

● LEGACY R&D TAKIMI DÖNÜŞÜM PROGRAMI ●

Agentiç AI Developer Müfredatı

LLM okuryazarlığından multi-agent SDLC pipeline'ına — yapısal, eval-driven bir yolculuk.

9

AŞAMA

68

MICRO DERS

28

PRATİK PROJE

26-33

HAFTA

8

EVAL & OBS DERSİ

9 aşama • 26–33 hafta

Her aşama micro derslerden, pratik projelerden ve ölçülebilir bir aşama çıktısından oluşur. Aşağıdaki tablo programın iskeletini, ardından gelen omurga eval & observability disiplininin müfredata nasıl dağıldığını gösterir.

AŞAMA	SÜRE	DERS	PRATİK
1 LLM okuryazarlığı GÜNCELLENDİ	2–3 hafta	7	3
2 Prompt engineering ve structured output GÜNCELLENDİ	2–3 hafta	7	3
3 Tool use, RAG ve entegrasyonlar GÜNCELLENDİ	3 hafta	7	4
4 Agentic workflow pattern'leri YENİ AŞAMA	3 hafta	7	3
5 Agent mimarisi, state ve hafıza GÜNCELLENDİ	3–4 hafta	10	3
6 MCP ve harici entegrasyonlar	2 hafta	4	2
7 Multi-agent sistemler GÜNCELLENDİ	3–4 hafta	9	4
8 Güvenilirlik ve production GÜNCELLENDİ	3–4 hafta	9	3
9 Multi-agent SDLC pipeline YENİ AŞAMA	3–4 hafta	8	3

● eval

● observability

● her ikisi



Aşama 2 **Eval mindset** 2.7

İlk golden dataset, basit metrikler, TDD köprüsü



Aşama 5 **Dev-time tracing** 5.10

Structured logging, trace format, step-through debug



Aşama 7 **Multi-agent obs** 7.7

Distributed tracing, correlation ID, audit trail



Aşama 8 **Offline eval** 8.1

Golden dataset, LLM-as-judge, assertion-based eval



Aşama 8 **Online eval & HITL** 8.2

Production feedback, A/B test, shadow mode



Aşama 8 **Production obs stack** 8.3

Metrics, alerting, anomaly detection, dashboards



Aşama 8 **CI/CD + eval gate** 8.9

Eval-gated deployment, canary release, rollback



Aşama 9 **Pipeline eval & obs** 9.8

SDLC metrikleri, pipeline tracing, bottleneck tespiti

Neden omurga? Eval, Aşama 2'de refleks olarak başlar; her agent projesinde golden dataset ve metrik beklenir. Observability, Aşama 5'te development-time tracing ile başlar, Aşama 7'de distributed tracing'e genişler, Aşama 8'de production-grade olgunlaşır, Aşama 9'da SDLC pipeline özelinde derinleşir. Sonuç: eval ve obs son aşamada öğrenilen bir eklenti değil, müfredatın her katmanında var olan bir disiplin.

◆ MICRO DERSLER

1.1 Generative AI ve LLM: ne, neden, neden şimdi?

YENİ

AI → ML → deep learning → generative AI → LLM kavram haritası. LLM'lerin ne yapabildiği ve yapamadığı, somut örneklerle. ChatGPT anı: neden 2022'de kırılma oldu (scale, instruction tuning, arayüz). Hype vs gerçeklik, yazılım geliştirme pratiğine etkisi. Deterministic yazılımdan probabilistic sistemlere geçişin anlamı.

1.2 Transformer mimarisi ve LLM'in doğası

GÜNCELLENDİ

Attention mekanizması, next-token prediction, training vs inference farkı. Modelin nasıl bu hale geldiği: pretraining → instruction tuning → RLHF (kavramsal düzeyde). Matematiksel detay değil, sezgisel anlama.

1.3 Tokenizasyon, context window ve vektör sezgisi

GÜNCELLENDİ

BPE tokenizer çalışma mantığı, token ≠ kelime, context window limitleri ve doluluk stratejileri. "Token'lar sayıya, sayılar vektöre": embedding sezgisine ilk dokunuş. Tiktoken ile hands-on token sayma, Türkçe vs İngilizce tokenizasyon farkı.

