

Prompt Döngü (Loop) Mühendisliği: Yapay Zekâ Mühendisliğinde Döngüsel Yaklaşım

ENGİN ARSLAN (19.07.2026) (Tahmini okuma süresi: 20 dk)

Giriş: Geleneksel *prompt* mühendisliği, yapay zekâ (YZ) modellerine tek seferlik komutlar yazmayı ifade eder. Döngüsel (loop) mühendislik ise “Yapay zekâ ajanının işi tamamlayana kadar kendi kendine soruları yeniden şekillendirdiği ve çıktıyı değerlendirdiği” bir yöntemdir. Örneğin Anthropic’in Claude Code’u yaratan Boris Cherny artık doğrudan prompt yazmayıp, kendisine sürekli prompt yazan döngüler oluşturduğunu belirtiyor. Bu yaklaşımda insan, her adımda girdiyi yazıp çıktıyı incelemek yerine, iş akışını yöneten sistemi inşa eder, geriye çekilir. Arkada model çağrılarına dayanan geri besleme döngüleri çalışır. Döngüsel mühendislik, yapay zekâ uygulamalarında çok adımlı, “açık hava koşulları” gerektiren işlerde verim ve kaliteyi artırmayı amaçlar.

Döngüsel mühendislik, *Prompt Mühendisliği*’nin tamamlayıcısıdır. Prompt mühendisi iyi bir soruyu belirleyip yapay zekadan dönen cevabı yorumlarken; döngü mühendisi bu soruları otomatik olarak üreten, sonuçları kontrol eden ve gerektiğinde yineleyen bir sistem tasarlar. Bu farkı basitçe ifade etmek gerekirse: Prompt bir el yordamıyla yönlendirmeyse, döngü mühendisliği geride durup işi yürüten otomatik bir sistemi kurma işidir.

Prompt Mühendisliği ve Döngü Mühendisliği Karşılaştırması

- **Prompt Mühendisliği:** İnsan her yinelemede modele yeni bir komut yazar. Tek seferlik görevlerde, insan denetimi mevcutken etkilidir. Yüksek hacimli veya uzun görevlerde ise kısıtlı kalır.
- **Döngü Mühendisliği:** İnsan ilk girdiyi verir, ancak her sonraki adımda sistem otomatik olarak hareket eder. Görev ilerleme kriterlerini ve durdurma koşullarını belirler. İnsan müdahalesine ihtiyaç kalmadan çok adımlı işleri yürütür.

- **Temel Bileşenler:** Modern döngüler genellikle beş ana kısımdan oluşur: Başlatıcı (trigger), Eylem (modellerin/görevlerin çalışması), Değerlendirme adımı, Karar aşaması (yeniden dene/sonlandır/human-in-the-loop) ve Döngü sonlandırma koşulu. Başka bir deyişle, iyi tasarlanmış bir döngü; ne zaman başlayacağını, ne yapacağını, sonucu nasıl olacağını ve ne zaman duracağını bilir. Abhijay Vuyyuru’nun tanımlamasıyla döngü (loop) üç temel unsur içerir: Amaç (Objective) – ulaşmak istenen hedef, Ölçüt (Metric) – her adımda sonucu değerlendiren başarı ölçüsü, Sınır (Boundary) – döngünün sınırları (maksimum yinleme sayısı veya başarının yakalanması). Bu üç öge net belirlenirse sistem sürekli çalışır, insan müdahalesine gerek kalmaz. Örneğin bir LinkedIn yazısı üretme döngüsünde amaç “200 kelime altında yararlı bir fikir öğretmek”, ölçüt “ilk cümle dikkat çeker mi; tek fikir var mı; okuyucu yeni bir şey öğrenir mi” ve sınır “puan 9 altında ise en fazla 10 iterasyona kadar düzeltme döngüsü” şeklinde belirlenebilir.

