Harika, verdiğin bilgiler doğrultusunda çalışmaya başlıyorum. Hedefin maksimum kar, prop firm kurallarına uyumluluk (günlük %5 ve toplamda %10 drawdown sınırı) ve EURUSD, GBPUSD gibi büyük forex çiftleri olduğu için, modül seçimini bu hedeflere göre optimize edeceğim.
Hangi RL modüllerinin bu senaryoda öncelikli olarak entegre edilmesi gerektiğini, hangilerinin sınırlı kaynaklar nedeniyle uygun olmadığını ve hangi sırayla ilerlemenin mantıklı olduğunu araştıracağım. Ayrıca MacBook M1'in sınırlamaları altında hangi algoritmaların pratik olarak eğitilebilir olduğunu da göz önünde bulunduracağım.
Kısa süre içinde bulgularımı seninle paylaşacağım.
Research completed in 6m· 16 kaynak· 78 arama
RL Modüllerinin Seçimi ve Sıralaması (MacBook M1 + Prop Firm Uyumlu Forex Botu)

Reinforcement Learning Tabanlı Trading Bot: Modül Öncelikleri ve Yol Haritası

Proje Verisi ve Hedefleri

Elinizde **24 yıllık 15-dakikalık fiyat verisi** ve buna eşlik eden **24 yıllık ekonomik takvim (makro olay) verisi**bulunuyor. Botu öncelikle döviz pariteleri (EURUSD, GBPUSD, USDJPY) üzerinde kullanmayı planlıyorsunuz; ileride US100, US500 endeksleri ve XAUUSD (altın) gibi varlıklara da genişleyebilir. Performans hedefiniz **maksimum kar**elde etmek ancak aynı zamanda **risk yönetimi** konusunda çok katı kısıtlar var: günlük en fazla %5 **gerileme (drawdown)**, toplamda en fazla %10 drawdown. Botun stratejisinin, FTMO gibi **prop firm** kurallarına uyum sağlaması gerekiyor (yani belirlenen kayıp limitlerini aşmaması şart). Bu hedefler, botun sadece kârlı değil aynı zamanda **istikrarlı ve kontrollü** olması gerektiğini gösteriyor. Dolayısıyla ödül fonksiyonunu ve eğitim stratejisini buna göre tasarlamak kritik olacak – ajan sadece karı maksimize etmeye değil, büyük kayıpları da **önlemeye** odaklanmalı (örneğin, aşırı düşüşleri cezalandıran bir bileşenle)milvus.io.

Donanım ve Hesaplama Kısıtları

Eğitimi **sadece bir MacBook M1** üzerinde yapabileceksiniz (yalnızca CPU gücü, sınırlı GPU **Metal** hızlandırma). Bu, derin RL algoritmalarını eğitirken ciddi bir hesaplama kısıtı demektir. Çok büyük sinir ağları veya yüz binlerce adımlık simülasyon gerektiren karmaşık algoritmalar M1 üzerinde **çok yavaş** çalışabilir. Bu nedenle, seçeceğimiz algoritmaların **verimliliği** önemli olacak. Mümkün olduğunca örnek-verimli (sample-efficient) yöntemler ve **hafif mimariler** tercih etmeli, aynı zamanda paralel hesaplama imkanı varsa (ör. A3C gibi) M1'in çok çekirdekli CPU'sunu kullanmayı düşünebiliriz. Ancak gene de, devasa transformer modelleri veya çok etmenli simülasyonlar gibi **hesaplaması ağır modülleri** büyük olasılıkla bu donanımda çalıştırmak pratik olmayacaktır. Bu yüzden, öncelikleri belirlerken **hesaplama yükünü** de göz önünde bulundurup, daha sade ve optimize çözümle başlamak en iyisi.

Temel RL Algoritmasının Seçimi

Listelenen modüller arasında birçok RL algoritması var: **Rainbow DQN**, **PPO**, **SAC**, **Distributional RL**, **DDPG/TD3**, **A3C/A2C** gibi. İlk adım, botun iskeletini oluşturacak **çekirdek RL algoritmasını** seçmek olmalı. Bu seçimi yaparken şunları dikkate alın:

- **Eylem uzayı (action space)**: Trading ortamında ajan ya **ayrık** eylemler yapabilir (örneğin *al, sat, bekle* gibi) ya da **sürekli** eylemler yapabilir (örneğin pozisyon boyutunu % olarak ayarla, lot büyüklüğünü belirle). Eğer eylemleri ayrık tanımlarsanız, DQN türü değer-temelli algoritmalar uygulanabilir. Sürekli aksiyonlar için ise aktör-kritik yaklaşımlar (**PPO, SAC, TD3** gibi) daha uygundur.
- **PPO vs A2C/A3C**: A3C/A2C, paralel ortamlar üzerinde çalışan eski bir politika gradyanı yöntemidir; **PPO** ise bunun daha yeni ve kararlı hale getirilmiş versiyonu gibi düşünülebilir. PPO, politika güncellemesini belli bir oranda kısıtlayarak kararlılığı artırır. Genelde PPO, A2C'ye kıyasla benzer ortamlarda daha yüksek performans ve stabil eğitim sağlar. Bu nedenle, eğer politika-tabanlı bir yöntem kullanılacaksa doğrudan PPO ile başlamak mantıklıdır (A3C/A2C'yi ayrı bir modül olarak entegre etmeye çok gerek yok, zira PPO onların yerini alabilir). Nitekim bir çalışmada PPO ajanı, aynı ortamda A2C ajanından **daha yüksek kümülatif getiri** elde etmiş ancak biraz daha yüksek oynaklık sergilemiştir; A2C ise daha düşük maksimum düşüş (maksimum drawdown) ile **daha risk kontrollü** davranabilmiştirdata-science-blog.com. Bu sonuçlar, PPO'nun trendi yakalama ve kar kovalamada iyi olduğunu, A2C'nin ise riski sınırlamada daha temkinli olduğunu gösteriyordata-science-blog.com. Sizin hedefiniz maksimum kar olsa da risk limitleri sıkı olduğu için, PPO kullanacaksanız dahi risk kontrolünü ayrıca ele almalısınız (aşağıda ele alacağız).
- **SAC vs TD3 vs DDPG**: Bunlar **off-policy** sürekli kontrol algoritmalarıdır. DDPG orijinal bir yöntemken, **TD3 (Twin Delayed DDPG)** DDPG'nin geliştirilmiş halidir (çift eleştirici ve gecikmeli politika güncellemesi ile aşırı tahmin sorununu giderir). **SAC (Soft Actor-Critic)** ise entropi bonusu ile keşfi artıran ve çok kararlı öğrenebilen bir yöntemdir. Off-policy algoritmaların avantajı, replay buffer kullanarak örnekleri tekrar tekrar kullanabilmeleri, dolayısıyla **örnek verimliliklerinin** yüksek olmasıdır. 24 yıllık tarihi veriniz olsa da, bunu verimli kullanmak önemli olacak – off-policy yöntemler bu sabit veriyi tekrar tekrar örnekleyerek daha hızlı öğrenebilir. PPO gibi on-policy yöntemler ise deneyimi bir kez kullanıp atar, benzer başarıya ulaşmak için daha fazla çevrim adımı gerekebilir. Bu açıdan **SAC** veya **TD3**, özellikle sürekli aksiyonlu bir ortam kuracaksanız, çok iyi adaylar. SAC, stokastik bir politika ve otomatik entropi ayarı ile genelde TD3 kadar (hatta bazen daha da) iyi sonuçlar verebiliyor. Bu algoritmaların başarılı uygulamaları mevcut; örneğin birçok açık kaynak proje ve kütüphane (Stable Baselines, FinRL vb.), finans verisiyle **PPO, A2C, DDPG, TD3, SAC** ajanlarını denemiş ve hepsinin çalışabildiğini

göstermiştirsgino209.medium.comsgino209.medium.com. Dolayısıyla, continuous action planlıyorsanız DDPG yerine **TD3 veya direkt SAC ile** başlamak en mantıklısı (DDPG modülünü ayrıca entegre etmenize gerek yok, TD3 onun yerine geçebilir).

