Claude Certified Architect — Foundations Sertifikası

Çalışma Kılavuzu (Resmî Sınav Kılavuzu'na Dayalı)

Giriş

Claude Certified Architect — Foundations sertifikası, bir uzmanın gerçek dünyadaki Claude tabanlı çözümleri uygularken isabetli ödünleşim kararları verebildiğini doğrular. Sınav; Claude ile üretim uygulamaları geliştirmeye yönelik temel teknolojiler olan Claude Code, Claude Agent SDK, Claude API ve Model Context Protocol (MCP) hakkında temel bilgileri değerlendirir.

Sınav soruları gerçekçi sektör senaryolarına dayanır: müşteri desteği için agentic sistemler (agentic systems) oluşturma, çoklu-ajan (multi-agent) araştırma işlem hatları (pipelines) tasarlama, Claude Code'u CI/CD'ye entegre etme, geliştirici üretkenliği araçları oluşturma ve yapılandırılmamış belgelerden yapılandırılmış veri çıkarma.

Hedef Aday

İdeal aday, Claude ile üretim uygulamaları tasarlayan ve yayına alan bir **çözüm mimarıdır**. Aşağıdaki konularda en az 6 aylık uygulamalı deneyiminiz olmalıdır:

- **Claude Agent SDK** — çoklu-ajan orkestrasyonu (orchestration), alt-ajanlara (subagent) görev devretme, araç entegrasyonu, yaşam döngüsü hook'ları
- **Claude Code** — CLAUDE.md, MCP sunucuları, Skills (Agent Skills), planlama modu (planning mode)
- **Model Context Protocol (MCP)** — backend entegrasyonu için araçlar ve kaynaklar
- **Prompt mühendisliği (prompt engineering)** — JSON şemaları, few-shot örnekler, veri çıkarma şablonları
- **Bağlam pencereleri (context windows)** — uzun belgelerle çalışma, çoklu-ajan bağlam aktarımı
- **CI/CD işlem hatları** — otomatik kod inceleme, test üretimi
- **Eskalasyon (insana yönlendirme) ve güvenilirlik (reliability)** — hata yönetimi (error handling), human-in-the-loop (döngüde insan)

Sınav Formatı

Parametre	Değer
Soru tipi	Çoktan seçmeli (multiple choice) (4 seçenekten 1 doğru)
Puanlama	100–1000 ölçeği, geçme puanı (passing score) **720**

Yanlış cevap cezası (guessing penalty)	Yok (her soruyu yanıtlayın!)
Senaryolar	Olası 8 senaryodan 4'ü (rastgele seçilir)

Sınav İçeriği: 5 Alan

Alan	Ağırlık
1. Ajan mimarisi ve orkestrasyon	**27%**
2. Araç tasarımı ve MCP entegrasyonu	**18%**
3. Claude Code yapılandırması ve iş akışları (workflows)	**20%**
4. Prompt mühendisliği ve yapılandırılmış çıktı (structured output)	**20%**
5. Bağlam yönetimi (context management) ve güvenilirlik	**15%**

Sınav Senaryoları

Senaryo 1: Müşteri Destek Ajanı

Claude Agent SDK'yı kullanarak iadeleri, fatura itirazlarını ve hesap sorunlarını ele alan bir ajan geliştirirsiniz. Ajan MCP araçlarını (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`) kullanır. Hedef, uygun eskalasyonla ilk temasta 80%+ çözüm sağlamaktır.

Senaryo 2: Claude Code ile Kod Üretimi

Geliştirmeyi hızlandırmak için Claude Code'u kullanırsınız: kod üretimi, refactoring, hata ayıklama, dokümantasyon. Bunu özel slash komutları (custom slash commands) ve CLAUDE.md yapılandırmasıyla entegre etmeniz ve planlama modunun ne zaman kullanılacağını anlamanız gerekir.

Senaryo 3: Çoklu-Ajan Araştırma Sistemi

Bir koordinatör (coordinator) ajan, görevleri özelleşmiş alt-ajanlara devreder: web araştırması, belge analizi, sentez (synthesis) ve rapor üretimi. Sistem, atıflı eksiksiz raporlar üretmelidir.

Senaryo 4: Geliştirici Üretkenliği Araçları

Ajan, mühendislerin aşina olmadıkları kod tabanlarını keşfetmesine, boilerplate kod üretmesine ve rutin görevleri otomatikleştirmesine yardımcı olur. Yerleşik araçlar (built-in tools) (Read, Write, Bash, Grep, Glob) ve MCP sunucuları kullanılır.

Senaryo 5: Sürekli Entegrasyon İçin Claude Code

Otomatik kod incelemeleri, test üretimi ve pull request geri bildirimi için Claude Code'u bir CI/CD işlem hattına entegre edin. Prompt'lar, hatalı pozitifleri en aza indirecek şekilde tasarlanmalıdır.

Senaryo 6: Yapılandırılmış Veri Çıkarma

Sistem, yapılandırılmamış belgelerden bilgi çıkarır, çıktı üzerinde JSON şemalarıyla doğrulama (validation) uygular ve yüksek doğruluk düzeyini korur. Uç durumları doğru şekilde ele almalıdır.

Senaryo 7: Konuşmaya Dayalı AI Mimari Kalıpları

Bağlam penceresi yönetimi, turlar arasında talimat kalıcılığı, bellek stratejileri, güvenli yürütme için araç tasarımı ve belirsiz ya da çelişkili kullanıcı girdilerini ele alma konularını kapsayan çok turlu konuşma sistemleri tasarlarsınız.

Senaryo 8: Agentic AI Araçları *(içerik eksik — tamamlamamıza yardım edin!)*

Bu senaryo sınav adayları tarafından bildirilmiştir, ancak henüz bu kılavuzda kapsanmamaktadır. Gerçek sınavda bu senaryodan sorularla karşılaştıysanız, tam kapsam ekleyebilmemiz için lütfen bunları GitHub Issues üzerinden paylaşın. Katkınız, sınava hazırlanan herkese yardımcı olacaktır.

Resmî Dokümantasyon

Kaynak	URL
Claude API — Messages	https://platform.claude.com/docs/en/api/messages
Claude API — Araç Kullanımı	https://platform.claude.com/docs/en/build-with-claude/tool-use
Claude API — Message Batches	https://platform.claude.com/docs/en/build-with-claude/message-batches
Claude Agent SDK — Genel Bakış	https://platform.claude.com/docs/en/agent-sdk/overview
Claude Agent SDK — Hook'lar	https://platform.claude.com/docs/en/agent-sdk/hooks
Claude Agent SDK — Alt-ajanlar	https://platform.claude.com/docs/en/agent-sdk/subagents
Claude Agent SDK — Oturumlar	https://platform.claude.com/docs/en/agent-sdk/sessions
Model Context Protocol (MCP)	https://modelcontextprotocol.io/
MCP — Araçlar	https://modelcontextprotocol.io/docs/concepts/tools
MCP — Kaynaklar	https://modelcontextprotocol.io/docs/concepts/resources
MCP — Sunucular	https://modelcontextprotocol.io/docs/concepts/servers
Claude Code — Dokümantasyon	https://code.claude.com/docs/en/overview
Claude Code — CLAUDE.md ve Bellek	https://code.claude.com/docs/en/memory
Claude Code — Skills (slash komutları dahil)	https://code.claude.com/docs/en/skills

Claude Code — Hook'lar	https://code.claude.com/docs/en/hooks
Claude Code — Alt-ajanlar	https://code.claude.com/docs/en/sub-agents
Claude Code — MCP Entegrasyonu	https://code.claude.com/docs/en/mcp
Claude Code — GitHub Actions CI/CD	https://code.claude.com/docs/en/github-actions
Claude Code — GitLab CI/CD	https://code.claude.com/docs/en/gitlab-ci-cd
Claude Code — Headless (etkileşimsiz mod)	https://code.claude.com/docs/en/headless
Prompt Mühendisliği Kılavuzu	https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview
Genişletilmiş Düşünme	https://platform.claude.com/docs/en/build-with-claude/extended-thinking
Anthropic Cookbook (kod örnekleri)	https://github.com/anthropics/anthropic-cookbook

---

KISIM I: TEORİK TEMELLER

Bu kısım, sınavı başarıyla geçmek için ihtiyaç duyduğunuz tüm teoriyi kapsar. Materyal, sınav alanlarına göre değil teknolojilere ve kavramlara göre düzenlenmiştir; bu da her konuyu daha derinlemesine anlamanıza yardımcı olur.

Bölüm 1: Claude API — Model Etkileşiminin Temelleri

> Dokümantasyon: Messages API | Prompt Mühendisliği

1.1 API İstek Yapısı

Claude API, istek-yanıt modelini izler. Claude Messages API'ye yapılan her istek şunları içerir:

```
{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "system": "You are a helpful assistant.",
  "messages": [
    {"role": "user", "content": "Hi!"},
    {"role": "assistant", "content": "Hello!"},
    {"role": "user", "content": "How are you?"}
  ],
  "tools": [...],
  "tool_choice": {"type": "auto"}
}
```

Temel alanlar:

- `model` — model seçimi (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — yanıttaki maksimum token sayısı
- `system` — sistem komutu (system prompt) (model davranışını tanımlar)
- `messages` — konuşma geçmişi (tutarlılığı korumak için **tam geçmişi göndermelisiniz**)
- `tools` — kullanılabilir araçların tanımları
- `tool_choice` — araç seçimi stratejisi

1.2 Mesaj Rolleri

`messages` dizisi iki konuşma rolü ile bir talimat rolü kullanır:

- `user` — kullanıcı mesajları; araç sonuçları da dahildir (ayrı bir `tool` rolü olarak değil, `user` rolündeki bir mesaj içinde `tool_result` içerik bloğu olarak gönderilir)
- `assistant` — model yanıtları (geçmiş gönderilirken dahil edilir); araç kullanım istekleri de dahildir (`tool_use` içerik blokları)
- `system` — üst düzey `system` alanı üzerinden (ilk turdan itibaren geçerlidir) ya da `messages` içinde satır arasında `{"role": "system", ...}` olarak (bu noktadan itibaren geçerlidir, belirli yerleştirme kurallarına tabidir — aşağıya bakın) ayarlanabilir

Araç sonuçları `"tool"` rolünde bir mesaj olarak gönderilmez. İçeriğinde bir `tool_result` içerik bloğu bulunan, `user` rolünde bir mesaj olarak gönderilir:

```
{
  "role": "user",
  "content": [
    {
      "type": "tool_result",
      "tool_use_id": "toolu_01...",
      "content": "..."
    }
  ]
}
```

`system`, yalnızca üst düzey `system` parametresiyle değil, `messages` dizisinde doğrudan bir rol olarak da yer alabilir. Bu, üst düzey `system` alanından gelen önbelleğe alınmış (cached) ön eki geçersiz kılmadan, konuşmanın ortasına talimat eklemek içindir. Bunun belirli yerleştirme kuralları vardır:

- `tool_result` blokları içeren biri dahil bir `user` turundan hemen sonra, ya da sunucu araç kullanımıyla biten bir `assistant` turundan hemen sonra gelmelidir.
- Bir `assistant` turundan önce gelmeli ya da dizinin sonunda yer almalıdır.
- Bir `tool_use` bloğu ile onun `tool_result`'ı arasında yer alamaz — bunu yapmak 400 hatası döndürür.
- Sonraki `system` mesajları (konuşma ortasında eklenenler dahil), sonraki turlar için önceki mesajlara ve üst düzey `system` alanına göre önceliklidir.

Kritik derecede önemli: Her API isteğinde **tam konuşma geçmişini** göndermelisiniz. Model, istekler arasında durumu kalıcı olarak saklamaz; her çağrı bağımsızdır.

1.3 Yanıttaki `stop_reason` Alanı

Claude API yanıtı, modelin üretimi neden durdurduğunu gösteren `stop_reason` alanını içerir:

Değer	Açıklama	Eylem
`"end_turn"`	Model yanıtını tamamladı	Sonucu kullanıcıya gösterin
`"tool_use"`	Model bir araç çağırmak istiyor	Aracı yürütün ve sonucu döndürün
`"max_tokens"`	Token sınırına ulaşıldı	Yanıt kesilmiştir; sınırı artırmanız gerekebilir
`"stop_sequence"`	Bir durdurma dizisiyle karşılaşıldı	Uygulama mantığınıza göre ele alın

Agentic sistemler için `"tool_use"` ve `"end_turn"` en önemli değerlerdir; ajan döngüsünü kontrol ederler.

1.4 Sistem Komutu

Sistem komutu, bağlamı ve davranış kurallarını tanımlayan özel bir talimattır. Şunları yapar:

- `messages` dizisinin parçası değildir; `system` alanında ayrı olarak geçirilir
- Kullanıcı mesajlarına göre önceliklidir
- Bir kez yüklenir ve konuşma boyunca uygulanır
- Rolü, kısıtları ve çıktı formatını tanımlamak için kullanılır

Sınav için önemli: sistem komutunun ifade edilişi, istenmeyen araç çağrışımları yaratabilir. Örneğin, “müşteriyi her zaman doğrula” gibi bir talimat, gereksiz olduğunda bile modelin `get_customer` aracını aşırı kullanmasına neden olabilir.

1.5 Bağlam Penceresi

Bağlam penceresi, modelin tek seferde işleyebileceği toplam metin miktarıdır (token cinsinden). Şunları içerir:

- Sistem komutu
- Tam mesaj geçmişi
- Araç tanımları
- Araç sonuçları

Başlıca bağlam penceresi sorunları:

1. **Ortada kaybolma etkisi:** modeller, uzun bir girdinin başındaki ve sonundaki bilgileri güvenilir şekilde işler; ancak ortadaki ayrıntıları kaçırabilir. Önlem: kritik bilgileri başlangıca veya sona yakın yerleştirin.
2. **Araç sonuçlarının birikmesi:** her araç çağrısı bağlama çıktı ekler. Bir araç 40+ alan döndürür, ancak yalnızca 5'i önemliyse bağlamın büyük bölümü boşa harcanır.
3. **Aşamalı özetleme:** geçmiş sıkıştırılırken sayısal değerler, yüzdeler ve tarihler çoğu zaman kaybolur ve muğlaklaşır (“yaklaşık”, “aşağı yukarı”, “birkaç”).

Bölüm 2: Araçlar ve `tool_use`

> Dokümantasyon: Araç Kullanımı

2.1 `tool_use` Nedir

`tool_use`, Claude'un harici fonksiyonları çağırmasını sağlayan bir mekanizmadır. Model kodu doğrudan çalıştırmaz; yapılandırılmış bir araç çağrısı isteği üretir, kodunuz bunu yürütür ve sonucu döndürür.

2.2 Araç Tanımı

Her araç bir JSON şeması kullanılarak tanımlanır:

```
{
  "name": "get_customer",
  "description": "Finds a customer by email or ID. Returns the customer profile,
including name, email, order history, and account status. Use this tool BEFORE
lookup_order to verify the customer's identity. Accepts an email (format:
user@domain.com) or a numeric customer_id.",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "Customer email"},
      "customer_id": {"type": "integer", "description": "Numeric customer ID"}
    },
    "required": []
  }
}
```

Bir araç açıklamasının kritik derecede önemli yönleri:

1. **Açıklama, birincil seçim mekanizmasıdır.** Bir LLM, araçları açıklamalarına göre seçer. Minimal açıklamalar (“Müşteri bilgilerini getirir”), araçlar çakıştığında hatalara yol açar.
2. **Açıklamaya şunları dahil edin:**
   - Aracın ne yaptığı ve ne döndürdüğü
   - Girdi formatları ve örnek değerler
   - Uç durumlar ve kısıtlar
   - Bu aracın benzer alternatiflere kıyasla ne zaman kullanılacağı
3. Araçlar arasında aynı veya çakışan açıklamalardan **kaçının**. `analyze_content` ve `analyze_document` neredeyse aynı açıklamalara sahipse model bunları karıştırır.
4. **Yerleşik araçlar ile MCP araçları:** ajanlar, benzer işlevlere sahip MCP araçları yerine yerleşik araçları (Read, Grep) tercih edebilir. Bunu önlemek için MCP araç açıklamalarını güçlendirin; somut avantajları, benzersiz verileri veya yerleşik araçların sağlayamayacağı bağlamı vurgulayın.

2.3 `tool_choice` Parametresi

`tool_choice`, modelin araçları nasıl seçeceğini kontrol eder:

Değer	Davranış	Ne zaman kullanılır
`{"type": "auto"}`	Model, araç çağırıp çağırmayacağına veya metinle yanıt verip vermeyeceğine karar verir	Çoğu durum için varsayılan
`{"type": "any"}`	Modelin bir araç çağırması **zorunludur**	Garantili yapılandırılmış çıktı gerektiğinde
`{"type": "tool", "name": "extract_metadata"}`	Modelin belirli bir aracı çağırması **zorunludur**	Zorunlu ilk adım / yürütme sırası gerektiğinde

Önemli senaryolar:

- `tool_choice: "any"` + birden çok çıkarma aracı → model en uygun olanı seçer, ancak yine de yapılandırılmış çıktı alırsınız
- Zorunlu seçim → belirli bir ilk eylemi garanti etmeniz gerektiğinde (ör. zenginleştirmeden önce `extract_metadata`)

2.4 Yapılandırılmış Çıktı İçin JSON Şemaları

JSON şemalarıyla `tool_use` kullanmak, Claude'dan yapılandırılmış çıktı elde etmenin **en güvenilir** yoludur. Şunları sağlar:

- Sözdizimsel olarak geçerli JSON'u garanti eder (eksik ayraç yok, sonda fazladan virgül yok)
- Gerekli yapıyı uygular (zorunlu alanlar bulunur)
- Anlamsal doğruluğu garanti **etmez** (değerler yine de yanlış olabilir)

Şema tasarımı — temel ilkeler:

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "Details if category = 'other' or 'unclear'"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "Null if the information was not found in the source"
    }
  },
  "required": ["category", "severity"]
}
```

Şema tasarımı kuralları:

1. **Zorunlu ve isteğe bağlı:** alanları yalnızca bilgi her zaman mevcutsa zorunlu olarak işaretleyin. Zorunlu alanlar, veri eksik olduğunda modeli değer uydurmaya iter.

2. **Null atanabilir alanlar:** bulunmayabilecek bilgiler için `"type": ["string", "null"]` kullanın. Model, halüsinasyon üretmek yerine `null` döndürebilir.
3. **`"other"` içeren enum'lar:** önceden tanımladığınız kategorilerin dışındaki verileri kaybetmemek için `"other"` + ayrıntı dizesi ekleyin.
4. **`"unclear"` enum'u:** modelin bir kategoriyi güvenle seçemediği durumlar içindir; dürüst bir `"unclear"`, yanlış bir kategoriden daha iyidir.

2.5 Sözdizimsel ve Anlamsal Hatalar

Hata türü	Örnek	Önlem
Sözdizimi	Geçersiz JSON, yanlış alan türü	JSON şemasıyla `tool_use` (ortadan kaldırır)
Anlamsal	Toplamlar tutmuyor, değer yanlış alanda, halüsinasyon	Doğrulama kontrolleri, geri bildirimle yeniden deneme (retry with feedback), öz düzeltme (self-correction)

Bölüm 3: Claude Agent SDK — Agentic Sistemler Kurma

> Dokümantasyon: Agent SDK | Hook'lar | Alt-ajanlar | Oturumlar

3.1 Ajan Döngüsü (agentic loop) Nedir

Ajan döngüsü, otonom görev yürütme için temel kalıptır. Model yalnızca yanıt vermez; bir eylemler dizisi gerçekleştirir:

```
1. Send a request to Claude with tools
2. Receive a response
3. Check stop_reason:
   - "tool_use" -> execute the tool, append the result to history, go back to step 1
   - "end_turn" -> the task is complete, show the result to the user
4. Repeat until completion
```

Bu, model güdümlü bir yaklaşımdır: Bir sonraki hangi aracın çağrılacağına, bağlama ve önceki araç sonuçlarına göre Claude karar verir. Bu, eylem sırasının sabit olduğu sabit kodlanmış karar ağaçlarından farklıdır.

Anti-pattern'ler (kötü kalıplar) (kaçının):

- Tamamlanmayı saptamak için asistan metnini ayrıştırmak (“Görev tamamlandı”)
- Birincil durma koşulu olarak keyfi bir yineleme sınırı (ör. `max_iterations=5`) kullanmak
- Tamamlanma sinyali olarak asistanın metinsel içerik üretip üretmediğini kontrol etmek

Doğru yaklaşım: tek güvenilir tamamlanma sinyali `stop_reason == "end_turn"` değeridir.

3.2 `AgentDefinition` Yapılandırması

`AgentDefinition`, Claude Agent SDK'da ajan yapılandırma nesnesidir:

```
agent = AgentDefinition(
    name="customer_support",
    description="Handles customer requests for returns and order issues",
    system_prompt="You are a customer support agent...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # For a coordinator:
    # allowed_tools=["Task", "get_customer", ...]
)
```

Temel parametreler:

- `name` / `description` — ajanın kimliği ve açıklaması
- `system_prompt` — yönergeleri içeren sistem komutu
- `allowed_tools` — izin verilen araçların listesi (en az ayrıcalık ilkesi)

3.3 Merkez-Uç (Hub-and-Spoke): Koordinatör (Coordinator) ve Alt-Ajanlar (Subagents)

Çoklu-ajan mimari genellikle merkez-uç topolojisi olarak kurulur:

```
        Coordinator
       /     |      \
  Subagent1  Subagent2  Subagent3
   (search)   (analysis)   (synthesis)
```

Koordinatör şunlardan sorumludur:

- Görevi alt görevlere ayrıştırmak
- Hangi alt-ajanların gerektiğine karar vermek (dinamik seçim)
- Çalışmayı alt-ajanlara devretmek
- Sonuçları toplamak ve doğrulamak
- Hataları ve yeniden denemeleri (retry) yönetmek
- Sonuçları kullanıcıya iletmek

Kritik ilke: alt-ajanların bağlamı yalıtılmıştır.

- Alt-ajanlar koordinatörün konuşma geçmişini otomatik olarak devralmaz
- Gerekli tüm bağlam, alt-ajan prompt'una **açıkça aktarılmalıdır**
- Alt-ajanlar çağrılar arasında bellek paylaşmaz
- Tüm iletişim koordinatör üzerinden akar (gözlemlenebilirlik ve hata denetimi için)

3.4 Alt-Ajanları Başlatmak İçin `Task` Aracı

Alt-ajanlar `Task` aracı üzerinden başlatılır:

```
# The coordinator's allowedTools must include "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

Bağlamın açıkça aktarılması zorunludur:

```
# Bad: the subagent has no context
Task: "Analyze the document"

# Good: full context in the prompt
Task: "Analyze the following document.
Document: [full document text]
Prior search results: [web search results]
Output format requirements: [schema]"
```

Paralel başlatma: bir koordinatör tek bir yanıtta birden fazla `Task` çağırabilir; alt-ajanlar paralel çalışır:

```
# One coordinator response contains:
Task 1: "Search for articles about X"
Task 2: "Analyze document Y"
Task 3: "Search for articles about Z"
# All three run concurrently
```

3.5 Agent SDK'da hook'lar

Hook'lar, ajan yaşam döngüsünün belirli noktalarında araya girmeyi ve dönüştürmeyi sağlar.

PostToolUse, bir araç sonucunu modele verilmeden önce yakalar:

```
# Example: normalize date formats from different MCP tools
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Convert Unix timestamp -> ISO 8601
    # Convert "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

Giden çağrıyı yakalayan hook, politikayı ihlal eden eylemleri engeller:

```python
# Example: block refunds above $500
@hook("PreToolUse")
def enforce_refund_limit(tool_call):
    if tool_call.name == "process_refund" and tool_call.args.amount > 500:
        return redirect_to_escalation(tool_call)
```

Temel fark: hook'lar ve prompt yönergeleri

Nitelik	Hook'lar	Prompt yönergeleri
Güvence	**Deterministik** (100%)	**Olasılıksal** (>90%, 100% değil)
Ne zaman kullanılır	Kritik iş kuralları, finansal işlemler, uyumluluk	Genel tercihler, öneriler, biçimlendirme
Örnek	$500 üzerindeki iadeleri engelle	“İnsana yönlendirmeden önce çözmeye çalış”

Kural: başarısızlığın finansal, hukuki veya güvenlik sonuçları varsa prompt'ları değil, hook'ları kullanın.

Bölüm 4: Model Context Protocol (MCP)

> Dokümantasyon: MCP | Araçlar | Kaynaklar | Sunucular

4.1 MCP Nedir

Model Context Protocol (MCP), harici sistemleri Claude'a bağlamak için kullanılan açık bir protokoldür. MCP üç temel kaynak türü tanımlar:

1. **Araçlar** — ajanın eylem gerçekleştirmek için çağırabileceği fonksiyonlar (CRUD işlemleri, API çağrıları, komut yürütme)
2. **Kaynaklar** — ajanın bağlam edinmek için okuyabileceği veriler (dokümantasyon, veritabanı şemaları, içerik katalogları)
3. **Prompt'lar** — yaygın görevler için önceden tanımlanmış prompt şablonları

4.2 MCP Sunucuları

MCP sunucusu, MCP protokolünü uygulayan ve araçlar/kaynaklar sağlayan bir süreçtir. Bir MCP sunucusuna bağlandığınızda:

- Tüm araçlar otomatik olarak keşfedilir
- Bağlı tüm sunuculardaki araçlar aynı anda kullanılabilir olur
- Araç açıklamaları, modelin bunları nasıl kullanacağını belirler

4.3 MCP Sunucularını Yapılandırma

Proje yapılandırması (`.mcp.json`) — ekip kullanımı için:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "jira": {
      "command": "npx",
      "args": ["-y", "mcp-server-jira"],
      "env": {
        "JIRA_TOKEN": "${JIRA_TOKEN}"
      }
    }
  }
}
```

Temel noktalar:

- `.mcp.json` proje kökünde tutulur ve sürüm kontrolünde yönetilir
- Ortam değişkenleri (`${GITHUB_TOKEN}`) gizli bilgiler için kullanılır; token'ların kendisi commit'lenmez
- Tüm proje katılımcıları tarafından kullanılabilir

Kullanıcı yapılandırması (`~/.claude.json`) — kişisel/deneysel sunucular için:

- Kullanıcının ev dizininde tutulur
- Sürüm kontrolü üzerinden paylaşılmaz
- Kişisel deneyler ve testler için uygundur

Sunucu seçimi:

- Standart entegrasyonlar (Jira, GitHub, Slack) için mevcut topluluk MCP sunucularını tercih edin
- Yalnızca benzersiz, ekibe özgü iş akışları (workflow) için kendi sunucularınızı oluşturun

4.4 MCP'de `isError` Bayrağı

Bir MCP aracı bir hatayla karşılaştığında, yanıtta `isError: true` kullanır. Bu, ajana çağrının başarısız olduğunu bildirir.

Yapılandırılmış hata (iyi):

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "The service is temporarily unavailable. Timeout while calling the
orders API.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

Genel hata (anti-pattern (kötü kalıp)):

```
{
  "isError": true,
  "content": "Operation failed"
}
```

Genel bir hata, ajana karar vermek için hiçbir bilgi sağlamaz: yeniden denemeli mi, sorguyu değiştirmeli mi, yoksa insana mı yönlendirmeli?

4.5 MCP Kaynakları

Kaynaklar, bir ajanın eylem gerçekleştirmeden bağlam elde etmek için isteyebileceği verilerdir:

- İçerik katalogları (ör. tüm proje görevlerinin listesi, hiyerarşik gezinme)
- Veritabanı şemaları (veri yapısını anlama)
- Dokümantasyon (API referansları, kurum içi kılavuzlar)
- Sorun/görev özetleri

Kaynağın avantajı: ajanın hangi verilerin mevcut olduğunu anlamak için keşif amaçlı araç çağrıları yapması gerekmez. Bir kaynak, anında bir “harita” sağlar.

Bölüm 5: Claude Code — Yapılandırma ve İş Akışları

> Dokümantasyon: Claude Code | Bellek / CLAUDE.md | Skills | MCP | Hook'lar | Alt-ajanlar | GitHub Actions | Headless

5.1 CLAUDE.md Hiyerarşisi

CLAUDE.md, Claude Code için yönerge dosyası/dosyalarıdır. Üç düzeyli bir hiyerarşi vardır:

```
1. User-level: ~/.claude/CLAUDE.md
   - Applies only to that user
   - NOT shared via VCS
   - Personal preferences and working style

2. Project-level: .claude/CLAUDE.md or a root CLAUDE.md
   - Applies to all project contributors
   - Managed via VCS
   - Coding standards, testing standards, architectural decisions

3. Directory-level: CLAUDE.md in subdirectories
   - Applies when working with files in that directory
   - Conventions specific to that part of the codebase
```

Yaygın hata: yeni bir ekip üyesi proje yönergelerini alamaz; çünkü bunlar `.claude/CLAUDE.md` (proje düzeyi) yerine `~/.claude/CLAUDE.md` (kullanıcı düzeyi) içine yerleştirilmiştir.

5.2 `@path` Söz Dizimi (Dosya İçe Aktarımları)

CLAUDE.md, `@path` kullanarak harici dosyalara başvurabilir; bu da yapılandırmayı modüler hale getirir:

```
# Project CLAUDE.md

Coding standards are described in @./standards/coding-style.md
Test requirements are in @./standards/testing-requirements.md
Project overview is in @README.md and dependencies are in @package.json
```

`@path` kuralları:

- Dosya yolunun hemen önünde `@` kullanın (boşluk olmadan)
- Göreli ve mutlak yollar desteklenir
- Göreli yollar, içe aktarımı içeren dosyaya göre çözümlenir
- Maksimum içe aktarma iç içe geçme derinliği 5'tir

Bu, yinelemeyi önler ve her paketin yalnızca ilgili standartları içermesine olanak tanır.