1.4 İlk API çağrısı

GÜNCELLENDİ

Claude/OpenAI API kurulumu, messages formatı, streaming response, error handling. cURL → SDK geçişi. Bu noktadan sonra tüm dersler canlı deneyle desteklenir.

1.5 Model parametreleri ve non-determinism

GÜNCELLENDİ

Temperature, top-p, top-k, max_tokens, stop sequences. Her parametrenin output'a etkisini API üzerinden deney yaparak gözlemlene. Non-determinism'in açıkça isimlendirilmesi: aynı input → farklı output. "LLM probabilistic bir bileşendir" zihinsel modeli.

1.6 Hallucination anatomisi

Neden oluyor (training data, probability distribution), türleri (factual, logical, contextual), tespit ve azaltma teknikleri, grounding konsepti. Önceki derslerdeki deneylerde gözlemlenen davranışların analizi.

1.7 Model landscape ve model seçimi

YENİ

Frontier vs küçük modeller, capability tiers, fiyat/performans dengesi, context window ve hız farkları, açık kaynak vs API modelleri. "Bu görev için hangi model?" karar çerçevesi: görev karmaşıklığı, latency, maliyet, veri gizliliği.

◆ PRATİK UYGULAMALAR

Token maliyet hesaplayıcı

bireysel

CLI aracı: girdi metnini tokenize et, model bazında maliyet hesapla, context window doluluk yüzdesi göster. Türkçe vs İngilizce token verimliliğini karşılaştır.

Model karşılaştırma lab'ı

bireysel

GÜNCELLENDİ

Aynı 10 prompt'u Claude, GPT-4o ve Gemini'ye gönder. Yanıt kalitesi, hız, token maliyeti ve hallucination oranını tablo halinde karşılaştır. Sonuçları 1.7'deki karar çerçevesiyle yorumla.

Hallucination dedektörü

ikili

LLM output'unu ikinci bir LLM ile doğrula. Factual accuracy skortlama sistemi oluştur. Ground truth karşılaştırma pipeline'ı kur. (LLM-as-judge tohumunu atar.)

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: API çağrısı yapıp streaming response alabilir · token maliyetini hesaplayabilir · model parametrelerini amaca göre ayarlayabilir · hallucination'ı tespit edip raporlayabilir · görev için uygun modeli gerekçeli seçebilir. — **Kavramsal kazanım:** LLM'in probabilistic doğasını, limitlerini ve güçlü yanlarını deneyimle kavramış, deterministic yazılımdan probabilistic sistemlere geçiş zihniyetini edinmiş olur.

2

Prompt engineering ve structured output

GÜNCELLENDİ

2-3 hafta · 7 micro ders · 3 pratik proje

◆ MICRO DERSLER

2.1 System prompt mimarisi

Rol tanımı, kısıtlar, output formatı. Prompt'u bir spesifikasyon dokümanı gibi yapılandırma. XML tag kullanımı, section'lara ayırma.

2.2 Few-shot ve chain-of-thought

Örnek vererek öğretme, adım adım düşündürme, "düşün sonra cevapla" pattern'i. Ne zaman zero-shot yeter, ne zaman few-shot gerekir kararı.

2.3 Structured output alma

JSON mode, XML çıktı, markdown formatı. LLM'den parse edilebilir, güvenilir veri yapıları çıkarma teknikleri.

2.4 Schema validation entegrasyonu GÜNCELLENDİ

Zod (TypeScript) ve Pydantic (Python) ile LLM output'unu tip-güvenli hale getirme. Nested/complex schema'lar, parse → validate → retry döngüsü, function calling ile structured output entegrasyonu.

2.5 Prompt chaining ve adım-arası validation YENİ

Tek çağrının yetmediği görevler: çok adımlı iş akışları. Bir adımın çıktısını sonraki adımın girdisi yapma, ara adımlarda schema validation gate'leri, hata durumunda fallback stratejileri. Tool use ve agent loop'larına geçişin kavramsal köprüsü.

2.6 Prompt iteration döngüsü

Sistematiik prompt iyileştirme: baseline ölç, değişken izole et, A/B karşılaştır, regresyon izle. Prompt versiyonlama.

