Sağladığınız kod, bir ticaret botu için kapsamlı bir ortam (`TicaretOrtami`) ve eğitim/test sürecini içeriyor. Önceki loglar ve tartışmalar ışığında, kodun genel yapısını, loglarda görülen sorunların (özellikle marjin çağrıları, düşük PNL, Sharpe oranı hataları) kaynaklarını ve iyileştirme önerilerini inceleyeceğim. Kod oldukça karmaşık, bu yüzden doğrudan loglarla ilgili sorunlara odaklanarak, önerilerimi yapılandırılmış bir şekilde sunacağım.

Kod Analizi ve Loglarla İlişkili Sorunların Kaynakları

1. **Marjin Çağrıları ve Yüksek Lot Büyüklükleri**:
   - **Sorun**: Loglarda USDJPY işlemlerinin sürekli ≈0.051 lot ile açıldığı ve hemen marjin çağrısı nedeniyle kapandığı görülüyor. Bu, `_lot_buyuklugu_hesapla` fonksiyonunun teminatı zorlayacak şekilde lot büyüklükleri ürettiğini gösteriyor.
   - **Koddaki İlgili Kısım**:
     ```python
     def _lot_buyuklugu_hesapla(self, sembol: str, atr: float, sinyal_gucu: float) -> float:
         risk_per_trade = self.alt_bakiyeler[sembol] * self.risk_orani
         pip_sizes = {'EURUSD': 0.0001, 'GBPUSD': 0.0001, 'USDJPY': 0.01}
         pip_size = pip_sizes.get(sembol, 0.0001)
         current_price = self.df_full_dict[sembol]['close'].iloc[self.mevcut_adim]
         pip_value = pip_size * 100000
         if sembol.endswith("JPY"):
             pip_value = pip_value / current_price
         atr = max(atr, np.nanmean(self.ozellikler_dict[sembol]['atr'].iloc[-20:]) or pip_size)
         stop_loss_pips = (atr * 2) / pip_size
         lot_size = risk_per_trade / (stop_loss_pips * pip_value)
         lot_size = max(0.01, min(lot_size * sinyal_gucu, MAX_LOT * 0.5))
         return lot_size
     ```
     - **Tespitler**:
       - **Pip Değeri Hesaplama**: USDJPY için `pip_value = pip_size * 100000 / current_price` doğru bir yaklaşım, ancak `current_price` (örneğin, 121.46000) büyük olduğundan, pip değeri (örneğin, 0.01 * 100000 / 121.46 ≈ 8.23 USD) yüksek oluyor. Bu, lot büyüklüğünü artırıyor.
       - **Stop-Loss Mesafesi**: `stop_loss_pips = (atr * 2) / pip_size` ile hesaplanan stop-loss mesafesi, ATR yüksek olduğunda (örneğin, 1.0) 200 pip gibi büyük bir değere ulaşıyor. Bu, lot büyüklüğünü artırıyor: `lot_size = risk_per_trade / (stop_loss_pips * pip_value)`.
       - **Sinyal Gücü**: `lot_size * sinyal_gucu` ile lot büyüklüğü ölçekleniyor, ancak loglarda sinyal gücü sürekli 0.50. Bu, lot büyüklüğünü %50 oranında azaltıyor, ama yine de teminatı zorlayacak kadar yüksek kalıyor.
       - **Margin Kontrolü Eksikliği**: Fonksiyon, hesaplanan lot büyüklüğünün

mevcut teminatı (`available_margin`) aşıp aşmadığını kontrol etmiyor. Bu, `_marjin_seviyesi_kontrol_et` fonksiyonunda kontrol edilse de, işlem açılmadan önce lot büyüklüğünü sınırlamak daha güvenli olur.