5.3 `.claude/rules/` Dizini

`.claude/rules/`, kuralları konuya göre düzenlemek için kullanılan, monolitik CLAUDE.md'ye bir alternatiftir:

```
.claude/rules/
  testing.md          -- testing conventions
  api-conventions.md  -- API conventions
  deployment.md       -- deployment rules
  react-patterns.md   -- React patterns
```

Temel özellik: koşullu yükleme için `paths` içeren YAML frontmatter:

```
---
paths: ["src/api/**/*"]
---

For API files, use async/await with explicit error handling.
Each endpoint must return a standard response wrapper.
```

```
---
paths: ["**/*.test.tsx", "**/*.test.ts"]
---

Tests must use describe/it blocks.
Use data factories instead of hardcoding.
Do not mock the database—use a test database.
```

Nasıl çalışır:

- Bir kural, **yalnızca** Claude Code, `paths` desenine uyan bir dosyayı düzenlediğinde yüklenir
- Bu, bağlamdan ve token'lardan tasarruf sağlar; ilgisiz kurallar yüklenmez
- Glob desenleri, konumdan bağımsız olarak dosya türüne göre kurallar uygulamanızı sağlar (kod tabanının geneline dağılmış testler için idealdir)

`paths` ile `.claude/rules/` ne zaman, dizin düzeyi CLAUDE.md ne zaman kullanılmalı:

- `.claude/rules/` + `paths` — kurallar birçok dizine yayılmış dosyalara uygulanıyorsa (testler, migrasyonlar)
- Dizin düzeyi CLAUDE.md — kurallar belirli bir dizine bağlıysa ve başka yerde gerekli değilse

5.4 Özel Slash Komutları ve Skills

> **Not:** mevcut Claude Code sürümünde, özel komutlar (`.claude/commands/`) Skills (`.claude/skills/`) ile birleştirilmiştir. Her iki format da `/name` komutları oluşturur. Sınav kılavuzu `.claude/commands/`'a atıfta bulunur; bu format hâlâ desteklenmektedir.

Eğik çizgi komutları (slash command), `/name` üzerinden çağrılan yeniden kullanılabilir prompt şablonlarıdır:

`.claude/commands/` formatı (eski, destekleniyor):

```
.claude/commands/
  review.md        -- /review -- standard code review
  test-gen.md      -- /test-gen -- test generation
```

`.claude/skills/` formatı (güncel):

```
.claude/skills/
  review/SKILL.md  -- /review -- with frontmatter configuration
  test-gen/SKILL.md
```

Proje komutları (`.claude/commands/` veya `.claude/skills/`):

- Sürüm kontrolünde saklanır ve repo klonlandığında herkes tarafından kullanılabilir
- Ekip genelinde tutarlı iş akışları sağlar

Kullanıcı komutları (`~/.claude/commands/` veya `~/.claude/skills/`):

- VCS üzerinden paylaşılmayan kişisel komutlar
- Bireysel iş akışları için

5.5 Skills — `.claude/skills/`

Skills, SKILL.md frontmatter'ı aracılığıyla yapılandırılan gelişmiş komutlardır:

```
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Path to the directory to analyze"
---

Analyze the code structure in the specified directory.
Output a report on dependencies and architectural patterns.
```

Frontmatter parametreleri:

Parametre	Açıklama
`context: fork`	Skill'i yalıtılmış bir alt-ajan içinde çalıştırır. Ayrıntılı çıktı ana oturumu kirletmez
`allowed-tools`	Hangi araçların kullanılabileceğini sınırlar (güvenlik; örneğin izin verilmemişse skill dosyaları silemez)
`argument-hint`	Parametresiz çağrıldığında argüman isteyen ipucu

Skill ile CLAUDE.md ne zaman kullanılmalı:

- **Skill** — belirli bir görev için isteğe bağlı çağrı (inceleme, analiz, üretim)
- **CLAUDE.md** — her zaman yüklenen genel standartlar ve konvansiyonlar

Kişisel skill'ler (`~/.claude/skills/`):

- Takım arkadaşlarınızı etkilememek için farklı adlar altında kişisel varyantlar oluşturun

5.6 Planlama Modu ve Doğrudan Yürütme

Planlama modu:

- Model yalnızca araştırır ve planlar; değişiklik yapmaz
- Kod tabanını keşfetmek için Read, Grep, Glob kullanır
- Kullanıcının onaylayacağı bir uygulama planı üretir
- Yan etkisiz, güvenli keşif sağlar

Planlama modu ne zaman kullanılmalı:

- Büyük değişiklikler (onlarca dosya)
- Birden fazla makul yaklaşım (mikroservisler: sınırlar nasıl tanımlanmalı?)
- Mimari kararlar (hangi framework? hangi yapı?)
- Aşina olunmayan kod tabanı (değiştirmeden önce anlamanız gerekir)
- 45+ dosyayı etkileyen kütüphane geçişleri

Doğrudan yürütme ne zaman kullanılmalı:

- Net bir stack trace'i olan tek dosyalık düzeltmeler
- Tek bir doğrulama kontrolü ekleme
- İyi anlaşılmış, belirsizlik taşımayan değişiklikler

Birleşik yaklaşım:

1. İnceleme ve tasarım için planlama modu
2. Kullanıcı planı onaylar
3. Onaylanan planı uygulamak için doğrudan yürütme

Keşif alt-ajanı — kod tabanını keşfetmeye yönelik uzmanlaşmış bir alt-ajan:

- Ayrıntılı çıktıyı ana bağlamdan yalıtır
- Yalnızca özet döndürür
- Çok aşamalı görevlerde bağlam penceresinin (context window) tükenmesini önler

5.7 `/compact` Komutu

`/compact`, bağlamı sıkıştırmaya yarayan yerleşik bir komuttur:

- Bağlam penceresinde yer açmak için önceki geçmişi özetler
- Bağlamın ayrıntılı araç çıktısıyla dolduğu uzun inceleme oturumlarında kullanılır
- Risk: kesin sayısal değerler, tarihler ve belirli ayrıntılar özetleme sırasında kaybolabilir

5.8 `/memory` Komutu

`/memory`, oturumlar arasında belleği yönetmeye yarayan yerleşik bir komuttur:

- Notları, tercihleri ve bağlamı kaydedebilmeniz için `CLAUDE.md` dosyasını düzenlemeye açar
- Bilgiler oturumlar arasında kalıcı olur ve başlangıçta otomatik olarak yüklenir
- Proje konvansiyonlarını, kullanıcı tercihlerini, sık kullanılan komutları ve mevcut çalışma bağlamını saklamak için kullanışlıdır
- Aynı talimatları her oturumda yeniden açıklamaya alternatiftir

5.9 CI/CD için Claude Code CLI

`-p` (veya `--print`) bayrağı:

```
claude -p "Analyze this pull request for security issues"
```

- Etkileşimsiz mod: prompt'u işler, stdout'a yazdırır ve çıkar
- Kullanıcı girdisi beklemez
- Claude'u CI/CD işlem hatlarında (pipeline) çalıştırmanın tek doğru yolu

CI için yapılandırılmış çıktı:

```
claude -p "Review this PR" --output-format json --json-schema
'{"type":"object",...}'
```

- `--output-format json` — JSON biçiminde çıktı
- `--json-schema` — çıktıyı bir şemaya göre doğrular
- Sonuç ayrıştırılarak satır içi PR yorumları otomatik olarak gönderilebilir

Oturum bağlamı izolasyonu: Kod üreten Claude oturumu, çoğu zaman aynı kodu incelemede daha az etkilidir (model akıl yürütme bağlamını korur ve kendi kararlarını sorgulama olasılığı azalır). İnceleme için bağımsız bir örnek kullanın.

Yinelenen yorumları önleme: Yeni commit'lerden sonra yeniden inceleme yaparken, önceki inceleme sonuçlarını bağlama dahil edin ve Claude'a yalnızca yeni veya çözülmemiş sorunları raporlamasını söyleyin.

5.10 `fork_session` ve Oturum Yönetimi

`--resume <session-name>` adlandırılmış bir oturumu sürdürür:

```
claude --resume investigation-auth-bug
```

- Kaydedilmiş bağlamla önceki bir konuşmayı sürdürür
- Birden fazla oturuma yayılan uzun incelemeler için kullanışlıdır
- Risk: önceki oturumdan sonra dosyalar değiştiyse araç sonuçları güncelliğini yitirmiş olabilir

`fork_session`, paylaşılan bağlamdan bağımsız bir dal oluşturur:

```
Codebase investigation
         |
    fork_session
    /           \
Approach A:      Approach B:
Redux            Context API
```

- İki fork da dal noktasına kadarki bağlamı devralır
- Sonrasında bağımsız biçimde ayrışırlar
- Yaklaşımları veya test stratejilerini karşılaştırmak için kullanışlıdır

Sürdürmek yerine ne zaman yeni oturum başlatılmalı:

- Araç sonuçları güncelliğini yitirmiştir (dosyalar değişmiştir)
- Aradan çok zaman geçmiştir ve bağlam zayıflamıştır
- Eski araç verileriyle sürdürmek yerine "Bulduklarımızın kısa bir özeti: ..." ile yeniden başlamak daha iyidir

---

Bölüm 6: Prompt Mühendisliği — İleri Teknikler

> Dokümantasyon: Prompt Mühendisliği | Anthropic Cookbook

6.1 Few-shot Prompt Kullanımı

Few-shot prompt kullanımı, beklenen davranışı göstermek için prompt'a 2–4 girdi/çıktı örneği eklemektir.

Few-shot neden metinsel açıklamalardan daha etkilidir:

- “Daha kesin ol” gibi belirsiz bir talimat birçok farklı şekilde yorumlanabilir
- Bir örnek, beklenen formatı ve karar mantığını açıkça gösterir
- Model kalıbı yeni vakalara geneller (yalnızca örnekleri tekrar etmez)

Few-shot örnek türleri ve ne zaman kullanılacakları:

1. **Belirsiz senaryolar için örnekler:**

```
Request: "My order is broken"
Action: Call get_customer -> lookup_order -> check status.
Rationale: “broken” may mean a damaged item; you need order details.

Request: "Get me a manager"
Action: Immediately call escalate_to_human.
Rationale: The customer explicitly requests a human. Do not attempt to solve
autonomously.
```

2. **Çıktı biçimlendirme için örnekler:**

```
Finding example:
{
  "location": "src/auth/login.ts:42",
  "issue": "SQL injection in the username parameter",
  "severity": "critical",
  "suggested_fix": "Use a parameterized query"
}
```

3. **Kabul edilebilir kodu sorunlu koddan ayırmak için örnekler:**

```
// Acceptable (do not flag):
const items = data.filter(x => x.active);

// Problem (flag):
const items = data.filter(x => x.active == true); // Use strict equality ===
```

4. **Farklı belge formatlarından çıkarım için örnekler:**

```
Document with inline citations:
"As shown in the study (Smith, 2023), the rate is 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Document with bibliography references:
"The rate is 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **Gayriresmî ölçümler için örnekler:**

```
Text: "about two handfuls of rice"
-> {"amount": "~100g", "original_text": "two handfuls", "precision": "approximate"}

Text: "a pinch of salt"
-> {"amount": "~1g", "original_text": "a pinch", "precision": "approximate"}
```

Few-shot, tamamen kural tabanlı talimatlar için fazla çeşitli olan gayriresmî ve standart dışı ölçü birimlerini çıkarmada özellikle etkilidir.

Prompt'larda format normalleştirme kuralları: Yapılandırılmış çıktı için katı JSON şemaları kullanırken prompt'a normalleştirme kuralları ekleyin:

```
Normalization:
- Dates: always ISO 8601 (YYYY-MM-DD); "yesterday" -> compute an absolute date
- Currency: numeric amount + currency code; "five bucks" -> {"amount": 5, "currency":
"USD"}
- Percentages: decimal fraction; "half" -> 0.5
```

Bu, JSON sözdizimsel olarak geçerli olsa bile değerlerin tutarsız olduğu semantik hataları önler.

6.2 Açık Kriterler ile Belirsiz Talimatlar

Kötü (belirsiz):

```
Check code comments for accuracy.
Be conservative—report only high-confidence findings.
```

İyi (açık kriterler):

```
Flag a comment as problematic ONLY if:
1. The comment describes behavior that CONTRADICTS the actual code behavior
2. The comment references a non-existent function or variable
3. A TODO/FIXME comment refers to a bug that has already been fixed in code

Do NOT flag:
- Comments that are merely stylistically outdated
- Comments with minor wording inaccuracies
- Missing comments (that is a separate category)
```

Önem derecesi kriterlerini örneklerle tanımlayın:

```
CRITICAL: Runtime failure for users
  Example: NullPointerException while processing a payment

HIGH: Security vulnerability
  Example: SQL injection, XSS, missing authorization checks

MEDIUM: Logic bug without immediate impact
  Example: Wrong sorting, off-by-one error

LOW: Code quality
  Example: Duplication, suboptimal algorithm for small data
```

6.3 Prompt Zincirleme (Prompt Chaining)

Prompt zincirleme, karmaşık bir görevi odaklı adımlar dizisine böler:

```
Step 1: Analyze auth.ts (local issues only)
       -> Output: list of issues in auth.ts

Step 2: Analyze database.ts (local issues only)
       -> Output: list of issues in database.ts

Step 3: Integration pass (cross-file dependencies)
       -> Output: issues at module boundaries
```

Bu neden önemlidir:

- Model aynı anda çok fazla dosya aldığında, bazı dosyalardaki bug'ları kaçırıp diğerleri hakkında yüzeysel yorumlar yapabileceği için **dikkat seyrelmesini** önler
- Dosya başına tutarlı analiz kalitesi sağlar
- Dosyalar arası etkileşimlerin ayrı analiz edilmesine olanak tanır

Prompt zincirleme ile dinamik ayrıştırma ne zaman kullanılmalı:

- **Prompt zincirleme** — öngörülebilir, tekrarlanabilir görevler (kod incelemesi, dosya geçişleri)
- **Dinamik ayrıştırma** — alt görevlerin ancak yürütme sırasında netleştiği açık uçlu incelemeler

6.4 “Görüşme” Kalıbı

Bir çözümü uygulamadan önce Claude netleştirici sorular sorar:

```
Claude: "Before implementing caching for the API, a few questions:
1. Which cache invalidation strategy do you prefer—TTL or event-based?
2. Is stale data acceptable when the cache is unavailable?
3. Should caching be per-user or global?
4. What is the expected data volume to cache?"
```

Bunun yararlı olduğu durumlar:

- Aşina olunmayan alanlar (fintech, sağlık, hukuk sistemleri)
- Dolaylı etkileri açık olmayan görevler (önbellek stratejileri, hata modları)
- En iyi seçimin bağlama bağlı olduğu birden fazla uygulanabilir yaklaşım

6.5 Doğrulama ve Geri Bildirimle Yeniden Deneme

Çıkarılan veri doğrulamadan geçemediğinde:

```
Step 1: Extract data from the document
Step 2: Validate (Pydantic, JSON Schema, business rules)
Step 3: If there's an error—retry with context:
  - The original document
  - The previous (incorrect) extraction
  - The specific error: "Field 'total' = 150, but sum(line_items) = 145. Re-check
values."
```

Yeniden denemenin etkili olacağı durumlar:

- Format hataları (tarih yanlış formatta)
- Yapısal hatalar (bir alan yanlış konuma yerleştirilmiş)
- Aritmetik tutarsızlıklar (model yeniden kontrol edebilir)

Yeniden denemenin yardımcı OLMAYACAĞI durumlar:

- Bilgi kaynak belgede yoktur
- Gerekli bağlam dışsaldır (veri, sağlanmamış başka bir belgededir)

Bir doğrulama aracı olarak Pydantic: Pydantic, şema tabanlı veri doğrulama için kullanılan bir Python kütüphanesidir. Sınav açısından temel noktalar şunlardır:

- **Yapısal doğrulama:** Claude'dan JSON aldıktan sonra kodda kontrol edilen tipler, zorunluluk ve enum kısıtları
- **Anlamsal doğrulama:** özel doğrulayıcılar iş mantığını zorunlu kılar (kalemlerin toplamı toplam tutara eşittir; start_date < end_date)
- **Doğrula-yeniden dene döngüleri:** Pydantic doğrulaması başarısız olduğunda bir hata mesajı oluşturun ve hata bağlamıyla Claude'a yeniden prompt verin
- **JSON Schema üretimi:** Pydantic modelleri `tool_use` için JSON Schema üretebilir ve tek doğruluk kaynağı sağlar

6.6 Öz düzeltme

İç çelişkileri saptamak için bir kalıp:

```
{
  "stated_total": "$150.00",
  "calculated_total": "$145.00",
  "conflict_detected": true,
  "line_items": [
    {"name": "Widget A", "price": 75.00},
    {"name": "Widget B", "price": 70.00}
  ]
}
```

Model hem belirtilen değeri hem de hesaplanan değeri çıkarır; bu ikisi farklıysa `conflict_detected`, tutarsızlığı ele almanızı sağlar.

Bölüm 7: Message Batches API

> Dokümantasyon: Message Batches

7.1 Genel bakış

Message Batches API, istek gruplarını asenkron olarak işlenmek üzere göndermenizi sağlar:

Öznitelik	Değer
Tasarruf	Eşzamanlı çağrılara kıyasla **50%**
İşleme penceresi	En fazla **24 saat** (gecikme için SLA garantisi yoktur)
Çok turlu araç çağırma	**Desteklenmez** (bir istek = bir yanıt)
Korelasyon	İsteği ve yanıtı ilişkilendirmek için `custom_id` alanı

7.2 Batch API ile eşzamanlı API ne zaman kullanılmalı?

Görev	API	Neden
Birleştirme öncesi PR kontrolü	**Eşzamanlı**	Geliştirici bekliyordur; 24 saat kabul edilemez
Gece çalışacak teknik borç raporu	**Batch**	Sonuç sabaha gereklidir; 50% tasarruf sağlar
Haftalık güvenlik denetimi	**Batch**	Acil değildir; 50% tasarruf sağlar
Etkileşimli kod incelemesi	**Eşzamanlı**	Anında yanıt gerekir
10.000 belge işleme	**Batch**	Toplu işleme yapılır; tasarruf belirgindir

7.3 `custom_id` kullanımı

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Extract data from: ..."}]
  }
}
```

`custom_id` şunları yapmanızı sağlar:

- Sonucu özgün belgeyle ilişkilendirme
- Hata durumunda yalnızca başarısız belgeleri yeniden gönderme
- Başarılı belgeleri yeniden işlemekten kaçınma

7.4 Toplu işlerde hataları yönetme

1. 100 belgelik bir toplu iş gönderin

2. 95'i başarılı olur; 5'i başarısız olur (bağlam sınırı aşılmıştır)
3. Hataları `custom_id` ile belirleyin
4. Stratejiyi değiştirin (ör. uzun belgeleri parçalara bölün)
5. Yalnızca başarısız olan 5 belgeyi yeniden gönderin

7.5 SLA planlaması

Bir sonuca 30 saat içinde ihtiyacınız varsa ve Batch API'nin tamamlanması 24 saate kadar sürebiliyorsa:

- Gönderim penceresi: 30 - 24 = **6 saat**
- Toplu işler, son teslim tarihinden en geç 24 saat önce gönderilmelidir
- Sık gönderimler için 4 saatlik pencerelere bölün

Bölüm 8: Görev ayrıştırma (task decomposition) stratejileri

8.1 Sabit işlem hatları (prompt zincirleme)

Her adım önceden tanımlanır:

```
Document -> Metadata extraction -> Data extraction -> Validation -> Enrichment -> Final output
```

Ne zaman kullanılır:

- Görev yapısı öngörülebilirdir (incelemeler her zaman aynı şablonu izler)
- Tüm adımlar baştan bellidir
- Kararlılık ve tekrarlanabilirlik gerekir

8.2 Dinamik uyarlamalı ayrıştırma

Alt görevler, ara sonuçlara göre oluşturulur:

```
1. "Add tests for a legacy codebase"
2. -> First: map the structure (Glob, Grep)
3. -> Found: 3 modules with no tests, 2 with partial coverage
4. -> Prioritize: start with the payments module (high risk)
5. -> During work: discovered a dependency on an external API
6. -> Adapt: add a mock for the external API before writing tests
```

Ne zaman kullanılır:

- Ucu açık araştırma görevleri
- Kapsamın tamamı baştan bilinmediğinde

- Her adım önceki adımın sonuçlarına bağlı olduğunda

8.3 Çok geçişli kod incelemesi

10+ dosyalı pull request'ler için:

```
Pass 1 (per-file): Analyze auth.ts -> list local issues
Pass 1 (per-file): Analyze database.ts -> list local issues
Pass 1 (per-file): Analyze routes.ts -> list local issues
...
Pass 2 (integration): Analyze relationships between files
  -> Cross-file issues: inconsistent types, circular dependencies
```

14 dosya üzerinden tek geçiş neden kötüdür:

- Dikkat seyrelmesi: bazı dosyalar derinlemesine, bazıları yüzeysel analiz edilir
- Tutarsız yorumlar: bir kalıp bir dosyada işaretlenirken başka bir dosyada onaylanır
- Kaçırılan hatalar: bilişsel yük fazlalığı nedeniyle bariz hatalar atlanır

---

Bölüm 9: Eskalasyon ve human-in-the-loop

9.1 Ne zaman insana yönlendirmek gerekir

Eskalasyon tetikleyicileri (net kurallar):

Durum	Eylem
Müşteri açıkça “bana bir yönetici bağlayın” der	Hemen insana yönlendirin; çözmeye çalışmayın
Politika isteği kapsamıyordur	İnsana yönlendirin (ör. politika bu konuda sessizken rakip fiyat eşleştirme talebi)
Ajan ilerleme kaydedemiyordur	Makul sayıda denemeden sonra insana yönlendirin
Bir eşiğin üzerindeki finansal işlem	İnsana yönlendirin (tercihen prompt ile değil, hook ile zorunlu kılınarak)
Müşteri ararken birden fazla eşleşme bulunur	Ek tanımlayıcılar isteyin; tahmin yürütmeyin

Güvenilir bir tetikleyici OLMAYANLAR:

Güvenilmez yöntem	Neden başarısız olur
Duygu analizi	Müşterinin ruh hâli, vakanın karmaşıklığıyla korelasyon göstermez

Modelin kendi verdiği güven puanı (1–10)	Model kendinden emin biçimde yanılıyor olabilir; kalibrasyonu zayıftır
Otomatik bir sınıflandırıcı	Aşırı mühendisliktir; elinizde olmayan eğitim verisi gerektirebilir

9.2 Eskalasyon kalıpları

Anında eskalasyon:

```
Customer: "I want to speak to a manager"
Agent: [immediately calls escalate_to_human]
NOT: "I can help with your issue, let me..."
```

Çözüm denemesinden sonra eskalasyon:

```
Customer: "My refrigerator broke two days after purchase"
Agent: [checks the order, offers a warranty replacement]
If the customer is not satisfied -> escalate
```

Nüanslı eskalasyon (kabul et → çöz → tekrarda insana yönlendir):

```
Customer: "This is outrageous, I'm very unhappy with the quality!"
Agent: [acknowledges frustration] "I understand your frustration."
       [offers resolution] "I can offer a replacement or a refund."
Customer: "No, I want to talk to someone!"
Agent: [customer insists again -> immediate escalation]
```

Temel ilke: önce duyguyu kabul edin, ardından somut bir çözüm önerin ve yalnızca müşteri bir insanla görüşme isteğini tekrarlarsa insana yönlendirin. İlk memnuniyetsizlik ifadesinde insana yönlendirmeyin (bu, yönetici istemekle aynı şey değildir).

Politika boşluğu için eskalasyon:

```
Customer: "Competitor X has this item 30% cheaper—give me a discount"
Policy: covers price adjustments only on your own site
Agent: [escalates — policy does not cover competitor price matching]
```

9.3 Yapılandırılmış devir (handoff) protokolleri

Eskalasyonda ajan, insana yapılandırılmış bir özet aktarmalıdır:

```
{
  "customer_id": "CUST-12345",
  "customer_name": "Ivan Petrov",
  "issue_summary": "Refund request for a damaged item",
  "order_id": "ORD-67890",
  "root_cause": "Item arrived damaged; photos attached",
  "actions_taken": [
    "Verified customer via get_customer",
    "Confirmed order via lookup_order",
    "Offered a standard replacement — customer insists on a refund"
  ],
  "refund_amount": "$89.99",
  "recommended_action": "Approve a full refund",
  "escalation_reason": "Customer requested to speak with a manager"
}
```

İnsan operatör tam konuşma dökümüne erişemez; yalnızca bu özeti görür. Bu nedenle özet eksiksiz ve kendi başına anlaşılır olmalıdır.

9.4 Güven kalibrasyonu ve insan gözetimi

Veri çıkarma sistemleri için:

1. **Alan düzeyinde güven puanları:** model, çıkarılan her alan için bir güven puanı üretir
2. **Kalibrasyon:** eşikleri ayarlamak için etiketli doğrulama kümeleri kullanın
3. **Yönlendirme:**
   - Yüksek güven + kararlı doğruluk -> otomatik işleme
   - Düşük güven veya belirsiz kaynaklar -> insan incelemesi

Katmanlı rastgele örnekleme:

- Yüksek güven düzeyli çıkarımlarda bile düzenli olarak bir örneği denetleyin
- Genel 97% doğruluk, belirli bir belge türünde 40% hatayı gizleyebilir
- Doğruluğu yalnızca genel toplamda değil, belge türüne ve alana göre analiz edin

---

Bölüm 10: Çoklu-ajan sistemlerde hata yönetimi

10.1 Hata kategorileri

Kategori	Örnekler	Yeniden denenebilir	Ajan eylemi
Geçici	Zaman aşımı, 503, ağ hatası	Evet	Üstel backoff ile yeniden deneyin
Doğrulama	Geçersiz girdi formatı, eksik zorunlu alan	Hayır (girdiyi düzeltin)	İsteği değiştirin ve yeniden deneyin

İş kuralı	Politika ihlali, eşik aşımı	Hayır	Kullanıcıya açıklayın; bir alternatif önerin
İzin	Erişim reddedildi	Hayır	İnsana yönlendirin

10.2 Hata yönetimi anti-pattern'leri (kötü kalıpları)

Anti-pattern	Sorun	Doğru yaklaşım
Genel “arama kullanılamıyor” durumu	Koordinatör nasıl toparlanacağına karar veremez	Hata türünü, sorguyu, kısmi sonuçları ve alternatifleri döndürün
Sessizce bastırma (boş sonuç = başarı)	Koordinatör eşleşme olmadığını düşünür, oysa bu bir hatadır	“Sonuç yok” ile “arama hatası”nı ayırın
Tek bir hatada tüm iş akışını durdurma	Tüm kısmi sonuçları kaybedersiniz	Kısmi sonuçlarla devam edin; boşlukları not edin
Alt-ajan içinde sonsuz yeniden denemeler	Gecikme ve boşa harcanan kaynaklar	Yerel toparlanma (1–2 yeniden deneme), ardından koordinatöre iletme

10.3 Yapılandırılmış bir alt-ajan hatası

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "AI impact on music industry 2024",
  "partial_results": [
    {"title": "AI Music Generation Report", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "Try a narrower query: 'AI music composition tools'",
    "Use an alternative data source"
  ],
  "coverage_impact": "Not covered: AI impact on music production"
}
```

Bu, koordinatöre karar vermek için gerekli bilgileri sağlar:

- Değiştirilmiş bir sorguyla yeniden denensin mi?
- Kısmi sonuçlar kullanılsın mı?
- Başka bir alt-ajana devredilsin mi?
- Bu bölüm olmadan devam edilip boşluk not edilsin mi?

10.4 Nihai sentezde kapsam notları (coverage annotation)

```
## Report: AI Impact on Creative Industries

### Visual Art (FULL COVERAGE)
[research results]

### Music (PARTIAL COVERAGE — search agent timeout)
[partial results]
⚠️ Note: coverage for this section is limited due to a timeout in the search agent.

### Literature (FULL COVERAGE)
[research results]
```

Bölüm 11: Üretim sistemlerinde bağlam yönetimi

11.1 Olguları ayrı bir bloğa çıkarın

Konuşma geçmişine güvenmek yerine (özetleme sırasında bozulabilir), temel olguları yapılandırılmış bir bloğa çıkarın:

```
=== CASE FACTS (updated whenever a new fact appears) ===
Customer ID: CUST-12345
Order ID: ORD-67890
Order Date: 2025-01-15
Order Amount: $89.99
Issue: Damaged item on delivery
Customer Request: Full refund
Status: Pending manager approval
===
```

Geçmiş nasıl özetlenirse özetlensin, bu bloğu her prompt'a ekleyin.

11.2 Araç sonuçlarını kırpma

`lookup_order` 40+ alan döndürüyorsa ancak mevcut görev için yalnızca 5 alana ihtiyacınız varsa:

```python
# PostToolUse hook: keep only relevant fields
@hook("PostToolUse", tool="lookup_order")
def trim_order_fields(result):
    return {
        "order_id": result["order_id"],
        "status": result["status"],
        "total": result["total"],
        "items": result["items"],
        "return_eligible": result["return_eligible"]
    }
```

Bu, bağlamı korur ve gürültüyü azaltır.

11.3 Konuma Duyarlı Girdi

Kritik bilgileri lost-in-the-middle etkisini dikkate alarak yerleştirin:

```
[KEY FINDINGS — at the top]
Found 3 critical vulnerabilities...

[DETAILED RESULTS — middle]
=== File auth.ts ===
...
=== File database.ts ===
...