2.7 Eval mindset: ölçmeden iyileştiremezsin YENİ EVAL & OBS

Prompt kalitesini sezgiyle değil metrikle değerlendirme. İlk golden dataset oluşturma (10-20 test case yeter), basit skorlama (exact match, contains, LLM-as-judge). "Her prompt değişikliği bir regresyon riski" refleksini kazandırma. TDD bilen developer için doğal köprü.

◆ PRATİK UYGULAMALAR

Otomatik code reviewer bireysel

Git diff alan, PR bazında code review yapan asistan. System prompt ile review standartlarını tanımla, structured JSON output ile sorunları severity/category/line bazında raporla.

Bug triage asistanı

ikili

Bug report metnini analiz et, Pydantic/Zod schema ile priority, component, estimated_effort, suggested_assignee döndür. Retry on validation failure.

Eval'li README üretici

bireysel

GÜNCELLENDİ

Repo analiz edip README üreten araç. Fark: 15 repo'luk golden dataset oluştur, her prompt değişikliğinde eval çalıştır, skoru düşürmeyen değişiklikleri kabul et. Prompt versiyonlama ile kaliteyi sistematik artır.

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: system prompt yazıp tutarlı structured JSON output alabilir · Zod/Pydantic ile output'u validate edebilir · çok adımlı prompt chain'i validation gate'leriyle kurabilir · 10+ test case'lik golden dataset oluşturup eval suite çalıştırabilir. — **Kavramsal kazanım:** "Her prompt değişikliği bir regresyon riski" refleksini kazanmış, eval-driven prompt geliştirme disiplini edinmiş olur.

3

Tool use, RAG ve entegrasyonlar

GÜNCELLENDİ

3 hafta · 7 micro ders · 4 pratik proje

◆ MICRO DERSLER

3.1 Function calling kavramı

LLM'in tool'u nasıl seçtiği ve argüman ürettiği. Tool call → execution → result → LLM response döngüsü.

3.2 Tool tanımı yazma sanatı

JSON schema ile tool açıklaması, parametre tipleri, required/optional ayrımı. İyi vs kötü tool description'ın LLM seçimine etkisi.

3.3 Çoklu tool orkestrasyonu

Birden fazla tool'un aynı conversation'da sunulması, LLM'in sıralama kararı. Parallel vs sequential tool calling.

3.4 Error handling ve fallback

Tool çağrısı başarısız olduğunda ne olur? Retry stratejileri, graceful degradation, kullanıcıya anlamlı hata mesajı üretme.

3.5 Güvenlik: input/output validation

Prompt injection via tool results, sandbox'lama, tehlikeli tool'ların korunması. LLM'in ürettiği argümanları doğrulama.

3.6 RAG temelleri YENİ

Embedding konsepti, vektör veritabanı (ChromaDB/Pinecone), document chunking stratejileri, semantic search vs keyword search. Retrieval → augmentation → generation pipeline'i.

3.7 Web search ve evidence grounding YENİ

Agent'a web arama yeteneği kazandırma, gürültülü sonuçları filtreleme, yanıtları kanıtlarla destekleme, hallucination'ı grounding ile azaltma.

◆ PRATİK UYGULAMALAR

DB sorgulayan asistan bireysel

Doğal dilde soru sor → LLM SQL üretsın → tool olarak çalıştır → sonucu LLM'e döndür → insan-
okunur yanıt üret. Read-only kısıtı, SQL injection koruması.

Multi-API asistan ikili

Hava durumu + takvim + e-posta API'lerini tool olarak tanımla. Zincirleme kullanım senaryosu
ile tek soruda 3 API birden kullan.

RAG dokümantasyon asistanı

bireysel

YENİ

Şirket dokümanlarını chunk'la, embed et, ChromaDB'ye yükle. Doğal dilde soru sor, ilgili chunk'ları retrieve et, LLM ile yanıt üret. Relevance skorlama ekle.

Güvenlik labı

takım

Kendi tool-use asistanına prompt injection saldırıları dene. Tool result üzerinden injection. Savunma katmanlarını kodla ve test et.