  - **Loglarla Bağlantı**: Loglarda görülen ≈0.051 lot, bu hesaplama mantığından kaynaklanıyor. Örneğin, `risk_per_trade = 25000/3 * 0.02 ≈ 166.67 USD`, `stop_loss_pips = 1.0 * 2 / 0.01 = 200`, `pip_value ≈ 8.23` ile `lot_size ≈ 166.67 / (200 * 8.23) ≈ 0.101`. Sinyal gücü (0.5) ile çarpıldığında ≈0.051 lot çıkıyor. Ancak, bu lot büyüklüğü için gerekli teminat (örneğin, 0.051 * 100000 / 25 ≈ 204 USD) mevcut teminatı zorluyor ve marjin çağrısına yol açıyor.

2. **PNL'nin Düşük veya Sıfır Olması**:
  - **Sorun**: Loglarda PNL genellikle 0.0 veya çok küçük (±0.03, ±0.04). Ayrıca, işlemlerin çoğu kapanmıyor, sadece marjin çağrısı ile sonlanıyor.
  - **Koddaki İlgili Kısım**:
    ```python
    def _yuzen_pnl_hesapla(self) -> float:
        total_pnl = 0.0
        for trade_id, position in list(self.pozisyonlar.items()):
            sembol = position['sembol']
            current_price = self.df_full_dict[sembol]['close'].iloc[self.mevcut_adim]
            position['pnl'] = (current_price - position['giris_fiyati']) * position['lot_buyuklugu'] * 10000 \
                if position['tip'] == 'buy' else (position['giris_fiyati'] - current_price) * position['lot_buyuklugu'] * 10000
            total_pnl += position['pnl']
    ```
    ```python
    def _pozisyonu_kapat(self, trade_id: str, reason: str, exit_price: float):
        position = self.pozisyonlar.pop(trade_id, None)
        if not position:
            return
        sembol = position['sembol']
        pnl = position['pnl'] if reason != 'SL' else -self.alt_bakiyeler[sembol] * self.risk_orani * 0.8
        self.gunluk_kapali_pnl += pnl
        self.bakiye += pnl
        self.alt_bakiyeler[sembol] += pnl
    ```
    - **Tespitler**:
      - **Yüzen PNL**: `_yuzen_pnl_hesapla` fonksiyonu, açık pozisyonların PNL'sini hesaplıyor, ancak loglarda bu değerler genellikle sıfır veya çok küçük. Bunun nedeni, işlemlerin çok kısa sürede (örneğin, bir adımda) marjin çağrısı ile kapanması olabilir. Bu, fiyat hareketlerinin (current_price - giris_fiyati) küçük kalmasına neden oluyor.
      - **Kapanış PNL'si**: `_pozisyonu_kapat` fonksiyonunda, stop-loss (SL) durumunda PNL sabit bir kayıp olarak hesaplanıyor: `-self.alt_bakiyeler[sembol]

* self.risk_orani * 0.8`. Bu, gerçek fiyat hareketine dayalı bir kayıp yerine sabit bir oran kullanıyor, bu yüzden loglarda küçük veya sabit PNL değerleri (örneğin, -0.03, 0.04) görülüyor.
    - **Marjin Çağrısı**: Loglarda işlemlerin çoğu "Marjin Çağrısı" nedeniyle kapanıyor. Bu durumda, `_tum_pozisyonlari_kapat` fonksiyonu çağrılıyor ve tüm pozisyonlar mevcut fiyatla kapatılıyor, ancak PNL genellikle sıfır veya düşük oluyor çünkü fiyat hareketleri küçük veya işlemler hemen kapanıyor.

  - **Loglarla Bağlantı**: Loglarda görülen küçük PNL değerleri (örneğin, GBPUSD için 0.03, -0.03), fiyat hareketlerinin küçük olması veya işlemlerin marjin çağrısı nedeniyle erken kapanmasından kaynaklanıyor. Ayrıca, `_yuzen_pnl_hesapla` fonksiyonunda `atr <= 0` kontrolü nedeniyle bazı pozisyonların PNL'si hesaplanmıyor olabilir.

