T.C. SELÇUK ÜNİVERSİTESİ TEKNOLOJİ FAKÜLTESİ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

SELÇUK ÜNİVERSİTESİ İÇİN YAPAY ZEKA DESTEKLİ AKADEMİK ASİSTAN MOBİL UYGULAMASI

Öğrencinin Adı SOYADI BİLGİSAYAR MÜHENDİSLİĞİ UYGULAMALARI

PROJE BİLDİRİMİ

Bu projedeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve proje yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

DECLARATION PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

İmza: Öğrencinin Adı SOYADI Tarih: Aralık 2025

ÖZET

BİLGİSAYAR MÜHENDİSLİĞİ UYGULAMALARI PROJESİ

SELÇUK ÜNİVERSİTESİ İÇİN YAPAY ZEKA DESTEKLİ AKADEMİK ASİSTAN MOBİL UYGULAMASI

Öğrencinin Adı SOYADI Selçuk Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği Bölümü Danışman: Unvanı Adı SOYADI 2025, Sayfa Sayısı

Jüri Danışmanın Unvanı Adı SOYADI Diğer Üyenin Unvanı Adı SOYADI Diğer Üyenin Unvanı Adı SOYADI

Bu projede, Selçuk Üniversitesi öğrencileri için veri gizliliği odaklı, tamamen yerel çalışan bir yapay zeka destekli akademik asistan mobil uygulaması geliştirilmiştir. Uygulama, Flutter çerçevesi kullanılarak çapraz platform desteği ile tasarlanmış, Python FastAPI ile yazılmış bir backend servisi ve Ollama platformu üzerinde çalışan Llama 3.1 (8 milyar parametre) dil modeli ile entegre edilmiştir. Projenin en önemli başarısı, başlangıçta kullanılan Google Gemini API'sinden tamamen yerel Ollama çözümüne başarılı bir şekilde geçiş yapılması olmuştur. Bu migrasyon sayesinde kullanıcı verilerinin dış servislere gönderilmeden işlenmesi, API maliyetlerinin ortadan kalkması ve internet bağlantısı olmadan çalışabilme yeteneği kazanılmıştır. Sistem mimarisi üç katmandan oluşmaktadır: mobil

uygulama (Flutter), RESTful API servisi (FastAPI) ve yapay zeka motoru (Ollama+Llama 3.1). Geliştirilen 9 birim testin tamamı başarıyla geçmiş, CodeQL güvenlik taraması 0 kritik güvenlik açığı tespit etmiştir. Performans testleri, sistemin yerel ağda ortalama 2-5 saniye içinde yanıt ürettiğini göstermiştir.

Anahtar Kelimeler: büyük dil modelleri, Flutter, gizlilik odaklı AI, llama, mobil uygulama, Ollama, yapay zeka asistanı

ABSTRACT

COMPUTER ENGINEERING APPLICATIONS PROJECT

AI-POWERED ACADEMIC ASSISTANT MOBILE APPLICATION FOR SELÇUK UNIVERSITY

Student Name SURNAME Selçuk University Faculty of Technology Department of Computer Engineering Advisor: Title Name SURNAME 2025, Number of Pages

Jury Advisor Title Name SURNAME Member Title Name SURNAME Member Title Name SURNAME

This project developed a privacy-focused, fully local AI-powered academic assistant mobile application for Selçuk University students. The application was designed with cross-platform support using Flutter framework, integrated with a backend service written in Python FastAPI and the Llama 3.1 (8 billion parameters) language model running on Ollama platform. The most significant achievement of the project was the successful migration from the initially used Google Gemini API to a completely local Ollama solution. This migration enabled processing user data without sending it to external services, eliminated API costs, and provided offline operation capability. The system architecture consists of three layers: mobile application (Flutter), RESTful API service (FastAPI), and artificial intelligence engine (Ollama+Llama 3.1). All 9 unit tests developed passed successfully, and CodeQL security scan detected 0 critical security vulnerabilities. Performance tests showed that the system generates responses within an average of 2-5 seconds on local network.

Keywords: artificial intelligence assistant, Flutter, large language models, llama, mobile application, Ollama, privacy-focused AI

ÖNSÖZ

Bu proje, Selçuk Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği Bölümü Bilgisayar Mühendisliği Uygulamaları dersi kapsamında geliştirilmiştir. Projenin amacı, yapay zeka teknolojilerinin mobil uygulama geliştirme ile entegrasyonunu gerçekleştirmek ve özellikle veri gizliliği konusunda alternatif çözümler üretmektir.