Döngü Mühendisliğinin Temel Unsurları

- **Hedef (Objective):** Döngünün nihai amacı. Örneğin bir rapor yazmak, testleri geçmek veya hata düzeltmek.
- **Ölçüt (Metric):** Döngünün kendi çıktısını değerlendireceği kıstas. Örneğin yazıda bilgi edinme değeri, kodda testlerin geçme durumu gibi nesnel göstergeler. İyi bir döngü, her yinlemede bu ölçüte göre otomatik puanlama ve iyileştirme yapar.
- **Sınır (Boundary):** Döngünün ne zaman duracağını belirler. Maksimum döngü sayısı, zaman aşımı veya başarının yakalanması gibi koşullar olabilir. Bu, hesaplamalı kaynakların gereksiz tüketilmemesi için elzemdir.
- **Araçlar ve Katmanlar:** Döngüler çoğu zaman LLM’in kendisinin ötesinde araçları (kod çalıştırma, veritabanı sorgusu, web isteği vb.) çağırır. Bu *tools* sayesinde ajan görevini gerçek dünyada gerçekleştirebilir. Döngü mühendisliği aynı zamanda bu araç setini yönetmeyi, paralel işleri (“worktree” izolasyonu) düzenlemeyi, loglama ve hata yönetimi gibi hususları da içerir.

Döngü Mühendisliği Türleri ve Kullanım Modelleri

Döngüsel yaklaşımlar çeşitli biçimlerde uygulanır, temel bazı modeller şunlardır:

- **Refleksiyon Döngüsü:** Ajan kendi önceki cevabını değerlendirir ve gerekirse düzeltir. Örneğin modele “Bu yanıtı değerlendir ve not düştüysen tekrar dene” komutu verilir. Böylece ilk seferde eksik kalan noktalar revize edilir.
- **Eleştirmen (Critic) Döngüsü:** Farklı bir model veya işlev, ajan tarafından üretilen çıktıyı bağımsız olarak kontrol eder. Örneğin bir LLM kod yazarken başka bir LLM veya test modülü kodu değerlendirir. Bu, onaylamayı modelin kendi öz-değerlendirmesine bırakmamak için önemlidir.
- **Planlayıcı Döngüsü:** Karmaşık bir görevi alt görevlere bölen ve bu görevleri farklı ajanlara atan döngü. Örneğin bir özellik geliştirmede önce “yapılacaklar listesini çıkar”, sonra her maddeyi alt ajana ilet. Böylece birden çok ajan paralel çalışır.
- **Yineleme (Retry) Döngüsü:** Ajan bir deneme yapar, başarısız olursa aynı veya farklı parametrelerle tekrar dener. Kod derlenmediğinde tekrar farklı bir strateji uygular gibi.
- **Doğrulama Döngüsü:** Ajan önce işi yapar, sonra ayrı bir model veya araç grubu yaptığı işi test eder. Örneğin kod yazan ajandan sonra ayrı bir test ajanının testleri çalıştırması. Hata varsa geri dönülür.
- **Çoklu Ajan Döngüsü:** Birden fazla özel ajanın işbirliği. Örneğin *Yürütücü Ajan* kod yazarken, *Test Ajanı* kodu otomatik testlere sokar, *Gözden Geçirme Ajanı* stil kontrolü yapar. Koordinatör ajan ise bu aşamaları yönetir. MindStudio’ya göre bu düzenleme multi-agent sistemleri oluşturur.
- **İnsan-Dâhil Döngü (Human-in-the-Loop):** Bazı adımlarda insan onayı alınır. Özellikle kritik kararlar (güvenlik, uyumluluk vb.) için insan denetimi önerilir. Döngü mühendisi, hangi aşamada insan kontrolü gerekip gerekmediğini belirler.
- **Araç Tabanlı Döngüler:** LangChain, OpenClaw vb. çerçeveler kullanılarak, ajanlar için tetikleyiciler ve görev akışları tanımlanır. Örneğin LangChain’de event-driven (cron, webhook) döngüler kurularak ajanlar arka planda sürekli çalıştırılabilir.