[ACTION ITEMS — at the end]
Priority: fix auth.ts vulnerabilities before merge.
```

11.4 Not Defteri (Scratchpad) Dosyaları

Uzun incelemelerde ajan, temel bulguları bir not defteri dosyasına yazabilir:

```
# investigation-scratchpad.md
## Key findings
- PaymentProcessor in src/payments/processor.ts inherits from BaseProcessor
- refund() is called from 3 places: OrderController, AdminPanel, CronJob
- External PaymentGateway API has a rate limit of 100 req/min
- Migration #47 added refund_reason (NOT NULL) — 2024-12-01
```

Bağlam zayıfladığında (veya yeni bir oturumda), ajan keşfi yeniden çalıştırmak yerine not defterine başvurabilir.

11.5 Bağlamı Korumak İçin Alt-Ajanlara (Subagents) Delege Etme

```
Main agent: "Investigate dependencies of the payments module"
  -> Subagent (Explore): reads 15 files, traces imports
  -> Returns: "Payments depends on AuthService, OrderModel, and the external
PaymentGateway API"

Main agent: keeps one line in context instead of 15 files
```

Ayrı bağlam katmanı: Çoklu-ajan sistemlerde her alt-ajan sınırlı bir bağlam bütçesi içinde çalışır; yalnızca görevi için gerekli bilgileri alır. Koordinatör ayrı bir bağlam katmanı gibi davranır: alt-ajan çıktılarını birleştirir, genel durumu saklar ve bağlam tahsis eder. Bu, bir ajanın bağlam penceresini diğerleriyle ilgisiz bilgilerle tüketmesi anlamına gelen “bağlam sızıntısını” önler.

Alt-ajanlar için kısıtlı bağlam bütçeleri:

- Asgari bağlam gönderin: belirli bir görev + gerekli veriler
- Alt-ajandan ham veri dökümleri değil, yapılandırılmış sonuçlar döndürmesini isteyin
- Alt-ajanın araç setini sınırlamak için `allowedTools` kullanın; daha az araç, daha az dikkat dağıtıcı unsur ve daha düşük bağlam maliyeti demektir

11.6 Yapılandırılmış Durum Kalıcılığı (State Persistence) (çökme kurtarma için)

Her ajan, durumunu bilinen bir konuma dışa aktarır:

```
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["AI music 2024", "AI music composition"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["music composition", "music production"],
  "gaps": ["music distribution", "music licensing"]
}
```

Koordinatör, sürdürme sırasında bir manifest yükler:

```
// agent-state/manifest.json
{
  "web-search": "completed",
  "doc-analysis": "in_progress",
  "synthesis": "not_started"
}
```

Bölüm 12: Köken Takibini (Provenance) Koruma

12.1 Atıf Kaybı Problemi

Birden çok kaynaktan gelen sonuçlar özetlenirken “iddia → kaynak” bağlantısı kaybolabilir:

```
Bad: "The AI music market is estimated at $3.2B." (No source, no year.)

Good:
{
  "claim": "The AI music market is estimated at $3.2B.",
  "source_url": "https://example.com/report",
  "source_name": "Global AI Music Report 2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

12.2 Çelişkili Verileri Ele Alma

İki kaynak farklı değerler verdiğinde:

```
{
  "claim": "Share of AI-generated music on streaming platforms",
  "values": [
    {
      "value": "12%",
      "source": "Spotify Annual Report 2024",
      "date": "2024-03",
      "methodology": "Automated classification"
    },
    {
      "value": "8%",
      "source": "Music Industry Association Survey",
      "date": "2024-07",
      "methodology": "Survey of 500 labels"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Difference in methodology and time period"
}
```

Bir değeri keyfi olarak seçmeyin. Her ikisini de atıfla koruyun ve kararı koordinatöre bırakın.

12.3 Doğru Yorumlama İçin Tarihleri Dahil Etme

Tarihler olmadığında, zamansal farklar yanlışlıkla çelişki gibi yorumlanabilir:

```
Bad: "Source A says 10%, source B says 15%. Contradiction."
Good: "Source A (2023) says 10%, source B (2024) says 15%. Likely +5% growth over a
year."
```

12.4 İçerik Türüne Göre Sunma

Her şeyi tek bir formata zorlamayın:

- Finansal veriler -> tablolar
- Haber ve analiz -> düz yazı
- Teknik bulgular -> yapılandırılmış listeler
- Zaman serileri -> kronolojik sıralama

Bölüm 13: Claude Code Yerleşik Araçları

13.1 Araç Seçimi Başvuru Tablosu

Görev	Araç	Örnek
Dosyaları ad/desene göre bul	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
Dosyalar içinde arama yap	**Grep**	İşlev adı, hata mesajı, import
Bir dosyanın tamamını oku	**Read**	Analiz için bir dosya yükle
Yeni bir dosya yaz	**Write**	Sıfırdan bir dosya oluştur
Mevcut bir dosyayı hassas biçimde düzenle	**Edit**	Benzersiz metin eşleşmesiyle belirli bir parçayı değiştir
Shell komutu çalıştır	**Bash**	git, npm, testleri çalıştırma, derleme

13.2 Kademeli İnceleme Stratejisi

Tüm dosyaları tek seferde okumayın. Anlayışı kademeli olarak kurun:

```
1. Grep: find entry points (function definition, export)
2. Read: read the found files
3. Grep: find usages (import, calls)
4. Read: read consumer files
5. Repeat until you have a complete picture
```

13.3 Edit Yerine Read + Write Geri Dönüşü

Edit, benzersiz olmayan metin eşleşmesi nedeniyle başarısız olduğunda:

1. Read — tam dosya içeriğini yükle
2. İçeriği programatik olarak değiştir
3. Write — güncellenmiş sürümü yaz

KISIM II: SINAV ALANI NOTLARI

Alan 1: Ajan Mimarisi ve Orkestrasyon (27%)

1.1 Otonom Görev Yürütme İçin Ajan Döngülerini (Agentic Loops) Tasarlama

Temel bilgi:

- Ajan döngüsünün yaşam döngüsü: bir Claude isteği gönderin, `stop_reason` değerini (`"tool_use"` ile `"end_turn"`) kontrol edin, araçları yürütün, sonuçları sonraki iterasyon için döndürün
- Araç sonuçları konuşma geçmişine eklenir; böylece model sonraki eyleme karar verebilir
- Model güdümlü karar alma (sonraki aracı Claude seçer) ile sabit kodlanmış karar ağaçları arasındaki fark

Temel beceriler:

- Akış denetimi: `stop_reason = "tool_use"` olduğunda döngüye devam edin ve `"end_turn"` durumunda durun
- İterasyonlar arasında araç sonuçlarını bağlama ekleme
- Kaçınılması gereken anti-pattern'ler (kötü kalıplar): tamamlanmayı anlamak için asistan metnini ayrıştırmak, birincil durdurma mekanizması olarak keyfi iterasyon sınırları kullanmak

1.2 Çoklu-Ajan Sistemleri Orkestre Etme (Koordinatör–Alt-Ajan)

Temel bilgi:

- Merkez-uç mimarisi: koordinatör, ajanlar arası tüm iletişimin, hata yönetiminin ve yönlendirmenin sahibidir
- Alt-ajanlar yalıtılmış bağlamla çalışır; koordinatörün geçmişini otomatik olarak miras almazlar
- Koordinatör sorumlulukları: görev ayrıştırma, delege etme, sonuç toplama, alt-ajanların dinamik seçimi
- Koordinatörün aşırı dar ayrıştırma yapması riski

Temel beceriler:

- Yinelemeyi en aza indirmek için araştırma kapsamını alt-ajanlar arasında bölüştürün
- İteratif iyileştirme döngüleri uygulayın (koordinatör sentezi değerlendirir ve görevleri yeniden yönlendirir)
- Gözlemlenebilirlik için tüm iletişimi koordinatör üzerinden yönlendirin

1.3 Alt-Ajan Çağrılarını, Bağlam Aktarımını ve Başlatmayı Yapılandırma

Temel bilgi:

- `Task` aracı alt-ajanları başlatır; koordinatörün `allowedTools` listesinde `"Task"` bulunmalıdır
- Alt-ajan bağlamı prompt'a açıkça dahil edilmelidir; alt-ajanlar üst bağlamı miras almaz
- `AgentDefinition` yapılandırması: açıklamalar, sistem komutları (system prompts), araç kısıtları
- Alternatifleri keşfetmek için `fork_session` üzerinden oturum yönetimi

Temel beceriler:

- Önceki ajanlardan gelen tam çıktıları alt-ajan prompt'una dahil edin
- Bağlam aktarırken veriyi meta veriden ayırmak için yapılandırılmış formatlar kullanın
- Tek bir koordinatör turunda birden fazla `Task` çağrısıyla paralel alt-ajanlar başlatın
- Koordinatör prompt'larını adım adım talimatlar yerine hedefler ve kalite ölçütleri üzerinden yazın

1.4 Kural Uygulatma ve Devir Kalıplarıyla Çok Adımlı İş Akışları Uygulama

Temel bilgi:

- Bir iş akışının sıralanması için **programatik kural uygulatma** (hook'lar, ön koşullar) ile **prompt yönlendirmesi** arasındaki fark
- Deterministik garantilere ihtiyaç duyduğunuzda (ör. finansal işlemlerden önce kimlik doğrulama), prompt'lar tek başına yeterli değildir
- Eskalasyon sırasında yapılandırılmış devir protokolleri (müşteri ID'si, gerekçe, önerilen eylem)

Temel beceriler:

- Önceki adımlar tamamlanana kadar aşağı akış çağrılarını engelleyen programatik ön koşullar (ör. `process_refund` çağrısını `get_customer` doğrulanmış bir ID döndürene kadar engelleyin)
- Çok boyutlu müşteri taleplerini ayrı maddelere ayrıştırın
- İnsana yönlendirirken yapılandırılmış özetler üretin

1.5 Araç Çağrılarını Yakalamak ve Veriyi Normalleştirmek İçin Agent SDK Hook'ları

Temel bilgi:

- Araç sonuçlarını model tüketmeden önce yakalamaya yönelik hook kalıpları (ör. `PostToolUse`)
- Giden çağrıları yakalayarak uyumluluk kurallarını uygulatmak için kullanılan hook'lar (ör. belirli bir eşiğin üzerindeki iadeleri engelleyin)
- Hook'lar **deterministik garantiler** sağlar; prompt talimatları ise **olasılıksal uyum** sağlar

Temel beceriler:

- Veri formatlarını (Unix zaman damgaları, ISO 8601, sayısal durum kodları) normalleştirmek için `PostToolUse` hook'ları
- Politika ihlali yapan eylemleri engelleyip eskalasyona yönlendiren yakalama hook'ları
- İş kuralları garantili uyum gerektirdiğinde prompt'lar yerine hook'ları seçin

1.6 Karmaşık İş Akışları İçin Görev Ayrıştırma Stratejileri

Temel bilgi:

- Ara sonuçlara göre **sabit işlem hatları** (prompt zincirleme) ile **dinamik uyarlamalı ayrıştırma** arasındaki fark
- Prompt zincirleme: sıralı adımlar (her dosyayı ayrı analiz edin, ardından bir entegrasyon geçişi çalıştırın)
- Keşfedilenlere göre alt görevler üreten uyarlamalı inceleme planları

Temel beceriler:

- Öngörülebilir çok boyutlu incelemeler için prompt zincirlemeyi; açık uçlu incelemeler için dinamik ayrıştırmayı kullanın
- Büyük kod incelemelerini dosya başına analiz ve ayrı bir çapraz dosya entegrasyon geçişi olarak bölün
- Açık uçlu görevleri ayrıştırın: önce yapıyı haritalayın, ardından önceliklendirilmiş bir plan oluşturun

1.7 Oturum Durumu, Sürdürme ve Çatallama

Temel bilgi:

- Adlandırılmış oturumları sürdürmek için `--resume <session-name>`
- Paylaşılan bağlamdan bağımsız inceleme dalları oluşturmak için `fork_session`
- Oturumları sürdürürken dosya değişiklikleri hakkında ajanı bilgilendirmenin önemi
- Yapılandırılmış bir özetle yeni oturum başlatmak, eski sonuçlarla sürdürmekten daha güvenilir olabilir

Temel beceriler:

- Adlandırılmış inceleme oturumlarını sürdürmek için `--resume` kullanın
- Yaklaşımları paralel karşılaştırmak için `fork_session` kullanın
- Sürdürme (bağlam hala güncel) ile yeni oturum başlatma (sonuçlar eski) arasında seçim yapın

---

Alan 2: Araç Tasarımı ve MCP Entegrasyonu (18%)

2.1 Net Açıklamalarla Araç Arayüzleri Tasarlama

Temel bilgi:

- Araç açıklamaları, bir LLM'in araç seçmek için kullandığı **birincil mekanizmadır**; yetersiz açıklamalar güvenilmez seçime yol açar
- Girdi formatlarını, örnek sorguları, uç durumları ve uygulanabilirlik sınırlarını dahil etmenin önemi
- Belirsiz veya örtüşen açıklamalar hatalı yönlendirmeye neden olur
- Sistem komutu ifadeleri, araçlarla istenmeyen çağrışımlar oluşturabilir

Temel beceriler:

- Her aracı benzer alternatiflerden net biçimde ayıran açıklamalar yazın
- İşlevsel örtüşmeyi ortadan kaldırmak için araçları yeniden adlandırın (ör. `analyze_content` -> `extract_web_results`)
- Genel amaçlı araçları net giriş/çıkış sözleşmeleri olan uzmanlaşmış araçlara bölün

2.2 MCP Araçları İçin Yapılandırılmış Hata Yanıtları Uygulama

Temel bilgi:

- MCP araç yanıtlarında `isError` bayrağı
- **Geçici hatalar** (zaman aşımı), **doğrulama hataları** (hatalı girdi), **iş hataları** (politika ihlalleri) ve **erişim/izin hataları** arasındaki fark
- Genel hatalar ("Operation failed") doğru kurtarma kararlarını engeller
- Yeniden denenebilir ve yeniden denenemez hatalar arasındaki fark

Temel beceriler:

- `errorCategory` (transient/validation/permission), `isRetryable` ve insan tarafından okunabilir bir mesaj gibi yapılandırılmış meta veriler döndürün
- İş kuralı ihlalleri için net kullanıcıya yönelik açıklamalarla `retryable: false` kullanın
- Geçici arızalar için yerel kurtarmayı alt-ajanların içinde yapın; yalnızca çözemedikleri hataları üst katmana iletin
- Erişim hatalarını (yeniden deneme kararı) geçerli boş sonuçlardan (eşleşme yok) ayırt edin

2.3 Araçları Ajanlar Arasında Tahsis Etme ve `tool_choice` Yapılandırması

Temel bilgi:

- Ajan başına çok fazla araç (ör. 4-5 yerine 18) araç seçimi güvenilirliğini **azaltır**
- Uzmanlık alanlarının dışında araçlara sahip ajanlar bunları yanlış kullanma eğilimindedir
- Kapsamı sınırlandırılmış araç erişimi: yalnızca rolle ilgili araçlar ve sınırlı sayıda roller arası yardımcı araç
- `tool_choice`: `"auto"`, `"any"` ve zorunlu araç seçimi (`{"type": "tool", "name": "..."}`)

Temel beceriler:

- Her alt-ajanın araç setini rolüyle ilgili olanlarla sınırlandırın
- Genel araçları kısıtlı alternatiflerle değiştirin (ör. `fetch_url` -> `load_document`)
- Metin yanıtı yerine araç çağrısını garanti etmek için `tool_choice: "any"` kullanın
- Yürütme sırasını güvenceye almak için belirli bir aracı zorunlu kılın

2.4 MCP Sunucularını Claude Code ve Ajan İş Akışlarına Entegre Etme

Temel bilgi:

- MCP sunucu kapsamı: ekipler için proje (`.mcp.json`), deneyler için kullanıcı (`~/.claude.json`)
- Gizli bilgi yönetimi için `.mcp.json` içinde ortam değişkeni ikamesi (ör. `${GITHUB_TOKEN}`)
- Bağlı tüm MCP sunucularından gelen araçlar bağlantı sırasında keşfedilir ve aynı anda kullanılabilir
- Keşif amaçlı araç çağrılarını azaltmak için “içerik katalogları” olarak MCP kaynakları (görev özetleri, veritabanı şemaları)

Temel beceriler:

- Paylaşılan MCP sunucularını proje `.mcp.json` dosyasında ortam değişkeni tabanlı token'larla yapılandırın
- Kişisel/deneysel sunucuları `~/.claude.json` içinde tutun
- Standart entegrasyonlar için özel sunucular yerine topluluk MCP sunucularını tercih edin

2.5 Yerleşik Araçları (Read, Write, Edit, Bash, Grep, Glob) Seçme ve Uygulama

Temel bilgi:

- **Grep**: dosya içeriklerinde arama yapma (işlev adları, hata mesajları, import'lar)
- **Glob**: ad/uzantı desenlerine göre dosya bulma
- **Read/Write**: tam dosya işlemleri; **Edit**: benzersiz metin eşleşmeleriyle hassas değişiklikler
- Edit, benzersiz olmayan eşleşmeler nedeniyle başarısız olursa Read + Write'a geri dönün

Temel beceriler:

- İçerik araması için Grep'i, desenlere göre dosya keşfi için Glob'u kullanın
- Anlayışı kademeli kurun: giriş noktalarını Grep ile bulun, ardından akışları izlemek için Read kullanın
- Sarmalayıcı modüller üzerinden işlev kullanımını izleyin

Alan 3: Claude Code Yapılandırması ve İş Akışları (20%)

3.1 CLAUDE.md'yi Hiyerarşi, Kapsam ve Modüler Organizasyonla Yapılandırma

Temel bilgi:

- CLAUDE.md hiyerarşisi: kullanıcı düzeyi (`~/.claude/CLAUDE.md`), proje düzeyi (`.claude/CLAUDE.md` veya kök `CLAUDE.md`) ve dizin düzeyi (alt dizinlerdeki CLAUDE.md)
- Kullanıcı düzeyi ayarlar yalnızca tek bir kullanıcı için geçerlidir ve VCS üzerinden paylaşılmaz
- CLAUDE.md'yi modülerleştirmek için dış dosyalara başvurmayı sağlayan `@path` sözdizimi (ör. `@./standards/coding-style.md`)
- Monolitik bir CLAUDE.md yerine konu odaklı kural dosyaları için `.claude/rules/` dizini

Temel beceriler:

- Hiyerarşi sorunlarını teşhis edin (yeni bir ekip üyesi, talimatlar proje düzeyi yerine kullanıcı düzeyinde olduğu için bunları kaçırır)
- Her paketin CLAUDE.md'sinde standartları seçici olarak dahil etmek için `@path` kullanın (ör. `@./standards/testing.md`)
- Büyük CLAUDE.md'yi birden çok `.claude/rules/` dosyasına bölün (testing.md, api-conventions.md, deployment.md)

3.2 Özel Slash Komutlarını ve Skills'i Oluşturma ve Yapılandırma

Temel bilgi:

- `.claude/commands/` içindeki **proje komutları** (VCS ile paylaşılır) ile `~/.claude/commands/` içindeki **kullanıcı komutları**
- `.claude/skills/` içindeki Skills; `SKILL.md` frontmatter alanları: `context: fork`, `allowed-tools`, `argument-hint`
- `context: fork`, skill'i ana oturumu kirletmemesi için yalıtılmış bir alt-ajan bağlamında çalıştırır
- Kişisel skill varyantları `~/.claude/skills/` altında farklı adlarla bulunabilir

Temel beceriler:

- Tüm ekibin erişebilmesi için proje slash komutlarını `.claude/commands/` içinde saklayın

- Ayrıntılı çıktı üreten Skills'i yalıtmak için `context: fork` kullanın
- Bir skill'in hangi araçları kullanabileceğini sınırlamak için `allowed-tools` kullanın
- Geliştiricilerden gerekli parametreleri istemek için `argument-hint` kullanın

3.3 Koşullu Konvansiyon Yükleme İçin Yola Özgü Kuralları Kullanma

Temel bilgi:

- `.claude/rules/` dosyaları, kuralları glob desenlerine göre etkinleştirmek için YAML frontmatter içinde `paths` içerebilir
- Yol kapsamlı kurallar, bağlam ve token tasarrufu sağlayarak eşleşen dosyalar düzenlenirken **yalnızca** o zaman yüklenir
- Konvansiyonlar çok sayıda dizine yayıldığında (ör. testler), glob tabanlı yol kuralları dizin düzeyi CLAUDE.md'ye tercih edilebilir

Temel beceriler:

- Yalnızca eşleşen dosyalar üzerinde çalışılırken yüklenmesi için `.claude/rules/` dosyalarını `paths: ["terraform/**/*"]` ile oluşturun
- Konumdan bağımsız olarak dosya türüne göre konvansiyon uygulamak için glob desenleri (`**/*.test.tsx`) kullanın
- Konvansiyonlar kod tabanının geneline yayıldığında dizin düzeyi CLAUDE.md yerine yola özgü kuralları tercih edin

3.4 Planlama Modunu mu Doğrudan Yürütmeyi (Direct Execution) mi Kullanacağınıza Karar Verme

Temel bilgi:

- **Planlama modu**: büyük değişiklikler, birden çok uygulanabilir yaklaşım ve mimari kararlar içeren karmaşık görevler için
- **Doğrudan yürütme**: basit, iyi anlaşılmış değişiklikler için (ör. tek bir doğrulama eklemek)
- Planlama modu, değişiklik yapmadan önce kod tabanının güvenli biçimde keşfedilmesini sağlar
- Explore alt-ajanı, ayrıntılı keşif çıktısını yalıtır

Temel beceriler:

- Mimari sonuçları olan görevlerde planlama modunu kullanın (mikroservisler, 45+ dosyaya dokunan migrasyonlar)
- Net bir stack trace ve tek bir dosya içeren düzeltmelerde doğrudan yürütmeyi kullanın
- Çok aşamalı görevlerde bağlam penceresinin tükenmesini önlemek için Explore alt-ajanını kullanın
- Yaklaşımları birleştirin: keşif için planlama yapın, ardından uygulama için yürütün

3.5 Kademeli İyileştirme İçin İteratif Rafine Etme

Temel bilgi:

- Somut girdi/çıktı örnekleri, beklentileri iletmenin en etkili yoludur
- **Test güdümlü iterasyon**: önce testleri yazın, ardından hatalara göre iterasyon yapın
- “Mülakat” deseni: Claude, bariz olmayan tasarım değerlendirmelerini ortaya çıkarmak için sorular sorar
- Tüm sorunları tek iletide ne zaman vereceğiniz (birbirine bağımlı) ile sırayla ne zaman vereceğiniz (bağımsız) arasındaki fark

Temel beceriler:

- Dönüşüm gereksinimlerini netleştirmek için 2-3 somut girdi/çıktı örneği sağlayın
- Uygulamadan önce beklenen davranış, sınır durumları ve performans gereksinimlerini içeren test kümeleri oluşturun
- Tasarım boyutlarını ortaya çıkarmak için mülakat desenini kullanın (önbellek geçersizleştirme, hata modları)
- Sınır durumları için örnek girdiler ve beklenen çıktılar içeren somut test vakaları sağlayın

3.6 Claude Code'u CI/CD İşlem Hatlarına Entegre Etme

Temel bilgi:

- Otomatik işlem hatlarında etkileşimsiz mod için `-p` (veya `--print`) bayrağı
- CI'da yapılandırılmış çıktı için `--output-format json` ve `--json-schema`
- CLAUDE.md, CI tarafından tetiklenen Claude Code için proje bağlamı (test standartları, inceleme ölçütleri) sağlar
- **Oturum bağlamı yalıtımı**: kodu üreten aynı oturum, onu incelemede bağımsız bir örnek kadar etkili değildir

Temel beceriler:

- Etkileşimli girdide takılmayı önlemek için Claude Code'u CI'da `-p` ile çalıştırın
- Yapılandırılmış sonuçlar için `--output-format json` + `--json-schema` kullanın (ör. satır içi PR yorumları)
- Yeni commit'lerden sonra yeniden çalıştırırken önceki inceleme sonuçlarını dahil edin (yalnızca yeni/düzeltilmemiş sorunları raporlayın)
- Test üretim kalitesini artırmak için test standartlarını ve kullanılabilir fixture'ları CLAUDE.md'de belgeleyin
- Tekrarı önlemek ve stili tutarlı tutmak için yeni testler üretirken mevcut test dosyalarını bağlama dahil edin

---

Alan 4: Prompt Mühendisliği ve Yapılandırılmış Çıktı (20%)

4.1 Doğruluğu Artırmak İçin Açık Ölçütlerle Prompt Tasarlama

Temel bilgi:

- Açık ölçütler belirsiz talimatlardan daha etkilidir (ör. “yorumları yalnızca kodla çeliştiklerinde işaretle” ifadesi, “yorum doğruluğunu kontrol et” ifadesinden daha iyidir)
- “Daha muhafazakar ol” gibi genel yönlendirmeler, somut kategorik ölçütlerden daha kötü çalışır
- Yanlış pozitiflerin geliştirici güvenine etkisi: bazı kategorilerdeki yüksek yanlış pozitif oranları, doğru kategorilere duyulan güveni zayıflatır

Temel beceriler:

- İnceleme ölçütlerini tanımlayın: neyin raporlanacağı (hatalar, güvenlik) ve neyin yok sayılacağı (küçük stil sorunları)
- Yanlış pozitif oranı yüksek kategorileri geçici olarak devre dışı bırakın
- Her seviye için kod örnekleriyle açık önem derecesi ölçütleri tanımlayın

4.2 Çıktı Tutarlılığını Artırmak İçin few-shot prompt'lama Kullanma

Temel bilgi:

- few-shot örnekleri, tutarlı biçimlendirilmiş ve eyleme dönüştürülebilir çıktı üretmenin en etkili yöntemidir
- few-shot, belirsiz durumların nasıl ele alınacağını gösterebilir (araç seçimi, test kapsamındaki boşluklar)
- few-shot, modelin yalnızca varsayılanları tekrarlamak yerine yeni desenlere genelleme yapmasına yardımcı olur
- few-shot, çıkarma görevlerinde halüsinasyonları azaltabilir

Temel beceriler:

- Belirsiz senaryolar için gerekçeleriyle birlikte 2-4 hedeflenmiş örnek sağlayın
- Çıktı formatını gösteren few-shot örnekleri dahil edin (konum, sorun, önem derecesi, önerilen düzeltme)
- Kabul edilebilir kod desenlerini gerçek sorunlardan ayıran örnekler sağlayın
- Farklı yapılara sahip belgelerden doğru çıkarma örnekleri sağlayın

4.3 `tool_use` ve JSON Şemalarıyla Yapılandırılmış Çıktıyı Zorunlu Kılma

Temel bilgi:

- JSON Şemaları ile `tool_use`, şemaya uygun çıktıyı garanti etmenin ve JSON sözdizimi hatalarını ortadan kaldırmanın en güvenilir yoludur
- `tool_choice: "auto"` ile model metin döndürebilir; `"any"` ile bir araç çağırmak zorundadır; zorunlu seçim belirli bir aracı seçer
- Katı JSON Şemaları sözdizimi hatalarını ortadan kaldırır, ancak semantik hataları önlemez (toplamlar tutmaz; değerler yanlış alanlardadır)
- Şema tasarımı: zorunlu ve opsiyonel alanlar; genişletilebilirlik için “other” içeren enum'lar ve ayrıntı metni

Temel beceriler:

- JSON Şemalarıyla çıkarma araçları tanımlayın ve `tool_use` sonuçlarından veri ayrıştırın
- Birden çok şema olduğunda yapılandırılmış çıktıyı garanti etmek için `tool_choice: "any"` kullanın
- Belirli bir araç çağrısını zorunlu kılın: `tool_choice: {"type": "tool", "name": "extract_metadata"}`
- Değer uydurmayı önlemek için kaynak bilgi içermeyebileceğinde alanları opsiyonel/null yapılabilir tanımlayın
- Genişletilebilir kategorizasyon için `"unclear"` ve `"other"` gibi enum değerlerini ayrıntı alanlarıyla birlikte kullanın

4.4 Çıkarma Kalitesi İçin Doğrulama, Yeniden Deneme ve Geri Bildirim Döngüleri Uygulama

Temel bilgi:

- Hata geri bildirimiyle yeniden deneme (retry-with-feedback): düzeltmeleri yönlendirmek için yeniden deneme prompt'una somut doğrulama hatalarını dahil edin
- Bilgi kaynakta gerçekten yoksa yeniden denemeler etkisizdir
- Geri bildirim döngüsü tasarımı: bir bulguyu tetikleyen deseni izleyin (`detected_pattern`)
- Semantik hatalar (toplamlar uzlaşmaz) ile sözdizimi hataları (`tool_use` tarafından ele alınır) arasındaki fark

Temel beceriler:

- Orijinal belge, yanlış bir çıkarma ve belirli doğrulama hataları içeren takip prompt'ları kullanın
- Yeniden denemenin ne zaman etkisiz olacağını belirleyin (gerekli bilgi yalnızca dış bir belgededir)
- Yanlış pozitifleri analiz etmek için bulgulara `detected_pattern` alanları ekleyin
- Tutarsızlıkları tespit etmek için hem `calculated_total` hem de `stated_total` çıkararak öz düzeltme tasarlayın

4.5 Verimli Toplu İşleme Stratejileri Tasarlama

Temel bilgi:

- Message Batches API: 50% tasarruf, 24 saate kadar işleme penceresi, gecikme SLA garantisi yok
- Toplu işleme, engelleyici olmayan görevler için uygundur (gecelik raporlar, denetimler); engelleyici görevler için uygun değildir (merge öncesi kontroller)
- Batch API, tek bir istek içinde çok turlu araç çağırmayı desteklemez
- `custom_id` alanları, batch'ler içinde istek/yanıt eşleştirmesi yapar

Temel beceriler:

