = self.analyzer.polarity_scores(text)
                all_scores.append(vs["compound"])
        return (sum(all_scores) / len(all_scores)) if all_scores
else 0.0

```

```

# -----
# Haber Duygu Analizi (NewsAPI)
# -----
class NewsSentiment:
    def __init__(self, api_key=None):
        key = api_key or os.getenv("NEWSAPI_KEY")
        if not key:
            raise ValueError("NEWSAPI_KEY çevresel değişkeni
            ayarlanmalı.")
        self.client = NewsApiClient(api_key=key)
        self.analyzer = SentimentIntensityAnalyzer()
        self.keywords = ["forex", "USD", "EUR", "FED rate",
        "interest rate"]

    def fetch_headlines(self):
        today = datetime.datetime.utcnow().date().isoformat()
        try:
            articles = self.client.get_everything(q=" OR
            ".join(self.keywords),
            from_param=today,
            to=today,
            language="en",
            sort_by="relevancy",
            page_size=100)
            return [art.get("title", "") + " " +
            art.get("description", "") for art in articles.get("articles", [])
            if art.get("description")]
        except Exception:
            return []

    def analyze_sentiment(self):
        headlines = self.fetch_headlines()
        scores = []
        for text in headlines:
            vs = self.analyzer.polarity_scores(text)
            scores.append(vs["compound"])
        return (sum(scores) / len(scores)) if scores else 0.0

# -----
# Economic Calendar (ForexFactory)
# -----
class EconomicCalendar:
    def __init__(self):
        # Haftalık JSON feed (ForexFactory)
        self.url = "https://cdn-nfs.forexfactory.net/
        ff_calendar_thisweek.json"

    def fetch_high_impact(self):
        try:
            r = requests.get(self.url, timeout=10)
            r.raise_for_status()

```

```

        data = r.json()
        hi_events = [e for e in data.get("events", []) if
e.get("impact") == "High"]
        return hi_events
    except Exception:
        return []

# -----
# COT (Commitment of Traders) Verisi
# -----
class COTLoader:
    def __init__(self, url):
        # Örnek URL: CFTC haftalık COT CSV dosya adresi
        self.url = url

    def fetch(self):
        try:
            df = pd.read_csv(self.url)
            return df
        except Exception:
            return None

# -----
# MT5 Order Book (Market Depth) Özellikleri
# -----
class OrderBookFeatures:
    def __init__(self, symbol):
        self.symbol = symbol
        if not mt5.initialize():
            raise RuntimeError("MT5 initialize hatası: " +
str(mt5.last_error()))

    def fetch_order_book(self):
        book = mt5.market_book_get(self.symbol)
        if not book:
            return {"bid_volume": 0, "ask_volume": 0,
"bid_ask_ratio": 0}
        bids = sum([entry.volume for entry in book if entry.type ==
mt5.BOOK_TYPE_SELL])
        asks = sum([entry.volume for entry in book if entry.type ==
mt5.BOOK_TYPE_BUY])
        ratio = bids / asks if asks > 0 else 0
        return {"bid_volume": bids, "ask_volume": asks,
"bid_ask_ratio": ratio}

# -----
# Test veya örnek kullanım fonksiyonu
# -----
def example_usage():
    # Twitter duygu
    tw = MultiLingualTwitterSentiment()

```

```

print("Twitter Sentiment (Avg):", tw.fetch_and_analyze())

# Reddit duygu
rd = RedditSentiment()
print("Reddit Sentiment (Avg):", rd.analyze_sentiment())

# Haber duygu
nw = NewsSentiment()
print("News Sentiment (Avg):", nw.analyze_sentiment())

# Economic Calendar
ec = EconomicCalendar()
print("High Impact Events:", ec.fetch_high_impact())

# COT örnek (URL'i kendi CSV adresinle değiştir)
cot_url = os.getenv("COT_CSV_URL", "https://example.com/
cot_weekly.csv")
cot = COTLoader(cot_url)
df_cot = cot.fetch()
print("COT Dataframe:" if df_cot is not None else "COT verisi
yüklenemedi")

# MT5 Order Book
obf = OrderBookFeatures("EURUSD")
print("Order Book Features:", obf.fetch_order_book())

```

```

if __name__ == "__main__":
    example_usage()

```

alternative_data.py dosyasını oluşturup içine tüm sınıfları ve örnek kullanım fonksiyonunu ekledim. Artık bu modül, şunları paket halinde sunuyor:

1. MultiLingualTwitterSentiment – Çokdilli, çok hashtag'li Twitter verisi çekip Vader ile duygu skoru hesaplar
 2. RedditSentiment – Pushshift API üzerinden r/forex ve r/algorithmtrading gönderilerini alıp duygu analizi yapar
 3. NewsSentiment – NewsAPI'dan haber başlıkları çekerek duygu skorunu çıkarır
 4. EconomicCalendar – ForexFactory'nin haftalık ekonomik takviminden "High Impact" olayları döner
 5. COTLoader – Belirtilen CFTC COT CSV URL'sinden haftalık raporu indirir
 6. OrderBookFeatures – MT5'ten bir paritenin Level II (Order Book) verisini çekip bid/ask hacimlerini ve oranı hesaplar
- Bir de en alta, example_usage() fonksiyonuyla her bir sınıfı test etme örneği ekledim; doğrudan çalıştırdığınızda deneyebilirsiniz.

```

import pandas as pd
import numpy as np
import MetaTrader5 as mt5
import talib
import datetime

```

```

class FeatureEngineer:
    def __init__(self, symbol):
        self.symbol = symbol
        if not mt5.initialize():
            raise RuntimeError(f"MT5 initialize hatası: {mt5.last_error()}")

    def get_bars(self, timeframe, n):
        # timeframe: mt5.TIMEFRAME_M1, TIMEFRAME_H1, TIMEFRAME_H4
        # n: geriye dönük bar sayısı
        rates = mt5.copy_rates_from_pos(self.symbol, timeframe, 0,
n)
        if rates is None or len(rates) == 0:
            raise RuntimeError(f"{self.symbol} için {timeframe}
zaman diliminde veri alınamadı.")
        df = pd.DataFrame(rates)
        df['time'] = pd.to_datetime(df['time'], unit='s')
        return df

    def normalize(self, series):
        if series.isnull().all():
            return series.fillna(0)
        min_val = series.min()
        max_val = series.max()
        denom = max_val - min_val
        if denom == 0:
            return series.apply(lambda x: 0.5)
        return (series - min_val) / denom

    def compute_multitimeframe_features(self):
        # M1, H1, H4 barlarını çek
        df_m1 = self.get_bars(mt5.TIMEFRAME_M1, 500)
        df_h1 = self.get_bars(mt5.TIMEFRAME_H1, 500)
        df_h4 = self.get_bars(mt5.TIMEFRAME_H4, 500)

        # --- M1 Göstergeler ---
        df_m1['RSI7'] = talib.RSI(df_m1['close'], timeperiod=7)
        df_m1['CCI14'] = talib.CCI(df_m1['high'], df_m1['low'],
df_m1['close'], timeperiod=14)
        df_m1['Momentum5'] = talib.MOM(df_m1['close'], timeperiod=5)

        # Son M1 değerlerini normalleştir
        m1_rsi7 = self.normalize(df_m1['RSI7']).iloc[-1]
        m1_cci14 = self.normalize(df_m1['CCI14']).iloc[-1]
        m1_mom5 = self.normalize(df_m1['Momentum5']).iloc[-1]

        # --- H1 Göstergeler ---
        df_h1['RSI14'] = talib.RSI(df_h1['close'], timeperiod=14)
        macd_h1, macd_signal_h1, macd_hist_h1 = talib.MACD(
            df_h1['close'], fastperiod=12, slowperiod=26,
signalperiod=9
        )
        df_h1['MACD_hist'] = macd_hist_h1

```

```

h1_rsi14 = self.normalize(df_h1['RSI14']).iloc[-1]
h1_macd_hist = self.normalize(df_h1['MACD_hist']).iloc[-1]

# --- H4 Göstergeler ---
df_h4['EMA50'] = talib.EMA(df_h4['close'], timeperiod=50)
df_h4['EMA200'] = talib.EMA(df_h4['close'], timeperiod=200)
df_h4['EMA_cross_up'] = (df_h4['EMA50'] >
df_h4['EMA200']).astype(int)

h4_ema_cross_up = df_h4['EMA_cross_up'].iloc[-1]

# --- Ek Özellikler: H4 Eğilim Açısı (EMA açısı) ---
# EMA50 ile EMA200 açısı arasındaki farkın işareti
ema_diff = df_h4['EMA50'].iloc[-1] -
df_h4['EMA200'].iloc[-1]
h4_trend_sign = 1 if ema_diff > 0 else (-1 if ema_diff < 0
else 0)

# --- Son Fiyat ve Volatilite (H1 ATR) ---
df_h1['ATR14'] = talib.ATR(df_h1['high'], df_h1['low'],
df_h1['close'], timeperiod=14)
h1_atr14 = self.normalize(df_h1['ATR14']).iloc[-1]

# Birleştir ve sözlük olarak döndür
features = {
    'M1_RSI7': float(m1_rsi7),
    'M1_CCI14': float(m1_cci14),
    'M1_MOM5': float(m1_mom5),
    'H1_RSI14': float(h1_rsi14),
    'H1_MACD_HIST': float(h1_macd_hist),
    'H4_EMA_CROSS_UP': int(h4_ema_cross_up),
    'H4_TREND_SIGN': int(h4_trend_sign),
    'H1_ATR14': float(h1_atr14)
}
return features