- **DQN ve türevleri (Rainbow, Distributional RL)**: Eğer aksiyonları basitçe **"Long pozisyon aç", "Short aç", "Pozisyon kapat/flat"** gibi sınırlı ayrık seçenekler olarak modellemeyi düşünürseniz, DQN tabanlı yöntemler de bir seçenek olabilir. Özellikle **Rainbow DQN**, DQN'nin tüm iyileştirmelerle zenginleştirilmiş hali (double DQN, dueling network, prioritized replay, n-step returns ve **distributional** öğrenme içeriyor). Rainbow ile finans verisinde ayrık al/sat kararları veren çalışmalar var, ancak genellikle oyun ortamlarına kıyasla finans zaman serilerinde DQN bazlı yaklaşımlar daha zorlanabilir çünkü ödül sinyali gürültülü ve gecikmeli olabiliyor. Ayrıca portföy veya pozisyon boyutu yönetimi DQN ile zor (ancak aksiyonu "al 10 lot, al 5 lot, sat 5 lot" gibi kademeli ayrık yaparak çözülebilir). Rainbow DQN'nin bir alt bileşeni olan **distributional RL**, ajanın sadece beklenen getiriyi değil **getiri dağılımını** tahmin etmesini sağlar. Bu, risk farkındalığı için değerli olabilir – ajan, eylemlerinin sonuç dağılımını öğrenerek tail-risk (olumsuz uç riskler) hakkında içgörü kazanır. Nitekim yeni bir araştırma, finans piyasasında distributional RL algoritmalarının (C51, QR-DQN, IQN gibi) klasik beklenti-getirisi odaklı RL'den anlamlı derecede daha iyi performans verdiğini ve **CVaR gibi risk ölçümlerini** optimize edecek şekilde eğitildiklerinde risk-yanıt veren politikalar elde edildiğini göstermiştirarxiv.org. Örneğin, **C51 algoritması** ile CVaR$_{0.1}$ (en kötü %10'luk dilimdeki getirilerin ortalaması) maksimize edilerek, riskten kaçınma düzeyi ayarlanabilir politikalar üretilmiş; CVaR eşiği düşürüldüğünde daha muhafazakar (düşük riskli) stratejiler elde edilmişarxiv.org. Bu sonuçlar distributional yaklaşımların risk kontrolü açısından çok umut vaat ettiğini gösteriyor.

Özetle, **ilk etapta bir veya iki çekirdek algoritmaya odaklanın**. Sürekli aksiyonlu bir model için **SAC** veya **PPO** iyi adaylardır. Eğer ayrık aksiyonlu basit bir stratejiyle başlamak isterseniz, **Rainbow DQN** de düşünülebilir, fakat bunun getiri dağılımını öğrenme avantajını daha sonra SAC/PPO'ya da entegre edebilirsiniz (örneğin dağılımsal bir kritik ağ kullanarak). Genel tavsiyem, **PPO ile başlamanız** yönünde olacak, çünkü uygulaması görece kolay, kararlı ve finans ortamlarında yaygın bir benchmark olarak kullanılıyordata-science-blog.com. PPO ile elde edeceğiniz sonuçları gördükten sonra, benzer ortamda SAC/TD3 gibi off-policy bir ajanı da deneme şansınız olur; böylece hangisinin performansı ve risk profili daha uygun bakabilirsiniz. Unutmayın, **DDPG'yi ayrı tutmanıza gerek yok**, zira TD3 onun geliştirilmiş hali; aynı şekilde A2C de PPO tarafından kapsanıyor. Bu sayede modül listenizi biraz sadeleştirmiş oluyoruz.

Risk Yönetimi ve Gerçek Dünya Kısıtlarının Entegrasyonu

Risk yönetimi, projenizin başarısı için teknik seçimler kadar önemli olacak. Hatta, maksimum kar hedeflerken belirttiğiniz sıkı drawdown limitleri nedeniyle,

ajanınızın ödül fonksiyonuna ve ortamına risk kontrol mekanizmalarını **en baştan entegre etmelisiniz**. Aşağıdaki adımları öneririm:

- **Ödül Fonksiyonunda Drawdown ve Risk Ceza/Puanları**: Sadece anlık kar/zarar odaklı bir ödül, ajanı agresif davranmaya itebilir ve bu da prop firm kurallarını ihlal eden sonuçlar doğurabilir. Bunun yerine ödüle bir **risk cezası** ekleyin. Örneğin, eğer herhangi bir günde %5'den fazla kayıp yaşarsa (hesap özsermayesinden), büyük bir negatif ödül verin veya o günü sonlandırın. Benzer şekilde toplam sermayeden %10 düşüş durumunu (simülasyon hesabının en yüksek noktasından itibaren) "oyunun sonu" olarak tanımlayıp epizodu bitirebilirsiniz. Ajan, bu durumun ne kadar ağır bir ceza getirdiğini öğrenecektir. Aslında bir makalede belirtildiği gibi, **risk yönetimi genellikle RL çerçevesine dahili olarak işlenir**: örneğin pozisyon büyüklüklerini sınırlamak, veya **aşırı drawdown'ları cezalandıran** terimler ödül fonksiyonuna eklemek yaygın bir yaklaşımdırmilvus.io. Bu sayede ajan, sadece getiriye değil, getiri-risk dengesine optimize olur.
- **Transaction Cost, Slipaj ve Gecikme**: Gerçekçi bir trading bot, işlem maliyetlerini göz önüne alır. Ortam simülasyonunuza makul bir **alış/satış spread'i**, komisyon veya fiyat kayması (slippage) modeli ekleyin. Her işlem yaptığında ödülden küçük bir maliyet düşülmesi, ajanın çok sık al-sat yaparak küçük dalgalanmalardan yararlanma stratejisini (ki bu işlem maliyetiyle aslında kârlı olmayacaktır) engeller. Nitekim literatürde bazı RL tabanlı trading çalışmalarında ajanların **işlem sıklığını azaltmayı** öğrenmesi için ödüle işlem maliyeti konulduğu bilinirmilvus.io. Sizin de botunuza en baştan bu maliyetleri koymanız, daha gerçekçi ve prop firm koşullarına uygun davranan bir stratejiye yol açacaktır.
- **Pozisyon Büyüklüğü Sınırlaması**: Prop firmalar genelde kaldıraç ve maksimum pozisyon konusunda da sınırlara sahiptir. Ajanınızın alabileceği maksimum pozisyonu (örneğin portföyün %x'i kadarlık bir pozisyon riske atabilir gibi) kısıtlayın. Bu, özellikle continuous aksiyon durumunda önemli; aksiyon uzayınızı $[-1,1]$ aralığında normalize ediyorsanız, bunu "hesabın %100'ü ile short/long ol" şeklinde yorumlayabilirsiniz. Ancak belki de hiçbir zaman %100'den fazlasıyla risk almamalı, hatta belki riskin bir kısmını yedekte tutmalı. Bu tür kısıtları da uygulayabilirsiniz. **Aksiyonları sınırlandırmak**, arzu edilmeyen politikaların oluşmasını baştan engeller (örneğin, hesabın tamamıyla tek bir işlem yapmak yerine belki yarısıyla yapmak gibi daha temkinli davranır).
- **Sharpe Ratio veya Risk Düzeltilmiş Ödül**: Ödül fonksiyonunu tasarlarken *reward = net_profit - λ * risk*formunda birleştirilmiş metrik kullanmayı düşünebilirsiniz. Örneğin her adımda anlık getiriyi (getiri = fiyat değişimi * pozisyon) verirken, aynı zamanda portföy volatilitesini ya da maksimum düşüşü takip edip ceza verebilirsiniz. Bazı çalışmalar doğrudan **Sharpe oranını** optimize eden bir ödül kullanıyor (Sharpe =

ortalama getiri / volatilite; bunun negatifini ceza olarak eklemek gibi). Hatta çok daha basit bir yöntem: eğer adımda portföy değeri önceki zirve değerine göre yeni bir **drawdown** seviyesindeyse, ekstra bir negatif ödül verin. Böylece ajan yeni bir maksimum düşüş yaratmaktan kaçınmaya çalışır. Bu tip yaklaşımlar literatürde dile getirilmiş durumda; örneğin bir çalışma, belirli bir dönem içindeki maksimum düşüşü hesaba katan bir ceza terimini ödüle ekleyerek **sert düşüşleri önlemeyi** başarmıştırietresearch.onlinelibrary.wiley.com. Sizin ortamınızda da benzer şekilde, keskin düşüşleri cezalandıran bir ödül, günlük %5 kayıp sınırını ihlal etmeme içgüdüsünü ajanınıza kazandırabilir.

