AIR LAB

GOOGLE ADK

NUR GÜMÜS
AI ENGINEER

AIR LAB

İÇERİK

- GİRİŞ
- AI ENGINEER
- AGENT NEDİR
- AGENT ANATOMİSİ
- BEYİN=LLM
- AGENT DESIGN PATTERNS
- TOOLBOX
- MULTI AGENTIC SİSTEMLER
- AGENTLARDA BİLİNMESİ GEREKEN KAVRAMLAR
- LIMITATIONS AND CHALLENGES
- GOOGLE ADK NEDİR
- GOOGLE ADK ÖZELLİKLERİ
- DİĞER TOOLLARLA KARŞILAŞTIRMA
- DEMO
- KİTAP, YOUTUBE KANALI VS KAYNAK ÖNERİLERİ
- Q&A

AIR LAB

GIRIS

- LLM'ler yeterince akıllılaştı (GPT, Gemini, GLM, Opus, Sonnet vs.)
- Function Calling standardı oturdu

  Şimdi: LLM direkt JSON döndürüyor, hangi tool'u çağıracağını biliyor
- Framework'ler olgunlaştı
- Maliyet 10x düştü
- Gerçek production use case'ler ortaya çıktı (Cursor, Copilot, other coding agents)
- MCP standartları (farklı appleri pipelinelara bağlama kolaylığı)

AIR LAB

AI ENGINEER

- 5 yıl önce bu title yoktu. Şimdi dünyanın en çok aranan mühendislik pozisyonu.

KLASİK ROLLER:

- Software Engineer → Kod yazar
- Data Scientist → Veriyi analiz eder
- ML Engineer → Model eğitir, deploy eder
- AI Engineer → Agent & LLM uygulamaları yapar

AI ENGINEER NE YAPAR?

- LLM'leri uygulamaya entegre eder
- Agent sistemleri tasarlar ve geliştirir
- Prompt engineering & optimization
- RAG pipeline'ları kurar
- Tool & API entegrasyonları yapar
- Production'a alır, monitor eder

ML Engineer	AI Engineer
Model eğitir	Model kullanır
Matematik ağır	Mühendislik ağır
Pytorch, TF vs.	Langchain, Google ADK, n8n vs.

AIR LAB

AGENT NEDIR?

Senaryo: Bir Ressam Agent İstiyorsunuz

Amacınız bir AI Agent'ın sizin için bir resim çizmesi. Bu agentın nelere ihtiyacı var?

Tools
(Tuval, Fırça, Boya vs.)

Instruction
(Fırçayı al, boyayı bandır, tuvale sür vs.)

Memory
(Kullanıcı rönesans eserlerini seviyordu)

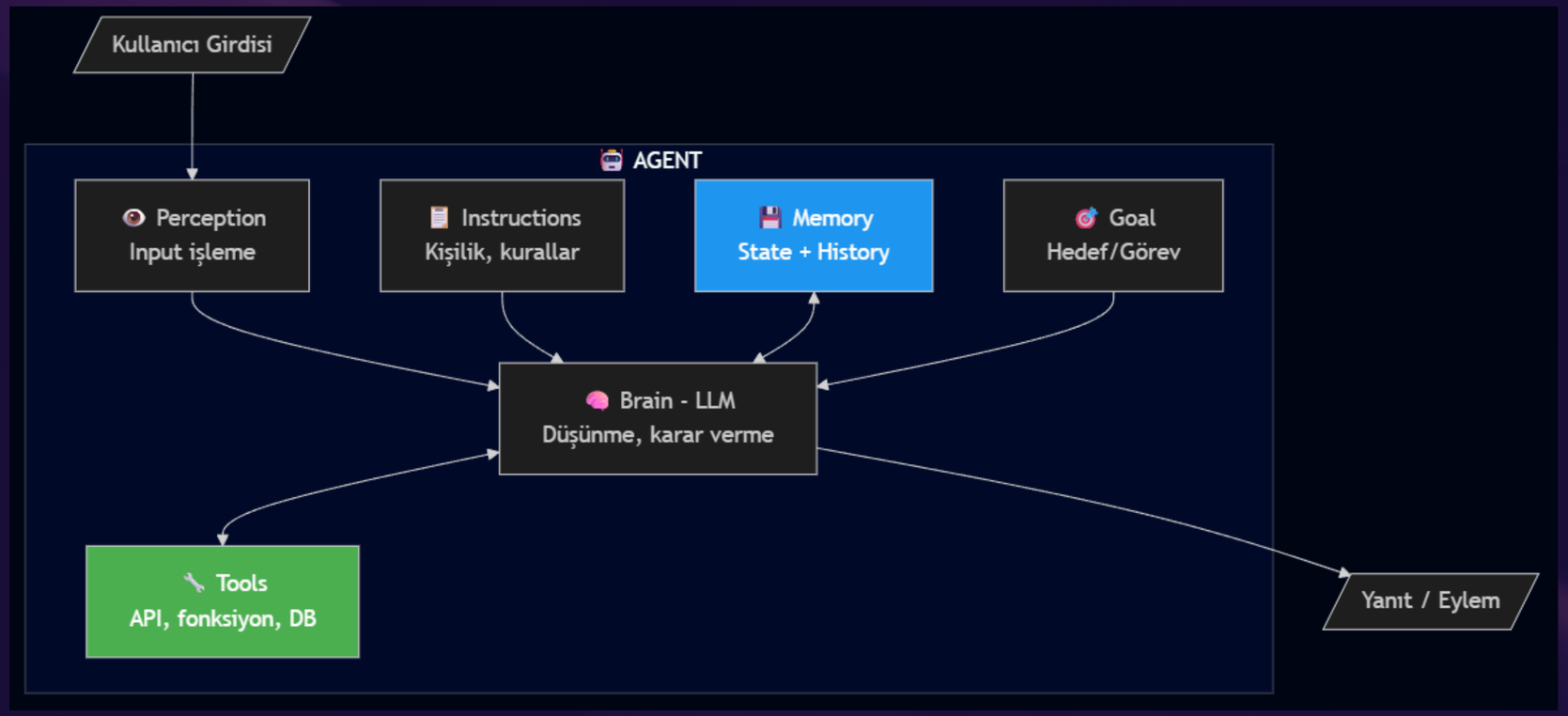
Session
(Mesai Saati)