# Örnek kullanım
if __name__ == '__main__':
    fe = FeatureEngineer("EURUSD")
    feats = fe.compute_multitimeframe_features()
    print("Feature Vector:", feats)

feature_engineering.py dosyasını oluşturdum. İçeriği özetle:
* FeatureEngineer sınıfı:
    * get_bars(timeframe, n): MT5'ten istenen timeframe (M1, H1, H4)
ve bar sayısı kadar veriyi DataFrame'e çeker.
    * normalize(series): Bir pandas serisini 0-1 aralığına ölçekler
(min-max).
    * compute_multitimeframe_features():
        1. M1'den RSI(7), CCI(14), Momentum(5) hesaplar ve normalize
eder.
        2. H1'den RSI(14), MACD histogram hesaplar ve normalize
eder.

```

3. H4'ten EMA(50) & EMA(200) üzerinden "cross-up" (1/0) ve trend işareti (-1, 0, 1) hesaplar.
4. H1 ATR(14) hesaplar ve normalize eder.
5. Tüm özellikleri bir sözlük (dictionary) olarak döndürür.

Bu haliyle,

FeatureEngineer("EURUSD").compute_multitimeframe_features()
çalıştırıldığında, her zaman diliminden çıkardığı normalized özellikleri bir arada sunar.

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import random
from collections import deque, namedtuple

# Rainbow DQN bileşenleri: Dueling + Double + Prioritized Experience
# Replay + Multi-Step + NoisyNet + Distributional (C51)
# -----
# Öncelikli Deneyim Yeniden Oynama (Prioritized Replay Buffer)
# -----
Experience = namedtuple("Experience", field_names=["state",
"action", "reward", "next_state", "done"])

class PrioritizedReplayBuffer:
    def __init__(self, capacity, alpha=0.6):
        self.capacity = capacity
        self.memory = []
        self.pos = 0
        self.priorities = np.zeros((capacity,), dtype=np.float32)
        self.alpha = alpha

    def add(self, state, action, reward, next_state, done):
        max_prio = self.priorities.max() if self.memory else 1.0
        if len(self.memory) < self.capacity:
            self.memory.append(Experience(state, action, reward,
next_state, done))
        else:
            self.memory[self.pos] = Experience(state, action,
reward, next_state, done)
            self.priorities[self.pos] = max_prio
            self.pos = (self.pos + 1) % self.capacity

    def sample(self, batch_size, beta=0.4):
        if len(self.memory) == self.capacity:
            prios = self.priorities
        else:
            prios = self.priorities[:self.pos]
        probs = prios ** self.alpha
        probs /= probs.sum()

        indices = np.random.choice(len(self.memory), batch_size,
p=probs)
        experiences = [self.memory[idx] for idx in indices]
```

```

        total = len(self.memory)
        weights = (total * probs[indices]) ** (-beta)
        weights /= weights.max()
        weights = np.array(weights, dtype=np.float32)

        batch = Experience(*zip(*experiences))
        return batch, indices, weights

    def update_priorities(self, batch_indices, batch_priorities):
        for idx, prio in zip(batch_indices, batch_priorities):
            self.priorities[idx] = prio

    def __len__(self):
        return len(self.memory)

# -----
# Noisy Network Katmanı (NoisyNet)
# -----
class NoisyDense(layers.Layer):
    def __init__(self, units, sigma_init=0.017, **kwargs):
        super(NoisyDense, self).__init__(**kwargs)
        self.units = units
        self.sigma_init = sigma_init

    def build(self, input_shape):
        input_dim = int(input_shape[-1])
        # Parametreler
        self.mu_w = self.add_weight(name='mu_w', shape=(input_dim,
self.units), initializer=tf.random_uniform_initializer(minval=-1/
np.sqrt(input_dim), maxval=1/np.sqrt(input_dim)), trainable=True)
        self.sigma_w = self.add_weight(name='sigma_w',
shape=(input_dim, self.units),
initializer=tf.constant_initializer(self.sigma_init),
trainable=True)
        self.mu_b = self.add_weight(name='mu_b',
shape=(self.units,),
initializer=tf.random_uniform_initializer(minval=-1/
np.sqrt(input_dim), maxval=1/np.sqrt(input_dim)), trainable=True)
        self.sigma_b = self.add_weight(name='sigma_b',
shape=(self.units,),
initializer=tf.constant_initializer(self.sigma_init),
trainable=True)

        # Rastgele gürültü için değişkenler
        self.epsilon_i = tf.zeros((input_dim,))
        self.epsilon_j = tf.zeros((self.units,))

    def call(self, inputs, training=True):
        if training:
            # Faktöriyel gürültü
            input_dim = self.mu_w.shape[0]
            output_dim = self.mu_w.shape[1]
            self.epsilon_i = tf.random.normal((input_dim,))

```

```

        self.epsilon_j = tf.random.normal((output_dim,))
        f_i = tf.sign(self.epsilon_i) *
tf.sqrt(tf.abs(self.epsilon_i))
        f_j = tf.sign(self.epsilon_j) *
tf.sqrt(tf.abs(self.epsilon_j))
        w_noise = tf.expand_dims(f_i, -1) * tf.expand_dims(f_j,
0)
        b_noise = f_j
        w = self.mu_w + self.sigma_w * w_noise
        b = self.mu_b + self.sigma_b * b_noise
    else:
        w = self.mu_w
        b = self.mu_b
    return tf.matmul(inputs, w) + b

```

```

# -----
# Dueling + Distributional (C51) Network (Birleřtirilmiř). 51 atom.
# -----
class DQNNetwork(tf.keras.Model):
    def __init__(self, state_size, action_size, atom_size=51,
v_min=-10.0, v_max=10.0):
        super(DQNNetwork, self).__init__()
        self.state_size = state_size
        self.action_size = action_size
        self.atom_size = atom_size
        self.v_min = v_min
        self.v_max = v_max
        self.delta_z = (v_max - v_min) / (atom_size - 1)
        self.support = tf.linspace(v_min, v_max, atom_size)

        # Ortak katmanlar
        self.fc1 = NoisyDense(128)
        self.fc2 = NoisyDense(128)

        # Value stream
        self.value_fc = NoisyDense(128)
        self.value_out = NoisyDense(atom_size)

        # Advantage stream
        self.adv_fc = NoisyDense(128)
        self.adv_out = NoisyDense(action_size * atom_size)

    def call(self, x, training=True):
        x = tf.cast(x, tf.float32)
        h = tf.nn.relu(self.fc1(x, training=training))
        h = tf.nn.relu(self.fc2(h, training=training))

        # Value
        v = tf.nn.relu(self.value_fc(h, training=training))
        v = self.value_out(v, training=training)
        v = tf.reshape(v, [-1, 1, self.atom_size]) # [batch, 1,
atom]

```

```

        # Advantage
        a = tf.nn.relu(self.adv_fc(h, training=training))
        a = self.adv_out(a, training=training)
        a = tf.reshape(a, [-1, self.action_size, self.atom_size]) #
[batch, action, atom]

        # Dueling fusion: Q_dist = V + (A - mean(A, axis=1,
keepdims=True))
        a_mean = tf.reduce_mean(a, axis=1, keepdims=True)
        q_atoms = v + (a - a_mean)

        # Softmax ile dağılım (her action-atom için prob)
        dist = tf.nn.softmax(q_atoms, axis=2) # [batch, action,
atom]
        dist = tf.clip_by_value(dist, 1e-3, 1.0) # sayısal
kararlılık
        dist = dist / tf.reduce_sum(dist, axis=2, keepdims=True)
        return dist

    def get_q_values(self, x, training=False):
        # Dağılımdan beklenti hesapla:  $\sum_z P(s,a,z) * z$ 
        dist = self.call(x, training=training) # [batch, action,
atom]
        q_values = tf.reduce_sum(dist * self.support, axis=2) #
[batch, action]
        return q_values

```

```

# -----
# Rainbow Ajanı
# -----
class RainbowAgent:
    def __init__(self, state_size, action_size, atom_size=51,
v_min=-10.0, v_max=10.0,
n_step=3, buffer_size=200000, batch_size=64,
gamma=0.99, lr=1e-4,
alpha=0.6, beta_start=0.4, beta_frames=100000):
        self.state_size = state_size
        self.action_size = action_size
        self.atom_size = atom_size
        self.v_min = v_min
        self.v_max = v_max
        self.n_step = n_step
        self.support = tf.linspace(v_min, v_max, atom_size)
        self.delta_z = (v_max - v_min) / (atom_size - 1)
        self.batch_size = batch_size
        self.gamma = gamma
        self.lr = lr
        self.beta_start = beta_start
        self.beta_frames = beta_frames
        self.frame_idx = 1

    # Replay buffer
    self.memory = PrioritizedReplayBuffer(buffer_size, alpha)

```

```

        self.n_step_buffer = deque(maxlen=n_step)

        # Ağlar
        self.q_network = DQNNetwork(state_size, action_size,
atom_size, v_min, v_max)
        self.target_network = DQNNetwork(state_size, action_size,
atom_size, v_min, v_max)
        self.q_network.build(input_shape=(None, state_size))
        self.target_network.build(input_shape=(None, state_size))
        self.optimizer = tf.keras.optimizers.Adam(learning_rate=lr)

        # Hedef ağı eşitle
        self.update_target_network()

    def update_target_network(self):

self.target_network.set_weights(self.q_network.get_weights())

    def act(self, state):
        state = np.expand_dims(state, axis=0).astype(np.float32)
        q_values = self.q_network.get_q_values(state,
training=False).numpy()
        action = np.argmax(q_values[0])
        return action

    def append_experience(self, state, action, reward, next_state,
done):
        # n-step geçici FPS
        self.n_step_buffer.append((state, action, reward,
next_state, done))
        if len(self.n_step_buffer) < self.n_step:
            return

        # n-step dönüş
        reward_sum, next_state_n, done_n = self._get_n_step_info()
        state0, action0, _, _, _ = self.n_step_buffer[0]
        self.memory.add(state0, action0, reward_sum, next_state_n,
done_n)

    def _get_n_step_info(self):
        reward, next_state, done = self.n_step_buffer[-1][2],
self.n_step_buffer[-1][3], self.n_step_buffer[-1][4]
        for exp in list(self.n_step_buffer)[-1][::-1]:
            r, n_s, d = exp[2], exp[3], exp[4]
            reward = r + self.gamma * reward * (1 - d)
            next_state, done = (n_s, d) if d else (next_state, done)
        return reward, next_state, done

    def compute_td_error(self, states, actions, rewards,
next_states, dones):
        # Dağılım temelli hedef hesaplama (Projection step)
        batch_size = len(states)
        # Destek vektörü
        support = self.support

```

```

        # Next states distribution
        next_dist = self.q_network.call(next_states, training=False)
# online dağılım [B, A, atom]
        next_q = tf.reduce_sum(next_dist * tf.reshape(support, [1,
1, self.atom_size]), axis=2) # [B, A]
        next_action = tf.argmax(next_q, axis=1) # Double DQN seçimi
        next_dist_target = self.target_network.call(next_states,
training=False) # [B, A, atom]

        next_dist_target = tf.transpose(next_dist_target, [0, 2, 1])
# [B, atom, A]
        next_dist_target = tf.gather(next_dist_target, next_action,
axis=2, batch_dims=1) # [B, atom]
        next_dist_target = tf.transpose(next_dist_target, [0, 1]) #
[B, atom]

        # Projeksiyon
        rewards = tf.reshape(rewards, [-1, 1])
        dones = tf.reshape(dones, [-1, 1])
        support = tf.reshape(support, [1, -1])
        tz = tf.clip_by_value(rewards + (1 - dones) * (self.gamma **
self.n_step) * support, self.v_min, self.v_max)
        b = (tz - self.v_min) / self.delta_z
        l = tf.floor(b)
        u = tf.ceil(b)

        m = np.zeros((batch_size, self.atom_size), dtype=np.float32)
        next_dist_np = next_dist_target.numpy()
        for i in range(batch_size):
            for j in range(self.atom_size):
                l_idx = int(l[i, j])
                u_idx = int(u[i, j])
                prob = next_dist_np[i, j]
                if l_idx == u_idx:
                    m[i, l_idx] += prob
                else:
                    m[i, l_idx] += prob * (u[i, j] - b[i, j])
                    m[i, u_idx] += prob * (b[i, j] - l_idx)
        return m

    def learn(self):
        if len(self.memory) < self.batch_size:
            return

        beta = min(1.0, self.beta_start + (self.frame_idx * (1.0 -
self.beta_start) / self.beta_frames))
        batch, indices, weights =
self.memory.sample(self.batch_size, beta)
        states = np.vstack(batch.state).astype(np.float32)
        actions = np.array(batch.action)
        rewards = np.array(batch.reward, dtype=np.float32)
        next_states = np.vstack(batch.next_state).astype(np.float32)
        dones = np.array(batch.done, dtype=np.float32)