- **CVaR (Conditional Value at Risk) Optimizasyonu**: Risk-aware RL denince özellikle bahsettiğiniz **CVaR** önemli bir kavram. CVaR$_\alpha$ genellikle en kötü α yüzdelik dilimdeki ortalama kaybı ölçer (örneğin $\alpha=0.05$ ise en kötü %5 senaryodaki ortalama getiri). Bunu doğrudan optimize etmek, "en kötü durumları" iyileştirmeye odaklanmayı sağlar. Standart RL bunu doğrudan yapmaz, ama distributional RL yöntemleri veya özel risk duyarlı politika gradyan yöntemleriyle CVaR optimize edilebilir. Bir araştırmada distributional DQN ajanları, doğrudan CVaR'ı maksimize edecek şekilde eğitilmiş ve böylece **ayarlanabilir risk iştahı** elde edilmiştirarxiv.org. CVaR temelli bir yaklaşımı uygulamak oldukça ileri seviye olabilir; belki başlangıçta tam olarak yapamazsınız ama aklınızda bulunsun: örneğin, agent'ın her bir episode sonu toplam getirisini alıp bunun belirli bir düşük yüzde dilimini optimize edecek şekilde bir objektif tasarlamak mümkündür. Alternatif pratik bir yaklaşım da: her adımda reward'u hesaplarken sadece *kâr* değil, *kârın belirli bir kötü senaryo değerini* kullanmak. Örneğin, agent'ın gelecek N adım içinde en kötü görmesi olası kaybı tahmin eden bir model entegre edip (belki bir ikinci ağ ile) bunu ceza olarak koymak. Bu oldukça karmaşık gelebilir; özetle demek istediğim, risk ölçülerini doğrudan hedeflemek de mümkün ve eğer ilk basit denemelerde strateji çok dalgalı çıkarsa bu yola başvurabilirsiniz.

Özetle, **ilk entegrasyonlarınızdan biri risk yönetimi olmalı**. Başlangıçta belki mükemmel bir Sharpe oranı yakalayamayabilirsiniz ancak en azından botun *büyük kayıplardan kaçınma* davranışını erken öğrenmesi önemli. Bu, prop firm sınavını geçebilmek için kritik.

Eğitim Stratejisi ve Süreci

Elimizdeki veriyle bir **öğrenme ortamı (environment)** kurarak ajanı eğiteceğiz. Bu noktada, önceki sorunuzdaki "süpervizyon (etiketli veri) vs çevrimdışı deneyim vs sıfırdan self-play" konusunu netleştirelim:

- **Süpervizyon (Gözetimli Öğrenme)**: Bu, her zaman adımında "doğru" veya uzman bir eylem etiketi gerektirir. Sizin durumunuzda böyle etiketli bir veri yok (24 yıllık veride, her adımda ne yapılması gerektiğini etikette veren bir uzman strateji bulunmuyor). Dolayısıyla klasik anlamda gözetimli öğrenme kullanmak mümkün değil. (Teorik olarak,

eğer geçmiş veride başarılı olmuş bir stratejinin sinyallerini çıkarsaydınız, ajanı önce o stratejiyi taklit ederek başlatabilirdiniz, ancak bu zorunlu değil ve belirtilmedi.)

- **Çevrimdışı RL vs Online RL**: **Çevrimdışı RL**, önceden toplanmış bir deneyim verisinden (state, action, reward, next state dörtlemesi) öğrenmeyi ifade eder ve ajan yeni aksiyonlar deneyemez, sadece verilene göre en iyisini genelleştirmeye çalışır. Sizin elinizde aslında böyle bir veri kümesi var (tüm geçmiş fiyat hareketleri ve belki bazı insan stratejileri?), ancak muhtemelen herhangi bir stratejinin eylemleri yok, sadece piyasa verisi var. Bu yüzden tamamen *batch RL* tarzı bir yöntem (ör. Conservative Q-Learning gibi) kullanmak yerine, biz **tarihsel veriyi bir simülasyon ortamı** olarak kullanacağız. Yani, aslında ajan "online RL" yapacak ama gerçek piyasa yerine kayıtlı geçmiş piyasa üzerinden bunu yapacak. Bu, birçok akademik çalışmada *backtesting ile RL eğitimi* olarak görülür. Ajan her episode'da belirli bir tarih aralığında gezinip al-sat yapacak, ödüllerini alacak.
- **Self-Play (Kendi kendine oyun)**: Bu terim genelde iki veya daha fazla ajanının birbirine karşı rekabet ederek öğrenmesini ifade eder (örneğin AlphaZero'da iki satranç ajanı birbirine karşı oynar ve güçlenir). Finans piyasalarında bu analojiyi kurmak zordur çünkü ortada tek bir rakip yok, dağıtık bir piyasa var ve "kazanan/kaybeden" net tanımlı değil, herkes para kazanmaya çalışıyor ve kazanç bir oyunun sonucu değil sürekli bir süreç. Yapılan çalışmalar self-play'i finansal ortama uyarlamanın zor olduğunu vurguluyorpmc.ncbi.nlm.nih.gov. Yani, bota karşı oynayacak ikinci bir ajan tanımlamak veya piyasanın davranışını bir "rakip" olarak modellemek pek pratik değil. Bu nedenle AlphaZero tarzı bir kendine karşı oynama durumunu **kullanmayacağız**. Onun yerine tek bir ajanımız olacak ve piyasa ortamı sabit bir simülasyon (geçmiş veri) olacak.

Dolayısıyla, eğitim sürecimiz aslında **offline verilere karşı online RL** şeklinde ilerleyecek. Ajanımız geçmiş veri üzerinde **deneme-yanılma** yaparak strateji öğrenecek, ancak bu deneme yanılma gerçek piyasayı etkilemeyecek (çünkü simüle ediyoruz). Bu sayede herhangi bir gerçek risk almadan, ajanı prop firm kurallarına uygun şekilde eğitebiliriz.

Eğitim Uygulaması: Bu noktada pratik bazı tavsiyeler:

- Ortamı bir **Gym (veya benzeri)** arayüzü şeklinde tasarlayın. Girdi olarak o andaki piyasa durumu (fiyatlar, indikatörler, vs), çıktı olarak ajan aksiyonu (örn. pozisyon değişikliği), ve step() fonksiyonunuz da yeni fiyattan kar/zararı hesaplayıp ödülü döndürsün.
- 24 yıllık veriyi tek bir episodeda tamamen kullanmak yerine, bunu daha küçük parçalara bölebilirsiniz. Örneğin her episode'ı 1 yıl yapıp 24 farklı yıl (veya ay) boyunca eğitim yapabilirsiniz. Ya da rastgele başlangıç noktalarıyla sabit uzunluklu parçalar alabilirsiniz (buna **experience replay**'e benzer bir çeşitlendirme denebilir). Bu, ajanınızın belirli bir

dönemine aşırı overfit olmasını engeller. **Walk-forward** veya **kaydırmalı pencere** yaklaşımları uygulayabilirsiniz: örneğin 1999-2015 arası veriyi eğitim, 2016-2020 arası doğrulama, 2021-2023 test gibi üçe bölebilirsiniz. Ajan eğitim sırasında 1999-2015 arasında defalarca gezinip öğrenir, ardından 2016-2020'yi ayrı tutup orada performansını ölçer, iyi ise en son 2021-23'de test edersiniz. Bu yöntem, stratejinin yeni görülen veride de çalıştığını doğrulamak için önemli.