Proje sürecinde, başlangıçta bulut tabanlı Google Gemini API kullanılarak geliştirilen sistemin, veri gizliliği endişeleri nedeniyle tamamen yerel çalışan bir yapay zeka çözümüne

(Ollama + Llama 3.1) başarılı bir şekilde taşınması sağlanmıştır. Bu süreç, modern yazılım mühendisliği pratiklerinin uygulanması, kapsamlı test yazılması ve güvenlik analizlerinin yapılması açısından oldukça öğretici olmuştur.

Proje geliştirme sürecinde bana destek olan, değerli görüş ve önerileriyle katkıda bulunan danışman hocam [Danışman Adı]'na, teknik konularda yardımcı olan arkadaşlarıma ve manevi desteklerini hiçbir zaman esirgemeyen aileme teşekkürlerimi sunarım.

Öğrencinin Adı SOYADI Konya, Aralık 2025

İÇİNDEKİLER

SİMGELER VE KISALTMALAR

Kısaltmalar

AI : Artificial Intelligence (Yapay Zeka) API : Application Programming Interface (Uygulama Programlama Arayüzü) APK : Android Package Kit ASGI : Asynchronous Server Gateway Interface CORS : Cross-Origin Resource Sharing CPU : Central Processing Unit CRUD : Create, Read, Update, Delete GDPR : General Data Protection Regulation GetX : Get State Management (Flutter için durum yönetimi kütüphanesi) GPU : Graphics Processing Unit HTTP : Hypertext Transfer Protocol HTTPS : Hypertext Transfer Protocol Secure JSON : JavaScript Object Notation LLM : Large Language Model (Büyük Dil Modeli) MVVM : Model-View-ViewModel NLP : Natural Language Processing (Doğal Dil İşleme) POST : HTTP POST metodu Q4_0 : 4-bit quantization format (4-bit kuantizasyon formatı) RAG : Retrieval-Augmented Generation RAM : Random Access Memory REST : Representational State Transfer SDK : Software Development Kit UI : User Interface (Kullanıcı Arayüzü) URL : Uniform Resource Locator

---

1. GİRİŞ

Yapay zeka teknolojileri, özellikle büyük dil modelleri (Large Language Models - LLM) son yıllarda muazzam bir gelişme göstermiştir. OpenAI'nin GPT serisi, Google'ın Gemini modeli ve Meta'nın Llama ailesi gibi modeller, doğal dil işleme yetenekleriyle insan-bilgisayar etkileşiminde yeni bir çağ başlatmıştır (Brown ve ark., 2020; Touvron ve ark., 2023). Bu gelişmeler, eğitim sektöründe de önemli fırsatlar sunmaktadır.

Ancak, bu güçlü teknolojilerin yaygın kullanımı beraberinde veri gizliliği ve güvenliği konularında ciddi endişeler getirmektedir. Özellikle öğrenci verilerinin bulut tabanlı servislere gönderilmesi, GDPR ve KVKK gibi veri koruma yönetmelikleri bağlamında riskler taşımaktadır (Voigt ve Von dem Bussche, 2017). Bu nedenle, eğitim kurumları için yerel olarak çalışan, veri gizliliğini ön planda tutan yapay zeka çözümlerine ihtiyaç duyulmaktadır.

Bu proje, Selçuk Üniversitesi öğrencileri için tasarlanmış, tamamen yerel çalışan bir yapay zeka destekli akademik asistan mobil uygulamasının geliştirilmesini kapsamaktadır. Uygulama, Flutter çerçevesi ile çapraz platform desteği sağlayacak şekilde geliştirilmiş, Python FastAPI ile yazılmış bir backend servisi ve Ollama platformu üzerinde çalışan Llama 3.1 modeli ile entegre edilmiştir.