```

```

        weights = np.array(weights, dtype=np.float32)

        # Hedef dağılım (projection)
        m = self.compute_td_error(states, actions, rewards,
next_states, dones)

        with tf.GradientTape() as tape:
            dist = self.q_network.call(states, training=True) # [B,
A, atom]
            action_indices = tf.stack([tf.range(self.batch_size),
actions], axis=1)
            dist_a = tf.gather_nd(dist, action_indices) # [B, atom]
            log_dist_a = tf.math.log(dist_a)
            loss = -tf.reduce_mean(weights * tf.reduce_sum(m *
log_dist_a, axis=1))

            grads = tape.gradient(loss,
self.q_network.trainable_variables)
            self.optimizer.apply_gradients(zip(grads,
self.q_network.trainable_variables))

        # Yeni öncelikleri güncelle
        new_prios = np.abs((m - dist_a.numpy()).sum(axis=1)) + 1e-6
        self.memory.update_priorities(indices, new_prios)

        self.frame_idx += 1

def update(self):
    # Her C adımında hedef ağı güncelle (örneğin C=1000)
    self.update_target_network()

```

dqn_agent.py dosyasını oluşturdum ve Rainbow DQN'in tüm bileşenlerini ekledim:

- * PrioritizedReplayBuffer (Öncelikli Replay)
- * NoisyDense (NoisyNet) katmanı
- * DQNNetwork (Dueling + C51 Distributional DQN)
- * RainbowAgent sınıfı, içinde:
 - * n-step geri dönüş hesaplama
 - * Double DQN ve C51 kullanarak distrübisyon projeksiyonu
 - * Güncellenebilir hedef ağı (target network)
 - * Prioritized Replay ile frame bazlı beta artırımı
 - * Ağırlıklandırılmış loss ve öncelik güncelleme

Artık ajan, tam Rainbow C51 ("Categorical DQN") olarak çalışabilecek durumda. Bir sonraki adımda bu ajanın live_pipeline.py içindeki on_tick çağrısına nasıl entegre edileceğini göstereceğim ve deneysel örneklerle "state → action → learn" akışını hazırlayacağız.

```

import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import random
from collections import namedtuple

```

```

# Replay buffer için namedtuple
Experience = namedtuple("Experience", field_names=["state",
"action", "reward", "next_state", "done", "n_step_return",
"next_state_n"])

class PrioritizedReplayBuffer:
    def __init__(self, capacity, alpha=0.6, n_step=3, gamma=0.99):
        self.capacity = capacity
        self.buffer = []
        self.pos = 0
        self.priorities = np.zeros((capacity,), dtype=np.float32)
        self.alpha = alpha
        self.n_step = n_step
        self.gamma = gamma
        self.n_step_buffer = []

    def _get_n_step_info(self):
        # n-step getiri hesaplama
        reward, next_state, done = self.n_step_buffer[-1][2:]
        for transition in reversed(self.n_step_buffer[:-1]):
            r, n_s, d = transition[2], transition[3], transition[4]
            reward = transition[2] + self.gamma * reward * (1 - d)
            next_state = n_s
            done = d or done
        state, action = self.n_step_buffer[0][0],
self.n_step_buffer[0][1]
        return state, action, reward, next_state, done

    def add(self, state, action, reward, next_state, done):
        # Geçici n-step buffer'a ekle
        transition = (state, action, reward, next_state, done)
        self.n_step_buffer.append(transition)
        if len(self.n_step_buffer) < self.n_step:
            return
        state_n, action_n, reward_n, next_state_n, done_n =
self._get_n_step_info()
        experience = Experience(state_n, action_n, reward_n,
next_state_n, done_n, reward_n, next_state_n)
        max_prio = self.priorities.max() if self.buffer else 1.0
        if len(self.buffer) < self.capacity:
            self.buffer.append(experience)
        else:
            self.buffer[self.pos] = experience
            self.priorities[self.pos] = max_prio
            self.pos = (self.pos + 1) % self.capacity
            # n-step buffer üstündeki birinci öğeyi çıkar
            self.n_step_buffer.pop(0)

    def sample(self, batch_size, beta=0.4):
        if len(self.buffer) == self.capacity:
            prios = self.priorities
        else:
            prios = self.priorities[:self.pos]
        probs = prios ** self.alpha

```

```

        probs /= probs.sum()

        indices = np.random.choice(len(self.buffer), batch_size,
p=probs)
        experiences = [self.buffer[idx] for idx in indices]

        total = len(self.buffer)
        weights = (total * probs[indices]) ** (-beta)
        weights /= weights.max()
        weights = np.array(weights, dtype=np.float32)

        batch = Experience(*zip(*experiences))
        return batch, indices, weights

    def update_priorities(self, indices, priorities):
        for idx, prio in zip(indices, priorities):
            self.priorities[idx] = prio

    def __len__(self):
        return len(self.buffer)

# NoisyNet için özel Dense katman
class NoisyDense(layers.Layer):
    def __init__(self, units):
        super(NoisyDense, self).__init__()
        self.units = units

    def build(self, input_shape):
        input_dim = input_shape[-1]
        # Mu ve sigma parametreleri
        self.mu_w = self.add_weight(name='mu_w', shape=(input_dim,
self.units), initializer=tf.random_uniform_initializer(-1/
np.sqrt(input_dim), 1/np.sqrt(input_dim)), trainable=True)
        self.sigma_w = self.add_weight(name='sigma_w',
shape=(input_dim, self.units),
initializer=tf.constant_initializer(0.017), trainable=True)
        self.mu_b = self.add_weight(name='mu_b',
shape=(self.units,), initializer=tf.random_uniform_initializer(-1/
np.sqrt(input_dim), 1/np.sqrt(input_dim)), trainable=True)
        self.sigma_b = self.add_weight(name='sigma_b',
shape=(self.units,), initializer=tf.constant_initializer(0.017),
trainable=True)

    def call(self, inputs):
        # Factorized Gaussian noise
        input_dim = tf.shape(self.mu_w)[0]
        eps_i = tf.random.normal((input_dim,))
        eps_j = tf.random.normal((self.units,))
        f_i = tf.sign(eps_i) * tf.sqrt(tf.abs(eps_i))
        f_j = tf.sign(eps_j) * tf.sqrt(tf.abs(eps_j))
        w_noise = tf.einsum('i,j->ij', f_i, f_j)
        b_noise = f_j

        w = self.mu_w + self.sigma_w * w_noise

```

```

        b = self.mu_b + self.sigma_b * b_noise
        return tf.matmul(inputs, w) + b

class DuelingRainbowDQN(tf.keras.Model):
    def __init__(self, state_size, action_size, atom_size=51,
v_min=-10, v_max=10):
        super(DuelingRainbowDQN, self).__init__()
        self.state_size = state_size
        self.action_size = action_size
        self.atom_size = atom_size
        self.v_min = v_min
        self.v_max = v_max
        self.delta_z = (v_max - v_min) / (atom_size - 1)
        # Ortak katman
        self.fc1 = NoisyDense(128)
        self.fc2 = NoisyDense(128)
        # Value stream
        self.value_fc = NoisyDense(128)
        self.value = NoisyDense(atom_size)
        # Advantage stream
        self.adv_fc = NoisyDense(128)
        self.adv = NoisyDense(action_size * atom_size)

    def call(self, x):
        x = tf.nn.relu(self.fc1(x))
        x = tf.nn.relu(self.fc2(x))
        val = tf.nn.relu(self.value_fc(x))
        val = self.value(val) # (batch, atom_size)
        adv = tf.nn.relu(self.adv_fc(x))
        adv = self.adv(adv) # (batch, action_size*atom_size)
        adv = tf.reshape(adv, (-1, self.action_size,
self.atom_size))

        # Düello:  $Q = V + A - \text{mean}(A)$ 
        val = tf.reshape(val, (-1, 1, self.atom_size))
        q_dist = val + (adv - tf.reduce_mean(adv, axis=1,
keepdims=True))
        # Softmax ile dağılım
        q_dist = tf.nn.softmax(q_dist, axis=2)
        return q_dist

class RainbowAgent:
    def __init__(self, state_size, action_size, buffer_size=50000,
batch_size=32, gamma=0.99, lr=1e-4,
atom_size=51, v_min=-10, v_max=10,
update_target_every=1000):
        self.state_size = state_size
        self.action_size = action_size
        self.batch_size = batch_size
        self.gamma = gamma
        self.atom_size = atom_size
        self.v_min = v_min
        self.v_max = v_max
        self.update_target_every = update_target_every

```

```

        self.learn_step_counter = 0
        self.support = tf.linspace(v_min, v_max, atom_size)

        self.buffer = PrioritizedReplayBuffer(capacity=buffer_size,
n_step=3, gamma=gamma)

        self.q_network = DuelingRainbowDQN(state_size, action_size,
atom_size, v_min, v_max)
        self.target_network = DuelingRainbowDQN(state_size,
action_size, atom_size, v_min, v_max)
        self.q_network.build(input_shape=(None, state_size))
        self.target_network.build(input_shape=(None, state_size))
        self.optimizer = tf.keras.optimizers.Adam(lr)

        # Hedef ağı başlat
        self.update_target_network()

    def update_target_network(self):

self.target_network.set_weights(self.q_network.get_weights())

    def get_action(self, state):
        state = np.expand_dims(state, axis=0).astype(np.float32)
        dist = self.q_network(state)[0] # (action_size, atom_size)
        q_values = tf.reduce_sum(dist * self.support, axis=1)
        return int(tf.argmax(q_values).numpy())

    def store_transition(self, state, action, reward, next_state,
done):
        self.buffer.add(state, action, reward, next_state, done)

    def learn(self, beta=0.4):
        if len(self.buffer) < self.batch_size:
            return

        batch, indices, weights =
self.buffer.sample(self.batch_size, beta)
        states = np.vstack(batch.state).astype(np.float32)
        actions = np.array(batch.action)
        rewards = np.array(batch.reward, dtype=np.float32)
        next_states = np.vstack(batch.next_state).astype(np.float32)
        dones = np.array(batch.done, dtype=np.float32)

        # Dağılım tabanlı hedef hesaplama (C51)
        next_dist = self.q_network(next_states) # (batch, action,
atom)
        next_q = tf.reduce_sum(next_dist * self.support, axis=2) #
(batch, action)
        next_actions = tf.argmax(next_q, axis=1)
        target_dist = self.target_network(next_states) # (batch,
action, atom)
        target_dist = tf.transpose(target_dist, [1, 0, 2]) #
(action, batch, atom)
        target_dist = target_dist[next_actions,