- **Örneklemeyi çeşitlendirme**: Off-policy bir algoritma kullanıyorsanız (mesela SAC), replay buffer'ınız doğası gereği eğitim verisini karıştıracak. On-policy (PPO) kullanıyorsanız da, her epoch'ta verinin sadece bir dönemine odaklanmamak için episode başlangıç noktalarını çeşitlendirin. Mesela her yeni episode'da başlangıç yılını rastgele seçebilirsiniz. Bu sayede ajan, tüm 24 yıllık döngüyü görür ve farklı koşullara maruz kalır.
- **Parallel Envs**: Mac M1'iniz var, 8 çekirdeğe kadar çıkabiliyor. Stable Baselines PPO gibi algoritmalar, birden fazla ortam örneğini paralel çalıştırıp veri toplayabiliyor (**vecenv**). Bunu kullanmak eğitimi hızlandırabilir. Örneğin paralelde 8 farklı dönemden veri çekerek PPO'nun batch'lerini doldurabilirsiniz. A3C tarzı yöntemler de benzer şekilde paralel çalışıyor. Bu M1 CPU için uygundur çünkü birden fazla çekirdeği kullanır.
- **Model boyutu ve karmaşıklığı**: Sinir ağınızı olabildiğince kompakt tutun. Finans verisinde devasa derin ağlar kullanmak overfitting riskini artırır ve donanımda yavaşlar. Örneğin, 2-3 katmanlı, her katmanda belki 64-128 nöronlu bir ağ ile başlayın (tabii durum vektörünüzün boyutuna göre de değişir ama). LSTM veya GRU gibi zaman dizisi hafızası tutan katmanlar kullanmak isterseniz, bunlar da CPU'da ağır olabilir. Öncelikle *feed-forward (MLP)*bir mimariyle başlayın; eğer ajan çok miyop davranıyorsa veya çok gürültüye takılıyorsa o zaman belki bir LSTM eklemeyi düşünürsünüz. Ancak unutmamak gerekir ki RL ajanı geçmiş birkaç adımı kendisi de gözlem olarak alabilir. Örneğin state tanımınıza son 10 barın fiyat hareketlerini veren özellikler koyarsanız, ağ belki LSTM olmadan da kısa vadeli trendi anlayabilir.
- **Doğrulama ve model seçimi**: RL eğitiminde overfitting'i ölçmek gözetimli öğrenme kadar net değil (çünkü bir loss değerine bakıp anlamak güç). Bu yüzden düzenli aralıklarla ajanı doğrulama verisinde test edin. Örneğin her X adımda bir, eğitim modunu durdurup 2016-2020 döneminde sabit bir politika ile trade ettirip sonuçlarına (Sharpe, max drawdown vs) bakabilirsiniz. Eğer doğrulama performansı bir noktadan sonra düşmeye başlıyorsa (overfit olduğunu ima eder) eğitim sürecini durdurup en iyi gördüğünüz politikayı seçebilirsiniz. Bu, özellikle uzun süre eğitim yapacaksanız önemli.

Aşamalı Geliştirme Planı

Botunuza bütün modülleri **aynı anda entegre etmek yerine** kademeli olarak eklemek en doğru yaklaşım olacaktır. Böylece hangi eklemenin ne fayda

sağladığını da gözlemleyebilirsiniz. Aşağıda mantıklı bir aşamalı plan özetlenmiştir:

1. Aşama – Temel RL Stratejisi ve Çekirdek Modül: Önce **tek bir varlıkta ve tek bir zaman diliminde** çalışan temel bir RL ajanı eğitin. Örneğin EURUSD paritesinde 15-dakikalık grafikten öğrenen bir PPO ajanı ile başlayabilirsiniz. Durum uzayı olarak sadece temel teknik göstergeleri kullanın (fiyatın kendisi, hareketli ortalamalar, RSI, MACD gibi birkaç indikatör). Aksiyon uzayını basit tutun (örn. pozisyonu -1, 0, +1 olarak değiştir, yani short/flat/long şeklinde). Bu aşamada **transaction cost ve temel risk limit cezalarını** mutlaka uygulayın ki başlangıçtan itibaren strateji gerçekçi olsun. Amaç, bu minimal kurulumu ayağa kaldırıp ajanın gerçekten öğrenip öğrenemediğini görmek. İlk denemelerde belki sürekli kaybeden veya rastgele gibi davranan bir politika elde edebilirsiniz – önemli değil, hiperparametreleri ayarlayarak ve belki state/features setini iyileştirerek **pozitif getiriye** ulaşmaya çalışın. Bu aşama, sistemin iskeletini doğrulamaktır.

2. Aşama – Risk Yönetimi ve İnce Ayarlar: Temel ajan çalışmaya başladığında (örneğin en azından kar etmeye yakın veya küçük kar yapan bir duruma geldiğinde), **risk yönetimini daha da sıkılaştırın**. Ödül fonksiyonunuzda drawdown cezasının ağırlığını ayarlayabilirsiniz (λ parametresi gibi). Hedefiniz, agent'ın getiri-eğrisinin mümkün olduğunca düzgün olması; günlük %5 kayıp sınırına eğitim sırasında nadiren bile yaklaşmaması. Bu aşamada, belki **risk odaklı ölçütlerle**performansa bakın: Sharpe oranı, Sortino oranı, maksimum drawdown yüzdesi vs. Bu metrikleri iyileştirmek için ajanınıza ufak müdahaleler yapabilirsiniz. Örneğin, eğer hala ara sıra büyük kayıplar gözleniyorsa, ceza katsayısını yükseltin veya belki **risk-fakir dönemlerde ödülü sınırlandırın** (agent çok yüksek getiri elde etmeye çalışırken risk alıyor olabilir; bunu dengelemek için bir *utility function* yaklaşımı düşünebilirsiniz: ödüldeki her bir birim getiri, portföy dalgalanması yükseldikçe daha az faydalı olsun gibi). Bu aşamada ayrıca **distributional RL** konseptini entegre etmeyi deneyebilirsiniz: Eğer PPO kullanıyorsanız doğrudan distributional yapması kolay değil, ancak DQN kullanan bir versiyon denerseniz C51 algoritmasını bir benchmark olarak test edebilirsiniz. Ya da actor-critic yapıda, kritiğin birden fazla dağılım dilimini tahmin ettiği bir yöntem araştırabilirsiniz. İleri düzey bir adım olarak, **CVaR optimizasyonunu**kısmen taklit eden bir eğitim yapabilirsiniz: Örneğin belli periyotlarla en kötü episode'ların gradientlerini daha fazla ölçeklendirmek gibi bir yöntem aklınızda olabilir (bu oldukça araştırma konusudur, uygulaması karmaşık olabilir). Unutmayın, bu risk odaklı ince ayarların amacı, stratejinizin **prop firm tarafından istenen güvenli bölge** içinde kalmasını garantilemek.

3. Aşama – Çoklu Zaman Çerçevesi ve Rejim Analizi: Temel ajan tek bir timeframe'deki veriden öğreniyordu. Gerçek trader'lar ise genelde **farklı zaman dilimlerini** dikkate alır (örneğin günlük trend yönü yukarıysa, 15-dk grafikte long pozisyonlar daha avantajlı olabilir gibi). Botunuza bu yetiyi kazandırmak için, durum vektörüne **daha yüksek zaman dilimlerinden** özellikler ekleyebilirsiniz. Örneğin 15-dk veriye ek olarak, **1 saatlik** grafikten gelen trend göstergeleri (MA, trend yönü), **4 saatlik** veya **günlük** grafikten bazı sinyaller eklenebilir. Bu sayede ajan sadece lokal fiyat hareketini değil, daha büyük resimdeki durumu da bilecek. Bu entegrasyonun eğitim performansını bir miktar