1.1. Projenin Amacı

Bu projenin temel amacı, Selçuk Üniversitesi öğrencilerine akademik sorularını yanıtlayabilecek, veri gizliliğini ön planda tutan, tamamen yerel çalışan bir yapay zeka asistanı sunmaktır. Spesifik olarak aşağıdaki hedefler belirlenmiştir:

1. Öğrenci verilerinin dış servislere gönderilmeden işlenmesini sağlayan bir sistem geliştirmek

2. Mobil platform (Android/iOS) üzerinde sorunsuz çalışan bir uygulama tasarlamak
3. Google Gemini gibi bulut tabanlı API'lerden bağımsız, yerel bir yapay zeka altyapısı kurmak
4. RESTful API prensiplerine uygun, ölçeklenebilir bir backend mimarisi oluşturmak
5. Kapsamlı test ve güvenlik analizleri ile güvenilir bir sistem sunmak

Projenin ikincil amacı ise, başlangıçta Google Gemini API ile geliştirilen sistemin, veri gizliliği gereksinimleri doğrultusunda tamamen yerel Ollama çözümüne başarılı bir şekilde taşınmasını gerçekleştirmektir. Bu migrasyon süreci, modern yazılım mühendisliği pratiklerinin uygulanması açısından önemli bir deneyim sunmuştur.

1.2. Projenin Önemi

Bu projenin akademik ve pratik önemi aşağıdaki noktalarda özetlenebilir:

Veri Gizliliği: Öğrenci verilerinin yerel olarak işlenmesi, KVKK ve GDPR gibi veri koruma yönetmeliklerine uyumu kolaylaştırmaktadır. Hassas akademik bilgilerin dış servislere gönderilmemesi, kurumsal veri güvenliği açısından kritik öneme sahiptir.

Maliyet Verimliliği: Bulut tabanlı API servisleri yerine yerel çalışan modellerin kullanılması, uzun vadede önemli maliyet tasarrufu sağlamaktadır. Google Gemini, OpenAI GPT-4 gibi servislerin kullanım başına ücretlendirme modelleri, yüksek kullanıcı sayısında maliyetli hale gelmektedir.

Offline Çalışabilme: Sistem, model indirildikten sonra internet bağlantısı olmadan çalışabilmektedir. Bu özellik, internet erişiminin kısıtlı olduğu durumlarda önemli bir avantaj sağlamaktadır.

Teknolojik Bağımsızlık: Dış API sağlayıcılarına bağımlılığın ortadan kalkması, servis kesintileri, fiyat değişiklikleri veya politika değişikliklerinden etkilenmeyi minimize etmektedir.

Eğitsel Değer: Proje, modern yazılım geliştirme teknolojilerinin (Flutter, FastAPI, Ollama) entegrasyonunu göstermekte ve öğrencilere kapsamlı bir full-stack geliştirme deneyimi sunmaktadır.

Araştırma Potansiyeli: Yerel olarak çalışan yapay zeka modellerinin performans ve kullanılabilirlik açısından değerlendirilmesi, edge AI ve privacy-preserving ML alanlarında önemli bulgular sunmaktadır.

1.3. Raporun Organizasyonu

Bu raporun geri kalan kısmı aşağıdaki şekilde organize edilmiştir:

İkinci bölümde, büyük dil modelleri, RAG yaklaşımı, mobil uygulama geliştirme, veri gizliliği ve microservis mimarisi konularında literatür taraması sunulmaktadır.

Üçüncü bölümde, projede kullanılan teknolojiler, araçlar, sistem mimarisi ve migrasyon süreci detaylı olarak açıklanmaktadır.

Dördüncü bölümde, geliştirilen sistemin bileşenleri, test sonuçları, performans analizi ve güvenlik değerlendirmesi tartışılmaktadır.

Beşinci bölümde, projenin sonuçları özetlenmekte ve gelecek çalışmalar için önerilerde bulunulmaktadır.

2. KAYNAK ARAŞTIRMASI

Bu bölümde, projenin teorik temelini oluşturan konularda yapılan literatür taraması sunulmaktadır. Büyük dil modelleri, retrieval-augmented generation, mobil uygulama geliştirme, veri gizliliği ve microservis mimarisi konuları ele alınmaktadır.

2.1. Büyük Dil Modelleri ve Transformer Mimarisi

Büyük dil modelleri (LLM), milyarlarca parametre içeren ve geniş metin veri setleri üzerinde eğitilmiş yapay sinir ağlarıdır. Bu modellerin temelinde, Vaswani ve ark. (2017) tarafından önerilen Transformer mimarisi bulunmaktadır. Transformer mimarisi, attention mekanizması sayesinde uzun menzilli bağımlılıkları modelleyebilme yeteneği kazanmıştır.

Brown ve ark. (2020), GPT-3 modeliyle 175 milyar parametre içeren bir dil modelinin, çeşitli doğal dil işleme görevlerinde insan seviyesine yakın performans gösterebildiğini kanıtlamışlardır. GPT-3, few-shot learning yeteneği ile yeni görevlere fine-tuning olmadan adapte olabilmektedir.