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: custom tool tanımlayıp function calling ile çalıştırabilir · çoklu tool'u orkestre edebilir · basic RAG pipeline (chunk → embed → retrieve → generate) kurabilir · tool input/output'unu güvenlik açısından validate edebilir. — **Kavramsal kazanım:** Agent'ın dış dünyayla güvenli etkileşim altyapısını kurmuş, LLM'i izole bir metin üreticiden aksiyon alan bir sisteme dönüştürmüş olur.

4

Agentic workflow pattern'leri

YENİ AŞAMA

3 hafta · 7 micro ders · 3 pratik proje

◆ MICRO DERSLER

4.1 Agentic use-case seçimi ve problem framing

YENİ

Hangi problemler agentic çözüm ister (belirsizlik, çok adımlılık, karar verme), hangileri klasik otomasyon veya basit LLM çağrısıyla çözülmeli? Karar ağacı: rule-based → single LLM call → prompt chain → single agent → multi-agent. Human-in-the-loop boundary tasarımı. ROI çerçevesi: development maliyeti, token maliyeti, hata maliyeti vs kazanılan zaman/kalite. Problem decomposition ve gerçek dünya case study'leri.

4.2 Agent anatomisi, workflow modelleme ve görselleştirme

GÜNCELLENDİ

Bir agent'in yapısal bileşenleri: persona (system prompt), knowledge (context/RAG), tools (function calling), interaction model. Agent tiplerinin taksonomisi. Workflow'u kodlamadan önce diagram olarak tasarlama, building block'larla görsel modelleme. Legacy developer'ların UML alışkanlığına köprü.

4.3 Agentic chaining pattern

GÜNCELLENDİ

2.5'teki saf prompt chaining'in üstüne inşa: zincirin her adımında tool call olabilir, bir adım dinamik olarak routing'e veya parallelization'a kayabilir. Conditional branching, chain'i bir agent state machine'in parçası olarak tasarlama.

4.4 Routing pattern

YENİ

Gelen isteği classify et → doğru uzman agent'a yönlendir. Router agent tasarımı, classification stratejileri, fallback routing, dynamic dispatch.

4.5 Parallelization pattern

YENİ

Birden fazla agent'ı concurrent çalıştırma. Task decomposition (sharding, aspect-based), threading/async, result aggregation ve synthesis.

4.6 Evaluator-optimizer pattern

YENİ

Creator + Critic döngüsü: üret → değerlendir → geri bildirim → tekrar üret. Clear evaluation criteria, actionable feedback, çıkış koşulu tasarımı.

4.7 Orchestrator-workers pattern

GÜNCELLENDİ

Merkezi orchestrator dinamik plan yapar, worker'lara dağıtır, sonuçları birleştirir. Plan kalitesinin execution'a etkisi, re-planning mekanizması. Burada worker'lar tek process içinde prompt chain adımları — 7.1'de aynı pattern gerçek multi-agent bağlama taşınır.

◆ PRATİK UYGULAMALAR

Akıllı destek router'ı

bireysel

YENİ

Müşteri mesajını classify et (teknik/fatura/şikayet/genel), doğru uzman agent'a yönlendir. Her uzman farklı system prompt ve tool set'e sahip. Fallback ve escalation.

Paralel doküman analizci

ikili

YENİ

Uzun dokümanı 3 uzman agent paralel analiz etsin (teknik kalite, iş etkisi, risk). Threading ile concurrent çalıştır, synthesizer agent birleştirsın.

İteratif kod üretici

bireysel

YENİ

Creator agent kod yazar → Critic agent unit test çalıştırır ve feedback verir → Creator düzeltir → tüm testler geçene kadar döngü. Max iteration limiti.

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: bir iş probleminin agent gerektirip gerektirmediğini analiz edip gerekçelendirebilir · ROI tahmini yapabilir · 5 workflow pattern'den probleme uygun olanı seçip implement edebilir · workflow'u önce diagram olarak tasarlayabilir. — **Kavramsal kazanım:** "Her problem agent gerektirmez" pragmatizmini ve "önce tasarla, sonra kodla" disiplinini edinmiş olur.

◆ MICRO DERSLER

5.1 ReAct pattern

Reason → Act → Observe döngüsü. LLM düşünür, aksiyon seçer, sonucu gözlemler, tekrar düşünür. While-loop implementasyonu, çıkış koşulları.