Human in the loop
(İstediğinde sorabilirsin)

AIR LAB

AGENT NEDIR?

Agent = LLM + Tools + Reasoning Loop + Memory



Bir hedefe ulaşmak için otonom olarak planlama yapabilen,
araçları kullanabilen ve çevresine tepki verebilen AI sistemi

AIR LAB

AGENT MIMARISI

1. PERCEPTION (Algılama)
   - Kullanıcıdan input alma
   - Dosya okuma, API'den veri çekme
   - 'Dış dünyayı görme'

2. BRAIN (Beyin = LLM)
   - Karar verme merkezi
   - 'Ne yapmalıyım?' sorusunu cevaplıyor
   - En kritik bileşen

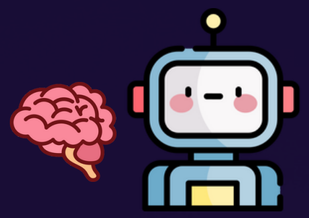
3. MEMORY (Hafıza)
   - Kısa vadeli: Bu conversation'da ne konuştuk?
   - Uzun vadeli: Bu kullanıcı kim, ne sever?

4. TOOLS (Araçlar)
   - Dış dünyaya etki etme
   - API çağırma, dosya yazma, email atma

5. OUTPUT (Çıktı)
   - Kullanıcıya yanıt
   - Veya bir aksiyon (dosya oluşturma, email gönderme)

AIR LAB

BEYIN = LLM



Bazı Önemli Paperlar

Chain-of-Thought (CoT) Prompting
"Chain-of-Thought Prompting Elicits Reasoning in Large Language Models"
Ana Fikir:
"LLM'lere adım adım düşünmesini söylersen, reasoning performansı dramatik şekilde artar"
Anahtar Bulgular:

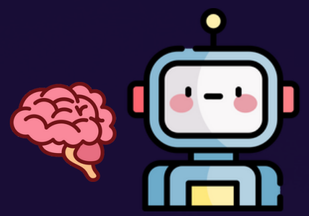
- "Let's think step by step" → %50+ performans artışı (matematik problemlerinde)
- Sadece büyük modellerde çalışıyor (100B+ parametre)
- Fine-tuning gerektirmiyor, sadece prompt değişikliği
- GSM8K benchmark'ta fine-tuned GPT-3'ü bile geçti

Örnek:

❌ Standart: "Roger'ın 5 tenisi var. 2 kutu daha aldı, her kutuda 3 top. Kaç topu var?"
→ Cevap: 11 (yanlış)
✅ CoT: "Adım adım düşünelim. Roger 5 topla başladı. 2 kutu aldı, her kutuda 3 top = 6 top. 5 + 6 = 11"
→ Cevap: 11 (doğru, ama reasoning gösteriliyor)