- **Hafıza Döngüsü (Memory Loop):** Uzun vadeli bilgi tutmak için harici bellek kullanılır. Modeller her iterasyonda önce sabit diskten veya veritabanından geçmiş bilgiyi çeker, sonra günceller. Çünkü modelin kendi bağlam penceresi sınırlı olduğundan, döngü ilerledikçe “dış” bellek olmazsa önceki bilgiler unutulur.
- **RAG (Retrieval-Augmented Generation) Döngüsü:** Bilgi tabanlı görevlerde her adımda veritabanı sorgusu yapılarak yeni bilgiler modele sunulur. Döngü her seferinde daha fazla kaynak tarayabilir.
- **Protokol Döngüleri (MCP/A2A):** Anthropic MCP veya Google A2A gibi standart protokollerle ajanlar arası iletişim ve çoklu alet entegrasyonu sağlanır. Bu katmanlar, döngü mühendisliğinin en üst altyapısını oluşturur (çoklu ajan keşfi, görev yönlendirmesi).

Bu modeller genellikle bir arada kullanılır. Örneğin bir kodlama döngüsü (Devin, Cursor gibi) birkaç alt ajandan ve yukarıdaki döngü türlerinden oluşur.

Kurumsal Kullanım Senaryoları ve Örnekler

Döngü mühendisliği, halen yazılım geliştirme ve iş süreçlerinde hız/kalite artırma amacıyla deneysel ve pilot aşamada olsa da bazı büyük şirketler tarafından kullanılmaktadır:

- **Yazılım Geliştirme ve Kod Refaktörü:** Brezilya’nın önde gelen fintech şirketi Nubank, kritik bir proje olarak 6 milyondan fazla satırlık ETL kodunu alt modüllere bölmek için Devins adlı bir yapay zeka ajanı kullandı. Devin, mühendislerin tanıdığı örneklerle ince ayarlandı ve görevleri otomatikleştirerek mühendislik süresini 8–12× kısalttı, maliyeti 20× azalttı. Burada Devin bir anlamda arka planda kesintisiz çalışan bir döngü gibiydi: İnsanlar yalnızca sürecin yönetimi ve onay için devredeydi.
- **İşgücü Otomasyonu (Otonom Takımlar):** GitHub Copilot, Anthropic Claude Code, OpenAI Codex, Cursor, Windsurf gibi platformlar, yazılım ekiplerinin ortak kod tabanlarında eşzamanlı çalışan arka plan ajanları sunmaya başladı. Örneğin Anysphere’in desteklediği Cursor, VS Code’da geliştiricilerin sadece düğmeye basarak arka planda multi-dosya düzenleme yapabilen arka plan ajanlarını etkinleştirir. OpenAI’nın Windsurf adı verilen gelecekteki IDE’si “Cascade” ajanın benzer bir işlevsellik sağlayacağı duyurulmuştur. Değişiklik istenen kod parçacıkları

“ajana görev” olarak verilir, ajanlar deęişiklik planlar, kodları yazar, test eder ve PR açar, geliştiriciler öğleden sonra sonuçları gözden geçirir. Bu, ‘gündüz mühendis kod yazar; gece ajan tamamlar’ senaryosudur.