yavaşlatabileceğini fakat orta vadede daha istikrarlı kararlar aldırabileceğini bekleyebiliriz.
Aynı zamanda, verinizde farklı **piyasa rejimleri** olduğunu unutmayın: Trendin güçlü olduğu dönemler, yatay piyasa dönemleri, yüksek volatilite dönemleri vs. Rejim analizi yapmak, bu rejimleri önceden belirleyip ona göre stratejiyi değiştirmek anlamına geliyor. Bir yaklaşım, her adımın state'ine bir **rejim göstergesi** eklemektir. Örneğin, son 1 ayın volatilitesi yüksek mi düşük mü, piyasa son dönemde trend mi yapıyor (Directional Movement göstergesi gibi) vb. parametreler ajan için birer girdi olabilir. Bu sayede agent, dolaylı olarak içinde bulunduğu rejimi **hisseder** ve aldığı aksiyonları ona göre ayarlamayı öğrenebilir. Alternatif (daha ileri) bir yaklaşım, **farklı rejimler için ayrı alt-ajanlar**eğitip bir üst seviye politika ile bunlar arasında geçiş yapmak (bu aslında bir tür meta-RL veya opsiyonel kompozisyon problemine giriyor). Bu ikinci yol çok daha karmaşık, bu yüzden başlangıçta tek bir ajanın rejimi feature'lardan öğrenmesine odaklanmak daha iyi. Yine de, eğer analizleriniz belirgin rejim farklılıkları gösteriyorsa (mesela 2008 finans krizi dönemi vs diğer dönemler gibi), belki bu aşamada veriyi rejimlere bölüp ajanı her birinde ayrı eğitip sonra bilgiyi birleştirmeyi düşünebilirsiniz. Ancak bu opsiyonel ve ileri seviye bir hamledir.
4. Aşama – Ekonomik Takvim ve Temel Veri Entegrasyonu: Elinizdeki 24 yıllık ekonomik takvim verisi, botun **makroekonomik olay farkındalığı** kazanmasını sağlayabilir. Özellikle FX ve endeks işlemlerinde, merkez bankası faiz kararları, istihdam raporları, enflasyon verileri gibi olaylar fiyatlarda ani ve büyük hareketlere yol açar. Bu olayları botunuza öğretmek istiyorsunuz, ancak bunu dikkatli yapmak lazım: Makro veri sayısı çok fazla ve her birinin etkisi farklı olabilir. Nasıl entegre edebilirsiniz?

- Öncelikle, takvim verisinden **önemli olayları seçin**. Örneğin "FED faiz kararı, ABD Tarım Dışı İstihdam, TÜFE enflasyonu, ECB toplantısı" gibi piyasa üzerinde bariz etkisi olanları belirleyin. Her olay için de belki bir "önem derecesi" vardır (verinizde muhtemelen önem seviyesi var – üç yıldızlı vs tek yıldızlı gibi).
- State vektörünüze, **yakın zamanda olacak olayların göstergelerini** ekleyin. Örneğin, önümüzdeki 1 gün içinde yüksek önem dereceli bir haber varsa, buna dair binary bir bayrak olabilir. Ya da tam olay anında iseniz, ajan belki pozisyon kapatmayı öğrenmeli – bunu sağlamak için o an bir "event volatilitiy" sinyali verebilirsiniz.
- Ayrıca gerçekleşen verilerin beklentiye göre sürprizlerini de dahil etmek isteyebilirsiniz (örneğin faiz oranı beklentiden yüksek çıktı mı vs.). Fakat bu seviye detay, RL ajanının anlayabileceği veya ihtiyaç duyacağı bir şey olmayabilir başlangıçta. Öncelikle **zamanlama farkındalığı** kazandırmak daha önemli: Ajan, dakikalar sonra çok önemli bir haber geleceğini biliyorsa, büyük pozisyon almaktan kaçınmalı, belki o an hiçbir işlem yapmamayı öğrenmeli. Bunu öğrenmesi için, state'ine "önümüzdeki X adım sonra büyük bir event var" bilgisi girmeli ve ödül mekanizmasında da dolaylı bir etkisi olmalı (örneğin, event sırasında oluşan yüksek belirsizlikte işlemde olduğu

için ani zarar ettiyse bunu görecek).

- Bu modülü entegre ederken dikkat: Makro veriler seyrek gerçekleşir (her gün değil) ve RL ajanı eğer bunların etkisini net görmezse yok sayabilir. Bu yüzden, belki eğitimin ilerleyen safhalarında, ajan zaten temel trade işini öğrendikten sonra bu bilgiyi eklemek mantıklı. Hatta yapılabilir bir yöntem: **İnce ayar (fine-tuning)**. Yani, agent'ı bir süre makro veri olmadan eğittiniz, sonra makro veriyi state'e ekleyip bir süre daha eğiterek adaptasyonunu sağladınız. Bu, transfer öğrenme gibi çalışabilir: Ajan eski stratejisini makro datayla günceller.
- Son olarak, makro veri entegrasyonunun başarısını ölçmek için, mesela geçmişteki **haber dönemlerindeki performansına** bakabilirsiniz. Makro entegre etmeden önce, diyelim ECB faiz kararı günlerinde büyük zararlar ettiğini gördünüz; entegre ettikten sonra bu zararlar azalmalı (çünkü belki o günleri pas geçiyordur artık).

5. Aşama – Çoklu Varlık ve Portföy Yönetimi: İlk başta EURUSD ile başladık diyelim; benzer şekilde GBPUSD, USDJPY gibi her birine ayrı birer ajan eğitebilirsiniz. Eğer **her varlık için ayrı bot** kullanmak sorun değilse, bu en kolayıdır – çünkü her biri kendi özel modelini öğrenir, ve belki hepsini ayrı ayrı FTMO hesabında çalıştırabilirsiniz. Ancak birden çok piyasada aynı anda işlem yapacak **tek bir bot** istiyorsanız, bu daha karmaşık bir modül gerektirir:

- **Portföy bazlı yaklaşım**: State, tüm ilgili varlıkların durumunu içerecek; aksiyon, her varlık için pozisyon değişimini verecek (örneğin 3 varlık varsa bir aksiyon vektörü 3 boyutlu olabilir, her biri -1,0,1 gibi). Bu aslında multi-dimensional continuous action demek. Bazı çalışmalar **multi-asset** RL ajanları denemiştir; örneğin 30 Dow Jones hissesi üzerinde aynı anda işlem yapan bir PPO ajanı referanslarda mevcuttudata-science-blog.comdata-science-blog.com. Bu tarz bir ajan portföy optimizasyonu gibi çalışır, korelasyonları vs de öğrenebilir. Fakat M1 üzerinde bu kadar büyük state-action uzayını öğrenmek çok zor ve yavaş olabilir.
- Alternatif olarak, **multi-agent** gibi düşünebilirsiniz: Her varlık için bir alt-ajan ve belki bunları yöneten bir üst politika. Ancak bu, multi-agent RL dediğimiz karmaşık yapıya girer ki aşağıda ayrıca tartışacağız (genelde önerilmez ilk aşamada).
- Pratikte, bence **her varlık için ayrı model** yaklaşımını tercih edin başlangıçta. Böylece EURUSD modelini iyice optimize edip FTMO sınavını onunla geçmeye odaklanabilirsiniz. Sonra aynı yapıyı diğerine uygularsınız. Hepsi başarılı olunca belki sermayeyi paylaştırıp hepsini bir sepette çalıştırırsınız. Tek dikkat edilmesi gereken, bu botlar aynı hesapta işlem yapacaksa, risk yönetimini toplamda düşünmeniz gerektiği. Örneğin EURUSD botu ve GBPUSD botu aynı gün ters pozisyonlar açıp birlikte portföyü %5 düşürebilir. Bunu önlemek için belki **hesap düzeyinde bir risk izleme** sistemi gerekir (her bot kendi riskini limitliyor ama toplam risk limitini de kimse aşmamalı). Bu organizasyonel bir konudur; teknik olarak her bot kendi verisinde

eğitildiği sürece sorun yok.