3. **Sharpe Oranı Hataları**:
  - **Sorun**: Loglarda `RuntimeWarning: divide by zero` hatası ve `inf`/ `-inf` Sharpe oranları görülüyor.
  - **Koddaki İlgili Kısım**:
    ```python
    def calculate_sharpe_ratio(self, returns: pd.Series, risk_free_rate: float = 0.02) -> float:
        try:
            sharpe = (returns.mean() - risk_free_rate) / returns.std() * np.sqrt(252)
            return sharpe if not np.isnan(sharpe) else 0.0
        except Exception:
            return 0.0
    ```
    - **Tespitler**:
      - **Sıfır Standart Sapma**: `returns.std()` sıfır olduğunda, bölme işlemi hata veriyor (`divide by zero`). Bu, getirilerin sabit olduğu (örneğin, tüm işlemler aynı PNL'yi üretiyorsa) veya veri eksikliği durumunda oluyor.
      - **Küçük Veri Seti**: Loglarda işlemlerin çoğu marjin çağrısı ile kapandığından, `returns` dizisi genellikle çok az veri içeriyor veya tüm değerler sıfır/küçük, bu da standart sapmayı sıfırlıyor.
      - **inf/-inf Değerler**: `returns.mean()` çok küçük veya sıfır olduğunda ve standart sapma da küçükse, Sharpe oranı `inf` veya `-inf` olabilir.

  - **Loglarla Bağlantı**: Loglarda görülen `Sharpe: inf`, `-inf`, veya sıfıra yakın değerler, bu fonksiyondaki standart sapma sorununun bir sonucu. Örneğin, GBPUSD işlemlerinde PNL'nin sürekli 0.03 veya -0.03 olması, getirilerin varyansını sıfırlıyor.

4. **Eğitim Sürecinin Verimliliği**:
  - **Sorun**: Loglarda epizodların sürekli marjin çağrısı nedeniyle erken bittiği görülüyor ("Marjin seviyesi düşük, epizod bitti"). Bu, modelin öğrenmesini engelliyor.
  - **Koddaki İlgili Kısım**:

```python
def _marjin_seviyesi_kontrol_et(self) -> bool:
    if not self.pozisyonlar:
        self.marjin_seviyesi = 100.0
        return True
    used_margin = sum(pos['lot_buyuklugu'] * 100000 * 2 for pos in
self.pozisyonlar.values())  # 2:1 kaldıraç
    self.marjin_seviyesi = (self.toplam_oz_sermaye / used_margin * 100) if
used_margin > 0 else 100.0
    if self.marjin_seviyesi < 50:  # %50 eşiği
        self.kayip_nedenleri.append(f"Marjin seviyesi düşük:
{self.marjin_seviyesi:.2f}%")
        self._tum_pozisyonlari_kapat('Marjin Çağrısı')
        self.telegram.sync_send_message(f"🚨 Marjin Çağrısı:
{self.marjin_seviyesi:.2f}%")
        return False
    return True
```
```python
def step(self, action: np.ndarray) -> tuple[np.ndarray, float, bool, Dict]:
    if self.pozisyonlar and not self._marjin_seviyesi_kontrol_et():
        reward = -500.0
        self.bitti = True
        info = self._get_info(total_profit)
        logging.info("Marjin seviyesi düşük, epizod bitti")
        return self._gozlem_al(), reward, self.bitti, truncated,
self._get_info(total_profit)
```
- **Tespitler**:
  - **Düşük Marjin Seviyesi**: `used_margin = sum(pos['lot_buyuklugu'] * 100000 * 2)` ile hesaplanan teminat kullanımı, 2:1 kaldıraç varsayımıyla yüksek çıkıyor. Örneğin, 0.051 lot için `0.051 * 100000 * 2 = 10,200 USD` teminat gerekiyor, bu da 25,000 USD'lik başlangıç bakiyesinin %40'ından fazlasını tüketiyor.
  - **Erken Epizod Sonu**: Marjin seviyesi %50'nin altına düştüğünde epizod hemen bitiyor ve model -500 ödül alıyor. Bu, modelin uzun vadeli stratejiler öğrenmesini engelliyor.
  - **Risk Oranı Dinamizmi**: `self.risk_orani` kayıplardan sonra azalıyor (`self.risk_orani * 0.8`), ancak başlangıç değeri (0.02) yüksek olduğu için ilk işlemler zaten marjin çağrısına yol açıyor.

