1️⃣ 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 reposu 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:

FTMO Uyumlu Forex Ticaret Botu

Bu proje, FTMO 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/algotrading 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 (Birleştirilmiş). 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 adımda 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 ağ (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 - 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 – FTMO 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.utcfromtimestamp(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, aşılırsa 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 oluştu.")


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 oluştu.")


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): FTMO 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