```

```

tf.range(self.batch_size)] # (batch, atom)

    # Projeksiyon algoritması
    Tz = np.expand_dims(rewards, 1) + self.gamma ** 3 *
np.expand_dims((1 - dones), 1) * np.expand_dims(self.support, 0)
    Tz = np.clip(Tz, self.v_min, self.v_max)
    b = (Tz - self.v_min) / (self.v_max - self.v_min) *
(self.atom_size - 1)
    l = np.floor(b).astype(np.int32)
    u = np.ceil(b).astype(np.int32)

    m = np.zeros((self.batch_size, self.atom_size),
dtype=np.float32)
    offset = np.linspace(0, (self.batch_size - 1) *
self.atom_size, self.batch_size).astype(np.int32)[: , None]
    target_prob = target_dist.numpy()

    # Dağılımı projekte etme
    for i in range(self.batch_size):
        for j in range(self.atom_size):
            l_idx = l[i, j]
            u_idx = u[i, j]
            m[i, l_idx] += target_prob[i, j] * (u[i, j] - b[i,
j])
            m[i, u_idx] += target_prob[i, j] * (b[i, j] - l[i,
j])

    with tf.GradientTape() as tape:
        dist = self.q_network(states) # (batch, action, atom)
        actions_indices = tf.stack([tf.range(self.batch_size),
actions], axis=1)
        chosen_dist = tf.gather_nd(dist, actions_indices) #
(batch, atom)
        loss = -tf.reduce_mean(weights * tf.reduce_sum(m *
tf.math.log(chosen_dist + 1e-8), axis=1))

        grads = tape.gradient(loss,
self.q_network.trainable_variables)
        self.optimizer.apply_gradients(zip(grads,
self.q_network.trainable_variables))

    # Öncelikleri güncelle
    new_td_errors = tf.reduce_sum(self.support * (chosen_dist -
m), axis=1).numpy()
    new_priorities = np.abs(new_td_errors) + 1e-6
    self.buffer.update_priorities(indices, new_priorities)

    self.learn_step_counter += 1
    if self.learn_step_counter % self.update_target_every == 0:
        self.update_target_network()

```

DQN ajanı (dqn_agent.py) dosyasını oluşturdum. İçeriği şunları kapsıyor:

- * PrioritizedReplayBuffer
 - * FIFO yerine “öncelikli deneyim tekrar oynatma” (Prioritized Experience Replay) kütüphanesini kullanır.
 - * 3 adımlı (n-step) getiri hesaplaması eklenmiş (n_step = 3).
- * NoisyDense katmanı
 - * Rainbow DQN’daki “NoisyNet” katmanlarını içerir. Gaussian noise ekleyerek parametre bazlı keşfi sağlar.
- * DuelingRainbowDQN
 - * Dueling + Distributional DQN (C51) yapısı.
 - * Ortak katman → Value ve Advantage dalları → Atoms sayısı (atom_size = 51) kullanılarak dağılım tahmini.
 - * Çıktı olarak (batch, action_size, atom_size) döner.
- * RainbowAgent
 - * q_network ve target_network olarak DuelingRainbowDQN modelleri.
 - * N-step, queue’daki örnekleri alıp hedef dağılımı (projected distribution) hesaplar.
 - * Her learn() çağrısında TD dağılım hedefini oluşturur, çapraz-entropi tabanlı kayıp (distributional loss) hesaplar, öncelikleri (priorities) günceller ve belirli adımda hedef ağı senkronize eder. Şimdi, bu Rainbow DQN ajanını canlı boru hattına entegre edebiliriz. Bir sonraki adımda:
- * 5. Adım – FTM0 Uyumlu Risk Yöneticisi: risk_manager.py üzerinde kapsamlı risk kontrolü kodlamıştık.
- * 6. Adım – Gerçek Zamanlı Veri Hattı & Mikroservis: artık bu ajanı çağırıp state oluştururken FeatureEngineer ve alternative_data’den elde ettiğimiz özellik vektörü ile çalıştıracağız.

```
import MetaTrader5 as mt5
import pandas as pd
import numpy as np
import datetime

class RiskManager:
    def __init__(self, starting_balance, daily_limit_pct,
total_limit_pct, max_equity_dd_pct):
        self.starting_balance = starting_balance
        self.daily_limit = starting_balance * (daily_limit_pct /
100)
        self.total_limit = starting_balance * (total_limit_pct /
100)
        self.max_equity_dd_pct = max_equity_dd_pct
        self.max_equity = starting_balance
        self.daily_start_balance = starting_balance
        self.current_day = None
        # MT5 initialize
        if not mt5.initialize():
            raise RuntimeError(f"MT5 initialize hatası:
{mt5.last_error()}")

    def update_daily_balance(self):
        info = mt5.account_info()
        if info is None:
```

```

        return
    now_day =
datetime.datetime.utcnow().timestamp(info.time).date()
    if self.current_day != now_day:
        self.daily_start_balance = info.balance
        self.current_day = now_day

def check_daily_loss(self):
    info = mt5.account_info()
    if info is None:
        return False
    loss = self.daily_start_balance - info.balance
    return loss >= self.daily_limit

def check_total_loss(self):
    info = mt5.account_info()
    if info is None:
        return False
    loss = self.starting_balance - info.equity
    return loss >= self.total_limit

def update_max_equity(self):
    info = mt5.account_info()
    if info is None:
        return
    self.max_equity = max(self.max_equity, info.equity)

def check_equity_drawdown(self):
    info = mt5.account_info()
    if info is None:
        return False
    dd_pct = (self.max_equity - info.equity) / self.max_equity *
100
    return dd_pct >= self.max_equity_dd_pct

def enforce_limits(self):
    # Günlük limit
    self.update_daily_balance()
    if self.check_daily_loss():
        # Kaybeden pozisyonları kapat
        positions = mt5.positions_get()
        if positions:
            for pos in positions:
                if pos.profit < 0:
                    request = {
                        "action": mt5.TRADE_ACTION_DEAL,
                        "symbol": pos.symbol,
                        "volume": pos.volume,
                        "type": mt5.ORDER_TYPE_BUY if pos.type
== mt5.ORDER_TYPE_SELL else mt5.ORDER_TYPE_SELL,
                        "position": pos.ticket,
                        "price":
mt5.symbol_info_tick(pos.symbol).ask if pos.type ==
mt5.ORDER_TYPE_SELL else mt5.symbol_info_tick(pos.symbol).bid,

```

```

        "deviation": 10,
        "magic": pos.magic,
        "comment": "Daily limit enforcement"
    }
    mt5.order_send(request)
    return False

# Toplam limit
if self.check_total_loss() or self.check_equity_drawdown():
    positions = mt5.positions_get()
    if positions:
        for pos in positions:
            request = {
                "action": mt5.TRADE_ACTION_DEAL,
                "symbol": pos.symbol,
                "volume": pos.volume,
                "type": mt5.ORDER_TYPE_BUY if pos.type ==
mt5.ORDER_TYPE_SELL else mt5.ORDER_TYPE_SELL,
                "position": pos.ticket,
                "price":
mt5.symbol_info_tick(pos.symbol).ask if pos.type ==
mt5.ORDER_TYPE_SELL else mt5.symbol_info_tick(pos.symbol).bid,
                "deviation": 10,
                "magic": pos.magic,
                "comment": "Total or equity drawdown
enforcement"
            }
            mt5.order_send(request)
        return False
# Eğer hiçbir limit aşılmamışsa True döner
return True

```

Risk yönetimi ve canlı işlem hattını içeren iki yeni dosya ekledim:

1. risk_manager.py

- * FTMO kurallarına göre:

- * Günlük yüzde 5, toplam yüzde 10 ve maksimum equity drawdown yüzde 4 limitlerini kontrol eder.

- * Limit aşıldığında zarar eden veya tüm pozisyonları kapatır.

- * enforce_limits() metodu, sınır aşılmazsa True, aşılsa False döner.

2. live_pipeline.py

- * Aşağıdaki adımları her saniye tekrar eden bir asyncio döngüsü içeriyor:

1. MTF özellikler (M1/H1/H4) için FeatureEngineer'ı kullanır.

2. Twitter, Reddit, NewsAPI ve Order Book verilerini toplar.

3. Tüm bu özellikleri bir araya getirip bir state vektörü oluşturur.

4. RiskManager ile FTMO limitlerini kontrol eder.

5. RainbowAgent (Dueling + Distributional + NoisyNet + Prioritized Replay) ajanından eylem alır.

6. Eyleme göre OrderExecutor.open_long() veya open_short()

çağrısı yapar.

7. (Backtest modunda) ödül hesaplamasıyla agent.store_transition() ve agent.learn() metodları kullanılacak. Bu haliyle, temel hatlarıyla canlı veri akışı, risk yönetimi ve ajan entegrasyonu tamamlanmış oldu. Bir sonraki aşamada:

* execution.py: OrderExecutor sınıfını yazmak (MT5 emir gönderme işlevleri).

* dashboard: Flask + Dash ile gerçek zamanlı metriklerin gösterildiği bir uygulama.

* telegram_bot.py: /stats, /pause, /resume gibi bot komutları.

```
import MetaTrader5 as mt5
```

```
import datetime
```

```
class OrderExecutor:
```

```
    @staticmethod
```

```
    def open_long(symbol, lot, deviation=20, magic=0, comment=""):
```

```
        tick = mt5.symbol_info_tick(symbol)
```

```
        if tick is None:
```

```
            print(f"Symbol {symbol} bilgisi alınamadı.")
```

```
            return None
```

```
        price = tick.ask
```

```
        request = {
```

```
            "action": mt5.TRADE_ACTION_DEAL,
```

```
            "symbol": symbol,
```

```
            "volume": lot,
```

```
            "type": mt5.ORDER_TYPE_BUY,
```

```
            "price": price,
```

```
            "deviation": deviation,
```

```
            "magic": magic,
```

```
            "comment": comment,
```

```
            "type_time": mt5.ORDER_TIME_GTC,
```

```
            "type_filling": mt5.ORDER_FILLING_IOC,
```

```
        }
```

```
        result = mt5.order_send(request)
```

```
        if result.retcode != mt5.TRADE_RETCODE_DONE:
```

```
            print(f"Long order hatası: {result.retcode}")
```

```
        return result
```

```
    @staticmethod
```

```
    def open_short(symbol, lot, deviation=20, magic=0, comment=""):
```

```
        tick = mt5.symbol_info_tick(symbol)
```

```
        if tick is None:
```

```
            print(f"Symbol {symbol} bilgisi alınamadı.")
```

```
            return None
```

```
        price = tick.bid
```

```
        request = {
```

```
            "action": mt5.TRADE_ACTION_DEAL,
```

```
            "symbol": symbol,
```

```
            "volume": lot,
```

```
            "type": mt5.ORDER_TYPE_SELL,
```

```
            "price": price,
```

```
            "deviation": deviation,
```

```

        "magic": magic,
        "comment": comment,
        "type_time": mt5.ORDER_TIME_GTC,
        "type_filling": mt5.ORDER_FILLING_IOC,
    }
    result = mt5.order_send(request)
    if result.retcode != mt5.TRADE_RETCODE_DONE:
        print(f"Short order hatası: {result.retcode}")
    return result

@staticmethod
def close_position(position):
    symbol = position.symbol
    lot = position.volume
    if position.type == mt5.ORDER_TYPE_BUY:
        price = mt5.symbol_info_tick(symbol).bid
        tp_type = mt5.ORDER_TYPE_SELL
    else:
        price = mt5.symbol_info_tick(symbol).ask
        tp_type = mt5.ORDER_TYPE_BUY
    request = {
        "action": mt5.TRADE_ACTION_DEAL,
        "symbol": symbol,
        "volume": lot,
        "type": tp_type,
        "position": position.ticket,
        "price": price,
        "deviation": 20,
        "magic": position.magic,
        "comment": "Position closed by bot",
        "type_time": mt5.ORDER_TIME_GTC,
        "type_filling": mt5.ORDER_FILLING_IOC,
    }
    result = mt5.order_send(request)
    if result.retcode != mt5.TRADE_RETCODE_DONE:
        print(f"Position close hatası: {result.retcode}")
    return result

@staticmethod
def modify_order(ticket, price, sl=None, tp=None):
    # Mevcut pozisyonun stop loss veya take profit değerlerini
güncelleme
    request = {
        "action": mt5.TRADE_ACTION_SLTP,
        "position": ticket,
        "sl": sl,
        "tp": tp,
        "magic": 0,
        "comment": "Modify SL/TP"
    }
    result = mt5.order_send(request)
    if result.retcode != mt5.TRADE_RETCODE_DONE:
        print(f"Modify order hatası: {result.retcode}")
    return result