- **Loglarla Bağlantı**: Loglarda görülen "Marjin seviyesi düşük, epizod bitti" mesajları, bu kontrol mekanizmasından kaynaklanıyor. Ayrıca, ardışık başarısızlık sayıları (örneğin, 6-7) modelin sürekli zarar eden işlemler yaptığını ve risk oranını yeterince hızlı düşüremediğini gösteriyor.

İyileştirme Önerileri

1. **Lot Büyüklüğü Hesaplamasını İyileştirme**:
   - **Sorun Çözümü**: Lot büyüklüğünü teminatla uyumlu hale getirmek için, `_lot_buyuklugu_hesapla` fonksiyonuna marjin kontrolü ekleyin. Ayrıca, USDJPY için pip değerini daha hassas hesaplayın.
   - **Önerilen Kod Değişikliği**:
     ```python
     def _lot_buyuklugu_hesapla(self, sembol: str, atr: float, sinyal_gucu: float) -> float:
         risk_per_trade = self.alt_bakiyeler[sembol] * min(self.risk_orani, 0.01)  # Risk oranını %1 ile sınırla
         pip_sizes = {'EURUSD': 0.0001, 'GBPUSD': 0.0001, 'USDJPY': 0.01}
         pip_size = pip_sizes.get(sembol, 0.0001)
         current_price = self.df_full_dict[sembol]['close'].iloc[self.mevcut_adim]

         if current_price <= 0:
             logging.warning(f"Geçersiz fiyat: {sembol}, current_price={current_price}")
             return 0.01

         pip_value = pip_size * 100000
         if sembol.endswith("JPY"):
             pip_value = pip_value / current_price

         atr = max(atr, np.nanmean(self.ozellikler_dict[sembol]['atr'].iloc[-20:]) or pip_size)
         stop_loss_pips = (atr * 2) / pip_size

         lot_size = risk_per_trade / (stop_loss_pips * pip_value)

         # Marjin kontrolü
         required_margin = lot_size * 100000 * 2 / current_price  # 2:1 kaldıraç
         available_margin = self.toplam_oz_sermaye * 0.5  # %50 marjin kullanılabilir
         if required_margin > available_margin:
             lot_size = available_margin * current_price / (100000 * 2)

         lot_size = max(0.01, min(lot_size * (sinyal_gucu ** 1.2), MAX_LOT * 0.5))

         logging.debug(f"Lot hesaplama: {sembol}, risk: {risk_per_trade:.2f}, atr: {atr:.5f}, "
                       f"pip_value: {pip_value:.2f}, stop_loss_pips: {stop_loss_pips:.1f}, "
                       f"lot: {lot_size:.3f}, required_margin: {required_margin:.2f}")
         return lot_size
     ```
   - **Faydalar**:

- Risk oranını %1 ile sınırlandırarak teminat baskısını azaltır.
- Marjin kontrolü, lot büyüklüğünü mevcut teminatla uyumlu hale getirir.
- Sinyal gücünün `**1.2` ile ölçeklenmesi, daha dinamik lot büyüklükleri sağlar.