5.2 Plan-and-execute pattern

Önce tam plan oluştur, sonra adım adım çalıştır. Re-planning ne zaman tetiklenir? Plan kalitesinin execution kalitesine etkisi.

5.3 Reflection ve self-correction

Agent kendi çıktısını değerlendirir, hataları saptar, düzeltir. Inner critic pattern'i. Sonsuz döngü riski ve korunma.

5.4 Agentic RAG: retrieval'dan reasoning'e YENİ

3.6'daki basic RAG pipeline'ına reflection ve self-correction ekleme. Retrieval kalitesi değerlendirmesi, query reformulation, retry loop, yeterlilik kararı. ReAct + evaluator-optimizer pattern'lerinin RAG'e uygulanması. 7.4'teki multi-agent RAG'e köprü.

5.5 State management ve state machine'ler YENİ

Agent state'ini explicit olarak yönetme. State machine tanımlama, transition kuralları, multi-turn state tracking. Schema-driven state management.

5.6 Kısa süreli hafıza

Conversation buffer, sliding window, summarization. Context window yönetimi, ne zaman özetle ne zaman at kararı.

5.7 Uzun süreli hafıza ve vektör DB YENİ

Semantic memory (ne biliyorum), episodic memory (ne yaşadım), procedural memory (nasıl yapıyorum). Vektör DB ile kalıcı hafıza, session persistence.

5.8 Framework tour: LangGraph ve CrewAI

Graph-based agent orkestrasyonu (LangGraph), rol-tabanlı multi-agent (CrewAI). Her framework'ün trade-off'ları, soyutlama maliyeti.

5.9 Sıfırdan agent loop yazma

Framework'süz, sadece API çağırısı + while loop + tool dispatch ile agent yaz. Framework'ün perde arkasında ne yaptığını bizzat kodlayarak anlama.

5.10

Agent debugging ve development-time tracing

YENİ

EVAL & OBS

Agent loop'unda her adımı structured log'a yaz: LLM input/output, tool seçimi, argümanlar, sonuç, karar gerekçesi. Trace formatı tasarımı. "Agent neden bu yolu seçti" sorusuna trace ile cevap verme. Step-through replay ile debugging.

◆ PRATİK UYGULAMALAR

Research agent

bireysel

Konu → web ara → kaynak oku → not çıkar → eksik bilgi tespit → tekrar ara → özet rapor üret.
ReAct loop + reflection ile kalite kontrolü. Max 10 iterasyon.

Sıfırdan vs framework yarışı

ikili

Aynı görevi hem sıfırdan agent loop ile hem LangGraph ile çöz. Kod karmaşıklığı, debug edilebilirlik, esneklik, geliştirme süresi karşılaştırması.

Traced stateful coding agent

takım

GÜNCELLENDİ

Doğal dil → plan → kod → test → düzelt. State machine ile durumları yönet. Long-term memory ile önceki session'lardan öğren. Her adımı trace'e yaz, trace viewer ile debug et. 20 test case'lik eval suite ile agent kalitesini ölç.

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: ReAct agent loop yazabilir (framework ile ve sıfırdan) · agentic RAG pipeline kurabilir · agent state'ini state machine ile yönetebilir · kısa ve uzun süreli hafıza implement edebilir · agent'ın her adımını structured trace log'a yazıp debug edebilir. — **Kavramsal kazanım:** Framework sihrine güvenmek yerine altta dönen mekanizmayı anlayan, gözlemlenemeyen agent'a güvenmeyen mühendislik refleksini edinmiş olur.

6

MCP ve harici entegrasyonlar

2 hafta · 4 micro ders · 2 pratik proje

◆ MICRO DERSLER

6.1 MCP protokolü temelleri

Model Context Protocol nedir, neden var? REST API'den farkı, server-client modeli, transport katmanları (stdio, SSE). Tool/resource/prompt kavramları.

6.2 MCP server geliştirme

TypeScript SDK ile MCP server yazma: tool tanımlama, resource expose etme, input validation. @modelcontextprotocol/sdk kullanımı.

6.3 MCP client entegrasyonu

Claude Desktop, Claude Code ve custom client'larda MCP server bağlantısı. Config dosyaları, auth flow, server discovery.

6.4 Ekosistem ve best practices

Mevcut MCP server registry, popüler connector'lar (GitHub, Slack, DB), güvenlik best practice'leri, rate limiting.