AIR LAB

BEYIN = LLM



Bazı Önemli Paperlar

ReAct (Reasoning + Acting)
"ReAct: Synergizing Reasoning and Acting in Language Models" https://arxiv.org/abs/2210.03629
"Reasoning ve Acting'i birleştir: Düşün → Eylem yap → Gözlemle → Tekrarla"
Neden Önemli:

- CoT'un eksikliği: Sadece düşünüyor, dış dünyayla etkileşim yok
- ReAct: Düşünme + Araç kullanma (Wikipedia API, hesaplama, vs.)
- Hallucination azalıyor çünkü gerçek veriye erişiyor
- Agent'ların teorik temeli bu paper

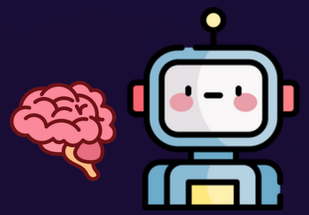
ReAct Döngüsü:

Thought: "Colorado'nun başkenti nedir öğrenmeliyim"
Action: Search["Colorado capital"]
Observation: "Denver is the capital of Colorado"
Thought: "Denver'ın nüfusunu bulmalıyım"
Action: Search["Denver population"]
...
Sonuçlar:

- HotpotQA: +%6 accuracy (CoT'a göre)
- ALFWorld: +%34 success rate
- WebShop: +%10 success rate

AIR LAB

BEYIN = LLM



Bazı Önemli Paperlar

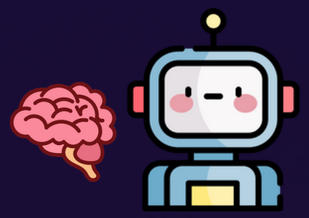
Tree of Thoughts (ToT)
"Tree of Thoughts: Deliberate Problem Solving with Large Language Models" https://arxiv.org/abs/2305.10601
Ana Fikir:
"Tek bir düşünce zinciri yerine, birden fazla yolu dene ve en iyisini seç"
CoT vs ToT:
CoT: A → B → C → D (tek yol)
ToT: B1 → C1 ✗
↗
A → B2 → C2 → D ✓
↘
B3 → C3 ✗

Neden Önemli:

- Backtracking (geri dönüş) mümkün
- Paralel keşif
- Self-evaluation: "Bu yol umut verici mi?"
- Karmaşık problemlerde (24 oyunu, sudoku) çok etkili

AIR LAB

BEYIN = LLM



Bazı Önemli Paperlar

PAPER	ANA KATKI	LİNK
SELF-CONSISTENCY (WANG ET AL., 2022)	BIRDEN FAZLA COT ÇALIŞTIR, EN TUTARLI CEVABI SEÇ	ARXIV.ORG/ABS/2203.11171
LEAST-TO-MOST (ZHOU ET AL., 2022	PROBLEMI ALT PROBLEMLERE BÖL	ARXIV.ORG/ABS/2205.10625
REFLEXION (SHINN ET AL., 2023)	AGENT HATALARINDAN ÖĞRENIR	ARXIV.ORG/ABS/2303.11366
TOOLFORMER (SCHICK ET AL., 2023)	LLM KENDI KENDINE TOOL KULLANMAYI ÖĞRENIR	ARXIV.ORG/ABS/2302.04761

AIR LAB

AGENT DESIGN PATTERNLARı

1. ReAct (Reason + Act)
   En basit ve yaygın pattern
   Düşün → Yap → Gözlemle → Tekrarla

2. Plan-and-Execute
   Önce tam plan yap, sonra execute et
   Daha uzun vadeli görevler için

3. Reflexion (Self-correction)
   Hata yaptığında kendini düzelt
   'Öğrenen agent'

4. Tool-use Patterns
   Hangi tool'u ne zaman çağıracağını bilme

AIR LAB

TOOLBOX

AGENT ORCHESTRATION

Google ADK, CrewAI, LangChain, LangGraph, n8n, AutoGen

DATABASE

MongoDB, Redis, PostgreSQL, SQLite

MONITORING & OBSERVABILITY

Langfuse, LangSmith, Weights & Biases, OpenTelemetry

API LAYER

Flask, FastAPI, Django REST

MCP

Tool'lar için USB-C standartı

VECTOR DATABASE

Pinecone, Qdrant, Chroma, pgvector

LLM PROVIDERS

OpenAI, Anthropic, Google Gemini, Ollama (local), Groq or local

DOCUMENT PROCESSING

Unstructured, PyPDF, LlamaParse, Docling, DeepParse

KNOWLEDGE GRAPH

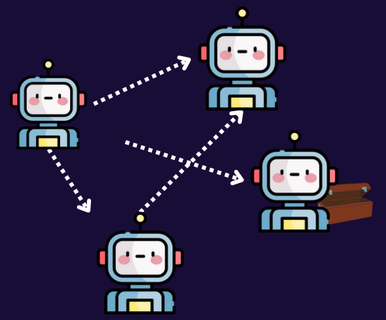
Neo4j, Amazon Neptune

FLOW PLANNING

Mermaid

AIR LAB

MULTI AGENTIC SISTEMLER



NEDEN MULTI-AGENT?