2. **PNL Hesaplamasını Düzeltme**:
   - **Sorun Çözümü**: Stop-loss durumunda sabit kayıp yerine gerçek fiyat hareketine dayalı PNL hesaplayın. Ayrıca, yüzen PNL'nin sıfır olması durumunu önlemek için fiyat farklarını daha hassas kontrol edin.
   - **Önerilen Kod Değişikliği**:
     ```python
     def _yuzen_pnl_hesapla(self) -> float:
         total_pnl = 0.0
         idx = self.mevcut_adim
         for trade_id, position in list(self.pozisyonlar.items()):
             sembol = position['sembol']
             current_price = self.df_full_dict[sembol]['close'].iloc[idx]
             atr = self.ozellikler_dict[sembol]['atr'].iloc[idx]
             if np.isnan(atr) or atr <= 0:
                 logging.warning(f"Geçersiz ATR: {sembol}, atr={atr}")
                 continue
             price_diff = current_price - position['giris_fiyati'] if position['tip'] == 'buy' else position['giris_fiyati'] - current_price
             position['pnl'] = price_diff * position['lot_buyuklugu'] * 100000 * pip_sizes.get(sembol, 0.0001)
             total_pnl += position['pnl']
             # ... (trailing stop ve SL/TP kontrolü aynı kalabilir)
         return total_pnl

     def _pozisyonu_kapat(self, trade_id: str, reason: str, exit_price: float):
         position = self.pozisyonlar.pop(trade_id, None)
         if not position:
             return
         sembol = position['sembol']
         pip_size = {'EURUSD': 0.0001, 'GBPUSD': 0.0001, 'USDJPY': 0.01}.get(sembol, 0.0001)
         price_diff = exit_price - position['giris_fiyati'] if position['tip'] == 'buy' else position['giris_fiyati'] - exit_price
         pnl = price_diff * position['lot_buyuklugu'] * 100000 * pip_size
         self.gunluk_kapali_pnl += pnl
         self.bakiye += pnl
         self.alt_bakiyeler[sembol] += pnl
         # ... (geri kalan kod aynı kalabilir)
     ```
   - **Faydalar**:
     - PNL hesaplaması, sembole özgü pip boyutlarını dikkate alarak daha doğru olur.

- Stop-loss durumunda sabit kayıp yerine gerçek fiyat farkına dayalı hesaplama yapılır.

3. **Sharpe Oranı Hatasını Düzeltme**:
   - **Sorun Çözümü**: Standart sapma sıfır olduğunda varsayılan bir Sharpe oranı döndürün ve getirileri filtreleyin.
   - **Önerilen Kod Değişikliği**:
     ```python
     def calculate_sharpe_ratio(self, returns: pd.Series, risk_free_rate: float = 0.02) -> float:
         try:
             if len(returns) < 2 or returns.std() == 0:
                 return 0.0
             sharpe = (returns.mean() - risk_free_rate) / returns.std() * np.sqrt(252)
             return np.clip(sharpe, -100, 100) if not np.isnan(sharpe) else 0.0
         except Exception:
             return 0.0
     ```
   - **Faydalar**:
     - Sıfır standart sapma durumunda hata önlenir.
     - `inf`/`-inf` değerler `np.clip` ile sınırlandırılır.

4. **Eğitim Sürecini İyileştirme**:
   - **Sorun Çözümü**: Marjin çağrılarını azaltmak ve epizod sürelerini uzatmak için risk yönetimini güçlendirin ve ödül fonksiyonunu iyileştirin.
   - **Önerilen Kod Değişikliği**:
     ```python
     def step(self, action: np.ndarray) -> tuple[np.ndarray, float, bool, Dict]:
         # ... (mevcut kodun üst kısmı aynı kalır)

         # Ödül fonksiyonunu iyileştir
         reward = (self.yuzen_pnl + self.gunluk_kapali_pnl) / self.baslangic_bakiyesi
         if self.volatility > 1.5:
             reward += 0.2 * signal_strength  # Volatiliteye bağlı ödül
         elif self.volatility < 0.5:
             reward -= 0.05 * (1 - signal_strength)  # Fırsat kaçırma cezası
         if black_swan_detected:
             reward += 0.3 if self.ardisik_basarisizlik == 0 else -0.3  # Black Swan başarısı/cezası
         reward -= 0.1 * self.ardisik_basarisizlik  # Ardışık başarısızlık cezası
         reward += 0.05 * self.sharpe_ratio if self.sharpe_ratio > 0 else 0.0  # Sharpe bonusu