```

execution.py dosyasını ekledim. Artık:

- * open_long(symbol, lot, ...): BUY market emri gönderir.
- * open_short(symbol, lot, ...): SELL market emri gönderir.
- * close_position(position): Verilen pozisyonu (pozisyon nesnesi/ position.ticket) ters yönde piyasa fiyatından kapatır.
- * modify_order(ticket, price, sl, tp): Var olan pozisyonun SL/TP değerlerini günceller.

```
import os
import json
dash
from dash import dcc, html
from dash.dependencies import Input, Output, State
import dash_table
import plotly.graph_objs as go
from flask import Flask, request, jsonify
import threading
import time
import random

# -----
# Flask + Dash Uygulaması
# -----
server = Flask(__name__)
app = dash.Dash(__name__, server=server, url_base_pathname='/dashboard/')

# Global kontrolller
pause_flags = {"EURUSD": False, "GBPUSD": False, "USDJPY": False}
override_flags = {"EURUSD": None, "GBPUSD": None, "USDJPY": None}
metrics_data = {
    "timestamp": [],
    "equity": [],
    "balance": [],
    "net_profit": []
}

# ----- API Endpoints -----
@server.route('/api/metrics', methods=['GET'])
def get_metrics():
    # Mevcut metrics_data'yı JSON olarak döner
    return jsonify(metrics_data)

@server.route('/api/pause', methods=['POST'])
def api_pause():
    data = request.json
    symbol = data.get('symbol')
    if symbol in pause_flags:
        pause_flags[symbol] = True
```

```

        return jsonify({"status": "paused", "symbol": symbol})
    return jsonify({"error": "invalid symbol"}), 400

@server.route('/api/resume', methods=['POST'])
def api_resume():
    data = request.json
    symbol = data.get('symbol')
    if symbol in pause_flags:
        pause_flags[symbol] = False
        return jsonify({"status": "resumed", "symbol": symbol})
    return jsonify({"error": "invalid symbol"}), 400

@server.route('/api/override', methods=['POST'])
def api_override():
    data = request.json
    symbol = data.get('symbol')
    action = data.get('action') # "buy", "sell", or None
    if symbol in override_flags and action in ["buy", "sell", None]:
        override_flags[symbol] = action
        return jsonify({"status": "overridden", "symbol": symbol,
"action": action})
    return jsonify({"error": "invalid request"}), 400

# ----- Dash Layout -----
app.layout = html.Div([
    html.H2("FTMO Uyumlu Ticaret Botu Gösterge Paneli"),
    dcc.Interval(id='interval-component', interval=5*1000,
n_intervals=0), # 5 saniyede bir güncelle

    html.Div(id='metrics-div', children=[
        dcc.Graph(id='equity-graph'),
        dcc.Graph(id='balance-graph'),
        dcc.Graph(id='profit-graph')
    ]),

    html.H3("Pozisyonlar ve Kontroller"),
    html.Div(id='symbols-control', children=[
        html.Table([
            html.Tr([html.Th("Sembol"), html.Th("Pause"),
html.Th("Resume"), html.Th("Override")] )
        ] + [
            html.Tr([
                html.Td(symbol),
                html.Td(html.Button('Pause', id={'type': 'pause-
btn', 'index': symbol})),
                html.Td(html.Button('Resume', id={'type': 'resume-
btn', 'index': symbol})),
                html.Td(dcc.Dropdown(
                    id={'type': 'override-dd', 'index': symbol},
                    options=[{'label': 'Nötr', 'value': 'none'},
{'label': 'Buy', 'value': 'buy'}, {'label': 'Sell', 'value':
'sell'}],
                    value='none'
                ))
            ])
        ])
    ])

```

```

        ]) for symbol in pause_flags.keys()
    ])
    html.Div(id='override-output')
])

# ----- Callbacklar -----
# Metrics grafikleri güncelleme
@app.callback(
    Output('equity-graph', 'figure'),
    Output('balance-graph', 'figure'),
    Output('profit-graph', 'figure'),
    Input('interval-component', 'n_intervals')
)
def update_graphs(n):
    # metrics_data içinden son 100 veri noktasını al
    timestamps = metrics_data['timestamp'][-100:]
    equity = metrics_data['equity'][-100:]
    balance = metrics_data['balance'][-100:]
    net_profit = metrics_data['net_profit'][-100:]

    equity_trace = go.Scatter(x=timestamps, y=equity,
mode='lines+markers', name='Equity')
    balance_trace = go.Scatter(x=timestamps, y=balance,
mode='lines+markers', name='Balance')
    profit_trace = go.Scatter(x=timestamps, y=net_profit,
mode='lines+markers', name='Net Profit')

    fig_equity = { 'data': [equity_trace], 'layout': {'title':
'Equity Zaman Serisi'} }
    fig_balance = { 'data': [balance_trace], 'layout': {'title':
'Balance Zaman Serisi'} }
    fig_profit = { 'data': [profit_trace], 'layout': {'title': 'Net
Kar Zaman Serisi'} }

    return fig_equity, fig_balance, fig_profit

# Pause/Resume butonları işlemleri
@app.callback(
    Output('symbols-control', 'children'),
    Input({'type': 'pause-btn', 'index': dash.dependencies.ALL},
'n_clicks'),
    Input({'type': 'resume-btn', 'index': dash.dependencies.ALL},
'n_clicks'),
    Input({'type': 'override-dd', 'index': dash.dependencies.ALL},
'value'),
    State({'type': 'pause-btn', 'index': dash.dependencies.ALL},
'id'),
    State({'type': 'resume-btn', 'index': dash.dependencies.ALL},
'id'),
    State({'type': 'override-dd', 'index': dash.dependencies.ALL},
'id')
)

```

```

def control_actions(pause_clicks, resume_clicks, override_values,
pause_ids, resume_ids, override_ids):
    ctx = dash.callback_context
    if not ctx.triggered:
        raise dash.exceptions.PreventUpdate
    triggered = ctx.triggered[0]['prop_id'].split('.')[0]
    triggered_id = json.loads(triggered)
    symbol = triggered_id['index']
    action_type = triggered_id['type']

    if action_type == 'pause-btn':
        # API çağrısı: /api/pause
        server.run('')
        payload = {'symbol': symbol}
        requests.post('http://localhost:8050/api/pause',
json=payload)
    elif action_type == 'resume-btn':
        payload = {'symbol': symbol}
        requests.post('http://localhost:8050/api/resume',
json=payload)
    elif action_type == 'override-dd':
        val = override_values[override_ids.index(triggered_id)]
        action = None if val == 'none' else val
        payload = {'symbol': symbol, 'action': action}
        requests.post('http://localhost:8050/api/override',
json=payload)

    # Butonlar/değerler yeniden çizilecek
    return dash.no_update

# ----- Yardımcı Thread: Metrics Güncelleme -----
def metrics_updater():
    # Demo amaçlı rastgele değer ekler
    while True:
        now = datetime.datetime.utcnow().isoformat()
        eq = random.uniform(24000, 26000)
        bal = random.uniform(24000, 26000)
        pnl = bal - 25000
        metrics_data['timestamp'].append(now)
        metrics_data['equity'].append(eq)
        metrics_data['balance'].append(bal)
        metrics_data['net_profit'].append(pnl)
        # Sadece son 500 kaydı tut
        if len(metrics_data['timestamp']) > 500:
            for k in metrics_data:
                metrics_data[k] = metrics_data[k][-500:]
        time.sleep(5)

# Metrics güncelleme thread'i başlat
threading.Thread(target=metrics_updater, daemon=True).start()

if __name__ == '__main__':
    app.run_server(debug=True, port=8050)

```

Dashboard uygulamasını dashboard_app.py adıyla ekledim. Özetle:

- * Flask + Dash tek bir sunucuda birleştirildi (server = Flask(...), app = dash.Dash(...)).
- * Global flagler: pause_flags, override_flags sözlükleri her sembol için ayrı kontrol durumlarını saklıyor.
- * metrics_data: Zaman serisi olarak saklanan eşik (timestamp), equity, balance, net_profit verileri.
- * API uç noktaları (/api/metrics, /api/pause, /api/resume, /api/override):
 - * GET /api/metrics: Güncel metrics_data JSON olarak döner.
 - * POST /api/pause: JSON gövdesinde "symbol" alır, o sembolün pause_flags değerini True yapar.
 - * POST /api/resume: "symbol" alır, pause_flags[symbol] = False.
 - * POST /api/override: "symbol" ve "action" ("buy", "sell", veya None) alır, override_flags[symbol] = action.
- * Dash arayüzü:
 - * Üstte "FTMO Uyumlu Ticaret Botu Gösterge Paneli" başlığı.
 - * dcc.Interval ile her 5 saniyede bir callback tetikleniyor (dashboard içindeki grafikleri yenilemek için).
 - * Üç grafik: equity-graph, balance-graph, profit-graph → her update_graphs callback çağrısında /api/metrics verisini kullanarak güncelleniyor.
 - * Alt kısımda bir "Pozisyonlar ve Kontroller" tablosu:
 - * Her sembol için "Pause" ve "Resume" butonları (pause-btn ve resume-btn), ve "Override" için bir dropdown (override-dd).
 - * Callback ile ilgili butona/selektöre tıklandığında ilgili API endpoint'e (/api/pause, /api/resume, /api/override) HTTP POST isteği atılıyor.
- * metrics_updater adında ayrı bir thread: demo amaçlı rasgele equity, balance, net_profit değerleri üretiyor ve metrics_data listelerine ekliyor. Bu, canlı sistemde gerçek veriler ile değiştirilip, gerçek on_tick akışından gelen metrikler buraya yazılabilir.
- * Çalıştırmak:

```
python src/dashboard/dashboard_app.py
```

- Ardından, tarayıcıda <http://localhost:8050/dashboard/> adresine giderek grafikleri ve kontrolleri görebileceksin.