6. Aşama – Gelişmiş Algoritma ve Modül Denemeleri: Temel yapı çalışıp, risk yönetimi ve veri entegrasyonlarıyla tatmin edici sonuç verdiğinde, hâlâ iyileştirme potansiyeli görüyorsanız daha **ileri modülleri** deneyebilirsiniz:

- **Distributional RL & Risk-Sensitive Learning:** Yukarıda bahsettiğimiz dağılımsal yöntemleri bu noktada daha ciddi uygulayabilirsiniz. Örneğin halihazırda çalışan PPO/SAC ajanınız varsa, benzer bir ortamda IQN (Implicit Quantile Network) veya QR-DQN gibi bir ajan eğitip performansına bakın. Son araştırmalar, dağılımsal algoritmaların klasik yöntemleri getiri açısından %30'dan fazla geride bırakabildiğini gösteriyorarxiv.org. Ayrıca bunlar, risk iştahını ayarlamak için doğal bir avantaj sunuyor: IQN veya C51 gibi yöntemler, farklı risk seviyelerine (CVaR gibi) optimize edilebiliyorarxiv.org. Yani belki stratejinizin çok agresif olduğunu düşünürseniz, C51'i CVaR$_{0.1}$'ı maksimize edecek şekilde eğitip daha korumacı bir politikayla kıyas yapabilirsiniz. Bu tür denemeler, zaten çalışan bir stratejiyi daha da **parlatmak** ve risk-getiri dengesini en iyi hale getirmek için akademik yaklaşımları uygulamak anlamına gelir.
- **Transformer-Tabanlı RL (Decision Transformer)**: Transformer mimarileri son yıllarda RL'de de kullanılmaya başlandı. **Decision Transformer**, geçmiş durum-eylem-ödül dizilerini girdi olarak alıp, bir dil modeli gibi işleyerek eylem üretir. Bu, özellikle **offline RL** için geliştirilmiş bir yöntemdir; yani elinizdeki tarihi veriden uzman davranışlar varsa onlardan öğrenebilir. Sizin durumunuz tam klasik offline RL değil (biz agent'ı simülasyonla eğittik), ancak belki elinizdeki en iyi ajanların deneyimlerini kullanarak bir decision transformer fine-tune etmek istersiniz veya doğrudan ham veriden bir sequence model kurmak istersiniz. Bu çok ileri bir modül, ve **hesaplama olarak ağır** olacaktır (transformer demek çok sayıda parametre ve matris işlemi demek). MacBook M1 üzerinde büyük bir transformer eğitmek neredeyse imkânsız olabilir – genelde bu işler için GPU şart. Fakat isterseniz küçük ölçekli bir deneme yapabilirsiniz: mesela GPT-2 tabanlı bir decision transformer'ı, en iyi ajanınızın geçmiş trade serileri üzerinde fine-tune etmek gibi. Yakın zamanlı bir çalışma, önceden GPT-2 olarak eğitilmiş bir dil modelini LoRA tekniğiyle ince ayar yaparak offline trading verisine uygulamış ve **CQL, IQL gibi** RL algoritmaları kadar iyi performans elde ettiğini raporlamıştırarxiv.orgarxiv.org. Bu, büyük dil modellerinin finans zaman serilerindeki karmaşık bağımlılıkları yakalama potansiyelini gösteriyor. Ancak bu yaklaşımın sizinki gibi bir proje için **doğrudan bir gereksinim olmadığını** vurgulamalıyım – daha çok araştırma amaçlı bir ek modül olur. Yine de gelecekte elinize daha iyi bir donanım geçerse veya bu projeyi derinleştirmek isterseniz, transformer tabanlı bir çözümü portföyünüze eklemeyi düşünebilirsiniz.
- **Meta-RL (Öğrenmeyi Öğrenme)**: Meta-RL, ajanınızın yeni bir ortama

veya değişen koşullara **hızla uyum sağlamayı öğrenmesi** demektir. Sizin senaryonuzda meta-RL kullanılabilecek iki alan var: (1) **Farklı varlıklar veya piyasalar arası adaptasyon** – mesela EURUSD'de eğitilen ajanı hafif bir güncellemeyle GBPUSD'ye uygulamak, (2) **Rejim değişikliklerine adaptasyon** – piyasa dinamikleri değiştiğinde (örneğin bir dönem momentum stratejileri iyi çalışırken başka bir dönem çalışmıyor), ajan bunu sezsin ve politikasını değiştirsin. Meta-RL yöntemleri (MAML, RL^2 gibi) genelde çok fazla deneyim ve hesaplama gerektirir. Bu nedenle, pratikte ancak elinizde bol compute olduğunda ve birçok farklı senaryoda ajanı eğitebildiğinizde işe yarar. İlk başta meta-RL'ye odaklanmayın; fakat ileride örneğin bir ajanı birden fazla döviz paritesinde ortak eğitmek isterseniz, meta-learning kullanarak her birine hızlı adapte olabilen bir "genel" ajan fikrini araştırabilirsiniz. Bu konsept, şimdilik **opsiyonel gelişmiş modül** olarak kalsın.

- **Multi-Agent RL**: Finans piyasaları doğası gereği çok sayıda ajanı (trader'ı) içerir, bu yüzden akademik olarak **çok-etmenli RL** ilgi çekici görünür. Örneğin bir çalışma, rekabetçi iki ajan PPO'yu aynı ortamda birbirine karşı işlemler yaptırarak tek başlarına olduğundan daha iyi performans elde ettiklerini gösterdiblogs.mathworks.com. Teoride, rekabet ortamı ajanı daha dayanıklı ve akıllı yapabilir çünkü karşısında sürekli onu alt etmeye çalışan bir rakip vardır. Ancak, pratikte sizin amacınız tek bir botu piyasada iyi kılmak olduğu için, kendi kendine rakip oluşturmak zor ve anlamsız olabilir. Multi-agent senaryolar genelde piyasanın bir bölümünü modelleyen yapay ajanlar (market maker, arbitrageur vs) ile öğrenen bir trading agent'ı bir arada simüle etmeye gider ki bu **çok karmaşık** bir çerçeve. MacBook üzerinde bunu simüle etmek de muhtemelen aşırı yavaş olacaktır çünkü her bir ekstra ajan, ortamın boyutunu katlar. Bu nedenle, multi-agent RL'yi gerçekçi bir modül olarak **dahil etmemenizi** öneririm (en azından başlangıçta ve orta vadede). İleride araştırma amaçlı merak ederseniz, iki ajanlı küçük bir oyun kurgulayabilirsiniz (mesela bir ajan alım satım yaparken diğeri fiyatı belirlesin gibi), ama bunun somut getirisi belirsiz. Tek ajan, geniş piyasa veriyle eğitildiğinde zaten dolaylı olarak pazarın tüm katılımcılarıyla etkileşimi öğreniyor denebilir.
- **Evrimsel Algoritmalar / Neuroevolution (NEAT, ES)**: Listelediğiniz modüller arasında bunlar da var. Evrimsel stratejiler (Evolution Strategies) veya NEAT gibi algoritmalar, **öğrenme yerine evrim yoluyla** ajan politikası bulmaya çalışır. Örneğin NEAT, bir popülasyon nöron ağını rastgele mutasyonlarla sürekli geliştirerek bir hedefe ulaşır. Avantajı, gradient descent kullanmadığı için her adımda türev hesaplamaz; dezavantajı, çok fazla deneme (genotip) gerektiği için çok **hesaplama** ister. Finans problemi gibi karmaşık bir alanda NEAT'in başarı sağlaması için binlerce nesil deneme yapmak gerekebilir ki M1

bunu makul sürede yapamaz. Yakın tarihli bir çalışma, NEAT algoritmasını borsada teknik indikatörlerle destekli bir stratejiye uygulamış ve elde ettiği modelin **buy-and-hold stratejisine benzer getiri** sağladığını ancak bunu daha düşük risk ve daha stabilite ile yaptığını raporlamıştırarxiv.org. Yani NEAT ile gelen çözüm, aşırı kâr getirmemiş ama riski azaltmış; bu aslında evrimsel algoritmaların tipik bir sonucudur – optimuma çok yakın gitmezler ama yerel daha güvenli bir çözüm bulurlar. Sizin hedefiniz maksimum kar olduğu için, belki NEAT ile benzer bir kâr seviyesine daha düşük riskle ulaşmak ilginizi çekebilir, ama görüldüğü gibi zaten risk kısıtını RL ile de entegre edebiliyoruz. **Evolution Strategies (ES)**ise OpenAI'nin de denediği, RL problemini saf bir optimizasyon problemi gibi ele alan bir yöntemdir. ES, paralel olarak milyonlarca rastgele politika deneme imkanı varsa iyi çalışır; sizde böyle bir imkan yok. Özetle, evrimsel yaklaşımları aktif geliştirme modülünüze almanızı **önermiyorum**. En iyi ihtimalle, belki bir hobi denemesi olarak bakılabilir. (Eğer RL eğitiminiz bir türlü stabil sonuç vermezse, "acaba bir genetic algorithm ile portföy optimizasyonu yapsam mı?" diye düşünebilirsiniz, ama bu ayrı bir proje gibidir.)