         # Marjin seviyesini daha erken kontrol et
         if self.pozisyonlar and not self._marjin_seviyesi_kontrol_et():
             reward = -100.0  # Daha hafif ceza
     ```

```
            self.bitti = True
            info = self._get_info(total_profit)
            logging.info("Marjin seviyesi düşük, epizod bitti")
            return self._gozlem_al(), reward, self.bitti, truncated,
self._get_info(total_profit)

        # ... (geri kalan kod aynı kalır)
```
  - **Faydalar**:
    - Ödül fonksiyonu, sinyal gücü, volatilite ve Sharpe oranını dikkate alarak daha dengeli olur.
    - Ardışık başarısızlıklar için ek ceza, modelin zararlı stratejilerden kaçınmasını teşvik eder.
    - Marjin çağrısı cezası (-500 yerine -100) epizodların tamamen kesilmesini önler.

5. **Logları Zenginleştirme**:
  - **Sorun Çözümü**: Hangi işlemlerin neden kapandığını ve marjin detaylarını daha ayrıntılı loglayın.
  - **Önerilen Kod Değişikliği**:
    ```python
    def _pozisyonu_kapat(self, trade_id: str, reason: str, exit_price: float):
        position = self.pozisyonlar.pop(trade_id, None)
        if not position:
            return
        sembol = position['sembol']
        pip_size = {'EURUSD': 0.0001, 'GBPUSD': 0.0001, 'USDJPY':
0.01}.get(sembol, 0.0001)
        price_diff = exit_price - position['giris_fiyati'] if position['tip'] == 'buy'
else position['giris_fiyati'] - exit_price
        pnl = price_diff * position['lot_buyuklugu'] * 100000 * pip_size
        self.gunluk_kapali_pnl += pnl
        self.bakiye += pnl
        self.alt_bakiyeler[sembol] += pnl
        trade_record = {
            'islem_id': trade_id, 'sembol': sembol, 'tip': position['tip'],
            'lot_buyuklugu': position['lot_buyuklugu'], 'giris_fiyati':
position['giris_fiyati'],
            'cikis_fiyati': exit_price, 'pnl': pnl, 'acilis_zamani':
position['acilis_zamani'],
            'kapanis_zamani': self.mevcut_tarih, 'neden': reason, 'sinyal_gucu':
position['sinyal_gucu'],
            'marjin_seviyesi': self.marjin_seviyesi
        }
        self.islemler.append(trade_record)
        self.gunluk_islemler.append(trade_record)
        if pnl < 0:
```

```
            self.ardisik_basarisizlik += 1
            self.risk_orani = max(0.005, self.risk_orani * 0.8)
        else:
            self.ardisik_basarisizlik = 0
            self.risk_orani = min(ISLEM_BASINA_MAKS_RISK, self.risk_orani * 1.1)
        returns = pd.Series([t['pnl'] for t in self.islemler if t['sembol'] == sembol])
        self.sharpe_ratio = self.calculate_sharpe_ratio(returns)
        logging.info(f"İşlem kapandı: {sembol}, neden: {reason}, PNL: {pnl:.2f},
Sharpe: {self.sharpe_ratio:.2f}, "
                     f"çıkış fiyatı: {exit_price:.5f}, ardışık başarısızlık:
{self.ardisik_basarisizlik}, "
                     f"marjin_seviyesi: {self.marjin_seviyesi:.2f}%")
    ```
  - **Faydalar**:
    - Marjin seviyesi ve diğer detaylar loglara eklenerek sorunların kökeni daha kolay tespit edilir.

Genel Değerlendirme
- **Marjin Çağrıları**: Lot büyüklüklerinin yüksek olması ve marjin kontrolünün eksikliği, loglarda görülen sürekli marjin çağrılarının ana nedeni. Önerilen lot hesaplama ve marjin kontrolü değişiklikleriyle bu sorun büyük ölçüde çözülebilir.
- **PNL Sorunları**: Küçük veya sıfır PNL, işlemlerin erken kapanmasından ve fiyat hareketlerinin küçük olmasından kaynaklanıyor. Gerçek fiyat farkına dayalı PNL hesaplama ve daha uzun epizodlar bu sorunu hafifletebilir.
- **Sharpe Oranı**: Standart sapma kontrolü ve sınırlandırma ile `inf`/ `-inf` hataları önlenebilir.
- **Eğitim Verimliliği**: Ödül fonksiyonuna eklenen bonuslar/cezalar ve daha hafif marjin cezaları, modelin daha uzun süre öğrenmesini sağlayacak.