- Engelleyici kontroller için senkron API'yi; gecelik/haftalık iş yükleri için Batch API'yi kullanın
- Toplu gönderim temposunu SLA ihtiyaçlarına göre planlayın (ör. 24 saatlik işleme ile 30 saatlik garanti için 4 saatlik pencereler)
- Hataları, yalnızca başarısız belgeleri yeniden göndererek ele alın (`custom_id` ile tanımlanır)
- Büyük ölçekli işleme çalıştırmadan önce bir örneklem kullanarak prompt'larda iterasyon yapın

4.6 Çoklu Örnek ve Çok Geçişli İnceleme Mimarileri Tasarlama

Temel bilgi:

- Öz inceleme sınırlamaları: model kendi muhakeme bağlamını korur ve kendi kararlarına meydan okuma olasılığı daha düşüktür
- Bağımsız inceleme örnekleri (üretim bağlamı olmadan), ince sorunları bulmada daha iyidir
- Çok geçişli inceleme: dikkat dağılmasını önlemek için dosya başına yerel analiz ve ardından dosyalar arası entegrasyon geçişi

Temel beceriler:

- Değişiklikleri üretim bağlamı olmadan incelemek için ikinci, bağımsız bir Claude örneği kullanın
- Çok dosyalı incelemeleri, dosya başına geçişlere ve dosyalar arası veri akışı analizi için entegrasyon geçişlerine bölün
- İncelemeleri kalibre edilmiş biçimde yönlendirmek için öz puanlanmış güven içeren doğrulama geçişleri kullanın

---

Alan 5: Bağlam Yönetimi ve Güvenilirlik (15%)

5.1 Kritik Bilgiyi Korumak İçin Konuşma Bağlamını Yönetme

Temel bilgi:

- Kademeli özetleme riskleri: sayısal değerler, yüzdeler ve tarihler belirsiz özetlere sıkışır

- Lost-in-the-middle etkisi: modeller uzun girdilerin başlangıcını ve sonunu güvenilir biçimde işler, ancak ortadaki bulguları kaçırabilir
- Araç çıktıları, alaka düzeylerine kıyasla bağlamda orantısız biçimde birikebilir (5 alan gerekirken 40+ alan)
- Sonraki API isteklerinde tam konuşma geçmişini göndermenin önemi

Temel beceriler:

- İşlemsel olguları, özetlenmiş geçmişin dışında kalıcı bir “vaka olguları” bloğuna çıkarın
- Ayrıntılı araç çıktılarını ilgili alanlara indirgeyin
- Temel bulguları, açık bölüm başlıklarıyla toplulaştırılmış verinin başına yerleştirin
- Alt-ajanların yapılandırılmış çıktılara metadata (tarihler, kaynaklar) eklemesini zorunlu kılın

5.2 Etkili Eskalasyon Desenleri Tasarlama ve Belirsizliği Çözme

Temel bilgi:

- Uygun eskalasyon tetikleyicileri: açık insan talebi, politika boşlukları/istisnaları, ilerleme kaydedememe
- Anında eskalasyon (açık talep) ile çözmeye çalışma (ajan kapsamı içinde) arasındaki fark
- Duygu analizi ve modelin kendi güven derecelendirmeleri, vaka karmaşıklığı için güvenilmez göstergelerdir
- Birden çok müşteri eşleşmesi, sezgisel tahmin değil ek tanımlayıcılar istemeyi gerektirir

Temel beceriler:

- Sistem komutunda few-shot örnekleriyle açık eskalasyon ölçütleri tanımlayın
- Bir insan talep eden açık istekleri ek araştırma yapmadan hemen yerine getirin
- Politika belirli bir istek için belirsiz veya sessiz kaldığında insana yönlendirin
- Araç sonuçları birden çok eşleşme içerdiğinde ek tanımlayıcılar isteyin

5.3 Çoklu-Ajan Sistemlerde Hata Yayılımı (Error Propagation) Stratejileri Uygulama

Temel bilgi:

- Yapılandırılmış hata bağlamı (hata türü, sorgu, kısmi sonuçlar, alternatifler), koordinatörün daha akıllı toparlanmasını sağlar
- Erişim hatalarını (zaman aşımları yeniden deneme kararı gerektirir), geçerli boş sonuçlardan (eşleşme yok) ayırın
- Genel hata durumları (“arama kullanılamıyor”), değerli bağlamı koordinatörden gizler
- Tek bir hata nedeniyle sessizce bastırma veya tüm iş akışını durdurma, ikisi de anti-pattern'lerdir (kötü kalıplardır)

Temel beceriler:

- Yapılandırılmış hata bağlamı döndürün: hata türü, neyin denendiği, kısmi sonuçlar, olası alternatifler

- Erişim hatalarını geçerli boş sonuçlardan ayırın
- Geçici hatalarda alt-ajanlarda yerel toparlanma yapın; yalnızca kurtarılamayan hataları kısmi sonuçlarla birlikte yayın
- Sentezde kapsam notu ekleyin: nelerin iyi desteklendiği ve boşlukların nerede kaldığı

5.4 Büyük Kod Tabanlarını İncelerken Bağlamı Verimli Yönetme

Temel bilgi:

- Uzun oturumlarda bağlam zayıflaması: model, belirli sınıflar yerine “tipik desenlere” atıf yaparak kararsız yanıtlar üretmeye başlar
- Not defteri dosyaları, temel bulguları bağlam sınırları boyunca korur
- Alt-ajanlara delege etmek, ayrıntılı keşif çıktısını yalıtır
- Yapılandırılmış durum kalıcılığı, çökme kurtarma sağlar

Temel beceriler:

- Üst düzey koordinasyonu ana ajanda tutarken belirli sorular için alt-ajanlar başlatın
- Temel bulguları saklamak ve daha sonra başvurmak için not defteri dosyaları kullanın
- Sonraki aşama alt-ajanlarını başlatmadan önce temel bulguları özetleyin
- Uzun incelemeler sırasında bağlam kullanımını azaltmak için `/compact` kullanın

5.5 İnsan Gözetimi ve Güven Kalibrasyonu ile İş Akışları Tasarlama

Temel bilgi:

- Toplu metrikler (ör. 97% genel doğruluk), belirli belge türleri veya alanlardaki zayıf performansı maskeleyebilir
- Katmanlı rastgele örnekleme, yüksek güven skorlu çıkarmalardaki hata oranlarını ölçer
- Etiketli doğrulama kümeleri kullanarak alan düzeyinde güven kalibrasyonu
- Otomasyondan önce doğruluğu belge türüne ve alan segmentine göre doğrulayın

Temel beceriler:

- Yeni hata desenlerini tespit etmek için katmanlı rastgele örnekleme uygulayın
- Kararlı performansı doğrulamak için doğruluğu belge türüne ve alana göre analiz edin
- Alan düzeyinde güven skorları üretin ve etiketli veriler kullanarak inceleme eşiklerini kalibre edin
- Düşük güven skorlu veya kaynağı belirsiz çıkarmaları insan incelemesine yönlendirin

5.6 Çok Kaynaklı Sentezde Köken Takibini Koruma ve Belirsizliği Ele Alma

Temel bilgi:

- “İddia → kaynak” eşlemeleri korunmadığında özetleme sırasında atıf kaybolur
- Yapılandırılmış eşlemeler toplulaştırma sırasında korunmalıdır
- Çelişen istatistikleri, keyfi olarak tek bir değer seçmek yerine atıfla birlikte çelişki notu ekleyerek ele alın
- Zamansal farkları çelişki olarak yanlış okumamak için yayınlanma/toplanma tarihlerini dahil edin

Temel beceriler:

- Alt-ajanların “iddia → kaynak” eşlemeleri (URL, belge adı, alıntılar) üretmesini zorunlu kılın
- Raporları, kararlı bulguları tartışmalı olanlardan ayıracak şekilde yapılandırın
- Çelişen değerleri notlarla koruyun ve uzlaştırma için koordinatöre aktarın
- Doğru zamansal yorumlama için yayın tarihlerini dahil edin
- İçeriği türüne göre sunun: finansal verileri tablo, haberleri düz yazı, teknik bulguları yapılandırılmış liste olarak

---

Açıklamalı Örnek Sınav Soruları

Soru 1 (Senaryo: Müşteri Destek Ajanı)

Durum: Veriler, vakaların 12%'sinde ajanın `get_customer` adımını atladığını ve yalnızca müşterinin adını kullanarak `lookup_order` çağırdığını; bunun da yanlış iadelere yol açtığını gösteriyor.

Hangi değişiklik en etkilidir?

- A) `lookup_order` ve `process_refund` çağrılarını, `get_customer` üzerinden bir ID elde edilene kadar engelleyen programatik bir ön koşul eklemek **[DOĞRU]**
- B) Sistem komutunu iyileştirmek
- C) few-shot örnekleri eklemek
- D) Bir yönlendirme sınıflandırıcısı uygulamak

Neden A: Kritik iş mantığı belirli bir araç sırası gerektirdiğinde yazılım, prompt tabanlı yaklaşımların (B, C) sağlayamayacağı **deterministik garantiler** sunar. D, erişilebilirliği ele alır; araç sıralamasını değil.

Soru 2 (Senaryo: Müşteri Destek Ajanı)

Durum: Ajan, siparişle ilgili sorularda çoğu zaman `get_customer` çağırıyor; oysa `lookup_order` kullanması gerekir. Araç açıklamaları minimal ve birbirine benzer.

İlk adım nedir?

- A) few-shot örnekleri
- B) Her aracın açıklamasını giriş formatları, örnekler ve sınırlarla genişletmek **[DOĞRU]**
- C) Bir yönlendirme katmanı eklemek
- D) Araçları birleştirmek

Neden B: Araç açıklamaları, modelin birincil seçim mekanizmasıdır. Bu, en düşük eforlu ve en yüksek etkili düzeltmedir. A, kök nedeni ele almadan token ekler. C, aşırı mühendisliktir. D, gerekçelendirilenden daha fazla çaba gerektirir.

Soru 3 (Senaryo: Müşteri Destek Ajanı)

Durum: Ajan, hedef 80% iken sorunların yalnızca 55%'ini çözüyor. Basit vakaları insana yönlendiriyor ve karmaşık politika istisnalarını otonom olarak ele almaya çalışıyor.

Kalibrasyonu nasıl iyileştirirsiniz?

- A) few-shot örnekleriyle açık eskalasyon ölçütleri eklemek **[DOĞRU]**
- B) Otomatik eskalasyonla öz puanlanmış güven (1-10)
- C) Geçmiş verilerle eğitilmiş ayrı bir sınıflandırıcı
- D) Duygu analizi

Neden A: Temel nedeni, yani belirsiz karar sınırlarını doğrudan ele alır. B güvenilmezdir (model kendinden emin biçimde yanılabilir). C aşırı mühendisliktir. D farklı bir problemi çözer (duygu durumu != karmaşıklık).

Soru 4 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Depoyu klonladıklarında tüm ekip için kullanılabilir olan, standart kod incelemesi için özel bir `/review` komutuna ihtiyacınız var.

Komut dosyasını nerede oluşturmalısınız?

- A) Proje deposundaki `.claude/commands/` **[DOĞRU]**
- B) `~/.claude/commands/`
- C) Kök `CLAUDE.md`
- D) `.claude/config.json`

Neden A: `.claude/commands/` içinde saklanan proje komutları sürüm kontrollüdür ve herkese otomatik olarak kullanılabilir olur. B kişisel komutlar içindir. C komut tanımları için değil, talimatlar içindir. D mevcut değildir.

Soru 5 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Bir monoliti mikroservislere yeniden yapılandırmanız gerekiyor (düzinelerce dosya, servis sınırı kararları).

Hangi yaklaşımı kullanmalısınız?

- A) Planlama modu: kod tabanını keşfetmek, bağımlılıkları anlamak, bir yaklaşım tasarlamak **[DOĞRU]**
- B) Kademeli olarak doğrudan yürütme
- C) Ayrıntılı ön talimatlarla doğrudan yürütme
- D) Doğrudan yürütme ve zorlaşınca planlamaya geçme

Neden A: Planlama modu büyük değişiklikler, birden çok olası yaklaşım ve mimari kararlar için tasarlanmıştır. B pahalı yeniden çalışmaya yol açabilir. C yapıyı zaten bildiğinizi varsayar. D tepkiseldir.

Soru 6 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Bir kod tabanında farklı alanlar arasında farklı konvansiyonlar var (React, API, veritabanı). Testler kodla aynı yerde duruyor. Konvansiyonların otomatik uygulanmasını istiyorsunuz.

Hangi yaklaşımı kullanmalısınız?

- A) YAML frontmatter ve glob desenleri içeren `.claude/rules/` dosyaları **[DOĞRU]**
- B) Her şeyi kök CLAUDE.md'ye koymak
- C) `.claude/skills/` içinde Skills
- D) Her dizinde CLAUDE.md

Neden A: `.claude/rules/`, glob desenleriyle (ör. `**/*.test.tsx`) dosya yollarına göre otomatik konvansiyon uygulanmasını sağlar; kod tabanının geneline yayılmış testler için idealdir. B model çıkarımına dayanır. C manuel/isteğe bağlıdır. D ilgili dosyalar çok sayıda dizinde olduğunda iyi çalışmaz.

Soru 7 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Sistem “AI'ın yaratıcı sektörlere etkisi” konusunu araştırıyor, ancak raporlar yalnızca görsel sanatı kapsıyor. Koordinatör konuyu şu başlıklara ayırdı: “dijital sanatta AI,” “grafik tasarımda AI,” “fotoğrafçılıkta AI.”

Nedeni nedir?

- A) Sentez ajanı boşlukları saptamıyor
- B) Koordinatör görevi fazla dar ayrıştırdı **[DOĞRU]**
- C) Web arama ajanı yeterince kapsamlı arama yapmıyor
- D) Belge analizi ajanı görsel olmayan kaynakları filtreliyor

Neden B: Günlükler, koordinatörün “yaratıcı sektörler” konusunu yalnızca görsel alt başlıklara ayırdığını; müzik, edebiyat ve filmi tamamen atladığını gösteriyor. Alt-ajan görevleri doğru şekilde yürütüldü; sorun onlara neyin atanmış olduğudur.

Soru 8 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Bir web arama alt-ajanı karmaşık bir konuyu araştırırken zaman aşımına uğruyor. Hata bilgisinin koordinatöre nasıl geri iletileceğini tasarlamanız gerekiyor.

Hangi hata yayılımı yaklaşımı akıllı kurtarmayı en iyi sağlar?

- A) Koordinatöre yapılandırılmış hata bağlamı döndürün: hata türü, sorgu, kısmi sonuçlar ve alternatifler **[DOĞRU]**
- B) Alt-ajan içinde üstel geri çekilmeli otomatik yeniden denemeler uygulayın, sonra genel bir "arama kullanılamıyor" durumu döndürün
- C) Alt-ajan içinde zaman aşımını yakalayın ve başarı olarak işaretlenmiş boş bir sonuç kümesi döndürün
- D) Zaman aşımı istisnasını tüm iş akışını sonlandıran üst düzey bir işleyiciye yayın

Neden A: Yapılandırılmış hata bağlamı, koordinatöre değiştirilmiş bir sorguyla yeniden deneme yapma, alternatif bir yaklaşımı deneme veya kısmi sonuçlarla devam etme kararını verebilmesi için gerekenleri sağlar. B, bağlamı genel bir durumun arkasına gizler. C, hatayı başarı gibi gösterir. D, tüm iş akışını gereksiz yere iptal eder.

Soru 9 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Sentez ajanı, sonuçları birleştirirken çoğu zaman belirli iddiaları doğrulamak zorunda kalıyor. Şu anda doğrulama gerektiğinde sentez ajanı denetimi koordinatöre geri veriyor; koordinatör web arama ajanını çağırıyor ve ardından yeni sonuçlarla sentezi yeniden çalıştırıyor. Bu, görev başına 2–3 ek gidiş dönüş ekliyor ve gecikmeyi 40% artırıyor. Değerlendirmeniz, bu kontrollerin 85% kadarının basit olgu kontrolleri (tarihler, adlar, istatistikler) olduğunu, 15% kadarının ise daha derin inceleme gerektirdiğini gösteriyor.

Güvenilirliği korurken ek yükü nasıl azaltırsınız?

- A) Sentez ajanına basit kontroller için sınırlı bir `verify_fact` aracı verin ve karmaşık doğrulamaları koordinatör üzerinden yönlendirmeye devam edin **[DOĞRU]**
- B) Tüm doğrulama ihtiyaçlarını bir toplu işte biriktirin ve sonunda koordinatöre döndürün
- C) Sentez ajanına tüm web arama araçlarına tam erişim verin
- D) Her kaynak etrafında ek bağlamı proaktif olarak önbelleğe alın

Neden A: Bu, en az ayrıcalık ilkesini uygular: sentez ajanı, 85% düzeyindeki yaygın durum (basit olgu kontrolleri) için tam olarak ihtiyaç duyduğu şeyi alırken karmaşık incelemeler için koordinatör aracılı yol korunur. B, engelleyici bağımlılıklar getirir (sonraki sentez adımları, daha önce doğrulanmış olgulara bağlı olabilir). C, sorumluluk ayrımını bozar. D, ihtiyaçları güvenilir biçimde öngöremeyen spekülatif önbelleğe almaya dayanır.

Soru 10 (Senaryo: CI için Claude Code)

Durum: Bir işlem hattı `claude "Analyze this pull request for security issues"` komutunu çalıştırıyor, ancak etkileşimli girdi beklediği için takılı kalıyor.

Doğru yaklaşım nedir?

- A) `-p` bayrağını kullanın: `claude -p "Analyze this pull request for security issues"` **[DOĞRU]**
- B) `CLAUDE_HEADLESS=true` ayarlayın
- C) stdin'i `/dev/null` üzerinden yönlendirin
- D) `--batch` kullanın

Neden A: `-p` (veya `--print`), Claude Code'u etkileşimsiz modda çalıştırmanın belgelenmiş yoludur. prompt'u işler, stdout'a yazdırır ve çıkar. Diğer seçenekler ya mevcut olmayan özelliklerdir ya da Unix geçici çözümleridir.

Soru 11 (Senaryo: CI için Claude Code)

Durum: Ekip, otomatik analiz için API maliyetini azaltmak istiyor. Claude şu anda iki iş akışına gerçek zamanlı hizmet veriyor: (1) geliştiriciler bir PR'yi birleştirebilmeden önce tamamlanması gereken engelleyici bir merge öncesi kontrol ve (2) sabah incelenmek üzere gece üretilen bir teknik borç raporu. Bir yönetici, 50% tasarruf etmek için ikisini de Message Batches API'ye taşımayı öneriyor.

Bu öneriyi nasıl değerlendirmelisiniz?

- A) Toplu işlemeyi yalnızca teknik borç raporları için kullanın; merge öncesi kontroller için gerçek zamanlı çağrıları koruyun **[DOĞRU]**
- B) Her iki iş akışını da toplu işlemeye taşıyın ve tamamlanmayı yoklayın
- C) Toplu sonuçlardaki sıralama sorunlarından kaçınmak için ikisinde de gerçek zamanlı çağrıları koruyun
- D) Her ikisini de toplu işlemeye taşıyın ve toplu iş çok uzun sürerse gerçek zamana geri dönüş ekleyin

Neden A: Message Batches API 50% tasarruf sağlar, ancak işlem süresi 24 saate kadar çıkabilir ve garantili gecikme SLA'sı yoktur. Bu, geliştiricilerin beklediği engelleyici merge öncesi kontroller için onu uygunsuz kılar; teknik borç raporları gibi gece çalışan toplu iş yükleri için ise idealdir.

Soru 12 (Senaryo: Çok Dosyalı Kod İncelemesi)

Durum: Bir pull request, envanter izleme modülünde 14 dosyayı değiştiriyor. Tüm dosyaların tek geçişli incelenmesi tutarsız sonuçlar üretiyor: bazı dosyalar için ayrıntılı yorumlar, bazıları için yüzeysel yorumlar, gözden kaçan bariz hatalar ve çelişkili geri bildirim (bir kalıp bir dosyada sorunlu olarak işaretlenirken başka bir dosyadaki aynı kod onaylanıyor).

İncelemeyi nasıl yeniden yapılandırmalısınız?

- A) Odaklı geçişlere bölün: her dosyayı yerel sorunlar için ayrı ayrı analiz edin, ardından dosyalar arası veri akışları için ayrı bir entegrasyon geçişi çalıştırın **[DOĞRU]**
- B) Geliştiricilerin büyük PR'leri 3–4 dosyalık gönderimlere bölmesini zorunlu kılın
- C) 14 dosyanın tamamını tek geçişte incelemek için daha büyük bağlam penceresine sahip daha üst seviye bir modele geçin
- D) Üç bağımsız tam PR inceleme geçişi çalıştırın ve yalnızca en az iki çalıştırmada bulunan sorunları raporlayın

Neden A: Odaklı geçişler, aynı anda birçok dosya işlenirken dikkatin dağılması olan kök nedeni doğrudan ele alır. Dosya başına analiz tutarlı derinlik sağlar; ayrı bir entegrasyon geçişi ise dosyalar arası sorunları yakalar. B, sistemi iyileştirmeden yükü geliştiricilere kaydırır. C bir yanılgıdır: daha büyük bağlam, dikkat kalitesini düzeltmez. D, tutarsız tespitler arasında fikir birliği gerektirerek gerçek hataları bastırır.

Deneme Sınavı

> 4 senaryoya yayılmış 60 soru. Format ve zorluk gerçek sınavla aynı düzeydedir.
>
> Alternatif olarak, bu soruları sınav benzeri bir HTML dosyasında çözebilirsiniz: Deneme Sınavı (EN)

Senaryo: Çoklu-ajan Araştırma Sistemi

Soru 1 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Bir belge analizi ajanı, iki güvenilir kaynağın önemli bir metrik için doğrudan çelişen istatistikler içerdiğini keşfediyor: bir hükümet raporu 40% büyüme bildirirken bir sektör analizi 12% belirtiyor. Her iki kaynak da güvenilir görünüyor ve bu tutarsızlık araştırma sonuçlarını maddi olarak etkileyebilir. Belge analizi ajanı bu durumu en etkili şekilde nasıl ele almalıdır?

Hangi yaklaşım en etkilidir?

- A) En olası doğru sayıyı seçmek için güvenilirlik sezgiselleri uygulayın, analizi bu değerle tamamlayın ve tutarsızlıktan bahseden bir dipnot ekleyin.
- B) Her iki sayıyı da çelişkili olarak işaretlemeden analiz çıktısına dahil edin ve sentez ajanının daha geniş bağlama göre hangisini kullanacağına karar vermesine izin verin.
- C) Analizi durdurun ve devam etmeden önce hangi kaynağın daha yetkin olduğuna karar vermesini isteyerek hemen koordinatöre eskale edin.
- D) Analizi her iki sayıyla tamamlayın, çelişkiyi kaynak atfıyla açıkça not edin ve senteze geçirmeden önce verinin nasıl uzlaştırılacağına koordinatörün karar vermesine izin verin. **[DOĞRU]**

Neden D: Bu yaklaşım sorumluluk ayrımını korur: analiz ajanı temel işini engellenmeden tamamlar, açık atıfla her iki çelişkili değeri de korur ve uzlaştırmayı, daha geniş bağlama sahip koordinatöre doğru şekilde devreder.

Soru 2 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Web arama ve belge analizi ajanları görevlerini tamamladı ve sonuçları koordinatöre döndürdü. Entegre bir araştırma raporu oluşturmak için sonraki adım nedir?

En uygun sonraki adım hangisidir?

- A) Her ajan, koordinatörü atlayarak sonuçlarını doğrudan rapor yazma ajanına gönderir.
- B) Belge analizi ajanı web arama sonuçlarını ister ve bunları kendi içinde birleştirir.
- C) Koordinatör, birleşik entegrasyon için her iki sonuç kümesini sentez ajanına iletir. **[DOĞRU]**
- D) Koordinatör, her iki ajanın ham çıktılarını art arda ekler ve nihai sonuç olarak döndürür.

Neden C: Koordinatör-alt ajan mimarisinde koordinatör, merkezi entegrasyon için her iki sonuç kümesini sentez ajanına iletir; bu da denetimi korur ve yüksek kaliteli birleştirme sağlar.

Soru 3 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Bir belge analizi alt-ajanı PDF dosyalarını işlerken sık sık başarısız oluyor: bazılarında ayrıştırma istisnalarını tetikleyen bozuk bölümler var, bazıları parola korumalı ve bazen ayrıştırma kitaplığı büyük dosyalarda takılı kalıyor. Şu anda herhangi bir istisna alt-ajanı hemen sonlandırıyor ve koordinatöre bir hata döndürüyor; koordinatör de yeniden deneme, atlama veya tüm görevi başarısız sayma kararını vermek zorunda kalıyor. Bu, rutin hata yönetiminde koordinatörün aşırı dahil olmasına neden oluyor. En etkili mimari iyileştirme nedir?

Hangi iyileştirme en etkilidir?

- A) Tüm hataları paylaşılan bir kuyruk üzerinden izleyen, kurtarma eylemlerine karar veren ve yeniden başlatma komutlarını doğrudan alt-ajanlara gönderen özel bir hata yönetimi ajanı oluşturun.
- B) Alt-ajanı her zaman başarı durumuyla kısmi sonuçlar döndürecek ve hata ayrıntılarını meta verilere gömecek şekilde yapılandırın; koordinatör tüm yanıtları başarılı kabul eder.
- C) Koordinatörün tüm belgeleri alt-ajana göndermeden önce doğrulamasını sağlayın ve hataya neden olabilecek belgeleri reddedin.
- D) Geçici hatalar için alt-ajan içinde yerel kurtarma uygulayın; yalnızca çözemediği hataları, denenen adımlar ve kısmi sonuçlarla birlikte koordinatöre eskale edin. **[DOĞRU]**

Neden D: Hataları, onları çözebilecek en düşük seviyede ele alın. Yerel kurtarma koordinatörün iş yükünü azaltırken, gerçekten kurtarılamayan sorunları tam bağlam ve kısmi ilerlemeyle eskale etmeye devam eder.

Soru 4 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Sistemi “AI'ın yaratıcı sektörlere etkisi” üzerinde çalıştırdıktan sonra her alt-ajanın başarıyla tamamladığını gözlemliyorsunuz: web arama ajanı ilgili makaleleri buluyor, belge analizi ajanı bunları doğru özetliyor ve sentez ajanı tutarlı metin üretiyor. Ancak nihai raporlar yalnızca görsel sanatı kapsıyor

ve müzik, edebiyat ile filmi tamamen kaçırıyor. Koordinatör günlüklerinde konuyu üç alt göreve ayırdığını görüyorsunuz: “dijital sanatta AI,” “grafik tasarımda AI” ve “fotoğrafçılıkta AI.” En olası kök neden nedir?

En olası kök neden nedir?

- A) Sentez ajanında kapsam boşluklarını saptama talimatları eksik.
- B) Belge analizi ajanı, aşırı katı alaka ölçütleri nedeniyle görsel olmayan kaynakları filtreliyor.
- C) Koordinatörün görev ayrıştırması fazla dar; alt-ajanlara ilgili tüm alanları kapsamayan işler atıyor. **[DOĞRU]**
- D) Web arama ajanının sorguları yetersiz ve daha fazla sektörü kapsayacak şekilde genişletilmeli.

Neden C: Koordinatör, geniş bir konuyu yalnızca görsel sanat alt görevlerine ayırarak müzik, edebiyat ve filmi tamamen kaçırdı. Alt-ajanlar kendilerine verilen işleri doğru yürüttüğünden, dar ayrıştırma açık kök nedendir.

Soru 5 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Web arama alt-ajanı istenen 5 kaynak kategorisinin yalnızca 3'ü için sonuç döndürüyor (rakip siteleri ve sektör raporları başarılı oluyor, ancak haber arşivleri ve sosyal akışlar zaman aşımına uğruyor). Belge analizi alt-ajanı sağlanan tüm belgeleri başarıyla işliyor. Sentez alt-ajanı, karışık kalitedeki üst aşama girdilerden bir özet üretmek zorunda. Hangi hata yayılımı stratejisi en etkilidir?

Hangi hata yayılımı stratejisi en etkilidir?

- A) Yalnızca başarılı kaynakları kullanarak senteze devam edin ve hangi verilerin kullanılamadığını belirtmeden çıktı üretin.
- B) Sentez alt-ajanı koordinatöre hata döndürür; eksik veri nedeniyle tam yeniden deneme veya görev başarısızlığı tetiklenir.
- C) Sentez alt-ajanı, senteze başlamadan önce koordinatörden zaman aşımına uğrayan kaynakları daha uzun zaman aşımıyla yeniden denemesini ister.
- D) Sentez çıktısını, hangi sonuçların iyi desteklendiğini ve kullanılamayan kaynaklar nedeniyle nerelerde boşluk olduğunu belirten kapsam notları ile yapılandırın. **[DOĞRU]**

Neden D: Kapsam notları, şeffaflıkla kontrollü işlev azaltımını uygular; tamamlanan işten elde edilen değeri korurken belirsizliği yayar ve güven konusunda bilinçli kararlar alınmasını sağlar.

Soru 6 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Belge analizi alt-ajanı ayrıştıramadığı bozuk bir PDF dosyasıyla karşılaşıyor. Sistemin hata yönetimi tasarlanırken bu hatayı ele almanın en etkili yolu nedir?

Hangi yaklaşım en etkilidir?

- A) Koordinatör ajanına bağlamla birlikte hata döndürerek nasıl ilerleyeceğine karar vermesini sağlayın. **[DOĞRU]**

- B) İş akışını kesintiye uğratmamak için bozuk belgeyi sessizce atlayın ve kalan dosyaları işlemeye devam edin.
- C) Hata bildirmeden önce belgeyi üstel geri çekilmeyle üç kez otomatik olarak yeniden ayrıştırmayı deneyin.
- D) Tüm araştırma iş akışını sonlandıran bir istisna fırlatın.