1. System Prompt Şişmesi
   Tek agent'a her şeyi öğretmek → 5000+ token prompt
   → Modelin kafası karışır

2. Uzmanlaşma (Separation of Concerns)
   Her agent TEK bir iş bilsin
   Frontend Agent, Backend Agent, Review Agent...

3. Maliyet Optimizasyonu
   Basit iş → Ucuz model (Gemini Flash)
   Zor iş → Pahalı model (GPT-4, Opus)

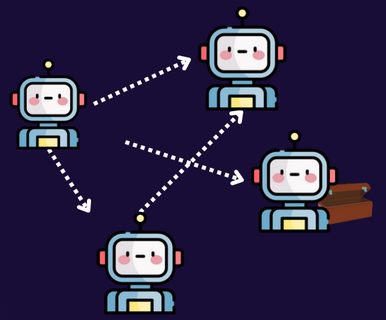
4. Paralel Çalışma Bağımsız agent'lar aynı anda çalışabilir

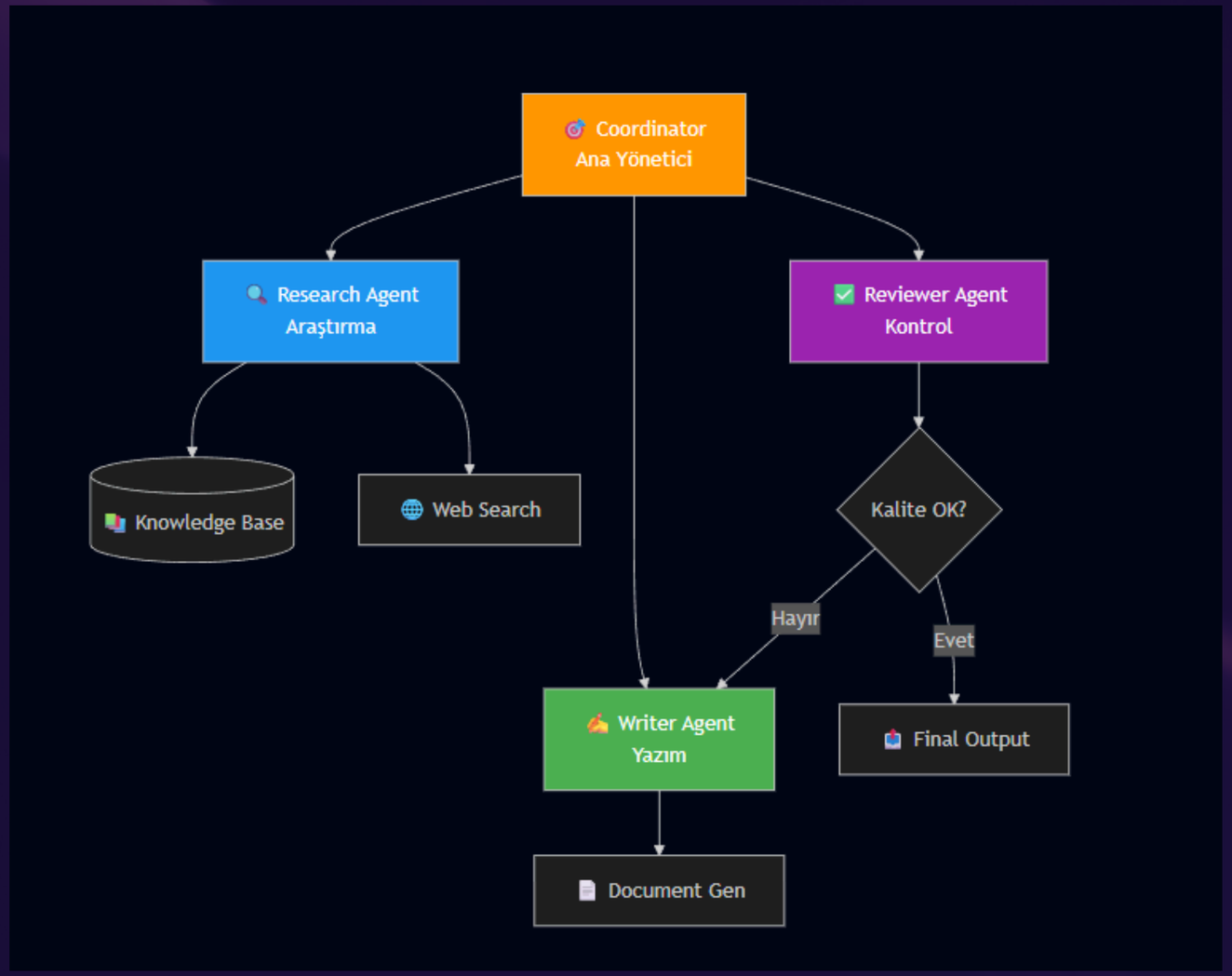
5. Debug & Test "Hangi agent hata yaptı?" sorusu cevaplanabilir