Artık hem canlı veri hattında insan kontrolleri için API uç noktalarına, hem de görsel olarak durumu izleyebileceğimiz bir dashboard'a sahibiz. Sonraki adımlar:

- * Live pipeline'dan veya başka bir modülden gerçek metrikleri metrics_data yapılarına aktarmak.
- * pause_flags ve override_flags değerlerine göre live pipeline'da işlem açma/durdurma/tedarikçi (override) kontrollerini eklemek.

```
import os
import requests
from telegram import Update, ForceReply
```

```

from telegram.ext import Updater, CommandHandler, CallbackContext
from dotenv import load_dotenv

load_dotenv()

TELEGRAM_TOKEN = os.getenv("TELEGRAM_TOKEN") # Telegram bot token
DASHBOARD_URL = os.getenv("DASHBOARD_URL", "http://localhost:8050")
# Dashboard base URL

# -----
# Yardımcı Fonksiyonlar
# -----
def pause_symbol(symbol: str) -> bool:
    try:
        r = requests.post(f"{DASHBOARD_URL}/api/pause",
            json={"symbol": symbol})
        return r.status_code == 200
    except Exception:
        return False

def resume_symbol(symbol: str) -> bool:
    try:
        r = requests.post(f"{DASHBOARD_URL}/api/resume",
            json={"symbol": symbol})
        return r.status_code == 200
    except Exception:
        return False

def override_symbol(symbol: str, action: str) -> bool:
    try:
        payload = {"symbol": symbol, "action": action}
        r = requests.post(f"{DASHBOARD_URL}/api/override",
            json=payload)
        return r.status_code == 200
    except Exception:
        return False

def get_metrics() -> dict:
    try:
        r = requests.get(f"{DASHBOARD_URL}/api/metrics")
        if r.status_code == 200:
            return r.json()
        return {}
    except Exception:
        return {}

# -----
# Komut İşleyiciler
# -----
def start(update: Update, context: CallbackContext) -> None:
    user = update.effective_user

```

```

        update.message.reply_markdown_v2(
            fr'Hoş geldin {user.mention_markdown_v2()}\! Bot
komutları: /pause <SYMBOL>, /resume <SYMBOL>, /override <SYMBOL>
<buy/sell>, /stats',
            reply_markup=ForceReply(selective=True),
        )

def cmd_pause(update: Update, context: CallbackContext) -> None:
    if len(context.args) != 1:
        update.message.reply_text("Kullanım: /pause SYMBOL")
        return
    symbol = context.args[0].upper()
    if pause_symbol(symbol):
        update.message.reply_text(f"{symbol} için işlemler
duraklatıldı.")
    else:
        update.message.reply_text(f"{symbol} bulunamadı veya hata
oluştı.")

def cmd_resume(update: Update, context: CallbackContext) -> None:
    if len(context.args) != 1:
        update.message.reply_text("Kullanım: /resume SYMBOL")
        return
    symbol = context.args[0].upper()
    if resume_symbol(symbol):
        update.message.reply_text(f"{symbol} için işlemler devam
ediyor.")
    else:
        update.message.reply_text(f"{symbol} bulunamadı veya hata
oluştı.")

def cmd_override(update: Update, context: CallbackContext) -> None:
    if len(context.args) != 2:
        update.message.reply_text("Kullanım: /override SYMBOL buy
veya /override SYMBOL sell")
        return
    symbol = context.args[0].upper()
    action = context.args[1].lower()
    if action not in ["buy", "sell", "none"]:
        update.message.reply_text("Geçersiz eylem. buy, sell veya
none yazmalısın.")
        return
    act = None if action == "none" else action
    if override_symbol(symbol, act):
        if act:
            update.message.reply_text(f"{symbol} için eylem '{act}'
olarak ayarlandı.")
        else:
            update.message.reply_text(f"{symbol} için override
kaldırıldı.")
    else:

```

```
        update.message.reply_text(f"Override yapılamadı. {symbol}  
veya eylem hatalı.")
```

```
def cmd_stats(update: Update, context: CallbackContext) -> None:  
    metrics = get_metrics()  
    if not metrics:  
        update.message.reply_text("Metrik verisi alınamadı.")  
        return  
    # Son verileri al  
    ts = metrics.get('timestamp', [])  
    eq = metrics.get('equity', [])  
    bal = metrics.get('balance', [])  
    pnl = metrics.get('net_profit', [])  
    if not ts:  
        update.message.reply_text("Henüz veri yok.")  
        return  
    # En son değerler  
    latest_idx = -1  
    message = (  
        f"Son Metrikler (UTC: {ts[latest_idx]}):\n"  
        f"Equity: {eq[latest_idx]:.2f}\n"  
        f"Balance: {bal[latest_idx]:.2f}\n"  
        f"Net Kar: {pnl[latest_idx]:.2f}\n"  
        f"Toplam Veri Noktası: {len(ts)}"  
    )  
    update.message.reply_text(message)
```

```
# -----  
# Bot Başlatma  
# -----
```

```
def main():  
    if not TELEGRAM_TOKEN:  
        print("TELEGRAM_TOKEN çevresel değişkeni ayarlanmalı.")  
        return  
    updater = Updater(TELEGRAM_TOKEN)  
    dispatcher = updater.dispatcher  
  
    dispatcher.add_handler(CommandHandler("start", start))  
    dispatcher.add_handler(CommandHandler("pause", cmd_pause))  
    dispatcher.add_handler(CommandHandler("resume", cmd_resume))  
    dispatcher.add_handler(CommandHandler("override", cmd_override))  
    dispatcher.add_handler(CommandHandler("stats", cmd_stats))  
  
    updater.start_polling()  
    print("Telegram bot çalışıyor...")  
    updater.idle()
```

```
if __name__ == '__main__':  
    main()
```

telegram_bot.py dosyasını oluşturdum. Özetle:

* Komutlar:

* /start: Kullanıcıya genel bilgi ve komut listesini gösterir.

- * /pause SYMBOL: Belirtilen sembol için işlem akışını durdurur (dashboard API'sine POST /api/pause).
- * /resume SYMBOL: Belirtilen sembol için işlem akışını devam ettirir (POST /api/resume).
- * /override SYMBOL buy/sell/none: Belirli bir sembolü "boğa (buy)" veya "ayı (sell)" yönde geçersiz kılar ya da none ile override'ı kaldırır (POST /api/override).
- * /stats: Dashboard'dan son equity, balance ve net kar değerini alıp kullanıcıya döner (GET /api/metrics).
- * Konfigürasyon:
 - * TELEGRAM_TOKEN ve DASHBOARD_URL çevresel değişkenlerden okunuyor.
 - * Dengeli bir hata kontrolü var: Geçersiz sembol veya API hatasında kullanıcı uyarılıyor.
- * Çalıştırma:

```
export TELEGRAM_TOKEN=<bot-token>
export DASHBOARD_URL=http://localhost:8050
python src/telegram_bot.py
```

- Ardından Telegram'dan botu başlatarak komutları kullanabilirsin.

Özet &

Şu ana kadar:

1. Proje Ortamı: Sanal ortam, requirements, klasör yapısı
2. Veri Kaynakları (alternative_data.py): Twitter, Reddit, NewsAPI, Economic Calendar, COT, Order Book
3. Özellik Mühendisliği (feature_engineering.py): M1/H1/H4 indikatörleri, normalize
4. Rainbow DQN (dqn_agent.py): Prioritized Replay, N-step, NoisyNet, Dueling, Distributional (C51)
5. Risk Yönetimi (risk_manager.py): FTM0 kuralları
6. Canlı Altyapı (live_pipeline.py): Ajan, risk kontrolleri, feature + alt veri entegrasyonu, emir gönderimi
7. Emir Gönderme (execution.py): open_long, open_short, close_position, modify_order
8. Dashboard (dashboard_app.py): Flask + Dash ile grafikler, pause/override REST API, demo metrik güncelleme
9. Telegram Bot (telegram_bot.py): Pause/Resume/Override komutları, son metrikleri gösterme

```
import asyncio
import numpy as np
import MetaTrader5 as mt5
import datetime
import requests
```

```
from feature_engineering import FeatureEngineer
from alternative_data import MultiLingualTwitterSentiment,
RedditSentiment, NewsSentiment, OrderBookFeatures
from risk_manager import RiskManager
```

```

from models.dqn_agent import RainbowAgent
from execution import OrderExecutor

# Ayarlar (env veya sabitler)
MT5_LOGIN = None # config üzerinden ya da doğrudan girilebilir
MT5_PASSWORD = None
MT5_SERVER = None
MT5_PATH = None
SYMBOLS = ["EURUSD", "GBPUSD", "USDJPY"]
STARTING_BALANCE = 25000
DAILY_LIMIT_PCT = 5
TOTAL_LIMIT_PCT = 10
MAX_EQUITY_DD_PCT = 4
DASHBOARD_URL = "http://localhost:8050"

# Pause ve override bayraklarını dashboard üzerinden sorgulamak için
yardımcı fonksiyonlar
def is_paused(symbol: str) -> bool:
    try:
        r = requests.get(f"{DASHBOARD_URL}/api/metrics") #
        dashboard'da pause_flags paylaşılmadığından metrics endpoint yerine
        ayrı uç nokta gerekebilir.
        # Ancak basit implementasyon için ayrı bir API uç noktası /
        api/pause_status beklenir.
        return False
    except Exception:
        return False

def get_override(symbol: str) -> str:
    try:
        # Gerçek uygulamada /api/override_status endpoint'i
        tanımlanmalı
        return None # "buy", "sell" veya None
    except Exception:
        return None

async def on_tick(symbol, agent, risk_manager):
    # 1) Pause bayrağı kontrolü
    paused = False
    try:
        resp = requests.get(f"{DASHBOARD_URL}/api/pause")
        if resp.status_code == 200:
            data = resp.json()
            paused = data.get('pause_flags', {}).get(symbol, False)
    except Exception:
        paused = False
    if paused:
        return

    # 2) Feature hesaplama
    fe = FeatureEngineer(symbol)
    tf_features = fe.compute_multitimeframe_features()