Neden A: Koordinatöre bağlamla hata döndürmek en etkili yaklaşımdır; çünkü koordinatörün dosyayı atlama, alternatif bir ayrıştırma yöntemi deneme veya kullanıcıyı bilgilendirme gibi bilinçli kararlar vermesini sağlarken hatanın görünürlüğünü korur.

Soru 7 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Üretim günlükleri kalıcı bir örüntü gösteriyor: “yüklenen çeyrek dönem raporunu analiz et” gibi istekler, zamanın 45% kadarında belge analizi ajanı yerine web arama ajanına yönlendiriliyor. Araç tanımlarını incelediğinizde web arama ajanında “içeriği analiz eder ve temel bilgileri çıkarır” diye açıklanan `analyze_content` adlı bir araç olduğunu, belge analizi ajanında ise “belgeleri analiz eder ve temel bilgileri çıkarır” diye açıklanan `analyze_document` adlı bir araç olduğunu görüyorsunuz. Yanlış yönlendirme sorununu nasıl düzeltmelisiniz?

Yanlış yönlendirme sorununu nasıl düzeltmelisiniz?

- A) Koordinatör yetkilendirmeye karar vermeden önce kullanıcının yüklenen dosyalardan mı yoksa web içeriğinden mi bahsettiğini saptayan bir ön yönlendirme sınıflandırıcısı ekleyin.
- B) Web arama aracını `extract_web_results` olarak yeniden adlandırın ve açıklamasını “web araması ve URL'lerden alınan bilgileri işler ve döndürür” şeklinde güncelleyin. **[DOĞRU]**
- C) Koordinatör prompt'una doğru yönlendirmeyi gösteren few-shot örnekler ekleyin: “Kullanıcı çeyrek dönem raporu yükler → belge analizi ajanı” ve “Kullanıcı bir web sayfasını sorar → web arama ajanı.”
- D) Belge analizi aracı açıklamasını “Yüklenen PDF'ler, Word belgeleri ve elektronik tablolar için kullanın” gibi kullanım örnekleriyle genişletin, web arama aracını değiştirmeden bırakın.

Neden B: Web arama aracını `extract_web_results` olarak yeniden adlandırmak ve açıklamasını açıkça web araması ve URL'lere referans verecek şekilde güncellemek, iki araç adı ve açıklaması arasındaki anlamsal örtüşmeyi ortadan kaldırarak kök nedeni doğrudan giderir. Bu, her aracın amacını belirsizliğe yer bırakmayacak hale getirir ve koordinatörün belge analizini web aramasından güvenilir biçimde ayırt etmesini sağlar.

Soru 8 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Bir iş arkadaşınız, belge analizi ajanının sonuçlarını koordinatörü atlayarak doğrudan sentez ajanına göndermesi gerektiğini öneriyor. Koordinatörü alt-ajanlar arasındaki tüm iletişim için merkezi düğüm olarak tutmanın temel avantajı nedir?

Koordinatörü merkezi düğüm olarak tutmanın temel avantajı nedir?

- A) Koordinatör tüm etkileşimleri gözlemleyebilir, hataları tek biçimde ele alabilir ve her alt-ajanın hangi bilgileri alması gerektiğine karar verebilir. **[DOĞRU]**

- B) Koordinatör, alt-ajanlara yapılan birden çok isteği toplu hale getirerek toplam API çağrılarını ve genel gecikmeyi azaltır.
- C) Koordinatör üzerinden yönlendirme, doğrudan ajanlar arası çağrıların destekleyemeyeceği otomatik yeniden deneme mantığını mümkün kılar.
- D) Alt-ajanlar yalıtılmış bellek kullanır ve doğrudan iletişim yalnızca koordinatörün gerçekleştirebileceği karmaşık serileştirme gerektirir.

Neden A: Koordinatör kalıbı tüm etkileşimlere merkezi görünürlük, sistem genelinde tek biçimli hata yönetimi ve her alt-ajanın hangi bilgileri alacağı üzerinde ince taneli denetim sağlar; bunlar yıldız biçimli iletişim topolojisinin başlıca avantajlarıdır.

Soru 9 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Web arama alt-ajanı karmaşık bir konuyu araştırırken zaman aşımına uğruyor. Bu hata hakkındaki bilginin koordinatöre nasıl döndürüleceğini tasarlamanız gerekiyor. Hangi hata yayılımı yaklaşımı akıllı kurtarmayı en iyi sağlar?

Hangi hata yayılımı yaklaşımı akıllı kurtarmayı en iyi sağlar?

- A) Hata türünü, yürütülen sorguyu, varsa kısmi sonuçları ve olası alternatif yaklaşımları içeren yapılandırılmış hata bağlamını koordinatöre döndürün. **[DOĞRU]**
- B) Alt-ajan içinde zaman aşımını yakalayın ve başarılı olarak işaretlenmiş boş bir sonuç kümesi döndürün.
- C) Alt-ajan içinde otomatik üstel geri çekilmeli yeniden denemeler uygulayın; yalnızca yeniden denemeler tükendikten sonra genel bir “arama kullanılamıyor” durumu döndürün.
- D) Zaman aşımı istisnasını doğrudan üst düzey işleyiciye yayın ve tüm araştırma iş akışını sonlandırın.

Neden A: Hata türü, yürütülen sorgu, kısmi sonuçlar ve alternatif yaklaşımlar dahil yapılandırılmış hata bağlamını döndürmek, koordinatöre akıllı kurtarma kararları vermesi için gereken her şeyi sağlar (ör. değiştirilmiş bir sorguyla yeniden denemek veya kısmi sonuçlarla devam etmek). Koordinasyon düzeyinde bilinçli karar verme için azami bağlamı korur.

Soru 10 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Sistem tasarımınızda, belge analizi ajanına URL üzerinden belge indirebilmesi için genel amaçlı `fetch_url` aracına erişim verdiniz. Üretim günlükleri, bu ajanın artık sık sık arama motoru sonuç sayfalarını indirerek plansız web araması yaptığını gösteriyor; bu davranış web arama ajanı üzerinden yönlendirilmeliydi ve tutarsız sonuçlara neden oluyor. Hangi düzeltme en etkilidir?

Hangi düzeltme en etkilidir?

- A) `fetch_url` yerine, URL'lerin belge biçimlerine işaret ettiğini doğrulayan bir `load_document` aracı koyun. **[DOĞRU]**
- B) Belge analizi ajanından `fetch_url` aracını kaldırın ve tüm URL getirme işlemlerini koordinatör üzerinden web arama ajanına yönlendirin.

- C) Bilinen arama motoru alan adlarına yapılan `fetch_url` çağrılarını engelleyen, diğer URL'lere izin veren filtreleme uygulayın.
- D) Belge analizi ajanı prompt'una `fetch_url` aracının arama yapmak için değil, yalnızca belge URL'lerini indirmek için kullanılması gerektiğine dair talimat ekleyin.

Neden A: Genel amaçlı bir aracı, URL'leri belge biçimlerine göre doğrulayan belgeye özgü bir araçla değiştirmek, yeteneği arayüz düzeyinde kısıtlayarak kök nedeni çözer. Bu, en az ayrıcalık ilkesini izler ve istenmeyen arama davranışını yalnızca caydırmak yerine imkansız hale getirir.

Soru 11 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Geniş bir konuyu araştırırken web arama ajanı ile belge analizi ajanının aynı alt konuları incelediğini ve çıktılarında ciddi tekrar oluştuğunu gözlemliyorsunuz. Token kullanımı, araştırma genişliği veya derinliğinde orantılı bir artış olmadan neredeyse ikiye katlanıyor. Bunu ele almanın en etkili yolu nedir?

Bunu ele almanın en etkili yolu nedir?

- A) Her iki ajanın paralel bitirmesine izin verin, ardından koordinatör örtüşen sonuçları sentez ajanına iletmeden önce tekilleştirsin.
- B) Koordinatör, yetkilendirmeden önce araştırma alanını açıkça bölümlendirerek her ajana ayrı alt konular veya kaynak türleri atar. **[DOĞRU]**
- C) Ajanların mevcut odak alanlarını günlüğe yazdığı, böylece diğer ajanların yürütme sırasında tekrardan dinamik olarak kaçınabileceği bir paylaşılan durum mekanizması uygulayın.
- D) Belge analizinin yalnızca web araması tamamlandıktan sonra çalıştığı, tekrardan kaçınmak için web arama sonuçlarını bağlam olarak kullanan sıralı yürütmeye geçin.

Neden B: Koordinatörün yetkilendirmeden önce araştırma alanını açıkça bölümlendirmesi en etkilidir; çünkü kök neden olan belirsiz görev sınırlarını herhangi bir iş başlamadan ele alır. Paralelliği korurken yinelenen çabayı ve boşa harcanan token'ları önler.

Soru 12 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Araştırma sırasında web arama alt-ajanı üç kaynak kategorisini farklı sonuçlarla sorguluyor: akademik veri tabanları 15 ilgili makale döndürüyor, sektör raporları “0 sonuç” döndürüyor ve patent veri tabanları “Bağlantı zaman aşımı” döndürüyor. Koordinatöre hata yayılımı tasarlanırken hangi yaklaşım en iyi kurtarma kararlarını mümkün kılar?

Hangi yaklaşım en iyi kurtarma kararlarını mümkün kılar?

- A) Sonuçları tek bir başarı yüzdesi metriğinde birleştirin (ör. “67% kaynak kapsamı”) ve ayrıntılı günlükleri isteğe bağlı kullanılabilir hale getirin.
- B) Hem “zaman aşımı” hem de “0 sonuç” durumlarını koordinatör müdahalesi gerektiren hatalar olarak raporlayın.
- C) Geçici hataları içeride yeniden deneyin ve yalnızca kalıcı hataları raporlayın.

- D) Yeniden deneme kararı gerektiren erişim hatalarını (zaman aşımı), başarılı sorguları temsil eden geçerli boş sonuçlardan (“0 sonuç”) ayırt edin. **[DOĞRU]**

Neden D: Zaman aşımı (erişim hatası) ile “0 sonuç” (geçerli boş sonuç), farklı yanıtlar gerektiren anlamsal olarak farklı sonuçlardır. Bunları ayırt etmek, koordinatörün patent veri tabanını yeniden denemesine, sektör raporlarındaki “0 sonuç” bulgusunu ise geçerli ve bilgilendirici bir bulgu olarak kabul etmesine olanak tanır.

Soru 13 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Üretim izleme, tutarsız sentez kalitesi gösteriyor. Birleştirilmiş sonuçlar yaklaşık 75K token olduğunda, sentez ajanı ilk 15K token'daki bilgileri (web arama başlıkları/parçacıkları) ve son 10K token'daki bilgileri (belge analizi sonuçları) güvenilir biçimde alıntılıyor; ancak araştırma sorusunu doğrudan yanıtlasalar bile ortadaki 50K token içindeki kritik bulguları çoğu zaman kaçırıyor. Birleştirilmiş girdiyi nasıl yeniden yapılandırmalısınız?

Birleştirilmiş girdiyi nasıl yeniden yapılandırmalısınız?

- A) İçeriği modelin güvenilir işleme aralığında tutmak için tüm alt-ajan çıktılarını birleştirmeden önce 20K token'ın altına özetleyin.
- B) Alt-ajan sonuçlarını sentez ajanına kademeli olarak akıtın; önce web arama sonuçlarını tamamlanana kadar işleyin, sonra belge analizi sonuçlarını ekleyin.
- C) Birleştirilmiş girdinin başına temel bulgular özetini yerleştirin ve ayrıntılı sonuçları daha kolay gezinme için açık bölüm başlıklarıyla düzenleyin. **[DOĞRU]**
- D) Zaman içinde her iki kaynağın da eşit üst konum elde etmesini sağlamak için araştırma görevleri arasında hangi alt-ajan sonuçlarının önce görüneceğini dönüşümlü hale getirin.

Neden C: Temel bulgular özetini başa koymak, kritik bilginin en güvenilir işlenen konumda durması için öncelik etkilerinden yararlanır. Baştan sona açık bölüm başlıkları eklemek, modelin girdinin orta bölümündeki içeriğe gitmesine ve dikkat etmesine yardımcı olarak “ortada kaybolma” olgusunu doğrudan azaltır.

Soru 14 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Testte, web arama ajanının çıktısı (sayfa içeriği dahil 85K token) ile belge analizi ajanının çıktısı (düşünce zincirleri dahil 70K token) toplam 155K token'a ulaşıyor; ancak sentez ajanı en iyi performansı 50K token'ın altındaki girdilerle gösteriyor. Hangi çözüm en etkilidir?

Hangi çözüm en etkilidir?

- A) Üst aşama ajanları, ayrıntılı içerik ve akıl yürütme yerine yapılandırılmış veri (temel olgular, alıntılar, alaka puanları) döndürecek şekilde değiştirin. **[DOĞRU]**
- B) Bulguları senteze aktarmadan önce yoğunlaştıran bir ara özetleme ajanı ekleyin.
- C) Sentez ajanının bulguları sıralı toplu işler halinde işlemesini ve çağrılar arasında durumu korumasını sağlayın.

- D) Bulguları bir vektör veri tabanında saklayın ve sentez ajanına çalışması sırasında sorgulama yapması için arama araçları verin.

Neden A: Üst aşama ajanları yapılandırılmış veri döndürecek şekilde değiştirmek, temel bilgiyi korurken token hacmini kaynağında azaltarak kök nedeni çözer. Sentez adımını iyileştirmeden token'ları şişiren hacimli sayfa içeriğini ve akıl yürütme izlerini aktarmaktan kaçınır.

Soru 15 (Senaryo: Çoklu-ajan Araştırma Sistemi)

Durum: Testte, sentez ajanının sonuçları birleştirirken sık sık belirli iddiaları doğrulaması gerektiğini gözlemliyorsunuz. Şu anda doğrulama gerektiğinde sentez ajanı denetimi koordinatöre geri veriyor; koordinatör web arama ajanını çağırıyor ve ardından sonuçlarla sentezi yeniden çağırıyor. Bu, görev başına 2–3 ek döngü ekliyor ve gecikmeyi 40% artırıyor. Değerlendirmeniz, bu doğrulamaların 85% kadarının basit olgu kontrolleri (tarihler, adlar, istatistikler) olduğunu ve 15% kadarının daha derin araştırma gerektirdiğini gösteriyor. Hangi yaklaşım sistem güvenilirliğini korurken ek yükü en etkili şekilde azaltır?

Hangi yaklaşım en etkilidir?

- A) Sentez ajanına tüm web arama araçlarına erişim verin, böylece koordinatör döngüleri olmadan her türlü doğrulama ihtiyacını doğrudan ele alabilsin.
- B) Sentez ajanı tüm doğrulama ihtiyaçlarını biriktirsin ve sonunda bunları toplu olarak koordinatöre döndürsün; koordinatör de hepsini tek seferde web arama ajanına göndersin.
- C) Web arama ajanı, sentezin doğrulamaya ihtiyaç duyabileceğini öngörerek ilk araştırma sırasında her kaynak etrafında ek bağlamı proaktif olarak önbelleğe alsın.
- D) Sentez ajanına basit kontroller için sınırlı kapsamlı bir `verify_fact` aracı verin; karmaşık doğrulamaları ise koordinatör üzerinden web arama ajanına yönlendirin. **[DOĞRU]**

Neden D: Sınırlı kapsamlı bir olgu doğrulama aracı, sentez ajanının basit kontrollerin 85% kadarını doğrudan ele almasını sağlar ve çoğu döngüyü ortadan kaldırır; karmaşık doğrulamaların 15% kadarı için ise koordinatör yetkilendirme yolunu korur. Bu, gecikmeyi önemli ölçüde azaltırken en az ayrıcalık ilkesini uygular.

Senaryo: Sürekli Entegrasyon için Claude Code

Soru 16 (Senaryo: Sürekli Entegrasyon için Claude Code)

Durum: CI işlem hattınız, kod incelemesi için proje bağlamı sağlamak üzere CLAUDE.md kullanarak Claude Code CLI'ı (`--print` modunda) çalıştırıyor ve geliştiriciler genellikle incelemeleri anlamlı buluyor. Ancak bulguları iş akışına entegre etmenin zor olduğunu bildiriyorlar: Claude, PR yorumlarına elle kopyalanması gereken anlatı biçiminde paragraflar üretiyor. Ekip, her bulguyu koddaki ilgili yere ayrı bir satır içi PR yorumu olarak otomatik göndermek istiyor; bunun için dosya yolu, satır numarası, önem düzeyi ve önerilen düzeltme içeren yapılandırılmış veri gerekiyor. Hangi yaklaşım en etkilidir?

Hangi yaklaşım en etkilidir?

- A) CLAUDE.md'ye yapılandırılmış bulgu örnekleri içeren bir “İnceleme için Çıktı Formatı” bölümü ekleyin; böylece Claude beklenen formatı proje bağlamından öğrenir.
- B) Yapılandırılmış bulguları zorunlu kılmak için `--output-format json` ve `--json-schema` CLI bayraklarını kullanın, ardından çıktıyı ayrıştırarak GitHub API aracılığıyla satır içi yorumlar gönderin. **[DOĞRU]**
- C) İnceleme prompt'una her bulgunun `[FILE:path] [LINE:n] [SEVERITY:level] ...` gibi ayrıştırılabilir bir şablonu izlemesini gerektiren açık biçimlendirme talimatları ekleyin.
- D) Anlatı biçimindeki inceleme formatını koruyun, ancak bulguların yapılandırılmış JSON özetini üretmek için Claude'u kullanan bir özetleme adımı ekleyin.

Neden B: `--output-format json` ile `--json-schema` kullanmak, CLI düzeyinde yapılandırılmış çıktıyı zorunlu kılar; dosya yolu, satır numarası, önem düzeyi ve önerilen düzeltme gibi gerekli alanlara sahip iyi biçimlendirilmiş JSON'un güvenilir şekilde ayrıştırılmasını ve GitHub API aracılığıyla satır içi PR yorumları olarak gönderilmesini sağlar. Bu, özellikle yapılandırılmış çıktı için tasarlanmış yerleşik CLI yeteneklerinden yararlanır.

Soru 17 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Ekibiniz kod önerileri üretmek için Claude Code kullanıyor, ancak bir örüntü fark ediyorsunuz: uç durumları bozan performans optimizasyonları, davranışı beklenmedik şekilde değiştiren temizlikler gibi bariz olmayan sorunlar ancak başka bir ekip üyesi PR'ı incelediğinde yakalanıyor. Claude'un üretim sırasındaki akıl yürütmesi bu durumları değerlendirdiğini, ancak yaklaşımının doğru olduğu sonucuna vardığını gösteriyor. Bu öz-denetim sınırlamasının temel nedenini doğrudan hangi yaklaşım ele alır?

Temel nedeni doğrudan hangi yaklaşım ele alır?

- A) Değişiklikleri, üreticinin akıl yürütmesine erişimi olmayan ikinci bağımsız bir Claude Code örneğine inceletmek. **[DOĞRU]**
- B) Önerileri üretmeden önce daha kapsamlı muhakeme yapılabilmesi için üretim aşamasında genişletilmiş düşünme modunu etkinleştirmek.
- C) Üretim prompt'una, çıktı sonlandırılmadan önce Claude'dan kendi önerilerini eleştirmesini isteyen açık öz-inceleme talimatları eklemek.
- D) Claude'un üretim sırasında beklenen davranışı daha iyi anlaması için tam test dosyalarını ve dokümantasyonu prompt bağlamına dahil etmek.

Neden A: Üreticinin akıl yürütmesine erişimi olmayan ikinci bağımsız bir Claude Code örneği, onaylama yanlılığından kaçınarak temel nedeni doğrudan ele alır. Bu bağımsız bakış, başka bir inceleyenin yazarın gerekçelendirdiği sorunları yakaladığı insan akran incelemesine benzer.

Soru 18 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Kod inceleme bileşeniniz yinelemelidir: Claude değiştirilen dosyayı analiz eder, ardından nihai geri bildirimi vermeden önce bağlamı anlamak için araç çağrılarıyla ilgili dosyaları (import'lar, temel sınıflar,

testler) isteyebilir. Uygulamanız, Claude'un dosya içeriklerini istemesini sağlayan bir araç tanımlar; Claude aracı çağırır, sonuçları alır ve analize devam eder. API maliyetini düşürmek için toplu işlemeyi değerlendiriyorsunuz. Bu iş akışı için toplu işlemeyi düşünürken temel teknik sınırlama nedir?

Temel teknik sınırlama nedir?

- A) Toplu işleme, çıktıları girdi istekleriyle eşleştirmek için korelasyon ID'leri içermez.
- B) Asenkron model, istek ortasında araçları çalıştırıp Claude'un analize devam etmesi için sonuçları geri döndüremez. **[DOĞRU]**
- C) Batch API, istek parametrelerinde araç tanımlarını desteklemez.
- D) 24 saate kadar çıkabilen toplu işleme gecikmesi pull request geri bildirimi için çok yavaştır, oysa iş akışı bunun dışında çalışırdı.

Neden B: “Gönder ve unut” tarzı asenkron Batch API modelinde, bir istek sırasında araç çağrısını yakalayacak, aracı çalıştıracak ve Claude'un analize devam etmesi için sonuçları geri döndürecek bir mekanizma yoktur. Bu, tek bir mantıksal etkileşim içinde birden fazla araç istek/yanıt turu gerektiren yinelemeli araç çağırma iş akışlarıyla temelden uyumsuzdur.

Soru 19 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: CI/CD sisteminiz Claude tabanlı üç analiz çalıştırıyor: (1) her PR'da tamamlanana kadar merge'ü engelleyen hızlı stil kontrolleri, (2) tüm kod tabanının kapsamlı haftalık güvenlik denetimleri ve (3) yakın zamanda değişen modüller için gecelik test senaryosu üretimi. Message Batches API 50% tasarruf sunuyor, ancak işlem 24 saate kadar sürebiliyor. Kabul edilebilir bir geliştirici deneyimini korurken API maliyetini optimize etmek istiyorsunuz. Hangi kombinasyon her görevi doğru API yaklaşımıyla eşleştirir?

Hangi kombinasyon doğrudur?

- A) 50% tasarrufu en üst düzeye çıkarmak için üç görevin tamamında Message Batches API kullanmak ve işlem hattını toplu işlemin tamamlanmasını yoklayacak şekilde yapılandırmak.
- B) PR stil kontrolleri için senkron çağrılar kullanmak; haftalık güvenlik denetimleri ve gecelik test üretimi için Message Batches API kullanmak. **[DOĞRU]**
- C) Tutarlı yanıt süreleri için üç görevin tamamında senkron çağrılar kullanmak ve iş yükleri genelinde maliyetleri düşürmek için prompt önbelleklemesine güvenmek.
- D) PR stil kontrolleri ve gecelik test üretimi için senkron çağrılar kullanmak; Message Batches API'yi yalnızca haftalık güvenlik denetimleri için kullanmak.

Neden B: PR stil kontrolleri geliştiricileri bloklar ve senkron çağrılarla anında yanıt gerektirir; haftalık güvenlik denetimleri ve gecelik test üretimi ise esnek son tarihlere sahip, 24 saate kadar toplu işlem penceresini tolere edebilen zamanlanmış görevlerdir; böylece ikisi için de 50% tasarruf elde edilir.

Soru 20 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Otomatik incelemeleriniz gerçek sorunlar buluyor, ancak geliştiriciler geri bildirimin eyleme dönüştürülebilir olmadığını bildiriyor. Bulgular, tam olarak neyin değiştirileceğini belirtmeden “karmaşık ticket yönlendirme mantığı” veya “potansiyel null pointer” gibi ifadeler içeriyor. “Her zaman somut düzeltme

önerileri ekle” gibi ayrıntılı talimatlar eklediğinizde bile model tutarsız çıktı üretiyor; bazen ayrıntılı, bazen belirsiz. Hangi prompt tekniği tutarlı şekilde eyleme dönüştürülebilir geri bildirimi en güvenilir biçimde üretir?

Hangi prompt tekniği en güvenilirdir?

- A) Geri bildirim formatının her bölümü için (konum, sorun, önem derecesi, önerilen düzeltme) daha açık gereksinimlerle talimatları daha da iyileştirmek.
- B) Modelin somut düzeltmeler önermek için yeterli bilgiye sahip olması amacıyla daha geniş çevre kod tabanını içerecek şekilde bağlam penceresini genişletmek.
- C) Bir prompt'un sorunları belirlediği, ikincisinin düzeltmeleri ürettiği iki geçişli bir yaklaşım uygulayarak uzmanlaşmaya olanak tanımak.
- D) Gerekli tam formatı gösteren 3–4 few-shot örnek eklemek: belirlenen sorun, koddaki konum, somut düzeltme önerisi. **[DOĞRU]**

Neden D: Tek başına talimatlar değişken sonuç verdiğinde, tutarlı çıktı formatı elde etmek için en etkili teknik few-shot örneklerdir. İstenen yapıyı (sorun, konum, somut düzeltme) bire bir gösteren 3–4 örnek sağlamak, modele izleyeceği somut bir örüntü verir; bu, soyut talimatlardan daha güvenilirdir.

Soru 21 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: CI işlem hattınız iki Claude tabanlı kod inceleme modu içeriyor: tamamlanana kadar PR merge'ünü engelleyen bir pre-merge-commit hook'u ve gece çalışan, toplu işlemin tamamlanmasını yoklayan ve PR'a ayrıntılı öneriler gönderen bir “derin analiz”. 50% tasarruf sunan ancak yoklama gerektiren ve 24 saate kadar sürebilen Message Batches API'yi kullanarak API maliyetini düşürmek istiyorsunuz. Hangi mod toplu işlemeyi kullanmalıdır?

Hangi mod toplu işlemeyi kullanmalıdır?

- A) Yalnızca pre-merge-commit hook'u.
- B) Yalnızca derin analiz. **[DOĞRU]**
- C) Her iki mod.
- D) Hiçbiri.

Neden B: Derin analiz, zaten gece çalıştığı, gecikmeyi tolere ettiği ve sonuçları yayımlamadan önce yoklama modelini kullandığı için toplu işleme açısından ideal bir adaydır; bu, 50% tasarruf sağlarken Message Batches API'nin asenkron, yoklama tabanlı mimarisiyle uyumludur.

Soru 22 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Otomatik incelemeniz yorumları ve docstring'leri analiz ediyor. Mevcut prompt, Claude'a “yorumların doğru ve güncel olup olmadığını kontrol et” talimatını veriyor. Bulgular çoğu zaman kabul edilebilir örüntüleri (TODO işaretleri, basit açıklamalar) işaretlerken, kodun artık uygulamadığı davranışı açıklayan yorumları kaçırıyor. Bu tutarsız analizin temel nedenini hangi değişiklik ele alır?

Hangi değişiklik temel nedeni ele alır?

- A) Claude'un yakın tarihli kod değişikliklerinden eski yorumları belirleyebilmesi için `git blame` verilerini dahil etmek.
- B) Modelin kod tabanındaki benzer örüntüleri tanımasına yardımcı olmak için yanıltıcı yorumlara dair few-shot örnekler eklemek.
- C) Gürültüyü azaltmak için analizden önce TODO, FIXME ve açıklayıcı yorum örüntülerini filtrelemek.
- D) Açık ölçütler belirtmek: yorumları yalnızca iddia ettikleri davranış kodun gerçek davranışıyla çeliştiğinde işaretlemek. **[DOĞRU]**

Neden D: Açık ölçütler, yani yorumları yalnızca iddia edilen davranış gerçek kod davranışıyla çeliştiğinde işaretlemek, belirsiz bir talimatı neyin sorun sayılacağına dair kesin bir tanımla değiştirerek temel nedeni doğrudan ele alır. Bu, kabul edilebilir örüntülerde yanlış pozitifleri ve gerçekten yanıltıcı yorumların kaçırılmasını azaltır.

Soru 23 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Otomatik kod inceleme sisteminiz tutarsız önem derecesi puanları gösteriyor; null pointer riskleri gibi benzer sorunlar bazı PR'larda “kritik”, bazılarında ise yalnızca “orta” olarak derecelendiriliyor. Geliştirici anketleri güvenin azaldığını gösteriyor; birçok kişi “yarısı yanlış” olduğu için bulguları okumadan yok saymaya başlıyor. Yüksek yanlış-pozitif kategoriler, doğru kategorilere duyulan güveni aşındırıyor. Hangi yaklaşım sistemi iyileştirirken geliştirici güvenini en iyi şekilde yeniden sağlar?

Hangi yaklaşım geliştirici güvenini en iyi şekilde yeniden sağlar?

- A) Yüksek yanlış-pozitif kategorileri (stil, adlandırma, dokümantasyon) geçici olarak devre dışı bırakmak ve prompt'ları iyileştirirken yalnızca yüksek kesinlikli kategorileri tutmak. **[DOĞRU]**
- B) Tüm kategorileri etkin tutmak, ancak geliştiricilerin neyi inceleyeceklerine karar verebilmesi için her bulguyla birlikte güven skorları göstermek.
- C) Tüm kategorileri etkin tutmak ve sonraki birkaç hafta içinde her kategori için doğruluğu artırmak üzere few-shot örnekler eklemek.
- D) Genel yanlış-pozitif oranını düşürmek için tüm kategorilere tek tip bir katılık azaltımı uygulamak.

Neden A: Yüksek yanlış-pozitif kategorileri geçici olarak devre dışı bırakmak, geliştiricilerin her şeyi yok saymasına yol açan gürültülü bulguları kaldırarak güven aşınmasını hemen durdurur; güvenlik ve doğruluk gibi yüksek kesinlikli kategorilerden gelen değeri de korur. Ayrıca, sorunlu kategorileri yeniden etkinleştirmeden önce prompt'ları iyileştirmek için alan açar.