ÖRNEK: Cursor IDE - 8 agent'a kadar paralel çalışma - Frontend, Backend, Reviewer, DevOps agent'ları - Günlük milyarlarca completion

AIR LAB

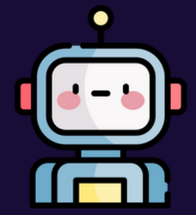
MULTI AGENTIC SISTEMLER





AIR LAB

PROMPT ENGINEERING



"System Prompt vs User Prompt:
- System: Agent'ın KİMLİĞİ (değişmez)
- User: Her istekte gelen mesaj

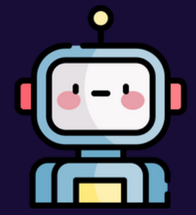
Few-shot Examples:
- 'Şöyle input gelirse, şöyle output ver'
- 2-3 örnek genellikle yeterli

Structured Output (JSON Mode):
- LLM'e 'JSON döndür' dersen, parse etmesi kolay
- Tool calling buna dayanıyor"

DsPy tarzı kütüphaneler

AIR LAB

TOKEN ECONOMICS



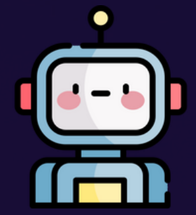
"Context Window:
- LLM'in bir seferde görebildiği maksimum token
- GPT-4 Turbo: 128K, Gemini: 1M+

Input vs Output Fiyatlandırma:
- Input (prompt) genellikle UCUZ
- Output (generation) genellikle PAHALI
- 10x fark olabilir!

Truncation:
- Context'e sığmıyorsa, eski mesajları kes
- Ya da özetle (summarization)"

AIR LAB

MODEL PARAMETRELERI



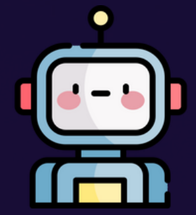
"Temperature:
- 0 = Deterministik (hep aynı cevap)
- 1 = Yaratıcı (her seferinde farklı)
- Agent'lar için genellikle 0-0.3

Top-p, Top-k:
- Hangi token'ları düşüneceğini sınırlar
- Genellikle default bırakılır

Max Tokens:
- Maksimum output uzunluğu
- Agent loop'larında dikkat: Çok uzun = pahalı"

AIR LAB

PRODUCTION CONCERNS



"Rate Limiting:
- API'ler saniyede X istek kabul eder
- Aşarsan 429 hatası

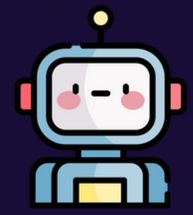
Retry Logic:
- API hata verirse, bekle ve tekrar dene
- Exponential backoff: 1s, 2s, 4s, 8s...

Cost Monitoring:
- Agent'lar loop'ta çalışır, maliyet hızla artar
- Max iteration, max token limitleri koy

Fallback:
- Ana model çökerse, yedek modele geç
- GPT-4 → GPT-3.5, Opus → Sonnet"

AIR LAB

MEMORY



MEMORY KATMANLARI

1. Short-term (Conversation History)
   - Bu oturumda ne konuştuk?
   - Context window içinde tutuluyor

2. Long-term (Persistent)
   - Kullanıcı tercihleri, geçmiş kararlar
   - Veritabanında saklanıyor

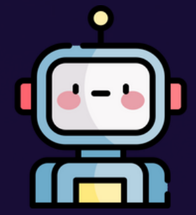
3. Episodic (Experience)
   - "Geçen sefer X yaptığımda Y olmuştu"
   - Reflexion pattern için

4. Semantic (Knowledge)
   - RAG ile dış bilgiye erişim
   - Vector database'de

⚠️ Memory = Context Window Yönetimi
Sonsuz hafıza yok, stratejik olmalı

AIR LAB

TOOLS



TOOL ÇAĞIRMA NASIL ÇALIŞIR?