- **Sürekli Test ve Kalite:** Döngü mühendisliği, özellikle yazılım testlerinde de kullanılmaktadır. Otomasyon testleri sürekli tekrar eden bir döngüye alıştırılarak (testi çalıştır, sonucu kontrol et, başarısızsa tekrarla) hızlı geri bildirim sağlanır. Birçok kuruluş, döngü tabanlı sürekli entegrasyon (CI) ile bir adım öteye geçerek “fail fast” modelini hayata geçiriyor. Örneğin, bir CI hattında her kod gönderimi bir ajana test ettirilir; testlerden geçen kod otomatikleştirilmiş gözden geçirme ile onaylanır. Bu sayede hatalar anında yakalanır, geliştirici işi gecikme olmadan sürdürür.
- **Alt Görev Dağılımı (Orkestrasyon):** Büyük projelerde iş bölümü, döngülerle otomatikleştirilebiliyor. Örneğin bir teknoloji firmasının Ürün Yöneticisi (PM) ajanı (*product management agent*) ürettiği döngülerle farklı yazılım ajanlarına görevler atayabilir. MindStudio örneğinde yer alan “Remy” ajanın işlevi tam da budur: kodlama ajanlarına ne yapılması gerektiğini bildiren bir *genel müteahhit* görevi görür.
- **Operasyonel Süreçler:** Türkçe bir kaynak olmamakla birlikte, dünya genelinde bakım/onarımlar gibi rutin işlerde döngü ajanları kullanılıyor. Örneğin, bir e-ticaret sitesinin müşteri destek ajanı, müşteri sorularını sıralayıp önceden belirlenmiş adımlarla cevaplayabilir; ardından alınan yanıtların yeterli olup olmadığını bir başka ajana onaylatabilir (Critic) ve gerekirse soruyu insan müşteri temsilcisine aktarır.
- **Otomatik Geri Bildirim ve DevOps:** Örneğin kod entegrasyonunda, ajanlar tüm testleri geçirene kadar düzeltme döngüsünde kalabilir. Ayrıca sistem performans optimizasyonu için sürekli benchmark döngüleri kurulabilir. NASA gibi kurumlar, simülasyon ve optimizasyon döngüleriyle otonom araç geliştirme döngülerini test ediyor.

•

Türkiye’de Durum: Türkiye’de döngü mühendisliği nispeten yeni bir kavram. Ancak Yapay Zekâ projelerine yatırım yapan bankalar, telekom operatörleri ve yazılım firmaları (Garanti BBVA, Turkcell, Havelsan vb.) pilot çalışmalar başlatmaktadır. Örneğin Microsoft Türkiye, finansal proselere AI ajan entegrasyonu üzerine hackathonlar düzenlemektedir. Yine de tam ölçekli döngü

uygulamalarında daha çok Ar-Ge aşaması vardır. KOBİ'lerin çoğu yapay zekâyı basit chatbot'lar veya otomasyon için kullanırken, büyük oyuncular %100 otomatik döngüsel çözümlere geçmeye hazırlanıyor. Gelecekte yasal düzenlemeler ve yerli bulut çözümleri (TÜBİTAK, ulusal veri merkezleri gibi) bu geçişi etkileyecektir.

Döngü Mühendisliğinde Performans Analizi

Döngü tasarımında performans, gecikme (latency), verim ve maliyet dengesi kritiktir:

- **Maliyet (Token Kullanımı):** Her döngü adımı yeni bir model çağrısıdır. Bu yüzden bir döngü 20 defa dönerse, tek bir iterasyondan çok daha fazla token tüketilir. Yetersiz kontrol ile gereksiz döngüler hızla maliyeti katlayabilir. Örneğin Gartner'a göre çok özerk agent projeleri beklenmedik maliyet patlamalarına yol açıyor. Bu nedenle döngülerde *iteration cap*, *zaman aşımı*, *kullanım analitiği* gibi mekanizmalar önceden tanımlanmalıdır. Türkiye'de bütçe ve token hesabı yapabilecek altyapılar henüz tam oluşmamakla birlikte, döngü tasarımında maliyet bilinci ilk gerekliliklerden biridir.
- **Gecikme (Latency):** Döngüler seri ya da paralel olarak çalışabilir. Seri döngüler (her adımda sonucu beklemek) doğal olarak tek seferlik prompt'a göre daha yavaştır. Ancak paralel ajanlar kurularak birden çok iş aynı anda ilerletilebilir. Daha önce bahsedilen "Nubank" örneğinde ajanlar her biri ayrı bacağına çalışabilecek şekilde planlanmıştır, böylece gecikme tolere edilebilir hale gelmiştir. Ücret-performans analizinde genellikle "mühendislik saatinin maliyeti vs. döngünün token maliyeti" karşılaştırılır (örn. Devin'in \$500/ay ek maliyeti bazı takımlar için kabul edilebilir bulunmuştur).
- **Verim (Throughput):** Döngü mühendisliği, insan beklemek zorunda olmadığı için eşzamanlı birden çok iş akışını mümkün kılar. Örneğin birkaç kod görevini aynı anda yürüten ajanlar sabah sonuçlarını rapor edebilir. Bu, daha yüksek toplu iş hacmi anlamına gelir. Dilimlendirmeden (sharding) veya iş tablolarından yararlanarak, mühendislerin iş yükü paylaştırılabilir. Eksik yönü, çok ajanlı sistemlerde çakışma (concurrency) sorunları olabilir (aynı dosyanın farklı versiyonları gibi). Ancak bu da modern yazılım mühendisliğinde izolasyon teknikleri (git worktree, konteyner vs.) ile yönetilebilir.