Soru 24 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Otomatik incelemeniz her PR için test senaryosu önerileri üretiyor. Ders tamamlama takibi ekleyen bir PR'ı incelerken Claude 10 test senaryosu öneriyor, ancak geliştirici geri bildirimi bunlardan 6'sının mevcut test paketinin zaten kapsadığı yinelenen senaryolar olduğunu gösteriyor. Hangi değişiklik yinelenen önerileri en etkili şekilde azaltır?

Hangi değişiklik en etkilidir?

- A) Claude'un hangi senaryoların zaten kapsandığını belirleyebilmesi için mevcut test dosyasını bağlama dahil etmek. **[DOĞRU]**
- B) Claude'un önce en değerli durumlara öncelik verdiğini varsayarak istenen öneri sayısını 10'dan 5'e düşürmek.
- C) Claude'u başarı yolları yerine yalnızca uç durumlara ve hata koşullarına odaklanmaya yönlendiren talimatlar eklemek.
- D) Açıklamaları anahtar kelime örtüşmesi yoluyla mevcut test adlarıyla eşleşen önerileri filtreleyen son işlem uygulamak.

Neden A: Mevcut test dosyasını dahil etmek, yinelemenin temel nedenini düzeltir: Claude, zaten hangi testlerin var olduğunu bilirse halihazırda kapsanan senaryoları önermemeyi başarabilir. Bu, Claude'a gerçekten yeni ve değerli testler önermek için gereken bilgiyi verir.

Soru 25 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: İlk otomatik inceleme 12 bulgu belirledikten sonra bir geliştirici sorunları gidermek için yeni commit'ler push ediyor. İncelemeyi yeniden çalıştırmak 8 bulgu üretiyor, ancak geliştiriciler bunların 5'inin yeni commit'lerde zaten düzeltilmiş koda ilişkin önceki yorumların kopyası olduğunu bildiriyor. Kapsamlılığı korurken bu gereksiz geri bildirimi ortadan kaldırmanın en etkili yolu nedir?

Gereksiz geri bildirimi ortadan kaldırmanın en etkili yolu nedir?

- A) İncelemeyi yalnızca PR oluşturulduğunda ve merge öncesi son durumda çalıştırıp ara commit'leri atlamak.
- B) Yorum göndermeden önce dosya yolları ve sorun açıklamaları bakımından önceki bulgularla eşleşen bulguları kaldıran bir son işlem filtresi eklemek.
- C) İnceleme kapsamını en son push'ta değişen dosyalarla sınırlayıp önceki commit'lerdeki dosyaları hariç tutmak.
- D) Önceki inceleme bulgularını bağlama dahil etmek ve Claude'a yalnızca yeni veya hâlâ çözülmemiş sorunları raporlamasını söylemek. **[DOĞRU]**

Neden D: Önceki inceleme bulgularını bağlama dahil etmek, Claude'un yeni problemler ile yakın tarihli commit'lerde zaten ele alınmış problemleri ayırt etmesini sağlar. Bu, Claude'un akıl yürütmesini kullanarak düzeltilmiş kod hakkında gereksiz geri bildirimden kaçınırken inceleme kapsamlılığını korur.

Soru 26 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: İşlem hattı betiğiniz `claude "Analyze this pull request for security issues"` komutunu çalıştırıyor, ancak iş süresiz olarak takılıyor. Log'lar Claude Code'un etkileşimli girdi beklediğini gösteriyor. Claude Code'u otomatik bir işlem hattında çalıştırmak için doğru yaklaşım nedir?

Doğru yaklaşım nedir?

- A) Bir `--batch` bayrağı eklemek: `claude --batch "Analyze this pull request for security issues"`.

- B) `-p` bayrağını eklemek: `claude -p "Analyze this pull request for security issues"`. **[DOĞRU]**
- C) stdin'i `/dev/null` üzerinden yönlendirmek: `claude "Analyze this pull request for security issues" < /dev/null`.
- D) Komutu çalıştırmadan önce `CLAUDE_HEADLESS=true` ortam değişkenini ayarlamak.

Neden B: `-p` (veya `--print`) bayrağı Claude Code'u etkileşimsiz çalıştırmanın dokümante edilmiş yoludur. Prompt'u işler, sonucu stdout'a yazdırır ve kullanıcı girdisi beklemeden çıkar; bu, CI/CD işlem hatları için idealdir.

Soru 27 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Bir pull request, envanter takip modülündeki 14 dosyayı değiştiriyor. Tüm dosyaları birlikte analiz eden tek geçişli bir inceleme tutarsız sonuçlar üretiyor: bazı dosyalarda ayrıntılı geri bildirim, bazılarında yüzeysel yorumlar, kaçırılan bariz hatalar ve çelişkili geri bildirimler (bir dosyada işaretlenen bir örüntünün aynı PR'daki başka bir dosyada onaylanması). İncelemeyi nasıl yeniden yapılandırmalısınız?

İncelemeyi nasıl yeniden yapılandırmalısınız?

- A) Üç bağımsız tam-PR inceleme geçişi çalıştırmak ve yalnızca üç çalıştırmanın en az ikisinde görünen sorunları işaretlemek.
- B) Odaklı geçişlere bölmek: yerel sorunlar için her dosyayı ayrı ayrı incelemek, ardından dosyalar arası veri akışlarını değerlendirmek için ayrı bir entegrasyon odaklı geçiş çalıştırmak. **[DOĞRU]**
- C) Geliştiricilerin otomatik inceleme çalıştırmadan önce büyük PR'ları 3–4 dosyalık daha küçük gönderimlere bölmesini zorunlu kılmak.
- D) Tek geçişte 14 dosyanın tamamına yeterli dikkat verebilmesi için daha büyük bağlam penceresine sahip daha büyük bir modele geçmek.

Neden B: Dosya başına odaklı geçişler, tutarlı derinlik ve güvenilir yerel sorun tespiti sağlayarak temel nedeni, yani dikkatin dağılmasını ele alır. Ayrı bir entegrasyon odaklı geçiş de bağımlılık ve veri akışı etkileşimleri gibi dosyalar arası konuları kapsar.

Soru 28 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Otomatik kod incelemeniz pull request başına ortalama 15 bulgu üretiyor ve geliştiriciler 40% yanlış-pozitif oranı bildiriyor. Darboğaz inceleme süresi: geliştiricilerin düzeltmeye mi yoksa reddetmeye mi karar vermeden önce Claude'un gerekçesini okumak için her bulguya tıklaması gerekiyor. CLAUDE.md dosyanız kabul edilebilir örüntüler için zaten kapsamlı kurallar içeriyor ve paydaşlar bulgular geliştiricilere görünmeden önce filtreleyen her yaklaşımı reddetti. Hangi değişiklik inceleme süresini en iyi şekilde ele alır?

Hangi değişiklik inceleme süresini en iyi şekilde ele alır?

- A) Claude'un gerekçesini ve güven tahminini doğrudan her bulguya dahil etmesini zorunlu kılmak. **[DOĞRU]**

- B) Bulgu örüntülerini analiz eden ve geçmiş yanlış-pozitif imzalarıyla eşleşenleri otomatik olarak baskılayan bir son işlemci eklemek.
- C) Bulguları “bloklayıcı sorunlar” ve “öneriler” olarak kategorize edip her seviye için farklı inceleme gereksinimleri tanımlamak.
- D) Claude'u yalnızca yüksek güvenli bulguları gösterecek şekilde yapılandırıp belirsiz işaretleri geliştiriciler görmeden önce filtrelemek.

Neden A: Gerekçeyi ve güveni doğrudan her bulguya dahil etmek, geliştiricilerin her bulguyu açmadan hızlı triage yapmasını sağlayarak inceleme süresini azaltır. “Filtreleme yok” kısıtını karşılar, çünkü tüm bulgular görünür kalırken geliştiricinin karar verme süreci hızlanır.

Soru 29 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Otomatik kod incelemenizin analizi, bulgu kategorilerine göre yanlış-pozitif oranlarında büyük farklar olduğunu gösteriyor: güvenlik/doğruluk bulgularında 8% yanlış pozitif, performans bulgularında 18%, stil/adlandırma bulgularında 52% ve dokümantasyon bulgularında 48%. Geliştirici anketleri güvenin azaldığını gösteriyor; birçok kişi “yarısı yanlış” olduğu için bulguları okumadan yok saymaya başlıyor. Yüksek yanlış-pozitif kategoriler, doğru kategorilere duyulan güveni aşındırıyor. Hangi yaklaşım sistemi iyileştirirken geliştirici güvenini en iyi şekilde yeniden sağlar?

Hangi yaklaşım geliştirici güvenini en iyi şekilde yeniden sağlar?

- A) Yüksek yanlış-pozitif kategorileri (stil, adlandırma, dokümantasyon) geçici olarak devre dışı bırakmak ve prompt'ları iyileştirirken yalnızca yüksek kesinlikli kategorileri tutmak. **[DOĞRU]**
- B) Tüm kategorileri etkin tutmak, ancak geliştiricilerin neyi inceleyeceklerine karar verebilmesi için her bulguyla birlikte güven skorları göstermek.
- C) Tüm kategorileri etkin tutmak ve sonraki birkaç hafta içinde her kategori için doğruluğu artırmak üzere few-shot örnekler eklemek.
- D) Genel yanlış-pozitif oranını düşürmek için tüm kategorilere tek tip bir katılık azaltımı uygulamak.

Neden A: Yüksek yanlış-pozitif kategorileri geçici olarak devre dışı bırakmak, geliştiricilerin her şeyi yok saymasına yol açan gürültülü bulguları kaldırarak güven aşınmasını hemen durdurur; güvenlik ve doğruluk gibi yüksek kesinlikli kategorilerden gelen değeri de korur. Ayrıca, sorunlu kategorileri yeniden etkinleştirmeden önce prompt'ları iyileştirmek için alan açar.

Soru 30 (Senaryo: Claude Code ile Sürekli Entegrasyon)

Durum: Ekibiniz otomatik analiz için API maliyetlerini düşürmek istiyor. Şu anda senkron Claude çağrıları iki iş akışını destekliyor: (1) geliştiriciler merge edebilmeden önce tamamlanması gereken bloklayıcı bir merge öncesi kontrol ve (2) ertesi sabah incelenmek üzere gece üretilen bir teknik borç raporu. Yöneticiniz 50% tasarruf etmek için ikisini de Message Batches API'ye taşımayı öneriyor. Bu öneriyi nasıl değerlendirmelisiniz?

Bu öneriyi nasıl değerlendirmelisiniz?

- A) Toplu işlemler çok uzun sürerse senkron çağrılara geri dönüşle ikisini de toplu işlemeye taşımak.

- B) Tamamlanmayı doğrulamak için durum yoklamasıyla iki iş akışını da toplu işlemeye taşımak.
- C) Toplu işlemeyi yalnızca teknik borç raporları için kullanmak; merge öncesi kontrollerde senkron çağrıları korumak. **[DOĞRU]**
- D) Toplu işlem sonuçlarının sıralamasıyla ilgili sorunlardan kaçınmak için iki iş akışında da senkron çağrıları korumak.

Neden C: Message Batches API işlemesi, gecikme SLA'sı olmadan 24 saate kadar sürebilir; bu, gecelik teknik borç raporları için kabul edilebilir, ancak geliştiricilerin beklediği bloklayıcı merge öncesi kontroller için kabul edilemez. Bu, her iş akışını gecikme gereksinimlerine göre doğru API ile eşleştirir.

Senaryo: Claude Code ile Kod Üretimi

Soru 31 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Claude Code'dan API yanıtlarını dahili normalize edilmiş bir formata dönüştüren bir işlev uygulamasını istediniz. İki iterasyondan sonra çıktı yapısı hâlâ beklentilerle eşleşmiyor; bazı alanlar farklı şekilde iç içe geçiyor ve zaman damgaları yanlış biçimlendiriliyor. Gereksinimleri düzyazıyla açıkladınız, ancak Claude bunları her seferinde farklı yorumluyor.

Sonraki iterasyon için en etkili yaklaşım hangisidir?

- A) Beklenen çıktı yapısını açıklayan bir JSON şeması yazmak ve her iterasyondan sonra Claude'un çıktısını buna göre doğrulamak.
- B) Temsili API yanıtları için beklenen dönüşümü gösteren 2–3 somut girdi-çıktı örneği sağlamak. **[DOĞRU]**
- C) Gereksinimleri daha teknik bir kesinlikle yeniden yazıp tam alan eşlemelerini, iç içe geçme kurallarını ve zaman damgası format dizgilerini belirtmek.
- D) Yorumların nerede ayrıştığını belirlemek için Claude'dan gereksinimlere ilişkin mevcut anlayışını açıklamasını istemek.

Neden B: Somut girdi-çıktı örnekleri, Claude'a beklenen dönüşüm sonuçlarını bire bir göstererek düzyazı açıklamalardaki doğal belirsizliği ortadan kaldırır. Bu, metinsel gereksinimlerin yanlış yorumlanması olan temel nedeni, alanların iç içe geçmesi ve zaman damgası biçimlendirme için açık örüntüler sağlayarak doğrudan ele alır.

Soru 32 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Yeni bir bildirim kanalı olarak Slack eklemeniz gerekiyor. Mevcut kod tabanında e-posta, SMS ve push kanalları için net, yerleşik örüntüler var. Ancak Slack'in API'si temelden farklı entegrasyon yaklaşımları sunuyor: incoming webhook'lar (basit, tek yönlü), bot token'ları (teslim onayı ve programatik kontrol destekler) veya Slack Apps (iki yönlü events, workspace onayı gerektirir). Göreviniz, entegrasyon yöntemini belirtmeden veya teslimat takibi gibi gelişmiş özellikler istemeden “Slack desteği ekle” diyor.

Bu göreve nasıl yaklaşmalısınız?

- A) Mevcut tek yönlü bildirim örüntüsüne uymak için incoming webhook'larla doğrudan yürütme modunda başlamak.
- B) Entegrasyon seçeneklerini ve mimari etkileri incelemek için planlama moduna geçmek, ardından uygulamadan önce bir öneri sunmak. **[DOĞRU]**
- C) Entegrasyon yöntemi kararını erteleyerek mevcut örüntüleri kullanan bir Slack kanal sınıfının iskeletini oluşturmak üzere doğrudan yürütme modunda başlamak.
- D) Teslim onayının mümkün olmasını sağlamak için bot-token yaklaşımıyla doğrudan yürütme modunda başlamak.

Neden B: Slack entegrasyonunda mimari etkileri ciddi ölçüde farklı olan birden fazla geçerli yaklaşım vardır ve gereksinimler belirsizdir. Planlama modu, uygulamadan önce webhook'lar, bot token'ları ve Slack Apps arasındaki ödünleşimleri değerlendirmenizi ve bir yaklaşım üzerinde uzlaşmanızı sağlar.

Soru 33 (Senaryo: Claude Code ile Kod Üretimi)

Durum: CLAUDE.md dosyanız, kodlama standartları, test kuralları, ayrıntılı bir PR inceleme kontrol listesi, deployment talimatları ve veritabanı migration prosedürleri içeren 400+ satıra ulaştı. Claude'un kodlama standartlarını ve test kurallarını her zaman izlemesini, ancak PR inceleme, deploy ve migration rehberliğini yalnızca bu görevleri yaparken uygulamasını istiyorsunuz.

Hangi yeniden yapılandırma yaklaşımı en etkilidir?

- A) Tüm rehberliği iş akışı türüne göre düzenlenmiş ayrı Skills dosyalarına taşımak ve CLAUDE.md'de yalnızca kısa bir proje açıklaması bırakmak.
- B) Her şeyi CLAUDE.md'de tutmak, ancak kategoriye göre ayrı sürdürülen dosyalara düzenlemek için `@import` söz dizimini kullanmak.
- C) CLAUDE.md'yi `.claude/rules/` altındaki dosyalara bölmek ve her kuralın yalnızca ilgili dosya türleri için yüklenmesi amacıyla yola bağlı glob örüntüleri kullanmak.
- D) Evrensel standartları CLAUDE.md'de tutmak ve iş akışına özgü rehberlik (PR inceleme, deploy, migration'lar) için tetikleyici anahtar sözcüklerle Skills oluşturmak. **[DOĞRU]**

Neden D: CLAUDE.md içeriği her oturumda yüklenir; bu da kodlama standartları ve test kurallarının her zaman uygulanmasını sağlar. Skills ise Claude tetikleyici anahtar sözcükleri algıladığında isteğe bağlı çağrılır; PR inceleme, deployment ve migration'lar gibi iş akışına özgü rehberlik için idealdir.

Soru 34 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibinizin monolitik uygulamasını mikroservislere yeniden yapılandırmakla görevlendirildiniz. Bu, onlarca dosyada değişiklikleri etkiler ve servis sınırları ile modül bağımlılıkları hakkında kararlar gerektirir.

Hangi yaklaşımı seçmelisiniz?

- A) Değişiklik yapmadan önce kod tabanını keşfetmek, bağımlılıkları anlamak ve uygulama yaklaşımını tasarlamak için planlama moduna geçmek. **[DOĞRU]**

- B) Doğrudan yürütme modunda başlamak ve yalnızca uygulama sırasında beklenmedik karmaşıklıkla karşılaştıktan sonra planlamaya geçmek.
- C) Doğrudan yürütme modunda başlamak ve doğal servis sınırlarını uygulamanın ortaya çıkarmasına izin vererek artımlı değişiklikler yapmak.
- D) Her servis yapısını belirten ayrıntılı peşin talimatlarla doğrudan yürütme kullanmak.

Neden A: Planlama modu, monoliti bölmek gibi karmaşık mimari yeniden yapılandırmalar için doğru stratejidir: birçok dosyada potansiyel olarak maliyetli değişikliklere girişmeden önce güvenli keşif ve sınırlar hakkında bilinçli kararlar sağlar.

Soru 35 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibiniz bağımlılık taraması, test kapsamı sayımları ve kod kalitesi metrikleri gibi derin kod analizi yapan bir `/analyze-codebase` skill'i oluşturdu. Komut çalıştırıldıktan sonra ekip üyeleri Claude'un oturumda daha az yanıt verir hale geldiğini ve özgün görevin bağlamını kaybettiğini bildiriyor.

Tam analiz kabiliyetlerini korurken bunu en etkili şekilde nasıl düzeltirsiniz?

- A) Analizi yalıtılmış bir alt-ajan bağlamında çalıştırmak için skill frontmatter'ına `context: fork` eklemek. **[DOĞRU]**
- B) Analiz için daha hızlı, daha ucuz bir model kullanmak üzere frontmatter'a `model: haiku` eklemek.
- C) Skill'i, her biri daha az çıktı üreten üç daha küçük skill'e bölmek.
- D) Skill'e, tüm sonuçları göstermeden önce kısa bir özete sıkıştırması için talimatlar eklemek.

Neden A: `context: fork`, analizi yalıtılmış bir alt-ajan bağlamında çalıştırır; böylece büyük çıktı ana oturumun bağlam penceresini kirletmez ve Claude özgün görevi kaybetmez. Ana oturumu yanıt verebilir tutarken tam analiz kabiliyetini korur.

Soru 36 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibiniz bir `/commit` skill'ini `.claude/skills/commit/SKILL.md` içinde kullanıyor. Bir geliştirici bunu kişisel iş akışı için (farklı commit mesajı formatı, ek kontroller) ekip arkadaşlarını etkilemeden özelleştirmek istiyor.

Ne önerirsiniz?

- A) `~/.claude/skills/` altında farklı bir adla kişisel bir sürüm oluşturmak, örneğin `/my-commit`. **[DOĞRU]**
- B) Proje skill frontmatter'ında kullanıcı adına dayalı koşullu mantık eklemek.
- C) Aynı adla `~/.claude/skills/commit/SKILL.md` konumunda kişisel bir sürüm oluşturmak.
- D) Kişisel skill frontmatter'ında proje sürümüne göre öncelik vermek için `override: true` ayarlamak.

Neden A: Kişisel skill'ler aynı ada sahip proje skill'lerine göre önceliklidir, bu yüzden `commit` adını yeniden kullanmak, ekibin skill'ini yalnızca bu geliştirici için sessizce gölgeler — ekip `/commit`'i her geliştirdiğinde güncellemeleri almayı bırakır ve aynı komut altında farklı bir skill çalıştırdığını hatırlaması gerekir. Kişisel sürümü `/my-commit` olarak adlandırmak bu çakışmayı tamamen ortadan kaldırır: geliştirici

ekibin bakımını yaptığı `/commit`'i kullanmaya devam eder ve kişisel iş akışı için açıkça adlandırılmış, ayrı bir skill elde eder; ikisini karıştırma veya ekip güncellemelerini kaçırma riski olmaz.

Soru 37 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibiniz aylardır Claude Code kullanıyor. Son zamanlarda üç geliştirici Claude'un “her zaman kapsamlı hata yönetimi ekle” rehberliğini izlediğini bildiriyor, ancak yeni katılan dördüncü bir geliştirici Claude'un bunu izlemediğini söylüyor. Dördü de aynı repo'da çalışıyor ve kodları güncel.

En olası neden ve çözüm nedir?

- A) Rehberlik, özgün geliştiricilerin kullanıcı düzeyi `~/.claude/CLAUDE.md` dosyalarında bulunuyor, proje `.claude/CLAUDE.md` dosyasında değil. Tüm ekip üyelerinin alması için talimatı proje düzeyi dosyaya taşıyın. **[DOĞRU]**
- B) Yeni geliştiricinin `~/.claude/CLAUDE.md` dosyası, proje ayarlarını geçersiz kılan çelişkili talimatlar içeriyor; çelişkili bölümü silmelidir.
- C) Claude Code kullanıcı tercihlerini zamanla kişi bazında öğrenir; yeni geliştirici, Claude bunu “hatırlayana” kadar gereksinimi tekrar etmelidir.
- D) Claude Code ilk okumadan sonra CLAUDE.md'yi önbelleğe alır; özgün geliştiriciler önbelleğe alınmış sürümleri kullanıyor. Herkes Claude Code önbelleğini temizlemelidir.

Neden A: Rehberlik yalnızca özgün geliştiricilerin kullanıcı düzeyi yapılandırmalarına eklenmişse ve proje düzeyi `.claude/CLAUDE.md` dosyasında yoksa, yeni ekip üyeleri bunu almaz. Proje düzeyi yapılandırmaya taşımak, mevcut ve gelecekteki tüm ekip üyelerinin rehberliği otomatik olarak almasını sağlar.

Soru 38 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Bağlam olarak 2–3 tam endpoint uygulama örneği dahil etmenin, yeni API endpoint'leri üretirken tutarlılığı önemli ölçüde artırdığını görüyorsunuz. Ancak bu bağlam yalnızca yeni endpoint'ler oluştururken faydalıdır; API dizininde hata ayıklama, kod inceleme veya başka işler yaparken değil.

Hangi yapılandırma yaklaşımı en etkilidir?

- A) Endpoint örneklerini ve örüntü dokümantasyonunu proje CLAUDE.md dosyasına ekleyerek her zaman kullanılabilir olmalarını sağlamak.
- B) Her üretim isteğinde kodu prompt'a kopyalayarak endpoint örneklerine manuel olarak referans vermek.
- C) API dizininde çalışırken etkinleşen ve endpoint örneklerini içeren `.claude/rules/api/` içinde yola özgü kurallar yapılandırmak.
- D) Endpoint örneklerine referans veren ve örüntü izleme talimatları içeren, bir eğik çizgi komutuyla isteğe bağlı çağrılan bir skill oluşturmak. **[DOĞRU]**

Neden D: İsteğe bağlı çağrılan bir skill, örnek bağlamı yalnızca yeni endpoint'ler üretirken yükler; hata ayıklama veya inceleme gibi ilgisiz görevlerde yüklemez. Bu, gerektiğinde yüksek kaliteli üretimi korurken ana bağlamı temiz tutar.

Soru 39 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibiniz veritabanı migration dosyaları üreten bir `/migration` skill'i oluşturdu. Migration adını `$ARGUMENTS` üzerinden alıyor. Üretimde üç sorun gözlemliyorsunuz: (1) geliştiriciler skill'i çoğu zaman argümansız çalıştırıyor ve bu kötü adlandırılmış dosyalara yol açıyor, (2) skill bazen ilgisiz önceki konuşmalardan veritabanı şema ayrıntılarını kullanıyor ve (3) bir geliştirici, skill geniş araç erişimine sahipken yanlışlıkla yıkıcı test temizliği çalıştırdı.

Hangi yapılandırma yaklaşımı üç problemi de çözer?

- A) Belirli girdileri zorunlu kılmak için `$1` ve `$2` konumsal parametrelerini `$ARGUMENTS` yerine kullanmak, bağlam kontrolü için `@` söz dizimiyle açık şema dosyası referansları dahil etmek ve yıkıcı işlemler hakkında uyarı veren bir frontmatter açıklaması eklemek.
- B) Gerekli parametreleri istemek için frontmatter'a `argument-hint` eklemek, yürütmeyi yalıtmak için `context: fork` kullanmak ve `allowed-tools`'u dosya yazma işlemleriyle sınırlamak. **[DOĞRU]**
- C) `/migration-create` ve `/migration-apply` skill'lerine bölmek, migration adı eksikse istemek için doğrulama talimatları eklemek ve her biri için farklı `allowed-tools` kapsamları kullanmak.
- D) `$ARGUMENTS`'ın geçerli bir ad olduğundan emin olmak için skill SKILL.md dosyasına doğrulama talimatları eklemek, önceki konuşma bağlamını yok sayacak prompt'lar eklemek ve kaçınılacak yasak işlemleri listelemek.

Neden B: Bu yaklaşım, her sorunu ele almak için üç ayrı yapılandırma özelliği kullanır: `argument-hint` argüman girişini iyileştirir ve eksik argümanları azaltır, `context: fork` önceki konuşmalardan bağlam sızıntısını önler ve `allowed-tools` skill'i güvenli dosya yazma işlemleriyle sınırlandırarak yıkıcı eylemleri engeller.

Soru 40 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Kod tabanınızda farklı kodlama kurallarına sahip alanlar var: React bileşenleri hook'larla fonksiyonel stili kullanıyor, API handler'ları belirli hata yönetimiyle async/await kullanıyor ve veritabanı modelleri repository pattern'ını izliyor. Test dosyaları kod tabanına, test edilen kodun yanına dağılmış durumda (ör. `Button.test.tsx`, `Button.tsx` dosyasının yanında) ve konumdan bağımsız olarak tüm testlerin aynı kuralları izlemesini istiyorsunuz.

Claude'un kod üretirken doğru kuralları otomatik olarak uygulamasını sağlamanın en çok desteklenen yolu nedir?

- A) Tüm kuralları, her alan için başlıklar altında kök CLAUDE.md'ye koymak ve hangi bölümün geçerli olduğunu Claude'un çıkarmasına güvenmek.
- B) Her kod türü için `.claude/skills/` içinde Skills oluşturup kuralları her SKILL.md'ye gömmek.
- C) Her alt dizine, o alanın kurallarını içeren ayrı bir CLAUDE.md dosyası yerleştirmek.
- D) Dosya yollarına göre kuralları koşullu uygulamak için glob desenleri belirten YAML frontmatter'a sahip kural dosyalarını `.claude/rules/` altında oluşturmak. **[DOĞRU]**

Neden D: `.claude/rules/` dosyaları, YAML frontmatter'a ve glob desenlerine (ör. `**/*.test.tsx`, `src/api/**/*.ts`) sahip olduğunda dizin yapısından bağımsız olarak deterministik ve yol tabanlı kural uygulamayı mümkün kılar. Dağıtılmış test dosyaları gibi kesişen kalıplar için en çok desteklenen yaklaşım budur.

Soru 41 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibinizin standart kod inceleme kontrol listesini çalıştıran özel eğik çizgi komutu `/review` oluşturmak istiyorsunuz. Bu komut, depoyu klonlayan veya güncelleyen her geliştirici tarafından kullanılabilir olmalı.

Komut dosyasını nerede oluşturmalısınız?

- A) Her geliştiricinin ana dizinindeki `~/.claude/commands/` içinde.
- B) Proje deposunda `.claude/commands/` altında. **[DOĞRU]**
- C) `.claude/config.json` içinde bir komut dizisi olarak.
- D) Kök proje CLAUDE.md dosyasında.

Neden B: Özel slash komutlarını proje deposunun içindeki `.claude/commands/` altına koymak, bunların sürüm kontrolünde tutulmasını ve depoyu klonlayan ya da güncelleyen her geliştirici için otomatik olarak kullanılabilir olmasını sağlar. Claude Code'da proje düzeyi özel komutlar için amaçlanan konum budur.

Soru 42 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibinizin CLAUDE.md dosyası TypeScript kurallarını, test rehberliğini, API kalıplarını ve dağıtım prosedürlerini karıştırarak 500 satırı aştı. Geliştiriciler doğru bölümleri bulmakta ve güncellemekte zorlanıyor.

Claude Code, proje düzeyi yönergeleri odaklı konu modülleri halinde düzenlemek için hangi yaklaşımı destekler?

- A) Dosya desenlerini CLAUDE.md içindeki belirli bölümlerle eşleyen bir `.claude/config.yaml` dosyası tanımlamak.
- B) `.claude/rules/` içinde, her biri tek bir konuyu kapsayan ayrı Markdown dosyaları oluşturmak (ör. `testing.md`, `api-conventions.md`). **[DOĞRU]**
- C) Yönergeleri, Claude'un otomatik olarak yönerge olarak yüklediği ilgili alt dizinlerdeki README.md dosyalarına bölmek.
- D) Dizin ağacının farklı düzeylerinde, her biri üst yönergeleri geçersiz kılan birden fazla CLAUDE.md adlı dosya oluşturmak.

Neden B: Claude Code, konu bazlı rehberlik için ayrı Markdown dosyaları oluşturabileceğiniz bir `.claude/rules/` dizinini destekler (ör. `testing.md`, `api-conventions.md`); bu da ekiplerin büyük yönerge kümelerini odaklı ve bakımı kolay modüller halinde düzenlemesine olanak tanır.