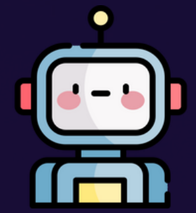
1. LLM tool listesini görür
2. "Bu görevi yapmak için X tool'u lazım" der
3. Framework tool'u çağırır
4. Sonuç LLM'e geri döner
5. LLM sonucu yorumlar

ÖRNEK:
User: "Hava nasıl?"
LLM: <tool_call>weather_api("Istanbul")</tool_call>
System: {"temp": 18, "condition": "sunny"}
LLM: "İstanbul'da hava güneşli, 18°C"

⚠️ LLM SADECE GEREKTİĞİNDE TOOL KULLANMALI
Basit soru → Direkt cevap
Dış veri lazım → Tool çağır

AIR LAB

SESSION



SESSION YÖNETİMİ

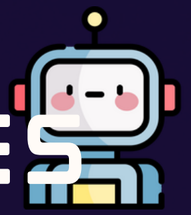
Neden önemli?
- Agent sonsuz çalışmamalı
- Maliyet kontrolü
- Timeout handling

LIMITLER:
Max Iterations: Kaç loop dönebilir?
Max Tokens: Ne kadar harcayabilir?
Timeout: Kaç saniye/dakika çalışabilir?
Cost Limit: Max $ harcama

BEST PRACTICE:
- Her session'a unique ID ver
- Logları sakla (debugging için)
- Graceful shutdown (yarıda kalan iş için)

AIR LAB

LIMITATIONS AND CHALLENGES



HALLUCINATION:
LLM yanlış bilgi uydurabilir.
Çözüm: Tool ile doğrulama, RAG kullanımı

INFINITE LOOP:
Agent aynı şeyi tekrar tekrar yapabilir.
Çözüm: Max iteration limiti, loop detection

MALIYET:
Her LLM çağrısı para. Loop = çok para.
Çözüm: Ucuz model tercih et, cache kullan

LATENCY:
Her LLM çağrısı 1-5 saniye.
5 turlu agent = 5-25 saniye bekleme
Çözüm: Streaming, paralel çağrılar

PROMPT INJECTION:
Kullanıcı agent'ı manipüle edebilir.
Çözüm: Input validation, sandboxing

EVALUATION:
'Agent iyi çalışıyor mu?' ölçmek zor.
Çözüm: Benchmark, A/B test, human eval, LLM AS A JUDGE

AIR LAB

OVERENGINEERING

⚠️ LLM HER ZAMAN GEREKLİ Mİ?

HAYIR!

❌ LLM GEREKMEZ:
- JSON parse
- Tarih hesaplama
- Basit API çağrısı
- Format dönüştürme

✅ LLM GEREKIR:
- Belirsiz input anlama
- Karar verme (trade-off)
- Özet/sentez
- Doğal dil → yapı

KURAL: Klasik kod yapabiliyorsa → klasik kod yap

Klasik otomasyon, algoritmalar karşılayabiliyorsa; onları kullan.

AIR LAB

GOOGLE ADK

GOOGLE ADK NEDİR?

Agent Development Kit
- Google'ın açık kaynak agent framework'ü
- Python-first
- Gemini-optimized (ama diğer LLM'ler de çalışır)

TEMEL BİLEŞENLER:
├── Agent: Ana class
├── Tool: Fonksiyon tanımlama
├── Runner: Çalıştırma
└── Session: State yönetimi

BASİT ÖRNEK:

```
from google.adk import Agent, Tool

@Tool
def get_weather(city: str) -> str:
 return f"{city}: 18°C, güneşli"

agent = Agent(
 model="gemini-2.0-flash",
 tools=[get_weather],
 instruction="Sen bir hava durumu
asistanısın"
)
```

AIR LAB

GOOGLE ADK

ADK'DA HER ŞEY HAZIR

Diğer framework'lerde KENDİN YAZARSIN:

- Konuşma geçmişi? → Kendin tut
- Kullanıcı tercihleri? → Kendin kaydet
- Oturumlar arası hafıza? → Kendin tasarla
- Veritabanı bağlantısı? → Kendin kur
- State yönetimi? → Kendin implement et

ADK'da HEPSİ BUILT-IN:

- Session → Konuşma takibi
- State → Veri saklama (4 farklı scope!)
- Memory → Uzun vadeli hafıza
- SessionService → Persistence
- MemoryService → Arama & geri çağırma