- **Bağlam Yönetimi (Context Window):** Uzun döngülerde AI modelinin bağlam hafızası dolar. Bu yüzden özetleme (context summarization) veya harici bellek kullanımı gerekebilir. Örneğin uzun bir rapor yazma döngüsünde ara çıktılar disk üzerinde tutulup modelin girişine gerektiği kadar özet olarak sunulur.
- **Kaynak Verimliliği:** Döngüler, aynı girdiyi tekrar tekrar hesaplamamak için önbellek mekanizmaları içerebilir. Ara sonuçlar cache'lenir, mümkünse hesaplama veya veri sorguları yeniden kullanılabilir. Bu, hem hız hem de maliyet açısından optimizasyondur.

Döngü Mühendisliğinin Riskleri ve Sınırlamaları

Otonom döngüler fayda sağlasa da yeni riskler getirir:

- **Maliyet Patlaması:** Kontrolsüz döngüler token maliyetini fırlatır. Örneğin “bir saat boyunca her dakika API çağıran bir döngü” hızla bütçeyi tüketir. Döngüde *iteration limit*, toplam harcama limiti gibi önlemler şarttır. Gartner da belirsiz iş değeri ve yetersiz yönetimin ajan projelerindeki başarısızlığın başlıca sebebi olduğunu vurgulamıştır.
- **Hatalı Genelleştirme (Hallucination):** Model yanlış bir adım atarsa, döngü bunu tekrar tekrar “olumlar”. Bir insan, anında hatayı fark edip düzeltirken, döngü hatalı bilgiyi pekiştirerek ilerleyebilir. Örneğin bir ajana “eksik kodu tamamla” dediniz ama model yanlış yapıyı algıladıysa; döngü bu hatalı varsayıma dayalı kodlar üretir. Bağımsız doğrulama ve test mekanizmaları kullanılmazsa bu risk büyür.
- **Çakışmalar ve Çakışmalar (Concurrency):** Birden çok ajan aynı anda çalışınca “aynı dosyayı iki farklı ajan değiştirme” veya “aynı sorunu iki kez çözme” gibi sorunlar ortaya çıkar. Takım içinde paralel dallanmalarda Git worktree, konteyner gibi izolasyon yöntemlerine başvurulur. Aksi halde agentlar birbirlerinin işini geçersiz kılabilir.
- **Tamamlanma Güvencesi (“Done” Yanılsaması):** Bir ajanın görevi bittiğini işaret etmesi, gerçekten işin doğru bittiği anlamına gelmez. Döngü sadece kendisine verilen ölçütleri bilir. Eğer ölçüt eksikse (örneğin kod sadece derleniyorsa, testler yoksa), ajan “bitti” diyebilir ama hâlen hatalar mevcuttur. Bu yüzden kritik işlerde insan onayı veya bağımsız testler gereklidir.

- **Güvenlik ve Uyumluluk:** (Kullanıcı istemezken detaylı güvenlik incelemesi yapılmasına gerek yok; ancak risk olarak belirtilebilir.) Döngüler veriye ve ağı sürekli eriştiği için güvenlik açıkları artabilir. Örneğin API anahtarlarının yanlışlıkla sızması veya ajanların kötü niyetli girdiye maruz kalması durumlarına karşı önlemler şarttır. Kurumsal ortamda *governance* (küpeşte politikası, izin kontrolleri) döngülerin sorunsuz çalışabilmesi için gereklidir.

Gelecek Öngörüler (2030 ve Sonrası)

Döngü mühendisliği, yazılım geliştirmenin geleceğinde önemli yer tutacak gibi görünüyor.