◆ PRATİK UYGULAMALAR

Custom MCP server

bireysel

Şirketin internal API'si için MCP server yaz. Ör: JIRA ticket'ları listele/oluştur, internal wiki'de arama yap. Claude Desktop'ta test et.

MCP gateway

ikili

Birden fazla MCP server'ı tek gateway arkasında birleştir. Unified auth, request logging, rate limiting. Kullanım dashboard'u.

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: MCP server geliştirip deploy edebilir · mevcut agent'ı MCP client olarak şirket iç araçlarına bağlayabilir · MCP ekosistemindeki mevcut server'ları keşfedip entegre edebilir. — **Kavramsal**

kazanım: Tool entegrasyonunu vendor-specific SDK'lardan standart bir protokole taşımış, agent-tool iletişimini ölçeklenebilir hale getirmiş olur.

◆ MICRO DERSLER

7.1 Multi-agent sistem tasarımı ve supervisor-worker pattern

GÜNCELLENDİ

Tek agent workflow'dan multi-agent'a geçiş: ne zaman, neden? Problem → rol ayrıştırma, agent interface contract tasarımı, iletişim topolojisi seçimi (hub-spoke, peer-to-peer, hierarchical), bağımlılık haritası. Supervisor-worker pattern: 4.7'deki orchestrator-workers'ın multi-agent bağlama taşınması — artık worker'lar ayrı LLM çağrıları, iletişim overhead'i gerçek, failure yayılabilir.

7.2 Agent arası iletişim

Message passing, shared blackboard, event-driven communication. Hangi pattern ne zaman uygun? İletişim overhead'inin performansa etkisi.

7.3 Shared state, izolasyon ve conflict resolution

GÜNCELLENDİ

State paylaşımı vs izolasyon. Git worktree pattern'i. Concurrent access, race condition, conflict detection ve resolution mekanizmaları.

7.4 Multi-agent RAG

YENİ

Her agent farklı veri kaynağında uzman (dokümanlar, API'ler, DB). Specialized retrieval agent'lar + synthesis agent. Query reformulation ve retry loop'lar.

7.5 Failure mode'lar ve dayanıklılık

Error cascading, token cost explosion, sonsuz delegasyon döngüsü, agent tartışması. Circuit breaker, timeout, cost ceiling pattern'leri.

7.6 Tek agent mi, çok agent mı?

Multi-agent overhead'i analizi. Karmaşıklık vs fayda eğrisi. Karar matrisi: ne zaman tek agent + iyi tool'lar yeter?

7.7 Multi-agent observability

YENİ

EVAL & OBS

Distributed tracing: her agent'ın trace'ini birleştirip uçtan uca görme. Agent-to-agent mesaj logları, shared state değişiklik geçmişi (audit trail). Correlation ID ile request takibi.

7.8 Multi-agent SDLC'ye giriş

YENİ

Requirements → Design → Code → Test → Deploy sürecinde agent rolleri. Hangi SDLC aşamasında agent'lar değer katar, hangileri riskli? Human-in-the-loop noktaları. Mevcut araçlarla (Cursor, Claude Code, Copilot) karşılaştırma.

7.9

SDLC agent prototipleme

YENİ

Basit bir requirements-to-test pipeline'i: doğal dil spec → kod üret → test üret → çalıştır. Tek agent vs multi-agent yaklaşım karşılaştırması. Prototip üzerinden trade-off analizi.

◆ PRATİK UYGULAMALAR

Code review pipeline

takım

3 agent: Planner (PR analiz, strateji), Reviewer (dosya incele, sorun raporla), Summarizer (birleştir, nihai review). Supervisor orkestrasyonu.

Multi-source RAG sistemi

ikili

YENİ

3 specialized retrieval agent (doküman DB, web search, SQL) + 1 synthesis agent. Paralel retrieve → birleştir → tutarlı yanıt üret.

Traced agent debate sistemi

ikili

GÜNCELLENDİ

Advocate + critic + judge ile teknik karar. Her agent'ın reasoning trace'i kaydedilir. Debate sonrası trace viewer ile "hangi argüman judge'ı ikna etti" analizi. End-to-end distributed trace.