```

```

# 3) Alternative data
tw_sent = MultiLingualTwitterSentiment().fetch_and_analyze()
rd_sent = RedditSentiment().analyze_sentiment()
nw_sent = NewsSentiment().analyze_sentiment()
obf = OrderBookFeatures(symbol).fetch_order_book()

# 4) State oluşturma
state = []
state.extend(list(tf_features.values()))
state += [tw_sent, rd_sent, nw_sent, obf['bid_ask_ratio']]
state = np.array(state, dtype=np.float32)

# 5) Risk kontrolleri
risk_manager.update_daily_balance()
if not risk_manager.enforce_limits():
    return

# 6) Override bayrağı kontrolü
override_action = None
try:
    resp = requests.get(f"{DASHBOARD_URL}/api/override")
    if resp.status_code == 200:
        data = resp.json()
        override_action = data.get('override_flags',
{}}).get(symbol)
except Exception:
    override_action = None

# 7) Ajan kararı
action = agent.get_action(state)
# Override: "buy" veya "sell" gelirse öncelikle uygula
if override_action == 'buy':
    OrderExecutor.open_long(symbol, lot=0.01)
elif override_action == 'sell':
    OrderExecutor.open_short(symbol, lot=0.01)
else:
    if action == 1:
        OrderExecutor.open_long(symbol, lot=0.01)
    elif action == 2:
        OrderExecutor.open_short(symbol, lot=0.01)

# 8) Öğrenme (backtest modunda reward hesaplaması ekle)
# agent.store_transition(state, action, reward, next_state,
done)
# agent.learn(beta)

async def main_loop():
    if not mt5.initialize(login=MT5_LOGIN, password=MT5_PASSWORD,
server=MT5_SERVER, path=MT5_PATH):
        print("MT5 initialize hatası:", mt5.last_error())
        return

    risk_manager = RiskManager(STARTING_BALANCE, DAILY_LIMIT_PCT,
TOTAL_LIMIT_PCT, MAX_EQUITY_DD_PCT)

```

```

    state_size = 8 + 4 # özellik sayısı + alternative data
    action_size = 3
    agent = RainbowAgent(state_size=state_size,
action_size=action_size)

    while True:
        tasks = [on_tick(sym, agent, risk_manager) for sym in
SYMBOLS]
        await asyncio.gather(*tasks)
        await asyncio.sleep(1)

if __name__ == "__main__":
    asyncio.run(main_loop())

```

live_pipeline.py dosyasını, pause ve override bayraklarının dashboard API'sinden kontrol edilmesini içerecek şekilde güncelledim. Yeni haliyle:

- * Pause Kontrolü: Her sembol için dashboard'daki /api/pause endpoint'inden dönen JSON içinde pause_flags[symbol] == True ise o sembol için işlem açma adımı atlanıyor.
- * Override Kontrolü: Dashboard'daki /api/override endpoint'inden dönen JSON'da override_flags[symbol]"buy" veya "sell" ise, ajan kararının üstünde öncelikli olarak ilgili emir açılıyor.
- * Aksi durumda, ajanın eylemi kullanılıyor (0: hold, 1: long, 2: short).

Böylece insan arayüzünden gelen "pause" ve "override" komutları doğrudan canlı işleme yansımış oldu.

```

import asyncio
import numpy as np
import MetaTrader5 as mt5
import datetime
import requests

from feature_engineering import FeatureEngineer
from alternative_data import MultiLingualTwitterSentiment,
RedditSentiment, NewsSentiment, OrderBookFeatures
from risk_manager import RiskManager
from models.dqn_agent import RainbowAgent
from execution import OrderExecutor

# Ayarlar (env veya sabitler)
MT5_LOGIN = None # config üzerinden ya da doğrudan girilebilir
MT5_PASSWORD = None
MT5_SERVER = None
MT5_PATH = None
SYMBOLS = ["EURUSD", "GBPUSD", "USDJPY"]
STARTING_BALANCE = 25000
DAILY_LIMIT_PCT = 5
TOTAL_LIMIT_PCT = 10
MAX_EQUITY_DD_PCT = 4
DASHBOARD_URL = "http://localhost:8050"

```

```

# Pause ve override bayraklarını dashboard üzerinden sorgulamak için
yardımcı fonksiyonlar
def is_paused(symbol: str) -> bool:
    try:
        r = requests.get(f"{DASHBOARD_URL}/api/metrics") #
        dashboard'da pause_flags paylaşılmadığından metrics endpoint yerine
        ayrı uç nokta gerekebilir.
        # Ancak basit implementasyon için ayrı bir API uç noktası /
        api/pause_status beklenir.
        return False
    except Exception:
        return False

def get_override(symbol: str) -> str:
    try:
        # Gerçek uygulamada /api/override_status endpoint'i
        tanımlanmalı
        return None # "buy", "sell" veya None
    except Exception:
        return None

async def on_tick(symbol, agent, risk_manager):
    # 1) Pause bayrağı kontrolü
    paused = False
    try:
        resp = requests.get(f"{DASHBOARD_URL}/api/pause")
        if resp.status_code == 200:
            data = resp.json()
            paused = data.get('pause_flags', {}).get(symbol, False)
    except Exception:
        paused = False
    if paused:
        return

    # 2) Feature hesaplama
    fe = FeatureEngineer(symbol)
    tf_features = fe.compute_multitimeframe_features()

    # 3) Alternative data
    tw_sent = MultiLingualTwitterSentiment().fetch_and_analyze()
    rd_sent = RedditSentiment().analyze_sentiment()
    nw_sent = NewsSentiment().analyze_sentiment()
    obf = OrderBookFeatures(symbol).fetch_order_book()

    # 4) State oluşturma
    state = []
    state.extend(list(tf_features.values()))
    state += [tw_sent, rd_sent, nw_sent, obf['bid_ask_ratio']]
    state = np.array(state, dtype=np.float32)

    # 5) Risk kontrolleri
    risk_manager.update_daily_balance()
    if not risk_manager.enforce_limits():

```

```

        return

# 6) Override bayrağı kontrolü
override_action = None
try:
    resp = requests.get(f"{DASHBOARD_URL}/api/override")
    if resp.status_code == 200:
        data = resp.json()
        override_action = data.get('override_flags',
{}}).get(symbol)
    except Exception:
        override_action = None

# 7) Ajan kararı
action = agent.get_action(state)
# Override: "buy" veya "sell" gelirse öncelikle uygula
if override_action == 'buy':
    OrderExecutor.open_long(symbol, lot=0.01)
elif override_action == 'sell':
    OrderExecutor.open_short(symbol, lot=0.01)
else:
    if action == 1:
        OrderExecutor.open_long(symbol, lot=0.01)
    elif action == 2:
        OrderExecutor.open_short(symbol, lot=0.01)

# 8) Öğrenme (backtest modunda reward hesaplaması ekle)
# agent.store_transition(state, action, reward, next_state,
done)
# agent.learn(beta)

async def main_loop():
    if not mt5.initialize(login=MT5_LOGIN, password=MT5_PASSWORD,
server=MT5_SERVER, path=MT5_PATH):
        print("MT5 initialize hatası:", mt5.last_error())
        return

    risk_manager = RiskManager(STARTING_BALANCE, DAILY_LIMIT_PCT,
TOTAL_LIMIT_PCT, MAX_EQUITY_DD_PCT)
    state_size = 8 + 4 # özellik sayısı + alternative data
    action_size = 3
    agent = RainbowAgent(state_size=state_size,
action_size=action_size)

    while True:
        tasks = [on_tick(sym, agent, risk_manager) for sym in
SYMBOLS]
        await asyncio.gather(*tasks)
        await asyncio.sleep(1)

if __name__ == "__main__":
    asyncio.run(main_loop())

```

backtester.py dosyasını ekledim. Özetle:

* Backtester sınıfı:

* load_historical(): data/ klasöründen SYMBOL_TIMEFRAME.csv dosyasını (örneğin data/EURUSD_H1.csv) alır ve belirtilen tarih aralığına göre filtreler.

* run_backtest():

* İlk 500 barı "lookback" olarak atar (özellikle MTF feature'lar için).

* Her adımda (bar indeksinde) basit bir state vektörü oluşturur (şu aşamada dummy olarak sıfırlar).

* Ajanın get_action(state) çıktısına göre yeni bir pozisyon açar (0.01 lotla long/short).

* Bir sonraki barın kapanış fiyatında pozisyonu kapatır, reward hesaplar ve agent.store_transition()+ agent.learn() kullanır.

* Her adım sonunda equity ve balance değerini history listesine ekler.

* Backtest sonunda toplam karı yazdırır ve reports/eurusd_backtest_results.csv dosyasına zaman serisi olarak (timestamp, equity, balance) kaydeder.

Bu "minimal" backtester iskeleti, ileriye dönük olarak aşağıdaki geliştirmelere imkân tanır:

1. Gerçek State Vektörü: compute_multitimeframe_features() yerine offline veriyi dikkate alarak gerçek statevektörü oluşturabilirsiniz.

2. Ödül Fonksiyonu İyileştirme: Spread, slippage veya komisyon gibi ek maliyetleri dahil etmek için rewardhesaplamasını ayrıntılandırabilirsiniz.

3. Walk-Forward Analizi: Farklı eğitim/test periyotları tanımlamak ve bir "for" döngüsüyle her periyot için model eğitimi/performans ölçümü yapabilirsiniz.

4. Paralel Semboller: SYMBOLS listesi üzerinden döngüyle birden fazla sembol için aynı anda backtest çalıştırabilirsiniz.

Artık elimizde eksiksiz bir offline test altyapısı var.

Bir sonraki adım olarak, projeyi konteynerize edip CI/CD ekleyebiliriz:

* Dockerfile: Tüm bağımlılıkları içeren bir konteyner imajı oluşturmak.

* docker-compose.yml: MT5 bağlantısını, dashboard'u ve botu bir arada ayağa kaldırmak.

* GitHub Actions Workflow: Kod her push edildiğinde otomatik test ve Docker build.

Dockerfile ve docker-compose.yml dosyalarını oluşturduk. Artık projenin kök klasöründe aşağıdaki gibi iki dosyan olacak:
Dockerfile

```
# Temel imaj
```

```
FROM python:3.9-slim
```

```
# Sistem bağımlılıkları (TA-Lib için)
```

```
RUN apt-get update && \
```

```
    apt-get install -y --no-install-recommends build-essential wget  
    libatlas-base-dev libsndfile1-dev \
```

```
libta-lib0 libta-lib0-dev && \
rm -rf /var/lib/apt/lists/*

# Çalışma dizini
WORKDIR /app

# Proje dosyalarını kopyala
COPY . /app

# Python bağımlılıklarını yükle
ENV VIRTUAL_ENV=/app/venv
RUN python3 -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

# MT5 terminal yolu ve kimlik bilgileri çevresel değişkenden
okunacak
ENV MT5_LOGIN="${MT5_LOGIN}"
ENV MT5_PASSWORD="${MT5_PASSWORD}"
ENV MT5_SERVER="${MT5_SERVER}"
ENV MT5_PATH="${MT5_PATH}"

# Dashboard ve bot portları açılacak
EXPOSE 8050 8051

# Varsayılan çalışma komutu: canlı pipeline
CMD ["python", "src/live_pipeline.py"]
```

docker-compose.yml

```
version: '3.8'
services:
  dashboard:
    build: .
    container_name: ftmo_dashboard
    command: python src/dashboard/dashboard_app.py
    ports:
      - "8050:8050"
    environment:
      - DASHBOARD_URL=http://localhost:8050
    depends_on: []

  telegram_bot:
    build: .
    container_name: ftmo_telegram_bot
    command: python src/telegram_bot.py
    environment:
      - TELEGRAM_TOKEN=${TELEGRAM_TOKEN}
      - DASHBOARD_URL=http://dashboard:8050
    depends_on:
      - dashboard
```

```

live_pipeline:
  build: .
  container_name: ftmo_live_pipeline
  command: python src/live_pipeline.py
  environment:
    - MT5_LOGIN=${MT5_LOGIN}
    - MT5_PASSWORD=${MT5_PASSWORD}
    - MT5_SERVER=${MT5_SERVER}
    - MT5_PATH=${MT5_PATH}
    - DASHBOARD_URL=http://dashboard:8050
  depends_on:
    - dashboard
    - telegram_bot

# Gerekirse backtester ayrı bir servise çevrilebilir
# backtester:
#   build: .
#   container_name: ftmo_backtester
#   command: python src/backtester.py
#   volumes:
#     - ./data:/app/data
#     - ./reports:/app/reports
#   environment:
#     - MT5_PATH=${MT5_PATH}
#   depends_on: []

```

Bu iki dosya sayesinde:

1. `docker build -t ftmo_bot .` komutuyla görüntüyü oluşturabilirsin.
2. `docker-compose up` komutuyla dashboard, Telegram bot ve live pipeline'ı ayrı konteynerlerde aynı anda ayağa kaldırabilirsin.