AIR LAB

GOOGLE ADK: SESSION

SESSION = BİR KONUŞMA THREAD'İ

Her kullanıcı etkileşimi = 1 Session

Session İçeriği:

- id → Benzersiz session ID
- user_id → Kim konuşuyor?
- app_name → Hangi agent?
- events → Konuşma geçmişi (kronolojik)
- state → Agent'ın not defteri
- last_update → Son aktivite zamanı

KOD:

```
from google.adk.sessions import
InMemorySessionService

service = InMemorySessionService()

session = await service.create_session(
 app_name="seyahat_asistani",
 user_id="ahmet_123")
```

AIR LAB

GOOGLE ADK: STATE

STATE = AGENT'IN NOT DEFTERİ (4 Scope!)

1) SESSION STATE (varsayılan)
Bu konuşmaya özel, konuşma bitince gider

```
state["current_step"] = 3
state["aradigi_ucus"] = "IST→JFK"
```

2) USER STATE (user: prefix)
Kullanıcıya özel, TÜM konuşmalarda kalır

```
state["user:name"] = "Ahmet"
state["user:preferred_language"] = "tr"
state["user:favorite_airline"] = "THY"
```

3) APP STATE (app: prefix)
Tüm kullanıcılar için geçerli, global

```
state["app:version"] = "2.1"
state["app:total_users"] = 1450
state["app:maintenance_mode"] = False
```

4) TEMP STATE (temp: prefix)
Sadece bu işlem sırasında, hiç kaydedilmez

```
state["temp:api_response"] = raw_data
state["temp:intermediate_calc"] = result
```

💡 TEMPLATE DESTEĞİ:

```
instruction = "Merhaba {user:name},
dil tercihin: {user:preferred_language}"
```

→ ADK otomatik olarak state'ten değerleri enjekte eder!

AIR LAB

GOOGLE ADK: MEMORY

MEMORY = SESSİONLAR ARASI HAFIZA

State → Kısa süreli (bir konuşma içinde)
Memory → Uzun süreli (konuşmalar ARASINDA)

1. Konuşma biter
 ↓
2. add_session_to_memory(session)
 Gemini önemli bilgileri çıkarır
 ↓
3. Memory Bank'a kaydeder
 (embedding olarak, aranabilir)
 ↓
4. Yeni konuşmada agent memory'yi arar
 search_memory("kullanıcının bütçesi")
 ↓
5. İlgili bilgileri bulur ve kullanır

HAZIR TOOL'LAR:
 ├─── PreloadMemory → Her turda otomatik hafıza yükle
 └─── LoadMemory → Agent karar versin, gerektiğinde yükle

MEMORY SERVICE SEÇENEKLERİ:
 ├─── InMemoryMemoryService → Test için (RAM)
 └─── VertexAiRagMemoryService → Production (RAG + embedding)

AIR LAB

DIGER FRAMEWORKLER

LANGCHAIN: En eski, en olgun. Devasa topluluk.
Ama 'chain' mantığıyla başladığı için multi-agent sonradan eklendi. Bazen gereksiz karmaşık.
Bir tool eklemek için 5 farklı class lazım.

LANGGRAPH: LangChain'in 'bunu düzelt' denemesi.
Graph yapısı - node'lar ve edge'ler.
Çok güçlü ama öğrenme eğrisi dik.
State machine biliyorsanız rahat edersiniz.

CREWAI: En kolay başlangıç. 'Rol bazlı takım' metaforu.
'Sen araştırmacısın, sen yazarsın, sen editörsün' diyorsun, geri kalanı framework hallediyor.
Ama production'da sınırları var.

N8N: Kod yazmadan agent! Visual drag-and-drop.
Teknik olmayan insanlar için mükemmel.
Ama karmaşık logic'te sınırlı.

GOOGLE ADK: En yeni ama Google'ın arkasında.
Multi-agent native, MCP native, deployment native.
Diğerlerinin hatalarından ders almış.
Dezavantaj: Topluluk henüz küçük, doküman az."