Ek Öneriler
- **Sinyal Gücü Çeşitliliği**: `_sinyal_gucu_hesapla` fonksiyonunda sinyal gücünün sürekli 0.50 olması, modelin öğrenmesini sınırlıyor. SHAP değerlerini daha etkin kullanmak veya ek teknik göstergeler (örneğin, RSI, ADX) eklemek faydalı olabilir:
  ```python
  def _sinyal_gucu_hesapla(self, sembol: str) -> float:
      try:
          features = self.ozellikler_dict[sembol].iloc[self.mevcut_adim]
          rsi = features.get('rsi', 50.0)
          adx = features.get('adx', 20.0)
          signal = (rsi - 50) / 50 * 0.5 + (adx - 20) / 20 * 0.5
          return np.clip(signal, 0.1, 0.9)
      except Exception:
          return 0.5
  ```
- **Epizod Süresini Uzatma**: Marjin çağrısı nedeniyle epizodların erken

bitmesini önlemek için, `_marjin_seviyesi_kontrol_et` fonksiyonunda bir "kurtarma" mekanizması eklenebilir:
```python
def _marjin_seviyesi_kontrol_et(self) -> bool:
    if not self.pozisyonlar:
        self.marjin_seviyesi = 100.0
        return True
    used_margin = sum(pos['lot_buyuklugu'] * 100000 * 2 for pos in self.pozisyonlar.values())
    self.marjin_seviyesi = (self.toplam_oz_sermaye / used_margin * 100) if used_margin > 0 else 100.0
    if self.marjin_seviyesi < 50:
        # Lotları yarıya indir ve tekrar dene
        for pos in self.pozisyonlar.values():
            pos['lot_buyuklugu'] *= 0.5
        used_margin = sum(pos['lot_buyuklugu'] * 100000 * 2 for pos in self.pozisyonlar.values())
        self.marjin_seviyesi = (self.toplam_oz_sermaye / used_margin * 100) if used_margin > 0 else 100.0
        if self.marjin_seviyesi < 50:
            self.kayip_nedenleri.append(f"Marjin seviyesi düşük: {self.marjin_seviyesi:.2f}%")
            self._tum_pozisyonlari_kapat('Marjin Çağrısı')
            self.telegram.sync_send_message(f"🚨 Marjin Çağrısı: {self.marjin_seviyesi:.2f}%")
            return False
    return True
```
- **Test Ortamını Genişletme**: `stress_test` fonksiyonunda farklı kriz dönemlerini ve volatilite senaryolarını daha sık test edin. Örneğin, `KRIZ_DONEMLERI`'ne daha fazla senaryo ekleyin:
```python
KRIZ_DONEMLERI = {
    2008: slice(0, 10000),
    2020: slice(50000, 60000),
    'high_vol': slice(10000, 20000),  # Yüksek volatilite dönemi
    'normal': slice(20000, 30000)
}
```

Sonuç
Kod, güçlü bir altyapıya sahip, ancak marjin çağrıları, düşük PNL ve Sharpe oranı hataları gibi sorunlar, lot hesaplama, marjin kontrolü ve ödül fonksiyonunda yapılacak iyileştirmelerle çözülebilir. Önerilen değişiklikler, loglarda görülen sorunları doğrudan ele alıyor ve eğitim sürecini daha verimli hale getirecek. Eğer ek test sonuçları veya belirli bir fonksiyonun detaylı analizi

gerekiyorsa, lütfen belirtin, daha derinlemesine inceleyebilirim!