

## **Genel Bakış: Botu Parçalara Ayıralım**

### **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 duygusal 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'de 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 aşılırsa 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**
  - Veri Toplama Servisi:** Tick/M1/H1/H4 verilerini toplar, state\_queue'ya yollar
  - Özellik Mühendisliği Servisi:** Kuyruktan aldığı raw veriyi işler, MTF feature vektörü oluşturur
  - RL Agent Servisi:** State vektörüne göre action (buy/sell/hold) üretir, order\_queueye yazar
  - Order Execution Servisi:** order\_queuedan komutları okur, mt5.order\_send ile emirleri geçer
  - Risk Manager Servisi:** Her açık pozisyonda equity/drawdown/ vaR kontrolü yapar, gereklirse 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 LONGLIVES")
- **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 aşındı", "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, göstergeler 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 hazırlım, 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 oluşturuktan 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
                                        # özelleştirilebilir

# 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 duygusal analizi var. İlk olarak, o dosyayı biraz genişleteceğiz.

### 2.1. Twitter Duygu Analizini Çok Dilli & Çok Hashtag'lı 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/algorithmic\_trading subreddit'lerinden veri çekebiliriz. Basit bir örnek:

```
import requests
import datetime

class RedditSentiment:
    def __init__(self):
        self.subreddits = ["forex", "algorithmic_trading"]
        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 edebilirsin.
- 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.ORDER_TYPE_SELL])
            asks = sum([entry.volume for entry in book if entry.type ==
mt5.ORDER_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 volatilite” veya “kur koruma” sinyali olarak state'e ekleyebilirsin.

## 3 Adım: Çok Zamanlı ve Çok Ölçülü 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
- **Distribütüyonel DQN (C51):** Q(s,a) yerine dağılım tahmini

#### 4.1. Basit "Dueling + Double DQN" İskeli

[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, distribütüyonel DQN, C51, kâr kitleme mekanizması, vb. daha sonra eklenecek. Bu iskelet, kod akışını anlaman için başlangıç.

## 5 Adım: FTMO 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() çağrıracagız. Eğer False dönerse, yeni pozisyon açılmasını 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).fetch_
    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 çalıştırtıp "state"leri doğru hesapladığından emin olmalısın.