7. Aşama – Modelin Açıklanabilirliği (XAI + RL): Botunuz iyi bir performansa ulaştıktan sonra, son olarak **neden-sonuç ilişkilerini anlamak** ve stratejinin **açıklanabilir** olmasını sağlamak isteyebilirsiniz. Özellikle prop firmalar veya yatırımcılar için, botun mantığını biraz olsun açıklayabilmek güven verici olabilir. RL modelleri genelde birer "kara kutu"dur; ancak burada XAI teknikleri devreye girebilir:

- **SHAP (Shapley Additive Explanations)**: SHAP, herhangi bir makine öğrenmesi modelinin çıktısına her bir özelliğin ne kadar katkı yaptığını hesaplayan bir yöntem. Finans verisinde eğitimli bir DQN ajanı üzerinde SHAP uygulanarak, ajan bir kararı alırken hangi indikatörlerin onu tetiklediği analiz edilebilmiştirarxiv.org. Sizin botunuz için de benzer bir yaklaşım kullanılabilir. Örneğin, botun belirli bir anda **"sat"** kararı aldığı bir durumu ele alın; o andaki state özelliklerini SHAP ile değerlendirin, belki göreceksiniz ki "10 barlık momentum göstergesi çok düşüktü ve yaklaşan bir haber vardı, bu yüzden satmaya karar verdi" gibi bir yorum çıkarabilirsiniz. Bu tip analizler, modeli geliştirmek için de faydalı olabilir – mesela botun aslında gereksiz bir özelliğe fazla önem verdiğini fark ederseniz o özelliği düzeltebilirsiniz.
- **Kural Tabanlı Yakınsama:** Bir diğer teknik, eğitilmiş modelin davranışını **basitleştirilmiş bir modelle taklit etmek**. Örneğin, RL ajanınızın yaptığı işlemleri toplayıp, bunları açıklayan bir **karar ağacı (decision tree)**eğitebilirsiniz. Bu ağaç, "eğer 5-günlük MA, 20-günlük MA'nın üzerindeyse ve RSI > 70 ise ve son haber etkisi düşükse, Sat" gibi kurallar çıkarabilir. Elbette karar ağacı her durumu kapsamaz ama önemli durumları yakalayabilir. Bu şekilde, botun stratejisini kabaca insan diline döken bir kural listesi elde edebilirsiniz.

- **Açıklanabilirlik ve Regülasyon:** Eğer bot gerçek hayatta kullanılsa, regülatörlerin veya risk departmanlarının bu tip açıklanabilirliğe ihtiyacı olabilir. Şu an için belki sadece prop firm sınavına yönelik düşündüğünüzden, bu kısım en son öncelik. Modeliniz başarılı olduktan sonra, performansı kadar **nasıl kararlar aldığını** da belgelemeniz uzun vadede faydalı olacak. Hatta bu sayede kendi stratejinizi de geliştirebilirsiniz.

Önerilmeyen veya Sonraki Aşamalara Bırakılacak Modüller

Yukarıdaki plan kapsamında, bazı modülleri özellikle **dahil etmedik** veya ileriki aşamalara öteledik. Sizin ilk sorunuzdaki "hangilerini hiç koymamalıyım" kısmını şöyle özetleyebiliriz:

- **AlphaZero Tarzı Self-Play RL:** Bu yaklaşımı **koymamalısınız**, çünkü finans piyasası bir satranç oyunu değil. Self-play gerektiren bir iki ajanlı ortam yok ortada; tek bir ajan ve büyük bir **piyasa** var. AlphaZero gibi algoritmalar, tam rekabetçi oyunlarda müthiş sonuçlar verdi ama bir trader'ın kendi kendine oynaması konsepti finans için uygun değil. Zaten akademik araştırmalar da self-play'i finans alanına uygulamanın net bir yöntemini bulabilmiş değil; finans örneklerinde genelde ya bir piyasa simülatörü ya da kısıtlı denemeler var ve çoğu hisse senedi gibi alanlarla sınırlı kalmış durumdapmc.ncbi.nlm.nih.gov. Kısacası, bu modül pratik fayda sağlamayacak.
- **Multi-Agent (Çok Etmenli) RL:** Yukarıda değindiğimiz gibi, çoklu ajanlar kurup birbirine karşı veya birlikte öğrenmelerini sağlamak gereksiz karmaşıklık getirecek. Piyasanın rekabetçi doğası teorik olarak multi-agent yaklaşımları çekici kılsa da, bunları uygulamak hem zor hem de hesaplama açısından masraflıdırblogs.mathworks.com. Sizinki gibi tek bot geliştirme projesinde, multi-agent dinamikleri **hiç eklememek** en iyisi. Birden fazla varlık için dahi, separate agent veya tek agent çözümleri varken, multi-agent öğrenme yoluna gitmek projeyi dağıtır.
- **Meta-RL (Öğrenmeyi öğrenme) Başlangıçta:** Meta-RL'yi direkt entegre etmeyin. Bu, ajanınızı farklı görevler üzerinde genel bir öğrenici yapma amacı güder. İlk etapta sizin net bir göreviniz var (belirli piyasada trade). Meta-RL ancak ileride, botun farklı piyasalara adaptasyonu veya değişen piyasa koşullarına anında uyumu sorun olursa gündeme gelebilir. Onu da belki veri augmentasyonu veya transfer learning ile çözebilirsiniz. Meta-RL akademik ve kompleks bir modül; **başlangıç modül listenizden çıkarın**.
- **Evrimsel Algoritmalar (NEAT, Genetik)**: Bunları da doğrudan entegre etmeyin. RL'nin alternatifi veya destekleyicisi olarak düşünebilirsiniz ama yukarıda tartıştığımız gibi **hesaplama yükü çok fazla** ve getirisi belirsiz. NEAT ile optimuma yakın bir strateji bulmak binlerce deneme gerektirebilir. Ayrıca literatürdeki sonuçlar da RL'ye üstün olduğunu göstermiyor – örneğin NEAT ile eğitilen bir borsa stratejisi, getiri açısından basit **buy-and-hold** ile benzer kalmış sadece riski biraz

azaltabilmişarxiv.org. Sizin hedefiniz buy-and-hold'u *geçmek* olduğuna göre, NEAT muhtemelen doğru araç değil. Kısacası, evrimsel modülleri listenizden çıkarın, en azından bu proje kapsamında.

- **A3C/A2C (Asenkron Aktör-Kritic):** Bunlar PPO öncesi popüler yöntemlerdi. Eğer PPO'yu uygulayabiliyorsanız, A2C'yi ayrıca koymanıza gerek yok. Yine de, belki kod basitliği açısından **A2C**, M1 gibi CPU tabanlı bir sistemde iyi paralel çalışabilir (PPO'dan biraz daha hafif bir algoritma). Ama optimum bakışla, **PPO varken A2C şart değil**. A3C de zaten aynı aile. Dolayısıyla bu modüller ayrı ayrı entegre edilmeyebilir. Sadece belki kıyas deneyleri için kullanabilirsiniz ama sürekli botun parçası olmaları gereksiz.
- **DDPG (Temel)**: DDPG modülünü de entegre etmeyin, bunun yerine TD3'ü alın dedik. Çünkü DDPG'nin bilinen kusurlarını TD3 çözüyor. Aynı şekilde, **Twin Delayed DDPG** zaten TD3 demek – listede ayrı yazmışsınız ama aslında aynı şeyden bahsediyoruz. Bu isim karmaşasını da giderip tek bir TD3 modülü yeterli olacaktır (DDPG'yi "hiç koymamak" anlamına gelir bu).
- **Rainbow DQN vs Diğerleri:** Rainbow DQN güçlü bir ajan ama continuous eylemlerde uygulanamaz dedik. Eğer continuous aksiyon yoluna giderseniz, **Rainbow'u koymamalısınız**. Discrete aksiyon deneyecekseniz Rainbow'u entegre edebilirsiniz; ancak hem Rainbow hem PPO hem SAC hepsini birden entegre etmek pratik değil – önce birini seçip denemek daha mantıklı. O yüzden, belki Rainbow'u ilk denemede **dahil etmeyebilirsiniz** ve önceliği PPO/SAC'ye verirsiniz. Distributional RL'yi tamamen kaldırmayın, zira ileride risk konusunda lazım olabilir, ama Rainbow'un bütün parçalarını uygulamak şart değil.
- **Transformer Tabanlı RL:** Bu modülü de aktif geliştirme listesinde **en sona atın** veya hiç koymayın derim. Gerekçesi donanım kısıtından ötürü: transformer'lar muazzam parametre sayılarıyla M1'de eğitilmesi günler, haftalar alacak modeller. Üstelik geleneksel RL'nin ötesinde uğraştırıcı. Decision Transformer gibi yöntemler ilgi çekici olsa da bunları şu an yapmaya kalkmak asıl projenizi (kâr eden bir bot yaratmayı) geciktirebilir. Bunun yerine, belki **daha küçük bir Sequence Model** (ör. bir tek-layer LSTM) ekleyerek başlayabilirsiniz (bu transformer olmasa da bir bellek katar). Transformer konusunu, eğer başka hiçbir yöntem işe yaramazsa ve elinizde hala geliştirme süresi kalırsa düşünün.
- **Açıklanabilirlik (XAI) Entegrasyonu:** XAI modülü, botun performansını artırmaz, sadece anlamayı artırır. Bu yüzden, bunu eğitim döngüsüne dahil etmeye gerek yok. Yani SHAP hesaplamalarını her eğitim adımında yapmayacağız tabii. XAI'yi bir **analiz aracı** olarak ele alıp en sona bırakmak doğru. Kodunuza belki entegre edebilirsiniz ama bu, core botun parçası değil, onu tüketen bir üst seviye araç gibi düşünebilirsiniz.