Araştırmalar ve piyasa tahminleri şunları gösteriyor:

- **Büyüyen Pazar:** Yapay zekâ ajan pazarının 2025'te ~7,8 milyar dolar iken 2030'da ~52,6 milyar dolara ulaşacağı öngörülüyor. Çok ajanlı çözümlere yatırım yüzde 46–48 civarı yıllık büyüyor. Türk firmalarının bu hızla aynı oranda ilerlemesi beklenmese de, geleneksel otomasyondan YZ ajanlarına yöneliş artacak.
- **Çoklu Ajan Başarısı:** Konvansiyonel bir ajanın aksine, birden çok uzman ajanla oluşturulan sistemlerin performansı çok daha yüksek çıkıyor. Örneğin çoklu ajan mimarileri, tek ajana göre %45–60 daha hızlı çözüm üretiyor. Bu da özellikle büyük şirketlerin birbirine entegre, modüler ajanlar geliştirmeye odaklandığını gösteriyor. Türkiye'de kurumlar da benzer modelleri denemeye başladı (ör. bankacılık müşterilerinin taleplerini işleyen ajan ağları).
- **Kurumsal Benimseme:** Şu anda şirketlerin yaklaşık %88'i en az bir AI uygulaması kullanırken, sadece %23'ü tam otomatik ajansistemleri üretime almış durumda. Bu “pilot-tren” (deneme-prototip) evresinden sonra, gelecek 5 yıl içinde *daha fazla Türkiye'deki büyük firmanın döngüsel YZ çözümlerine geçiş yapması* bekleniyor.
- **Günlük Hayatta Ajanlar:** Marton Schneider gibi düşünürler, 2030'da yazılım ekiplerinin yapısını kökten değiştireceğini öngörüyor. Kod inceleme, veri analizi, operasyon gibi görevlerde uzman ajanlar hep arkada çalışacak, mühendisler strateji ve mimariye odaklanacak. Örneğin bir “mühendis arkadaşı olarak çalışan ajan” altyapıları görüp sorunlara öngörü ile çözüm sunacak, sistem çökmeden arızayı önleyecek. Ya da tasarım aşamasında, bir mimari ajan

sürekli sistemin bütünüyle konuşup sınırları koruyacak. Bu vizyon, döngü mühendisliğinin de “mühendis kozu” olarak daha da önemli olacağını gösteriyor.

- **Eğitim ve Yetenek Gelişimi:** Gelecekte “Loop Mühendisi” yeni bir unvan olabilir. Bu alanda uzmanlar LLM’leri, otonom sistem tasarımını ve bulut veri yönetimini bilmelidir. Türkiye’de üniversiteler ve online kurslar hâlihazırda yapay zekâ ajanlarını müfredata eklemeye başladı. 2026 itibarıyla aktif topluluklar (örn. tosea.ai, Kaggle yarışmaları) bu konuyu yoğun işliyor.

Yol Haritası: Bir Loop Mühendisi Nasıl Olunur?

1. **Temel Bilgiler:** Büyük dil modellerini (LLM’ler), prompt mühendisliğini ve yapay zekâ mimarisini öğrenin. (OpenAI/Anthropic dökümantasyonları, kurslar).
2. **Deneme ve Pratik:** ChatGPT/Claude’a basit görevler vererek *feedback loop* konseptini kavrayın. Kendinize sorun, model nasıl iyileştirir, nerede durur?
3. **Araçları Keşfetme:** LangChain, DeepAgents, OpenClaw gibi açık kaynak çerçevelerle küçük ajanlar geliştirin. Geliştirme ortamında otomatik görev tetikleme ve değerlendirme ekleyin.
4. **Otomasyon ve DevOps:** Sürekli entegrasyon (CI/CD), bulut sunucuları, zamanlanmış işler (cron, GitHub Actions) gibi altyapıları öğrenin. Döngülerin kesintisiz çalışabilmesi için bu alt yapıya hakim olun.
5. **Değerlendirme:** Model çıktılarını testlerle, dış denetlemeyle sürekli ölçün. Evals (değerlendirme çerçeveleri) ve otomatik test altyapıları oluşturun. Başarı kriterleriyle oynayıp geri besleme loop’ları kurun.
6. **Çoklu Ajan & Protokoller:** Agent-to-Agent (A2A) ve Model Context Protocol (MCP) gibi standartları inceleyin. Farklı ajansistemlerinin nasıl işbirliği yaptığını öğrenin.
7. **Gerçek Dünya Projeleri:** Kendi küçük projenizi planlayın: bir görevi (örneğin otomatik rapor hazırlama) küçük adımlara bölün, her adımı otomatikleştiren loop’lar yazın. Hataları ve performansı not edin. Türkiye’deki hackathon veya yarışmalara katılın.
8. **Risk ve Güvenlik:** Döngülerin potansiyel risklerini (maliyet, hatalar, ihlaller) nasıl yöneteceğinizi öğrenin. İzin yönetimi, şifreleme, güvenlik taramaları konularına çalışın.