Mini SDLC pipeline

ikili

YENİ

2 agent'lık basit pipeline: Spec agent (doğal dil → structured requirements) + Coder agent (requirements → implementasyon + test). Handoff protokolü tasarla, kaliteyi ölç.

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: multi-agent sistem tasarlayabilir (rol ayrıştırma, interface contract, iletişim topolojisi) · 3+ agent'lı çalışan sistem kurabilir · distributed trace ile multi-agent debug yapabilir · tek agent vs multi-agent kararını gerekçelendirebilir · basit SDLC agent prototipi kurabilir. — **Kavramsal kazanım:** Multi-agent sistemlerin hem gücünü hem maliyetini (complexity, failure modes, overhead) bilerek bilinçli mimari kararlar verebilir hale gelmiş olur.

◆ MICRO DERSLER

- 8.1 **Offline eval: golden dataset ve metrikler** GÜNCELLENDİ EVAL & OBS
Deterministic eval (exact match, regex, JSON schema validation), LLM-as-judge (rubric tasarımı, judge prompt kalibrasyonu), assertion-based eval. Golden dataset curation: edge case seçimi, class balance, veri zehirlenmesi riski. Eval suite'i CI'da çalıştırma.
- 8.2 **Online eval ve human-in-the-loop** YENİ EVAL & OBS
Production'da eval: kullanıcı feedback toplama, implicit signals (retry, abandon, thumbs down), sampling-based human review. Annotation workflow, inter-annotator agreement. A/B testing, shadow mode.
- 8.3 **Production observability stack** GÜNCELLENDİ EVAL & OBS
Development tracing'den (5.10) production'a geçiş: structured logging, metrics (latency p50/p95/p99, token usage, error rate), distributed tracing (OpenTelemetry), dashboard tasarımı. Alerting ve alert fatigue, anomaly detection.
- 8.4 **Guardrails ve safety katmanları** GÜNCELLENDİ
3.5'teki prompt injection farkındalığını production-grade katmanlı sisteme taşıma. Input guardrails, output guardrails (PII, toxicity, factual grounding), tool guardrails (tehlikeli operasyon onayı, rate limiting). Defence in depth.
- 8.5 **AI governance ve enterprise risk** YENİ
Model kullanım politikaları, veri gizliliği ve compliance (KVKK/GDPR, PII, data residency). Audit trail ve accountability. Vendor risk, data retention, vendor lock-in. İç politika tasarımı: acceptable use policy, model onay süreci. Teknik guardrails + kurumsal governance = defence in depth.
- 8.6 **Agent serving ve deployment mimarisi** YENİ
Agent'ı service olarak sunma: REST API, WebSocket (streaming), async queue. Serverless vs long-running trade-off'lar (cold start, timeout, stateful agent'lar). Horizontal scaling (session affinity, shared memory), rate limiting. Klasik web service deployment'tan farklar.
- 8.7 **Cost optimization**
Token bütçesi yönetimi, prompt caching (semantic cache), model routing (kolay görev → küçük model), batch processing. Cost-per-task analizi, real-time token bütçe dashboard'u.
- 8.8 **Error handling ve degradation**
API timeout, rate limit, model downtime senaryoları. Fallback model, cached response, human escalation. SLA tanımlama ve izleme.

Prompt deployment pipeline: eval gate (her PR eval'den geçmeli), canary release, rollback mekanizması. Eval regression → otomatik blok. Prompt/config versiyonlama, infrastructure as code.

◆ PRATİK UYGULAMALAR

End-to-end eval pipeline

bireysel

GÜNCELLENDİ

Önceki agent'lardan birini seç. 50+ test case golden dataset yaz. Offline eval (deterministic + LLM-as-judge) + online eval (shadow mode simülasyonu). Eval CI'da çalışsın, prompt değişikliğinde otomatik regresyon raporu.

Full-stack observability

ikili

GÜNCELLENDİ

Multi-agent sisteme production-grade observability ekle. Structured logging, OpenTelemetry tracing, Grafana dashboard (latency, token, error rate, cost). Alert kuralları, anomaly detection ile davranış kayması tespiti.