Soru 43 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibinizin bir yaklaşım seçmeden önce uygulama seçeneklerini beyin fırtınasıyla üretmek ve değerlendirmek için kullandığı özel Skill `/explore-alternatives` oluşturuyorsunuz. Geliştiriciler, Skill çalıştırıldıktan sonra Claude'un sonraki yanıtlarının alternatifler tartışmasından etkilendiğini bildiriyor; bazen reddedilen yaklaşımlara atıf yapıyor veya gerçek uygulamayı engelleyen keşif bağlamını koruyor.

Bu Skill'i en etkili şekilde nasıl yapılandırmalısınız?

- A) Keşif mantığını bir bash alt süreci olarak çalıştırmak için Skill'de `!` önekini kullanmak.
- B) Skill frontmatter'ına `context: fork` eklemek. **[DOĞRU]**
- C) Keşif bağlamının ne zaman atılması gerektiğini sınırlamak için iki Skill'e bölmek: `/explore-start` ve `/explore-end`.
- D) Skill'i `~/.claude/skills/` içinde, `.claude/skills/` yerine oluşturmak.

Neden B: `context: fork`, Skill'i yalıtılmış bir alt-ajan bağlamında çalıştırır; böylece keşif tartışmaları ana konuşma geçmişini kirletmez. Bu, reddedilen yaklaşımların ve beyin fırtınası bağlamının sonraki uygulama çalışmalarını etkilemesini önler.

Soru 44 (Senaryo: Claude Code ile Kod Üretimi)

Durum: Ekibiniz Claude Code üzerinden PR aramak ve CI durumunu kontrol etmek için bir GitHub MCP sunucusu eklemek istiyor. Altı geliştiricinin her birinin kendi kişisel GitHub erişim token'ı var. Kimlik bilgilerini sürüm kontrolüne eklemeden ekip genelinde tutarlı araçlar istiyorsunuz.

Hangi yapılandırma yaklaşımı en etkilidir?

- A) Her geliştiricinin sunucuyu kullanıcı kapsamına `claude mcp add --scope user` ile eklemesini sağlamak.
- B) Token'ları bir `.env` dosyasından okuyan ve GitHub API çağrılarını proxy'leyen bir MCP sunucu sarmalayıcısı oluşturup bu sarmalayıcıyı proje `.mcp.json` dosyasına eklemek.
- C) Sunucuyu proje `.mcp.json` dosyasına, yetkilendirme için ortam değişkeni yerine koymayı (`${GITHUB_TOKEN}`) kullanarak eklemek ve gerekli ortam değişkenini proje README'sinde belgelemek. **[DOĞRU]**
- D) Sunucuyu yer tutucu bir token ile proje kapsamında yapılandırmak, ardından geliştiricilere bunu yerel yapılandırmalarında geçersiz kılmalarını söylemek.

Neden C: Ortam değişkeni yerine koyma kullanan proje `.mcp.json` dosyası idiomatiktir: MCP yapılandırması için sürüm kontrolünde tutulan tek bir doğruluk kaynağı sağlar ve her geliştiricinin kimlik bilgilerini ortam değişkenleri üzerinden vermesine olanak tanır. Değişkeni belgelemek, sırları commit etmeden yeni geliştirici katılımını kolaylaştırır.

Soru 45 (Senaryo: Claude Code ile Kod Üretimi)

Durum: 120 dosyalık bir kod tabanı genelinde harici API çağrılarının etrafına hata yönetimi sarmalayıcıları ekliyorsunuz. İşin üç aşaması var: (1) tüm çağrı noktalarını ve kalıpları keşfetmek, (2) hata yönetimi yaklaşımını iş birliğiyle tasarlamak ve (3) sarmalayıcıları tutarlı biçimde uygulamak. 1. Aşamada Claude, yüzlerce çağrı noktasını bağlamıyla listeleyen büyük çıktılar üretiyor ve keşif bitmeden bağlam penceresini hızla dolduruyor.

Uygulama tutarlılığını korurken görevi tamamlamak için hangi yaklaşım en etkilidir?

- A) 1. Aşama için ayrıntılı keşif çıktısını yalıtıp bir özet döndürmesi amacıyla bir Explore alt-ajanı kullanmak, ardından 2–3. Aşamaları ana konuşmada sürdürmek. **[DOĞRU]**
- B) Tüm aşamaları ana konuşmada yapmak, dosyalar arasında ilerlerken bağlam kullanımını azaltmak için periyodik olarak `/compact` kullanmak.
- C) Sürekliliği korumak için toplu çağrılar arasında açık bağlam özetleri geçirerek `--continue` ile headless moda geçmek.
- D) Hata yönetimi kalıbını CLAUDE.md'de tanımlamak, ardından tutarlılık için paylaşılan bellek dosyasına güvenerek dosyaları birden fazla oturumda gruplar halinde işlemek.

Neden A: Explore alt-ajanı, ayrıntılı keşif çıktısını ayrı bir bağlamda yalıtır ve ana konuşmaya yalnızca kısa bir özet döndürür. Bu, ana bağlam penceresini, korunan bağlamın en değerli olduğu iş birliğine dayalı tasarım ve tutarlı uygulama aşamaları için saklar.

Senaryo: Müşteri Destek Ajanı

Soru 46 (Senaryo: Müşteri Destek Ajanı)

Durum: Test sırasında, kullanıcılar sipariş durumu hakkında soru sorduğunda ajanın sık sık `get_customer` çağırdığını, oysa `lookup_order`'ın daha uygun olduğunu fark ediyorsunuz. Bu sorunu ele almak için ilk olarak neyi kontrol etmelisiniz?

İlk olarak neyi kontrol etmelisiniz?

- A) Siparişle ilgili istekleri algılayıp doğrudan `lookup_order`'a yönlendiren bir ön işleme sınıflandırıcısı uygulamak.
- B) Seçimi basitleştirmek için ajanın kullanabileceği araç sayısını azaltmak.
- C) Araç seçimini iyileştirmek için sistem komutuna, tüm olası sipariş isteği kalıplarını kapsayan few-shot örnekler eklemek.
- D) Her aracın amacını açıkça ayırt ettiklerinden emin olmak için araç açıklamalarını kontrol etmek. **[DOĞRU]**

Neden D: Araç açıklamaları, modelin hangi aracı çağıracağına karar vermek için kullandığı birincil girdidir. Bir ajan sürekli yanlış aracı seçtiğinde ilk tanılama adımı, araç açıklamalarının her aracın amacını ve kullanım sınırlarını açıkça ayırdığını doğrulamaktır.

Soru 47 (Senaryo: Müşteri Destek Ajanı)

Durum: Ajanınız tek sorunlu istekleri %94 doğrulukla ele alıyor (ör. “#1234 numaralı sipariş için iade istiyorum”). Ancak müşteriler tek mesajda birden fazla sorun belirttiğinde (ör. “#1234 numaralı sipariş için iade istiyorum ve ayrıca #5678 numaralı siparişin teslimat adresini güncellemek istiyorum”), araç seçimi doğruluğu %58'e düşüyor. Ajan genellikle yalnızca bir sorunu çözüyor veya istekler arasında parametreleri karıştırıyor. Çok sorunlu isteklerde güvenilirliği en etkili hangi yaklaşım iyileştirir?

Hangi yaklaşım en etkilidir?

- A) Çok sorunlu mesajları ayrı isteklere ayrıştırmak, her birini bağımsız ele almak ve sonuçları birleştirmek için ayrı bir model çağrısı kullanan bir ön işleme katmanı uygulamak.
- B) İlgili araçları daha az sayıda evrensel araçta birleştirmek.
- C) Prompt'a, çok sorunlu istekler için doğru akıl yürütmeyi ve araç sıralamasını gösteren few-shot örnekler eklemek. **[DOĞRU]**
- D) Eksik yanıtları algılayan ve kaçırılan sorunları çözmesi için ajanı otomatik olarak yeniden prompt'layan yanıt doğrulaması uygulamak.

Neden C: Çok sorunlu isteklerde doğru akıl yürütmeyi ve araç sıralamasını gösteren few-shot örnekler en etkilidir; çünkü ajan tek sorunlarda zaten iyi performans gösteriyor. İhtiyaç duyduğu şey, birden çok sorunu ayrıştırma ve yönlendirme ile parametreleri ayrı tutma kalıbına ilişkin rehberliktir.

Soru 48 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim log'ları, “#1234 numaralı sipariş için iade” gibi basit isteklerde ajanın sorunu 3–4 araç çağrısıyla ve %91 başarıyla çözdüğünü gösteriyor. Ancak “Benden iki kez ücret alındı, indirimim uygulanmadı ve iptal etmek istiyorum” gibi karmaşık isteklerde ajan ortalama 12+ araç çağrısı yapıyor ve yalnızca %54 başarı sağlıyor; çoğu zaman sorunları sırayla inceliyor ve her biri için gereksiz müşteri verilerini tekrar çekiyor. Karmaşık isteklerin ele alınmasını en etkili hangi değişiklik iyileştirir?

Hangi değişiklik en etkilidir?

- A) Aşamalar arasına açık doğrulama kontrol noktaları ekleyerek, ajanın bir sonraki soruna geçmeden önce her sorunu çözdükten sonra ilerlemeyi kaydetmesini zorunlu kılmak.
- B) `get_customer`, `lookup_order` ve faturalamayla ilgili araçları tek bir `investigate_issue` aracında birleştirerek araç sayısını azaltmak.
- C) İsteği ayrı sorunlara ayrıştırmak, ardından nihai çözümü sentezlemeden önce paylaşılan müşteri bağlamını kullanarak her birini paralel incelemek. **[DOĞRU]**
- D) Çeşitli çok yönlü faturalama senaryoları için ideal araç çağrısı dizilerini gösteren few-shot örnekleri sistem komutuna eklemek.

Neden C: Ayrı sorunlara ayrıştırmak ve paylaşılan müşteri bağlamıyla paralel incelemek iki temel problemi de düzeltir: sorunlar arasında paylaşılan bağlamı yeniden kullanarak gereksiz veri çekmeyi ortadan kaldırır ve tek bir çözümü sentezlemeden önce incelemeyi paralelleştirerek toplam araç çağrısı döngülerini azaltır.

Soru 49 (Senaryo: Müşteri Destek Ajanı)

Durum: Ajanınız %55 ilk temasta çözüm oranına ulaşıyor; bu, %80 hedefinin oldukça altında. Log'lar, basit vakaları (fotoğraf kanıtı olan hasarlı ürünlerde standart değişimler) eskale ederken politika istisnası gerektiren karmaşık durumları otonom olarak ele almaya çalıştığını gösteriyor. Eskalasyon kalibrasyonunu iyileştirmenin en etkili yolu nedir?

Eskalasyon kalibrasyonunu iyileştirmenin en etkili yolu nedir?

- A) Ajandan her yanıttan önce 1–10 ölçeğinde güvenini puanlamasını istemek ve güven bir eşiğin altına düştüğünde otomatik olarak insanlara yönlendirmek.
- B) Ana ajan işlemeye başlamadan önce hangi isteklerin eskalasyon gerektirdiğini tahmin etmek için geçmiş destek kayıtlarıyla eğitilmiş ayrı bir sınıflandırıcı model devreye almak.
- C) Sistem komutuna, ne zaman eskale edileceğini ve ne zaman otonom çözüleceğini gösteren few-shot örneklerle birlikte açık eskalasyon kriterleri eklemek. **[DOĞRU]**
- D) Müşteri hayal kırıklığı düzeyini belirlemek ve negatif duygu eşiği aşıldığında otomatik eskale etmek için duygu analizi uygulamak.

Neden C: Açık eskalasyon kriterleri ve few-shot örnekler kök nedeni doğrudan ele alır: basit ve karmaşık vakalar arasındaki karar sınırlarının belirsiz olması. Bu, ek altyapı olmadan ajana ne zaman insana yönlendireceğini ve ne zaman otonom çözeceğini öğreten, en orantılı ve etkili ilk müdahaledir.

Soru 50 (Senaryo: Müşteri Destek Ajanı)

Durum: Ajan `get_customer` ve `lookup_order` çağırdıktan sonra mevcut tüm sistem verilerine sahip, ancak hâlâ belirsizlikle karşı karşıya. Hangi durum `escalate_to_human` çağırmak için en haklı tetikleyicidir?

Hangi durum eskalasyon için en haklıdır?

- A) Müşteri dün gönderilen ve yarın ulaşacak bir siparişi iptal etmek istiyor. Ajan eskale etmelidir, çünkü müşteri paketi aldıktan sonra fikrini değiştirebilir.
- B) Müşteri bir siparişi almadığını iddia ediyor, ancak takip bilgisi siparişin üç gün önce adresinde teslim edildiğini ve imzalandığını gösteriyor. Ajan eskale etmelidir, çünkü çelişkili kanıt sunmak müşteri ilişkisine zarar verebilir.
- C) Müşteri rakip fiyat eşleştirmesi istiyor. Politikalarınız, kendi sitenizdeki fiyat düşüşleri için 14 gün içinde fiyat ayarlamasına izin veriyor, ancak rakip fiyatları hakkında hiçbir şey söylemiyor. Ajan politika yorumu için eskale etmelidir. **[DOĞRU]**
- D) Müşteri mesajı hem bir faturalama sorusu hem de bir ürün iadesi içeriyor. Ajan eskale etmelidir ki bir insan iki konuyu tek etkileşimde koordine edebilsin.

Neden C: Bu gerçek bir politika boşluğudur: şirket kuralları kendi sitenizdeki fiyat düşüşlerini kapsar, ancak rakip fiyat eşleştirmesini ele almaz. Ajan politika uydurmamalı ve mevcut kuralların nasıl yorumlanacağı ya da genişletileceği konusunda insan kararı için eskale etmelidir.

Soru 51 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim log'ları, vakaların %12'sinde ajanınızın `get_customer` adımını atlayıp yalnızca müşterinin verdiği adı kullanarak doğrudan `lookup_order` çağırdığını; bunun bazen yanlış tanımlanan hesaplara ve hatalı iadelere yol açtığını gösteriyor. Bu güvenilirlik problemini en etkili hangi değişiklik düzeltir?

Hangi değişiklik en etkilidir?

- A) Müşteriler gönüllü olarak sipariş ayrıntıları verse bile ajanın her zaman önce `get_customer` çağırdığını gösteren few-shot örnekler eklemek.
- B) Her isteği analiz eden ve o istek türüne uygun araçların yalnızca bir alt kümesini etkinleştiren bir yönlendirme sınıflandırıcısı uygulamak.
- C) `lookup_order` ve `process_refund` çağrılarını, `get_customer` doğrulanmış bir müşteri tanımlayıcısı döndürene kadar engelleyen programatik bir ön koşul eklemek. **[DOĞRU]**
- D) Herhangi bir sipariş işleminden önce `get_customer` ile müşteri doğrulamasının zorunlu olduğunu belirten sistem komutunu güçlendirmek.

Neden C: Programatik bir ön koşul, gerekli sıralamanın izlendiğine dair deterministik bir garanti sağlar. LLM davranışından bağımsız olarak doğrulamanın atlanma olasılığını ortadan kaldırdığı için en etkili yaklaşımdır.

Soru 52 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim metrikleri, karmaşık faturalama anlaşmazlıkları veya çok siparişli iadeler çözülürken müşteri memnuniyeti puanlarının, çözüm teknik olarak doğru olsa bile basit vakalara göre %15 daha düşük olduğunu gösteriyor. Kök neden analizi, ajanın doğru çözümler sunduğunu ancak gerekçeyi tutarsız açıkladığını gösteriyor: bazen ilgili politika ayrıntılarını atlıyor, bazen zaman çizelgesi bilgisini veya sonraki adımları kaçırıyor. Belirli bağlam boşlukları vakadan vakaya değişiyor. İnsan gözetimi eklemeden çözüm kalitesini iyileştirmek istiyorsunuz. Hangi yaklaşım en etkilidir?

Hangi yaklaşım en etkilidir?

- A) Ajanın taslak yanıtı eksiksizlik açısından değerlendirdiği bir öz eleştiri aşaması eklemek: müşterinin sorununu çözdüğünden, ilgili bağlamı içerdiğinden ve takip sorularını öngördüğünden emin olmak. **[DOĞRU]**
- B) Kapatmadan önce ajanın “Bu sorununuzu tamamen çözüyor mu?” diye sorduğu ve müşterilerin gerekirse ek bilgi isteyebildiği bir onay aşaması eklemek.
- C) Tanımlı bir karmaşıklık metriğine göre yönlendirme yaparak karmaşık vakalar için modeli Haiku'den Sonnet'e yükseltmek.
- D) Sistem komutunda, politika bağlamının, zaman çizelgelerinin ve sonraki adımların nasıl dahil edileceğini gösteren beş yaygın karmaşık vaka türü için eksiksiz açıklamalar içeren few-shot örnekler uygulamak.

Neden A: Öz eleştiri aşaması (değerlendirici-iyileştirici kalıbı), ajanın yanıtı sunmadan önce kendi taslağını politika bağlamı, zaman çizelgeleri ve sonraki adımlar gibi somut kriterlere göre değerlendirmesini zorunlu kılarak açıklama eksiksizliğindeki tutarsızlığı doğrudan ele alır. Bu, insan gözetimi olmadan vaka özelindeki boşlukları yakalar.

Soru 53 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim metrikleri, ajanınızın çözüm başına ortalama 4+ API döngüsü yaptığını gösteriyor. Analiz, her ikisi de başlangıçta gerekli olduğunda bile Claude'un çoğu zaman `get_customer` ve `lookup_order` isteklerini ayrı ardışık turlarda yaptığını ortaya koyuyor. Döngü sayısını azaltmanın en etkili yolu nedir?

Döngüleri azaltmanın en etkili yolu nedir?

- A) İstenen herhangi bir araçla paralel olarak muhtemelen gerekli araçları otomatik çağıran ve ne istendiğine bakmadan tüm sonuçları döndüren spekülatif yürütme uygulamak.
- B) Claude'a planlama için daha fazla alan vermek ve araç isteklerini doğal olarak birleştirmesini sağlamak için `max_tokens` değerini artırmak.
- C) Yaygın arama kombinasyonlarını tek çağrıda paketleyen `get_customer_with_orders` gibi bileşik araçlar oluşturmak.
- D) Prompt'ta Claude'a araç isteklerini tek turda paketlemesini ve sonraki API çağrısından önce tüm sonuçları birlikte döndürmesini söylemek. **[DOĞRU]**

Neden D: Claude'a ilgili araç isteklerini tek turda paketlemesini prompt ile belirtmek, Claude'un aynı anda birden fazla araç isteme konusundaki yerleşik yeteneğinden yararlanır. Bu, en az mimari değişiklikle ardışık çağrı kalıbını doğrudan düzeltir.

Soru 54 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim log'ları şu kalıbı gösteriyor: müşteriler belirli tutarlara atıfta bulunuyor (ör. “bahsettiğim %15 indirim”), ancak ajan yanlış değerlerle yanıt veriyor. İnceleme, bu ayrıntıların 20+ tur önce belirtildiğini ve “promosyon fiyatlandırması tartışıldı” gibi belirsiz özetlere sıkıştırıldığını gösteriyor. En etkili düzeltme nedir?

Hangi düzeltme en etkilidir?

- A) Özetleme eşiğini %70'ten %85'e çıkarmak; böylece özetleme tetiklenmeden önce konuşmalara daha fazla alan kalır.
- B) Tam konuşma geçmişini harici depolamada saklamak ve ajan “bahsettiğim gibi” benzeri atıfları algıladığında getirme uygulamak.
- C) İşlemsel olguları (tutarlar, tarihler, sipariş numaraları) özetlenmiş geçmişin dışında her prompt'a dahil edilen kalıcı bir “vaka olguları” bloğuna çıkarmak. **[DOĞRU]**
- D) Özetleme prompt'unu, tüm sayıları, yüzdeleri, tarihleri ve müşterinin belirttiği beklentileri birebir koruyacak şekilde açıkça revize etmek.

Neden C: Özetleme doğası gereği hassas ayrıntıları kaybeder. İşlemsel olguları özetlenmiş geçmişin dışındaki yapılandırılmış bir “vaka olguları” bloğuna çıkarmak, kritik bilgileri korur; böylece kaç turun özetlendiğinden bağımsız olarak her prompt'ta güvenilir biçimde kullanılabilirler.

Soru 55 (Senaryo: Müşteri Destek Ajanı)

Durum: `get_customer` aracınız ada göre arama yaparken tüm eşleşmeleri döndürüyor. Şu anda birden fazla sonuç olduğunda Claude en yeni siparişi olan müşteriyi seçiyor, ancak üretim verileri bunun belirsiz eşleşmelerde zamanın %15'inde yanlış hesabı seçtiğini gösteriyor. Bunu nasıl ele almalısınız?

Bunu nasıl ele almalısınız?

- A) %85 güvenin üzerinde otonom hareket eden ve eşiğin altında açıklama isteyen bir güven puanlama sistemi uygulamak.
- B) `get_customer` birden fazla eşleşme döndürdüğünde, müşteriye özgü herhangi bir işlem yapmadan önce Claude'a ek bir tanımlayıcı (e-posta, telefon veya sipariş numarası) istemesini söylemek. **[DOĞRU]**
- C) `get_customer` aracını, belirsizliği ortadan kaldıracak şekilde sıralama algoritmasına göre yalnızca tek ve en olası eşleşmeyi döndürecek biçimde değiştirmek.
- D) Belirsiz eşleşmeler için doğru akıl yürütmeyi ve araç sıralamasını gösteren few-shot örnekleri prompt'a eklemek.

Neden B: Kullanıcıdan ek bir tanımlayıcı istemek, belirsizliği gidermenin en güvenilir yoludur; çünkü kullanıcı kendi kimliği hakkında kesin bilgiye sahiptir. Yanlış hesabı seçmenin yol açtığı %15 hata oranını ortadan kaldırmak için fazladan bir konuşma turu küçük bir bedeldir.

Soru 56 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim log'ları tutarlı bir kalıp gösteriyor: müşteriler mesajlarında “hesap” sözcüğünü kullandığında (ör. “Dün verdiğim bir sipariş için hesabımı kontrol etmek istiyorum”), ajan zamanın %78'inde önce `get_customer` çağırıyor. Müşteriler benzer istekleri “hesap” olmadan ifade ettiğinde (ör. “Dün verdiğim bir siparişi kontrol etmek istiyorum”), zamanın %93'ünde önce `lookup_order` çağırıyor. Araç açıklamaları açık ve belirsiz değil. Bu tutarsızlığın en olası kök nedeni nedir?

En olası kök neden nedir?

- A) Sistem komutu, “hesap” gibi terimlere göre davranışı yönlendiren ve istenmeyen araç seçimi kalıpları oluşturan anahtar sözcüğe duyarlı yönergeler içeriyor. **[DOĞRU]**
- B) Modelin temel eğitimi, araç açıklamalarını geçersiz kılan “hesap” terminolojisi ile müşteriyle ilgili işlemler arasında çağrışımlar oluşturuyor.
- C) Modelin çok kavramlı mesajlar konusunda daha fazla eğitim verisine ihtiyacı var ve hem hesap hem sipariş terminolojisi içeren örneklerle fine-tune edilmelidir.
- D) Bu anahtar sözcük kaynaklı karışıklığı önlemek için araç açıklamalarına, her aracın ne zaman kullanılmaması gerektiğini belirten ek negatif örnekler eklenmelidir.

Neden A: Sistematik anahtar sözcük güdümlü kalıp (%78'e karşı %93), sistem komutunda “hesap” sözcüğüne tepki veren ve ajanı müşteriyle ilgili araçlara yönlendiren açık yönlendirme mantığını güçlü biçimde gösterir. Araç açıklamaları zaten açık olduğundan, tutarsızlık prompt düzeyi yönergelerin istenmeyen davranış yönlendirmesi oluşturduğuna işaret eder.

Soru 57 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim log'ları, kullanıcılar siparişler hakkında soru sorduğunda (ör. “#12345 numaralı siparişimi kontrol et”) ajanın sık sık `get_customer` çağırdığını, `lookup_order` çağırmadığını gösteriyor. Her iki araç da çok kısa açıklamalara sahip (“Müşteri bilgilerini alır” / “Sipariş ayrıntılarını alır”) ve benzer görünen tanımlayıcı biçimlerini kabul ediyor. Araç seçimi güvenilirliğini iyileştirmek için en etkili ilk adım nedir?

En etkili ilk adım nedir?

- A) Her turdan önce kullanıcı girdisini analiz eden ve algılanan anahtar sözcüklere ve ID kalıplarına göre doğru aracı önceden seçen bir yönlendirme katmanı uygulamak.
- B) Herhangi bir tanımlayıcıyı kabul eden ve hangi backend'in sorgulanacağına dahili olarak karar veren tek bir `lookup_entity` aracında iki aracı birleştirmek.
- C) Sistem komutuna, siparişle ilgili sorguları `lookup_order`'a yönlendiren 5–8 örnekle doğru araç seçimi kalıplarını gösteren few-shot örnekler eklemek.
- D) Her aracın açıklamasını; girdi formatlarını, örnek sorguları, uç durumları ve benzer araçlara kıyasla ne zaman kullanılacağını açıklayan sınırları içerecek şekilde genişletmek. **[DOĞRU]**

Neden D: Araç açıklamalarını girdi formatları, örnek sorgular, uç durumlar ve açık sınırlarla genişletmek kök nedeni doğrudan düzeltir: LLM'e benzer araçları ayırt etmek için yeterli bilgi vermeyen minimal açıklamalar. Bu, LLM'in araç seçimi için kullandığı birincil mekanizmayı iyileştiren düşük eforlu ve yüksek etkili ilk adımdır.

Soru 58 (Senaryo: Müşteri Destek Ajanı)

Durum: Destek ajanınız için ajan döngüsünü uyguluyorsunuz. Her Claude API çağrısından sonra döngüye devam edip etmeyeceğinize (istenen araçları çalıştırıp Claude'u tekrar çağırma) veya durup durmayacağınıza (nihai yanıtı müşteriye sunma) karar vermelisiniz. Bu kararı ne belirler?

Bu kararı ne belirler?

- A) Claude'un yanıtındaki `stop_reason` alanını kontrol etmek; `tool_use` ise devam etmek, `end_turn` ise durmak. **[DOĞRU]**
- B) Claude'un metninde “Bitirdim” veya “Başka bir konuda yardımcı olabilir miyim?” gibi ifadeleri ayrıştırmak; doğal dil sinyalleri görev tamamlanmasını gösterir.
- C) Maksimum iterasyon sayısı belirlemek (ör. 10 çağrı) ve Claude'un daha fazla iş gerektiğini belirtip belirtmediğinden bağımsız olarak bu sayıya ulaşıldığında durmak.
- D) Yanıtın asistan metni içerip içermediğini kontrol etmek; Claude açıklayıcı metin ürettiyse döngü sonlanmalıdır.

Neden A: `stop_reason`, döngü kontrolü için Claude'un açık yapılandırılmış sinyalidir: `tool_use`, Claude'un bir aracı çalıştırmak ve sonuçları geri almak istediğini; `end_turn` ise Claude'un yanıtını tamamladığını ve döngünün sona ermesi gerektiğini gösterir.

Soru 59 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim log'ları, ajanın MCP araçlarınızdan gelen çıktıları yanlış yorumladığını gösteriyor: `get_customer`'dan Unix zaman damgaları, `lookup_order`'dan ISO 8601 tarihleri ve sayısal durum kodları (1=beklemede, 2=gönderildi). Bazı araçlar değiştiremeyeceğiniz üçüncü taraf MCP sunucuları. Veri formatı normalleştirmesi için en sürdürülebilir yaklaşım hangisidir?

Hangi yaklaşım en sürdürülebilirdir?

- A) Ajan işlemeden önce araç çıktılarını yakalayıp biçimlendirme dönüşümleri uygulamak için bir PostToolUse hook'u kullanmak. **[DOĞRU]**
- B) Kontrol ettiğiniz araçları insan tarafından okunabilir formatlar döndürecek şekilde değiştirmek ve üçüncü taraf araçlar için sarmalayıcılar oluşturmak.
- C) Ajanın her veri getirmeden sonra değerleri dönüştürmek için çağırdığı bir `normalize_data` aracı oluşturmak.
- D) Sistem komutuna, her aracın veri kurallarını açıklayan ayrıntılı format dokümantasyonu eklemek.

Neden A: Bir PostToolUse hook'u, ajanın işlemesinden önce tüm araç çıktılarını, üçüncü taraf MCP sunucusu verileri dahil, yakalayıp normalleştirmek için merkezi ve deterministik bir nokta sağlar. Dönüşümlerin kodda yaşaması ve LLM yorumuna güvenmek yerine tek biçimde uygulanması nedeniyle daha sürdürülebilirdir.

Soru 60 (Senaryo: Müşteri Destek Ajanı)

Durum: Üretim log'ları, özellikle “son alışverişimle ilgili yardıma ihtiyacım var” gibi belirsiz sorgularda ajanın bazen `get_customer` seçtiğini, oysa `lookup_order`'ın daha uygun olacağını gösteriyor. Araç seçimini iyileştirmek için sistem komutuna few-shot örnekler eklemeye karar veriyorsunuz. Hangi yaklaşım problemi en etkili şekilde ele alır?

Hangi yaklaşım en etkilidir?

- A) Her araç açıklamasına, belirsiz vakaları kapsayan açık “ne zaman kullanılır” ve “ne zaman kullanılmaz” rehberliği eklemek.
- B) Örnekleri araca göre gruplamak: önce tüm `get_customer` senaryoları, sonra tüm `lookup_order` senaryoları.
- C) Belirsiz senaryolara hedeflenmiş 4–6 örnek eklemek ve her birinde neden olası alternatifler yerine bir aracın seçildiğine dair gerekçe vermek. **[DOĞRU]**
- D) Her araç için tipik senaryolarda doğru araç seçimini gösteren açık ve belirsiz olmayan isteklerden 10–15 örnek eklemek.