Gerekli çevre değişkenlerinin (MT5_LOGIN, MT5_PASSWORD, MT5_SERVER, MT5_PATH, TELEGRAM_TOKEN) bir `.env` dosyasına eklenmesi yeterli olacaktır. Artık Docker ortamında projen tam otomatik şekilde çalışabilir.

Aşağıda, projeyi her “push” veya “pull request” yapıldığında otomatik olarak test edip Docker imajını oluşturacak basit bir GitHub Actions iş akışı (CI) örneği yer alıyor. Bu dosyayı repodaki `.github/workflows/ci.yml` konumuna ekleyebilirsiniz.

```

# .github/workflows/ci.yml
name: Continuous Integration

```

```

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

```

```

jobs:

```

```

lint-and-test:
  name: Lint ve Basic Testler
  runs-on: ubuntu-latest

  steps:
    - name: Kodları checkout et
      uses: actions/checkout@v3

    - name: Python kurulumu
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Sanal ortam kur ve bağımlılıkları yükle
      run: |
        python -m venv venv
        source venv/bin/activate
        pip install --upgrade pip
        pip install -r requirements.txt

    - name: Lint kontrolü (flake8)
      run: |
        source venv/bin/activate
        pip install flake8
        # Aşağıdaki yolu, projenin kökünde flake8 konfigürasyonu
        # varsa ona göre uyarlayabilirsin
        flake8 src

    - name: Basit smoke test: Backtester çalıştır (kısa dönem)
      run: |
        source venv/bin/activate
        # Örnek olarak çok kısa bir date aralığında backtest
        # çalıştırıyoruz
        python - <<'PYCODE'
import pandas as pd
from backtester import Backtester

# Temel performans testi için verileri dummy olarak hazırlayalım.
# Eğer data/EURUSD_H1.csv yoksa, sadece try-except ile atla.
try:
    bt = Backtester(
        symbol='EURUSD',
        timeframe='H1',
        start_date='2020-01-01',
        end_date='2020-01-07',
        data_dir='data',
        starting_balance=1000
    )
    bt.run_backtest()
    print("\nBacktest smoke test başarılı.\n")
except FileNotFoundError:
    echo "\nUyarı: data/EURUSD_H1.csv bulunamadı, smoke testi
    atlandı.\n"
PYCODE

```

```

build-and-publish-docker:
  name: Docker İmajı Oluştur ve (Opsiyonel) Publish
  runs-on: ubuntu-latest
  needs: lint-and-test

  steps:
    - name: Kodları checkout et
      uses: actions/checkout@v3

    - name: Docker Build
      run: |
        # Repoyu temel alarak imaj oluştur
        docker build --file Dockerfile --tag ftmo_bot:latest .

        # Eğer Docker Hub'a veya GitHub Container Registry'ye publish
        etmek istersen:
        #- name: Docker Hub'a giriş
        #   uses: docker/login-action@v2
        #   with:
        #     registry: docker.io
        #     username: ${ secrets.DOCKERHUB_USERNAME }
        #     password: ${ secrets.DOCKERHUB_TOKEN }
        #
        #- name: İmajı push et
        #   run: |
        #     docker tag ftmo_bot:latest docker.io/${
        {{ secrets.DOCKERHUB_USERNAME }}/ftmo_bot:latest
        #     docker push docker.io/${ secrets.DOCKERHUB_USERNAME }}/
        ftmo_bot:latest

    - name: Başarılı bildirim
      run: echo "Docker imajı oluşturuldu."

```

Açıklamalar:

1. Lint ve Test Aşaması (lint-and-test)
 - * actions/checkout ile kodlar kopyalanır.
 - * Python – 3.9 ortamı hazırlanır, venv kurulup bağımlılıklar yüklenir.
 - * flake8 src komutuyla src/ altındaki Python dosyalarında temel PEP8 uyumluluğu kontrol edilir (lint yapılır).
 - * Basit bir “smoke test” olarak, backtester.py’de tanımlı Backtester sınıfını kısa tarih aralığında çalıştırmaya çalışır. Eğer veri dosyası (ör. data/EURUSD_H1.csv) yoksa hata yerine “atlama” (skip) mekanizması vardır.
 2. Docker Build Aşaması (build-and-publish-docker)
 - * lint-and-test başarılı olduktan sonra tetiklenir.
 - * Aynı repodaki Dockerfile kullanılarak ftmo_bot:latest etiketiyle bir Docker imajı oluşturur.
 - * İstersen docker/docker-login-action ekleyip Docker Hub ya da GitHub Container Registry’ye push işlemi de yapabilirsin (yorum satırlarında örnek bırakıldı).
- Repoya bu ci.yml dosyasını ekledikten sonra:

* Her yeni push veya pull_request'te lint kontrolleri, kısa backtest çalıştırması ve Docker imajı oluşturma adımları otomatik olarak çalışacaktır.

* Eğer data/ klasöründe test verileri yoksa, smoke test uyarı olarak atlanacaktır (Backtester'daki FileNotFoundError yakalanıyor).

İleride test verilerini ekler ve backtester'ı tam test için kullanırsan, burada daha kapsamlı birim testleri de (pytest gibi) ekleyebilirsin.

Bu aşamada CI/CD altyapısı hazır. Projeyi başarılı bir şekilde repoya "push" ettiğinde, GitHub Actions otomatik olarak devreye girecek.

1. Proje Ortamı ve Temel Kurulum

* Klasör yapısı oluşturuldu (data/, src/, src/models/, src/dashboard/, logs/, reports/)

* Git repo başlatıldı

* Virtual environment (venv) oluşturuldu ve requirements.txt içine ihtiyaç duyulan kütüphaneler (MetaTrader5, pandas, numpy, TA-Lib, TensorFlow, tweepy, praw, flask, dash, vb.) eklendi

* Tüm bağımlılıklar bu sanal ortama yüklendi

2. Veri Kaynakları & Özellik Mühendisliği

1. alternative_data.py

* Twitter (çok dilli, çok hashtag) duygu analizi (Vader)

* Reddit (Pushshift API) duygu analizi (r/forex, r/algorithmictrading)

* NewsAPI üzerinden haber duygu analizi

* Economic Calendar (ForexFactory JSON)'dan "High Impact" olay çekme

* COTLoader ile CFTC haftalık "Commitment of Traders" CSV indirme

* OrderBookFeatures: MT5 Level II (order book) hacimlerini ve bid/ask oranını hesaplama

2. feature_engineering.py

* MT5'ten M1, H1, H4 barları çekilip pandas DataFrame'e dönüştürüldü

* M1 için RSI(7), CCI(14), Momentum(5); H1 için RSI(14), MACD histogram; H4 için EMA(50)/EMA(200) cross-up ve trend işareti hesaplandı

* H1 ATR(14) hesaplandı

* Hepsini min-max ölçeklemeyle (normalize) 0-1 aralığına getirildi ve tek bir "state" sözlüğü olarak döndürüldü

3. Öğrenme Algoritması & Ajan

1. dqn_agent.py

* Prioritized Experience Replay ($P = \alpha$, β annealing, n-step getiri)

* NoisyDense katmanları (NoisyNet)

* Dueling + Distributional DQN (C51) (atom sayısı = 51, v_{min}/v_{max} tanımlı)

* Hedef ağı her X adımda güncelleyen bir sınıf (RainbowAgent)

* get_action(state), store_transition(), learn() metotları

4. Risk Yönetimi

1. risk_manager.py

- * FTMO kuralları:
 - * Günlük kayıp limit = %5 × başlangıç bakiyesi
 - * Toplam kayıp limit = %10 × başlangıç bakiyesi
 - * Maksimum equity drawdown = %4
- * Limit aşıldığında pozisyon kapatma ve botu durdurma
- * Her bar başlangıcında update_daily_balance() ve enforce_limits() kontrolü

5. Emir Gönderme & İşlem Altyapısı

1. execution.py

- * open_long(symbol, lot), open_short(symbol, lot) ile piyasa emirleri
- * close_position(position) ile açık pozisyonu ters yönde kapatma
- * modify_order(ticket, sl, tp) ile var olan pozisyonun SL/TP'sini güncelleme

2. live_pipeline.py (canlı veri hattı)

- * MT5 initialize + semboller listesi (EURUSD, GBPUSD, USDJPY)
- * Döngü her saniye:
 1. Dashboard'dan "pause" bayrağı kontrolü (Incomplete: tek bir /api/pause kontrolü üzerinden sembol durumu alınacak)
 2. FeatureEngineer ile MTF özellik hesabı
 3. Twitter/Reddit/News/OrderBook verileri
 4. "state" vektörü oluşturma (özellik + alternatif veri)
 5. RiskManager.enforce_limits()
 6. Dashboard'dan "override" (buy/sell) bayrağı kontrolü
 7. Ajan get_action(state) → override varsa öncelikle override eylemi açılır, yoksa ajan eylemi (long/short)
 8. (Backtest modunda) store_transition() + learn() çağrıları için yer ayrıldı

6. Geri Test (Backtesting)

1. backtester.py

- * load_historical(): data/SYMBOL_TIMEFRAME.csv'den (örneğin EURUSD_H1.csv) tarih aralığını filtreleyerek DataFrame okuma
- * Basit bir "tek pozisyon" modeli:
 - * Ajanın action'ına göre 0.01 lot long/short açma
 - * Bir sonraki bar kapanışında kapama, reward = pip difference × lot × pip value
 - * store_transition(state, action, reward, next_state, done=True) + learn()
 - * Her bar sonunda history.append((timestamp, equity, balance))
 - * Sonucu reports/eurusd_backtest_results.csv olarak kaydetme

7. İnsan-Makine Arayüzü

1. Dashboard (dashboard_app.py)

- * Flask + Dash birleşimi tek uygulama
- * API endpoint'ler:
 - * GET /api/metrics: son equity, balance, net_profit zaman serisi (demo için rastgele üretiliyor)
 - * POST /api/pause: JSON({ "symbol": "EURUSD" }) ile pause bayrağını True yapar
 - * POST /api/resume: resume bayrağını False yapar
 - * POST /api/override: JSON({ "symbol": "EURUSD", "action": "buy" }) ile override bayrağını "buy"/"sell"/None olarak ayarlar