Production-ready agent (capstone)

takım

Tüm aşamaların birleşimi: gerçek iş problemi çözen agent. Eval pipeline (offline + online), full observability, guardrails, cost ceiling, fallback, CI/CD with eval gate. Demo + post-mortem analiz.

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: eval pipeline (offline + online) kurup CI'a entegre edebilir · production observability stack deploy edebilir · guardrails katmanı implement edebilir · governance checklist'i hazırlayabilir · agent'ı service olarak deploy edebilir · cost ceiling ve fallback kurabilir. — **Kavramsal kazanım:** SW engineering disiplini (CI/CD, monitoring, SLA) ile agentic sistemleri tam olarak buluşturmış, production-grade agent operasyonu yapabilir hale gelmiş olur.

9

Multi-agent SDLC pipeline

YENİ AŞAMA

3-4 hafta · 8 micro ders · 3 pratik proje

◆ MICRO DERSLER

9.1 SDLC pipeline mimarisi YENİ

Requirements agent → Design agent → Coder agent → Reviewer agent → Tester agent → Deployer agent. End-to-end orchestration stratejisi, agent sorumluluk sınırları, handoff protokolü tasarımı.

9.2 Requirements agent YENİ

Doğal dil → structured spec. Belirsizlik tespiti ve soru sorma, acceptance criteria üretimi, edge case çıkarımı. Spec kalitesinin downstream etkisi.

9.3 Design agent YENİ

Spec → teknik tasarım dokümanı. API contract üretimi, data model tasarımı, component breakdown. Mevcut codebase context'ini kullanarak uyumlu tasarım.

9.4 Coder agent YENİ

Tasarım → implementasyon. Context-aware kod üretimi, mevcut codebase ile uyum, incremental generation. Büyük dosyalarda edit vs rewrite stratejisi. Code style consistency.

9.5 Reviewer + Tester agent pipeline YENİ

Otomatik code review + test generation + test execution. Review agent'ın review standartlarını structured uygulaması. Tester agent'ın unit/integration test üretmesi. Feedback loop ile düzeltme döngüsü.

9.6 CI/CD entegrasyonu ve deployer agent YENİ

Agent pipeline'ını CI/CD'ye bağlama. Automated PR oluşturma, merge gate konfigürasyonu, deployment validation. Git workflow entegrasyonu, branch stratejisi.

9.7 End-to-end pipeline orkestrasyonu YENİ

Tüm agent'ların koordinasyonu: handoff protokolü, state management, failure recovery, re-planning. Pipeline konfigürasyonu ve customization. Human-in-the-loop gate'leri.

9.8 Pipeline observability ve eval YENİ EVAL & OBS

SDLC pipeline'a özel metrikler: code quality skoru, test coverage, review accuracy, spec-to-code fidelity. Pipeline-level tracing ile bottleneck tespiti. Eval suite: pipeline output kalitesini sistematik ölçme.

◆ PRATİK UYGULAMALAR

Feature factory

takım

YENİ

Doğal dil feature request → PR-ready kod. End-to-end pipeline ile 5 farklı feature implement et. Her feature için spec → design → code → review → test → PR akışı. Üretilen kodu manual review ile karşılaştır.

Pipeline benchmark

ikili

YENİ

Aynı 10 feature'ı manual geliştirme vs SDLC pipeline ile geliştir. Kalite (bug sayısı, test coverage), hız, maliyet (token usage) karşılaştırması. Hangi feature türlerinde pipeline başarılı?

SDLC pipeline capstone

takım

YENİ

Gerçek repo üzerinde production-grade SDLC pipeline. Eval suite (code quality, test coverage, spec fidelity), full observability, guardrails (dangerous op blocking, cost ceiling), human-in-the-loop gates. Demo + ROI analiz.

◆ AŞAMA ÇIKTISI

Minimum Viable Skill: requirements → design → code → review → test → deploy SDLC pipeline'ını multi-agent olarak tasarlayıp uçtan uca çalıştırabilir · pipeline çıktı kalitesini eval ile ölçebilir · manual vs agent-assisted karşılaştırması yapabilir. — **Kavramsal kazanım:** Tüm müfredat boyunca edinilen becerileri tek bir gerçek dünya pipeline'ında sentezlemiş, agent-assisted development'ın güçlü ve zayıf yönlerini deneyimle anlamış olur.