9. **Sürekli Öğrenme:** Bu alan hızla gelişiyor. Konferansları takip edin, bilimsel makaleler okuyun (ör. ArXiv'daki Loop Engineering çalışmaları), topluluk forumlarında aktif olun.
10. **Ekip Çalışması:** Son olarak unutmayın; döngü mühendisliği bir ekip oyunudur. Diğer geliştiriciler, veri uzmanları ve güvenlik mühendisleriyle iş birliği yapmayı öğrenin. Birlikte ölçeklenebilir, sürdürülebilir otomasyon mimarileri oluşturun.

Sıkça Sorulan Sorular (SSS)

- **Prompt Döngüsü ve Döngü Mühendisliği nedir?**

Prompt döngüsü, ajana adım adım sorular soran ve her cevabı değerlendirip gerektiğinde yeniden sorgu üreten sürekli bir süreçtir. Döngü mühendisliği ise bu süreci otomatikleştiren sistemi tasarlamaktır.
- **Prompt mühendisliğiyle arasındaki fark nedir?**

Prompt mühendisliği el yordamıyla soruları iyileştirme ustalığıdır. Döngü mühendisliği ise prompt'ları otomatik üreten, sonuçları sürekli kontrol eden bir yazılım bütünü inşa etmektir.
- **Hangi durumlarda döngü kullanılmalı?**

Döngüler, çok adımlı, otomatik tekrar gerektiren veya sürekli optimizasyon isteyen işlerde kullanışlıdır. Örneğin otomatik test döngüleri, büyük kod refaktör işlemleri, ara rapor üretimi gibi. Tek seferlik içerik üretimlerinde sınırlı etki yaratabilir.
- **Döngü oluşturma'nın temel riski ne?**

En büyük risk kontrolsüz işlem maliyetidir. Bir döngü önlem alınmazsa çok sayıda API çağrısına dönüşebilir. Ayrıca, model hatalarını pekiştirebilir (yanlış bilgiyi düzeltmez). Bu nedenle 'durma koşulu' ve doğrulayıcı mekanizmalar tasarlamak zorunludur.
- **Loop Mühendisi olmak için hangi yetenekler gerekir?**

YZ modellerine hakimiyet, sistem tasarımı, yazılım mimarisi bilgisi önemlidir. Ek olarak performans optimizasyonu, güvenlik tedbirleri, veritabanı ve bulut teknolojilerinde tecrübe gereklidir. Sürekli öğrenmeye açık, analitik düşünebilen mühendisler başarı şansını artırır.

- **Loop (Döngü) Mühendisliği her projeye uygun mu?**

Hayır. Döngü, belirlenebilir ölçüt ve otomasyon potansiyeli olmayan, belirsiz/yaratıcı işler için uygun değildir. İnsan odaklı strateji belirleme veya inovasyon gerektiren işlerde döngü sınırlı kalır.

Kaynaklar: Makalede belirtilen bilgiler, Endüstri blogları (MindStudio, ShiftAsia), Business Insider ve akademik makalelerden alınmıştır. Gerekli yerlerde alıntılar belirtilmiştir.