AIR LAB

DIGER FRAMEWORKLER

LANGCHAIN
Felsefe: "Her şeyi chain'le, her LLM çağrısını bileşenlere ayır"
Güçlü: 100+ entegrasyon, devasa ekosistem
Zayıf: Aşırı abstraction, "chain of chains" karmaşıklığı
Kod: Chain → Prompt → LLM → Parser → Tool → Memory (her biri ayrı class)
İdeal: RAG uygulamaları, çok API'li projeler

LANGGRAPH
Felsefe: "Agent = State Machine, her adım bir node"
Güçlü: Tam kontrol, conditional branching, checkpoints
Zayıf: Öğrenme eğrisi dik, basit işler için overkill
Kod: Graph() → add_node() → add_edge() → compile()
İdeal: Karmaşık workflow, retry logic, human-in-the-loop

AIR LAB

DIGER FRAMEWORKLER

CREWAI
Felsefe: "Agent = Takım üyesi, görev dağılımı yap"
Güçlü: En kolay syntax, hızlı prototip
Zayıf: Production sınırları, debugging zor
Kod: Agent(role=...) → Task(agent=...) → Crew([...])
İdeal: İçerik üretimi, araştırma, basit otomasyon

N8N
Felsefe: "Kod yazmadan otomasyon"
Güçlü: Visual builder, 400+ entegrasyon, self-host
Zayıf: Karmaşık agent logic'te sınırlı
Kod: Yok! Drag-and-drop
İdeal: İş akışı otomasyonu, non-developer kullanıcılar

GOOGLE ADK
Felsefe: "Agent lifecycle: build → test → deploy → monitor"
Güçlü: Multi-agent native, MCP, Google ekosistemi
Zayıf: Yeni, küçük topluluk
Kod: Agent(model=..., tools=[...], sub_agents=[...])
İdeal: Google Cloud projeleri, multi-agent sistemler

AIR LAB

DEMO: DERS PLANLAYICI

AIR LAB

GERCEK ÖRNEK:DRONE KONTROL

📄 Paper: arxiv.org/abs/2601.15486 (Ocak 2025) "A Universal LLM - Drone Command and Control Interface"

Bir MCP Server yazıyorsunuz - drone'a MAVLink komutları gönderiyor. Sonra herhangi bir LLM'i bağlıyorsunuz.
Claude'a diyorsunuz ki: 'En yakın markete uç'
Claude ne yapıyor?
1. Önce Google Maps MCP'ye soruyor: 'En yakın market nerede?'
2. Koordinatları alıyor
3. Sonra Drone MCP'ye diyor: 'Bu koordinata uç'
4. Drone kalkıyor ve uçuyor 2 farklı MCP server, 1 LLM, 1 drone. Hepsi doğal dille kontrol ediliyor

Bu neden önemli? Eskiden drone uçurmak için:
- MAVLink komutlarını bilmen lazımdı
- Koordinat hesaplaması yapman lazımdı
- Özel yazılım kullanman lazımdı

Şimdi: 'Şu binayı bul ve fotoğrafını çek' diyorsun.

Paper Ocak 2025'te yayınlandı. Hem simülasyonda hem GERÇEK drone'da test edildi. Claude, GPT, Gemini hepsiyle çalışıyor."

AIR LAB

GERÇEK ÖRNEK:THY MCP SERVER

Claude, ChatGPT veya herhangi bir LLM'e doğal dille THY hizmetlerine erişim sağlıyor:

├── ✈️ Uçuş Arama & Durum Takibi
│ "TK123 uçuşunun durumu ne?"
│ "İstanbul-New York arası yarınki uçuşlar?"
│
├── 📋 Rezervasyon Yönetimi
│ "PNR ABC123 ile check-in yapabilir miyim?"
│
├── 💳 Miles&Smiles Bilgisi
│ "Kaç milim var?"
│
└── 🗺️ Destinasyon Bilgisi
"Tokyo'ya bagaj hakkım ne kadar?"

KULLANIM:
Claude Desktop'ta THY MCP entegrasyonu:
claude.ai/settings/integrations
URL: https://mcp.turkishtechlab.com/sse

ŞU AN: Digital Lab projesi (henüz resmi ürün değil)
AMA: Gerçek API'lerle çalışan, fonksiyonel bir sistem

AIR LAB

USE CASES & PROJE FIKIRLERI

AIR LAB

KAYNAKLAR

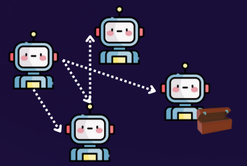
📚 KİTAPLAR
- "Building LLM Apps" - Chip Huyen
- "AI Engineering" - Chip Huyen
  - Oceanofpdf tarzı siteler en iyi dostunuz

🎥 YOUTUBE
- AI Jason (agent tutorials)
- Sam Witteveen (Google ADK)
- Matt Shumer (HyperWrite)

📄 PAPER'LAR
- CoT: arxiv.org/abs/2201.11903
- ReAct: arxiv.org/abs/2210.03629
- ToT: arxiv.org/abs/2305.10601

🔧 FRAMEWORK DOCS

🌐 TOPLULUK
- Reddit
- X
- AI Discord'lar



AIR LAB

THANK YOU

www.airlab.yildizskylab.com