- **Prop Firm Uyumu**: Aslında bu bir modülden çok bir hedef. Fakat yine de belirtelim: Prop firm kurallarına uyumu sağlamak için yaptığımız risk yönetimi entegrasyonlarından ödün vermeyin. Bu kurallar bir modül olarak çıkarılmamalı, tam tersine esastır. Yani "hiç koymamak" burada geçerli değil; tam aksine koymadan asla olmaz. Dolayısıyla prop firm uyum modülünü zaten baştan entegre etmiştik (risk limitleri ve cezaları ile).

Sonuç olarak, ilk etapta **en yalın, gerekli parçaları** tutup geri kalan her şeyi atmalısınız. Bu, geliştirme hızınızı artırır ve karmaşıklık yüzünden çıkabilecek hataları engeller. Unutmayın, **basit stratejiler çoğu zaman karmaşık olanları yenebilir** – özellikle de iyi optimize edilmişse. Önce basit bir şeyi çalışır hale getirin (küçük kar da olsa, kurallara uyuyor olsun), sonra adım adım karmaşıklık ekleyerek onu geliştirin. Bu yaklaşım, modüllerin sıralanmasında da rehberiniz olsun.

Sonuç ve Özet Yol Haritası

Toparlarsak, MacBook M1 üzerinde, prop firm kurallarına uygun maksimum kâr hedefleyen bir RL botu geliştirmek için şu yol haritasını izleyin:

- **Temel algoritma seçimi:** PPO gibi güvenilir bir yöntemle başlayın (continuous aksiyon gerekiyorsa SAC/TD3 de uygun). A2C/DDPG gibi eski sürümleri ayrı tutmayın, doğrudan gelişmiş versiyonlarını kullanındata-science-blog.com.
- **Gerçekçilik ve risk entegrasyonu:** En baştan ortamınıza işlem maliyeti, spread, slipaj ekleyin ve ödül fonksiyonuna drawdown kısıtlarını yansıtın. Ajanın fazla risk alması durumunda ağır cezalar olduğunu öğrenmesi gerekiyormilvus.io.
- **Kademeli eğitim:** Tarihsel verinizi simüle ederek ajanı eğitin. Önce tek bir piyasada ve basit özelliklerle başlayın; performansı kabul edilir düzeye gelince yeni veri kaynakları (diğer zaman dilimi, makro olaylar) ekleyin. Bu aşamalı yaklaşım, hem her eklemenin katkısını izlemenizi hem de overfit olmadan genelleşmeyi sağlar.
- **Gelişmiş teknikler:** Temel botu oluşturduktan sonra risk-ayarlı optimizasyonu ilerletmek için distributional RL ve/veya CVaR odaklı yaklaşımları deneyebilirsinizarxiv.org. Ancak bunlar ikinci planda – önce basit getiri odaklı eğitimde istikrar yakalayın.
- **Ağır modüllerden kaçınma:** Multi-agent, self-play gibi proje hedefinize direkt hizmet etmeyen yöntemlere vakit harcamayınpmc.ncbi.nlm.nih.govblogs.mathworks.com. Aynı şekilde, donanımınızı zorlayacak devasa modelleri (transformer gibi) hemen gündeme almayın.
- **Açıklanabilirlik ve değerlendirme:** Son olarak, stratejiniz belirli bir seviyeye geldiğinde test edin ve açıklayın. Özellikle out-of-sample testlerde prop firm kriterlerini sağladığından emin olun. Ardından, SHAP veya benzeri araçlarla modelin kararlarını anlamaya çalışınarxiv.org. Bu, stratejinin tutarlılığını ve öngörülebilirliğini artıracaktır.

Bu planı izlerseniz, **önce küçük başlayıp sonra büyüterek** ilerlemiş olacaksınız. İlk başta belki yalnızca %1-2 kazanan ama kural ihlali yapmayan bir botunuz olacak; sonra bunu hyperparameter tuning ile %5 kazanan yapacaksınız; sonra haber etkisini vs ekleyip belki %7-10 kazanan ama yine de drawdown limitlerini aşmayan bir seviyeye getireceksiniz. Her modül eklemesi sonrasında bolca **backtest** ve analiz yapmayı unutmayın. Sonuçta hedefiniz sadece sınavı geçmek değil, gerçek piyasa koşullarında da mantıklı bir sistem elde etmek olmalı.

Kaynaklar:

- S. Gino, *"Deep Reinforcement Learning for Automated Stock Trading"* – Çeşitli RL algoritmalarının finans piyasasında uygulanmasısgino209.medium.comsgino209.medium.com.
- Data Science Blog – *"Actor-Critic Tabanlı RL Algoritmaların Hisse Senedi Trading Performansı"* – PPO'nun yüksek getiri, A2C'nin düşük risk özellikleri üzerine analizdata-science-blog.com.
- Milvus AI Blog – *"How does RL work in financial trading?"* – RL ticaret sistemlerinde risk yönetimi ve ödül tasarımı tavsiyelerimilvus.io.
- J. Posth ve ark., Frontiers in AI 2021 – *"Self-Play Algorithms in Trading"* – Self-play yöntemlerinin finans piyasalarında uygulanabilirliği tartışmasıpmc.ncbi.nlm.nih.gov.
- F. Hêche ve ark., arXiv 2025 – *"Risk-averse policies for futures trading using distributional RL"* – Distributional RL ile CVaR optimizasyonunun faydaları üzerine çalışmaarxiv.org.
- L.-C. Huang, arXiv 2024 – *"NEAT algorithm-based stock trading strategy"* – NEAT ile evrimsel strateji uygulaması ve sonuçlarıarxiv.org.
- S. Kumar ve ark., arXiv 2022 – *"Explainable RL on Financial Trading using SHAP"* – DQN ajanlarının SHAP ile açıklanması üzerine çalışmaarxiv.org.
- MathWorks Blog 2024 – *"Multiagent RL for Financial Trading"* – Çok etmenli rekabetçi ajan örneklerinin tek ajanı nasıl geçtiğini gösteren demoblogs.mathworks.com.
- S. Yun, arXiv 2024 – *"Pretrained LLM as Decision Transformer for Trading"* – GPT-2 tabanlı decision transformer ile offline trading stratejisi öğrenme denemesiarxiv.org.

Bu kaynaklar, söylediklerimizin dayanak noktalarını oluşturuyor ve daha derin teknik detaylar sunuyor. Son olarak, **disiplinli bir geliştirme süreci** dilediğiniz sonuca ulaşmanıza yardımcı olacaktır: Adım adım ilerleyin, her aşamada sonuçları değerlendirin ve gerektiğinde geri adım atmaktan çekinmeyin. Başarılar dilerim!