Neden C: Few-shot örnekleri, hataların oluştuğu belirli belirsiz senaryolara hedeflemek ve bir aracın alternatiflere neden tercih edildiğine dair açık gerekçe vermek, modele uç durumlar için gereken karşılaştırmalı karar sürecini öğretir. Bu, genel örneklerden veya bildirimsel kurallardan daha etkilidir.

Soru 61 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: `remove_team_member` aracınız, yürütmeden önce etkileri önizlemek için `dry_run: boolean` parametresini kullanıyor. Üretim izlemesi, ajanın doğrudan `dry_run=false` ile çağırarak önizleme adımını atladığını gösteriyor. Her kaldırma işleminin, kullanıcının açıkça onayladığı bir önizleme ile başlamasını sağlamanız gerekiyor.

En güvenilir yaklaşım nedir?

- A) `dry_run=false` değerine yalnızca son 60 saniye içinde aynı parametrelerle `dry_run=true` çağrısı gerçekleşmişse izin veren sunucu tarafı doğrulaması eklemek.
- B) Aracı onay gerektiriyor olarak işaretlemek ve orkestrasyon katmanını, işaretli araçlara yapılan herhangi bir çağrıyı iletmeden önce kullanıcıdan onay isteyecek şekilde yapılandırmak.
- C) Araç açıklamasına, ajanın her zaman önce `dry_run=true` ile çağırmasını ve tekrar çağırmadan önce kullanıcı onayını beklemesini gerektiren ayrıntılı yönergeler ve few-shot örnekler eklemek.
- D) İki araçla değiştirmek: `preview_remove_member` etki ayrıntılarını ve tek kullanımlık bir onay token'ı döndürür; `execute_remove_member` bu token'ı gerektirir ve yürütmeyi önizlemeye bağlar. **[DOĞRU]**

Neden D: İki araçlı token bağlama yaklaşımı, önceden önizleme olmadan yürütmeyi mimari olarak imkânsız kılar; yürütme aracı, yalnızca önizleme aracının üretebildiği bir token'ı kelimenin tam anlamıyla gerektirir. Bu, kısıtı LLM'in yönergelere uyumuna (C), zamanlama sezgisellerine (A) veya orkestrasyon altyapısına (B) güvenmek yerine kod düzeyinde zorunlu kılan tek yaklaşımdır.

Soru 62 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Üretim izlemesi, `search_catalog` aracınızın zamanın %12'sinde başarısız olduğunu gösteriyor: %8'i yeniden denendiğinde başarılı olan ağ zaman aşımı, %4'ü ise kaç kez yeniden denenirse denensin başarılı olmayan sorgu söz dizimi hatası. Şu anda her iki hata türü de aynı şekilde döndürülüyor ve bu boşa yeniden denemelere yol açıyor.

Aracın hata yönetimini nasıl değiştirmelisiniz?

- A) Sistem komutunuza, ağ hatalarını söz dizimi hatalarından nasıl ayırt edeceğinizi gösteren few-shot örnekler eklemek.
- B) Tüm hatalara tek biçimde üstel geri çekilmeli yeniden deneme mantığı uygulamak.
- C) Ağ zaman aşımları için aracın içinde backoff ile otomatik yeniden deneme uygulamak; söz dizimi hatalarını parametre doğrulama ayrıntılarıyla hemen döndürmek. **[DOĞRU]**
- D) Tüm hataları bir `retryable` boolean bayrağı ve hata türü ayrıntılarıyla döndürmek.

Neden C: Geçici hatalar için yeniden denemeleri araç düzeyinde ele almak doğru soyutlama sınırıdır; araç hata türü hakkında kesin bilgiye sahiptir ve ajanın bir bayrağı yorumlamasına (D) veya prompt düzeyi yönergeleri izlemesine (A) güvenmeden deterministik yeniden deneme mantığı uygulayabilir. Tek biçimli backoff (B), asla başarılı olmayacak söz dizimi hatalarında zaman kaybettirir.

Soru 63 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Yatırım stratejisini tartışan birkaç tur boyunca bir kullanıcı “Risk toleransım çok düşük” dedi ve daha sonra “Getirilerimi maksimize etmek istiyorum” diye belirtti. Şimdi şunu soruyor: “Neye yatırım yapmalıyım?”

Hangi yaklaşım, önerinin kullanıcının gerçek önceliğiyle uyumlu olmasını en iyi sağlar?

- A) Çelişkiyi görünür kılın ve kullanıcıdan hangisinin daha önemli olduğunu netleştirmesini isteyin. **[DOĞRU]**
- B) Her iki senaryo için ayrı öneriler sunun.
- C) En son belirtilen tercihle ilerleyin.
- D) Çatışmayı ele almadan dengeli bir portföy önerin.

Neden A: Kullanıcı tercihleri birbiriyle doğrudan çeliştiğinde, çatışmayı görünür kılmak ve netleştirme istemek, önerinin kullanıcının gerçek niyetiyle uyumlu olmasını garanti etmenin tek yoludur. Diğer tüm yaklaşımlar yanlış olabilecek bir varsayım yapmayı gerektirir; getirileri maksimize etmek ve düşük risk toleransı, temelde uyumsuz hedeflerdir ve insan kararı gerektirir.

Soru 64 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Kullanıcılar çalma listesi tercihlerini birden fazla konuşma turunda rafine ediyor. Bir kullanıcı “Cazı çok seviyorum” dedikten iki mesaj sonra Claude “Hangi türlerden hoşlanırsınız?” diye soruyor.

En olası neden nedir?

- A) Claude'un konuşma belleğini sürdürmek için bir vektör veritabanı bağlantısına ihtiyacı vardır.
- B) Modelin bağlam penceresi aşılmıştır.
- C) Claude API bir `session_id` parametresi gerektirir.
- D) Uygulamanız önceki mesajları `messages` dizisine dahil etmiyor. **[DOĞRU]**

Neden D: Claude'un sunucu tarafı belleği yoktur; her API çağrısı durumsuzdur. Her istekte tam konuşma geçmişi `messages` dizisine dahil edilmezse Claude önceki turları bilemez. Vektör veritabanları (A) ve `session_id` (C), Claude'un mimarisinin parçası değildir; iki mesajlık alışverişlerde bağlam penceresi taşması (B) imkânsızdır.

Soru 65 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: 40 dakikalık bir yemek pişirme oturumundan sonra konuşma 78,000 token'a ulaşıyor. Geçmişte alerjiler, tarif ölçekleme, netleştirilmiş mutfak terimleri ve genel tartışma yer alıyor. Önemli bilgileri korurken token sayısını azaltmanız gerekiyor.

Hangi yaklaşım, koruma ile token azaltma arasında en iyi dengeyi kurar?

- A) Tüm konuşma geçmişini özetleyin.
- B) Yalnızca en son 20,000 token'ı tutun.

- C) Kritik yapılandırılmış verileri (alerjiler, miktarlar, tercihler) çıkarın, genel tartışmayı özetleyin ve yakın tarihli alışverişleri birebir koruyun. **[DOĞRU]**
- D) Tam konuşmayı dışarıda saklayın ve ilgili bölümleri semantik arama yoluyla alın.

Neden C: Hibrit yaklaşım, en yüksek değerli bilgiyi en düşük maliyetle korur. Alerjiler ve tarif miktarları gibi kritik olgular kompakt bir yapılandırılmış bloğa çıkarılır (özetleme sırasında oluşan hassasiyet kaybını önler), genel tartışma özetlenir ve yakın tarihli alışverişler konuşma tutarlılığı için birebir tutulur. A ve B seçenekleri kritik beslenme bilgilerini kaybetme riski taşır; D ise tek bir yemek pişirme oturumu için mimari açıdan gereğinden ağırdır.

Soru 66 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Kullanıcılar, uzun konuşmalarda asistanın önceki konuları ve tercihleri izlemeyi kaybettiğini bildiriyor. Mevcut uygulamanız yalnızca son 25 mesaj çiftini tutuyor.

En etkili çözüm nedir?

- A) Hibrit yaklaşım: eski mesajları özetlerken yakın tarihli olanları birebir tutun. **[DOĞRU]**
- B) Tam konuşma geçmişi üzerinde vektör benzerliği araması.
- C) Pencereyi 50 mesaj çiftine çıkarın.
- D) Atılan mesajları her turda özetleyin ve çalışan özeti başa ekleyin.

Neden A: Hibrit yaklaşım sorunun iki boyutunu da ele alır: yakın tarihli bağlamı aynen korur (konuşma tutarlılığı için kritiktir) ve önceki tercihlerin sıkıştırılmış bir temsilini sürdürür (çiftler atıldığında tamamen kaybolmalarını önler). Pencereyi artırmak (C) aynı sorunu yalnızca geciktirir. Vektör araması (B), mevcut sorguyla semantik olarak benzer olmayan önemli bağlamı kaçırabilir. Her turda tam özetleme (D) ek yük getirir ve özetleme hatalarını biriktirir.

Soru 67 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Kullanıcılar, konuşmalar 50 turu aştığında gecikmenin arttığını ve maliyetlerin yükseldiğini bildiriyor.

Birincil neden nedir?

- A) Her API isteğine tüm konuşma geçmişi dahil edilir. **[DOĞRU]**
- B) Model giderek daha uzun yanıtlar üretir.
- C) Geçmiş büyüdükçe veritabanı işlemleri yavaşlar.
- D) Model daha fazla işlem gerektiren dahili bir kullanıcı profili oluşturur.

Neden A: Claude API'si tamamen durumsuzdur; her istek, tam konuşma geçmişini `messages` dizisinde içermelidir. Konuşmalar büyüdükçe her istek daha fazla token taşır; bu da hem işleme gecikmesini hem de maliyeti doğrudan artırır. Model çağrılar arasında herhangi bir dahili durum tutmaz (D yanlıştır) ve yanıt uzunluğu doğası gereği konuşma uzunluğuna bağlı değildir (B).

Soru 68 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Üç ay boyunca haftalık oturumlardan sonra konuşma geçmişi 85,000 token'a ulaşıyor. Kullanıcı “Yalıtılmışlık teması hakkında ne sonuca varmıştık?” diye sorduğunda asistan, önceki tartışmalara atıf yapmak yerine genel yanıtlar veriyor.

En etkili yaklaşım nedir?

- A) Kayan pencere kırpması.
- B) Temel sonuçları yakalayan kademeli özetleme.
- C) İlgili alışverişlerin getirilmesiyle semantik embedding'ler. **[DOĞRU]**
- D) Tartışma sonuçlarını işaretleyen yapılandırılmış XML etiketleri ekleyin.

Neden C: Konuşma geçmişi üzerinde semantik arama, üç aylık tartışmaya ölçeklenirken belirli ilgili alışverişleri talep üzerine yüzeye çıkarabilen tek yaklaşımdır. Kayan pencere (A) geçmişin çoğunu atar. Kademeli özetleme (B), tartışmaları kullanıcıların sorduğu belirli sonuçları kaybettiren soyutlamalara sıkıştırır. XML etiketleri (D), tüm geçmiş içeriğin yeniden yapılandırılmasını gerektirir ve bu ölçekte getirme problemini çözmez.

Soru 69 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: QA testi sırasında Claude ilk 10–15 turda sistem komutu yönergelerini izliyor, ancak sonraki yanıtlar sapıyor. Konuşma hâlâ token sınırları içinde.

En iyi çözüm nedir?

- A) Davranış yönergelerini ilk kullanıcı mesajına taşıyın.
- B) 20 turdan sonra yeni bir konuşma başlatın.
- C) Konuşma kırılma noktalarında yönergeleri pekiştiren kullanıcı rolü mesajları ekleyin. **[DOĞRU]**
- D) Uyumlu olmayan yanıtları yeniden üretmek için yanıt sonrası doğrulama kullanın.

Neden C: Davranış hatırlatmalarının periyodik olarak eklenmesi, konuşma geçmişi biriktikçe kısıtları düzenli aralıklarla yeniden tesis ederek talimat kaymasıyla doğrudan mücadele eder. Yönergeleri ilk kullanıcı mesajına taşımak (A) otoritelerini azaltır. Yeni konuşma başlatmak (B) bağlamı yok eder. Yanıt sonrası doğrulama (D) önleyici değil düzelticidir ve ciddi gecikme ekler.

Soru 70 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: AI eğitmeninizin öğretim metodolojisini ve uyarlama kurallarını tanımlayan 2,800 token'lık bir sistem komutu var. 12 turdan sonra asistan yeterlilik seviyelerini göz ardı etmeye başlıyor.

En etkili düzeltme nedir?

- A) Her 4–5 turda bir hatırlatmalar ekleyin.
- B) Ayrıntılı kuralları, yeterlilik seviyesi uyarlamasını gösteren few-shot örneklerle değiştirin. **[DOĞRU]**
- C) Kritik kuralları sistem komutunun sonuna yerleştirin.

- D) Yanıtları değerlendirin ve zorluk seviyesi uyuşmuyorsa yeniden üretin.

Neden B: Bildirime dayalı kurallardan oluşan 2,800 token'lık bir sistem komutu kaymaya açıktır; çünkü soyut kurallar modelin her turda onlar hakkında akıl yürütmesini gerektirir. Ayrıntılı kuralları, doğru yeterlilik seviyesi uyarlamasını gösteren somut few-shot örneklerle değiştirmek, modele eşleştireceği net davranış kalıpları verir; bu, çok sayıda tur boyunca soyut talimatlardan daha güvenilir biçimde izlenir. Hatırlatma ekleme (A) yardımcı olur ama belirtileri ele alır; sona yerleştirme (C) başlangıçta yardımcı olur ama tur düzeyindeki kaymada yeterli değildir; yeniden üretme (D) pahalıdır ve düzelticidir.

Soru 71 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Asistanınızın coşkulu bir tonu sürdürmesi, akıl yürütmesini açıklaması ve netleştirici sorular sorması gerekiyor. Bu davranış yönergeleri nerede tanımlanmalıdır?

Bu davranış yönergeleri nerede tanımlanmalıdır?

- A) Her kullanıcı mesajının başına eklenmelidir.
- B) Sistem komutunda. **[DOĞRU]**
- C) İlk asistan mesajında.
- D) Ortam değişkenlerinde.

Neden B: Sistem komutu, tüm konuşma boyunca geçerli olan kalıcı davranış kısıtları ve yönergeleri için özel olarak tasarlanmıştır. Her kullanıcı mesajının başına eklemek (A) gereksiz ek yüktür. İlk asistan mesajı (C) güvenilir değildir; çünkü model kendi önceki ifadelerinden sapabilir. Ortam değişkenlerinin (D) model davranışı üzerinde etkisi yoktur.

Soru 72 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Kullanıcılar “Elbette!” ve “Yardımcı olmaktan memnuniyet duyarım!” gibi tekrarlı yanıt açılışları bildiriyor.

En etkili yaklaşım nedir?

- A) Doğrudan bir yanıt açılışıyla kısmi bir asistan mesajı ekleyin. **[DOĞRU]**
- B) Temperature ayarını düşürün.
- C) Selamlamaları kaldırmak için yanıtları sonradan işleyin.
- D) Bu ifadelerden kaçınmak için sistem komutu talimatları ekleyin.

Neden A: Asistan yanıtını doğrudan bir cevabın başlangıcıyla önceden doldurmak, selamlama kalıplarını üretim düzeyinde engeller; model yeni açılış ifadeleri üretmek yerine ön doldurmadan devam eder. Sistem komutu talimatları (D) yardımcı olabilir, ancak model yine de varyantlar üretebileceği için daha az güvenilirdir. Sonradan işleme (C) kırılgan bir geçici çözümdür. Temperature (B) rastlantısallığı kontrol eder, belirli ifade kalıplarını değil.

Soru 73 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Kullanıcı aktif olarak sohbet ederken bir webhook, sisteminize kullanıcının paketinin kargoya verildiğini bildiriyor. Asistanın bunu bir sonraki yanıta doğal biçimde dahil etmesini istiyorsunuz.

En iyi yaklaşım nedir?

- A) Kargo durumunu sistem komutuna ekleyin.
- B) Hemen sentetik bir kullanıcı mesajı gönderin.
- C) Asistanı her turda bir durum aracı çağırmaya zorlayın.
- D) Durum güncellemesini bir sonraki kullanıcı mesajının ön eki olarak ekleyin. **[DOĞRU]**

Neden D: Durum güncellemesini bir sonraki kullanıcı mesajının ön eki yapmak, gerçek zamanlı bağlamı akışı bozmadan doğal bir konuşma sınırında enjekte eder. Sistem komutunu değiştirmek (A), oturumu yeniden kurmayı gerektirir veya mimari olarak zahmetlidir. Sentetik kullanıcı mesajı (B), doğal diyalog akışını bozabilir ve atfı karıştırabilir. Her turda araç çağrısını zorunlu kılmak (C), olaylar nadirken israftır.

Soru 74 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Kullanıcılar sık sık “Parti için bir mekan ayır” gibi istekler gönderiyor. Asistan 4+ netleştirici soru soruyor ve bu 35% terk oranına yol açıyor.

Hangi yaklaşım bu ödünleşimi en iyi iyileştirir?

- A) Gizli varsayımlarla ilerleyin.
- B) Tüm netleştirici soruları tek bir birleşik mesajda sorun.
- C) Varsayımları açıkça belirtin ve düzeltmeleri davet ederek ilerleyin. **[DOĞRU]**
- D) Yapılandırılmış bir bilgi alma formu kullanın.

Neden C: Varsayımları açıkça belirtmek ve ilerlemek, kullanıcıya anında yararlı bir yanıt verirken yanlış varsayımları düzeltme olanağını korur. Gizli varsayımlar (A), kullanıcının neyin varsayıldığından habersiz kalmasına neden olur. Birleşik soru listesi (B) hâlâ kullanıcıdan baştan çaba ister. Yapılandırılmış form (D), daha az değil daha fazla sürtünme ekler ve terk oranını azaltma hedefiyle çelişir.

Soru 75 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Asistanınız yüklenici personası içeren bir sistem komutu kullanıyor. İlk turlar kuralları izliyor, ancak 7. turda asistan genel tavsiyeler veriyor. Konuşma uzunluğu yalnızca 2,500 token.

En olası neden nedir?

- A) Sistem komutları yalnızca başlangıç davranışını oluşturur.
- B) Turlar biriktikçe model dikkati zayıflar.
- C) Biriken asistan yanıtları sistem komutunun etkisini seyreltir. **[DOĞRU]**
- D) Sistem komutu yalnızca bir kez gönderilir.

Neden C: Asistan yanıtları konuşma geçmişinde biriktikçe, sistem komutunun davranış kısıtlarını yansıtan metnin oranı, asistan tarafından üretilen ve büyüyen içerik gövdesine göre azalır. Model, sistem komutundan çok kendi önceki çıktılarına giderek daha fazla kalıp eşleştirir; bu da kısa token uzunluklarında bile kaymayı bileşik hale getirir. Sistem komutu her API çağrısına dahil edilir (D tek başına açıklama olarak yanlıştır) ve model dikkat zayıflaması (B) 2,500 token'da bu şekilde işlemez.

Soru 76 (Senaryo: Konuşmaya Dayalı AI Mimari Kalıpları)

Durum: Kullanıcılar “Rapor konusunda yardım edebilir misin?” gibi belirsiz istekler soruyor. Asistan birden fazla soru sorarak yanıtlıyor (hangi rapor? ne tür yardım? son tarih?) ve bu 40% terk oranına yol açıyor.

En iyi çözüm nedir?

- A) Makul varsayımlar yapın, bunları açıkça belirtin ve ayarlamayı teklif edin. **[DOĞRU]**
- B) Yanıtlamadan önce belirsizliği daha küçük bir modelle sınıflandırın.
- C) Varsayımları belirtmeden önceden tanımlı yorumlar kullanın.
- D) Asistanı tur başına bir netleştirici soruyla sınırlayın.

Neden A: Makul ve açıkça belirtilmiş varsayımlarla ilerlemek, kullanıcıyı bilgilendirip kontrolü onda tutarken gidip gelmeyi tamamen ortadan kaldırır. Önceden tanımlı sessiz yorumlar (C), yanıt kullanıcının niyetiyle eşleşmediğinde kullanıcıları şaşırtır. Tek soru sınırı (D) yine de karşılıklı ek turlar gerektirir. Daha küçük bir sınıflandırma modeli (B), temel UX sorununu çözmeden gecikme ve altyapı karmaşıklığı ekler.

Uygulamalı Alıştırmalar

Alıştırma 1: Eskalasyon Mantığına Sahip Çok Araçlı Ajan

Hedef: Araç entegrasyonu, yapılandırılmış hata yönetimi ve eskalasyon içeren bir ajan döngüsü tasarlayın.

Adımlar:

1. Ayrıntılı açıklamalara sahip 3–4 MCP aracı tanımlayın (araç seçimini test etmek için iki benzer araç dahil edin)
2. `stop_reason` değerini (`"tool_use"` / `"end_turn"`) kontrol eden bir ajan döngüsü uygulayın
3. Yapılandırılmış hata yanıtları ekleyin: `errorCategory` , `isRetryable` , açıklama
4. Bir eşik üzerindeki operasyonları engelleyen ve eskalasyona yönlendiren bir interceptor hook'u uygulayın
5. Çok yönlü isteklerle test edin

Alanlar: 1 (Ajan mimarisi), 2 (Araçlar ve MCP), 5 (Bağlam ve güvenilirlik)

Alıştırma 2: Ekip Geliştirme İçin Claude Code'u Yapılandırma

Hedef: CLAUDE.md, özel komutlar, yola özgü kurallar ve MCP sunucuları yapılandırın.

Adımlar:

1. Evrensel standartlara sahip proje düzeyinde bir CLAUDE.md oluşturun
2. Farklı kod alanları için YAML frontmatter içeren `.claude/rules/` dosyaları oluşturun (`paths: ["src/api/**/*"]`, `paths: ["**/*.test.*"]`)
3. `.claude/skills/` altında `context: fork` ve `allowed-tools` içeren bir proje skill'i oluşturun
4. Ortam değişkenleriyle birlikte `.mcp.json` içinde bir MCP sunucusu + `~/.claude.json` içinde kişisel bir geçersiz kılma yapılandırın
5. Farklı karmaşıklıktaki görevlerde planlama modu ile doğrudan yürütmeyi test edin

Alanlar: 3 (Claude Code yapılandırması), 2 (Araçlar ve MCP)

Alıştırma 3: Yapılandırılmış Veri Çıkarma İşlem Hattı

Hedef: JSON şemaları, yapılandırılmış çıktı için `tool_use`, doğrulama/yeniden deneme döngüleri, toplu işleme.

Adımlar:

1. JSON şemasıyla bir çıkarım aracı tanımlayın (zorunlu/isteğe bağlı alanlar, "other" içeren enum'lar, nullable alanlar)
2. Bir doğrulama döngüsü oluşturun: hata durumunda belge, hatalı çıkarım ve belirli doğrulama hatasıyla yeniden deneyin
3. Farklı yapılara sahip belgeler için few-shot örnekler ekleyin
4. Message Batches API üzerinden toplu işleme kullanın: 100 belge, hataları `custom_id` ile ele alın
5. İnsanlara yönlendirin: alan düzeyinde güven skorları, belge türü analizi

Alanlar: 4 (Prompt mühendisliği), 5 (Bağlam ve güvenilirlik)

Alıştırma 4: Çoklu-Ajan Araştırma İşlem Hattı Tasarlama ve Hata Ayıklama

Hedef: Alt-ajan orkestrasyonu, bağlam aktarımı, hata yayılımı, kaynak takibiyle sentez.

Adımlar:

1. 2+ alt-ajana sahip bir koordinatör (`allowedTools` içinde `"Task"` yer alır, bağlam prompt'larda açıkça aktarılır)
2. Tek bir yanıtta birden fazla `Task` çağrısı üzerinden alt-ajanları paralel çalıştırın
3. Yapılandırılmış alt-ajan çıktısını zorunlu kılın: iddia, alıntı, kaynak URL'si, yayın tarihi
4. Bir alt-ajan zaman aşımını simüle edin: koordinatöre yapılandırılmış hata bağlamı döndürün ve kısmi sonuçlarla devam edin

5. Çelişkili verilerle test edin: her iki değeri de atıfla koruyun; doğrulanmış bulguları tartışmalı bulgulardan ayırın

Alanlar: 1 (Ajan mimarisi), 2 (Araçlar ve MCP), 5 (Bağlam ve güvenilirlik)

Ek: Teknolojiler ve Kavramlar

Teknoloji	Temel yönler
Claude Agent SDK	AgentDefinition, ajan döngüleri, `stop_reason`, hook'lar (PostToolUse), Task üzerinden alt-ajan başlatma, `allowedTools`
Model Context Protocol (MCP)	MCP sunucuları, araçlar, kaynaklar, `isError`, araç açıklamaları, `.mcp.json`, ortam değişkenleri
Claude Code	CLAUDE.md hiyerarşisi, glob kalıplarıyla `.claude/rules/`, `.claude/commands/`, SKILL.md ile `.claude/skills/`, planlama modu, `/compact`, `--resume`, `fork_session`
Claude Code CLI	Etkileşimsiz mod için `-p` / `--print`, `--output-format json`, `--json-schema`
Claude API	JSON şemalarıyla `tool_use`, `tool_choice` ("auto"/"any"/forced), `stop_reason`, `max_tokens`, sistem komutları
Message Batches API	50% tasarruf, 24 saate kadar pencere, `custom_id`, çok turlu araç çağırma yok
JSON Schema	Zorunlu vs isteğe bağlı, nullable alanlar, enum türleri, "other" + ayrıntı, katı mod
Pydantic	Şema doğrulama, semantik hatalar, doğrulama/yeniden deneme döngüleri
Yerleşik araçlar	Read, Write, Edit, Bash, Grep, Glob — amaç ve seçim kriterleri
Few-shot prompting	Belirsiz durumlar için hedeflenmiş örnekler, yeni kalıplara genelleme
Prompt zincirleme	Odaklanmış geçişlere ardışık ayrıştırma
Bağlam penceresi	Token bütçeleri, kademeli özetleme, "lost in the middle", not defteri dosyaları
Oturum yönetimi	Sürdürme, `fork_session`, adlandırılmış oturumlar, bağlam izolasyonu
Güven kalibrasyonu	Alan düzeyinde puanlama, etiketli kümelerde kalibrasyon, katmanlı örnekleme

---

Kapsam Dışı Konular

Aşağıdaki ilişkili konular sınavda yer **ALMAYACAKTIR**:

- Claude modellerine fine-tuning yapmak veya özel modeller eğitmek
- Claude API kimlik doğrulaması, faturalandırma veya hesap yönetimi
- Belirli programlama dilleri veya framework'lerde ayrıntılı uygulama (araç/şema yapılandırması için gerekenin ötesinde)
- MCP sunucularını dağıtma veya barındırma (altyapı, ağ, konteyner orkestrasyonu)
- Claude'un iç mimarisi, eğitim süreci veya model ağırlıkları
- Constitutional AI, RLHF veya güvenlik eğitimi metodolojileri
- Embedding modelleri veya vektör veritabanı uygulama ayrıntıları
- Bilgisayar kullanımı (tarayıcı otomasyonu, masaüstü etkileşimi)
- Görüntü analizi yetenekleri (Vision)
- Streaming API veya server-sent events
- Hız sınırlama, kotalar veya ayrıntılı API maliyet hesaplamaları
- OAuth, API anahtarı rotasyonu veya kimlik doğrulama protokolü ayrıntıları
- Bulut sağlayıcısına özgü yapılandırmalar (AWS, GCP, Azure)
- Performans benchmark'ları veya model karşılaştırma metrikleri
- Prompt önbelleğe alma uygulama ayrıntıları (var olduğunu bilmenin ötesinde)
- Token sayma algoritmaları veya tokenizasyon ayrıntıları

Hazırlık Önerileri

1. **Claude Agent SDK ile bir ajan oluşturun** — araç çağırma, hata yönetimi ve oturum yönetimi içeren eksiksiz bir ajan döngüsü uygulayın. Alt-ajanlar ve açık bağlam aktarımı üzerinde pratik yapın.
2. **Claude Code'u gerçek bir proje için yapılandırın** — CLAUDE.md hiyerarşisini, `.claude/rules/` içinde yola özgü kuralları, `context: fork` ve `allowed-tools` içeren skills'i ve MCP sunucusu entegrasyonunu kullanın.
3. **MCP araçları tasarlayıp test edin** — benzer araçları ayırt eden açıklamalar yazın, kategoriler ve yeniden deneme bayrakları içeren yapılandırılmış hatalar döndürün ve belirsiz kullanıcı isteklerine karşı test edin.
4. **Bir veri çıkarma işlem hattı oluşturun** — JSON şemalarıyla `tool_use`, doğrulama/yeniden deneme döngüleri, isteğe bağlı/nullable alanlar ve Message Batches API üzerinden toplu işleme kullanın.
5. **Prompt mühendisliği pratiği yapın** — belirsiz senaryolar için few-shot örnekler, açık inceleme kriterleri ve büyük kod incelemeleri için çok geçişli mimariler ekleyin.
6. **Bağlam yönetimi kalıplarını çalışın** — ayrıntılı çıktılardan olguları çıkarın, not defteri dosyaları kullanın ve bağlam sınırlarını yönetmek için keşfi alt-ajanlara delege edin.

7. **Eskalasyonu ve human-in-the-loop yaklaşımını anlayın** — ne zaman insana yönlendirmek gerektiğini (politika boşlukları, açık kullanıcı isteği, ilerleme sağlayamama) ve güvene dayalı yönlendirme iş akışlarını öğrenin.
8. **Gerçek sınavdan önce bir deneme sınavı çözün**. Aynı senaryoları ve formatı kullanır.