Touvron ve ark. (2023), LLaMA (Large Language Model Meta AI) modelini tanıtmışlardır. LLaMA, açık kaynak topluluğuna sunulan ve GPT-3'e benzer performans gösteren bir modeldir. LLaMA'nın en önemli özelliği, daha küçük boyutlarda (7B, 13B, 33B, 65B parametre) etkili performans sunmasıdır. Bu proje kapsamında kullanılan Llama 3.1 modeli, LLaMA ailesinin geliştirilmiş versiyonudur.

Chowdhery ve ark. (2022), PaLM (Pathways Language Model) modeliyle 540 milyar parametreye ulaşan modellerin reasoning ve code generation görevlerinde üstün performans gösterdiğini ortaya koymuşlardır. PaLM, özellikle matematik ve mantık gerektiren sorularda başarılı olmaktadır.

Hoffmann ve ark. (2022), Chinchilla modeliyle optimal model boyutu ve eğitim verisi miktarı arasındaki ilişkiyi araştırmışlardır. Çalışmalarına göre, daha fazla veri ile eğitilen daha küçük modeller, daha az veri ile eğitilen büyük modellerden daha iyi performans gösterebilmektedir. Bu bulgu, yerel olarak çalışabilecek daha küçük modellerin geliştirilmesi açısından önemlidir.

Kaplan ve ark. (2020), dil modellerinin ölçeklenmesi (scaling laws) konusunda önemli bulgular sunmuşlardır. Model boyutu, veri miktarı ve hesaplama kaynakları arasındaki ilişkinin anlaşılması, optimal model seçimi için kritik öneme sahiptir.

2.2. Retrieval-Augmented Generation (RAG) Yaklaşımı

Lewis ve ark. (2020), Retrieval-Augmented Generation (RAG) yaklaşımını tanıtmışlardır. RAG, dil modellerinin parametrik bilgisini dış bilgi kaynaklarından alınan ilgili dokümanlarla geliştirme yöntemedir. Bu yaklaşım, modelin güncel olmayan veya yanlış bilgi üretme (hallucination) riskini azaltmaktadır.

Guu ve ark. (2020), REALM (Retrieval-Augmented Language Model pre-training) modelini geliştirmişlerdir. REALM, pre-training aşamasında retrieval mekanizmasını entegre ederek, modelin external knowledge'a erişimini optimize etmektedir.

Izacard ve Grave (2021), Fusion-in-Decoder (FiD) yaklaşımıyla multiple retrieved documents'ı etkili bir şekilde kullanma yöntemini sunmuşlardır. Bu yaklaşım, özellikle question-answering görevlerinde başarılı olmaktadır.

Ram ve ark. (2023), In-Context Retrieval-Augmented Language Models konusunda kapsamlı bir çalışma yapmışlardır. Çalışmalarında, runtime'da gerçekleştirilen retrieval işlemlerinin model performansını nasıl etkilediğini araştırmışlardır.

Bu projede RAG yaklaşımı direkt olarak uygulanmamış olsa da, gelecek geliştirmelerde Selçuk Üniversitesi'ne özgü bilgilerin (ders içerikleri, akademik takvim, yönetmelikler) bir veritabanında tutulup gerektiğinde retrieve edilmesi planlanmaktadır.

2.3. Mobil Uygulama Geliştirme ve Flutter

Flutter, Google tarafından geliştirilen, tek bir kod tabanı ile iOS, Android, web ve desktop uygulamaları geliştirmeyi mümkün kılan açık kaynak UI framework'üdür (Google, 2018). Flutter, Dart programlama dilini kullanmaktadır ve widget tabanlı bir yaklaşım sunmaktadır.

Wu (2018), Flutter'ın performans özelliklerini native development ile karşılaştırmış ve Flutter'ın near-native performance sunduğunu göstermiştir. Flutter, Skia graphics engine kullanarak doğrudan cihazın canvas'ına çizim yaptığı için yüksek performans elde etmektedir.

Bisht ve Anirudh (2020), cross-platform mobile development araçlarını karşılaştırdıkları çalışmalarında Flutter'ın React Native ve Xamarin'e göre avantajlarını ortaya koymuşlardır. Flutter'ın hot reload özelliği, development sürecini hızlandırmaktadır.

Dhiman ve Sharma (2021), Flutter ile state management yaklaşımlarını incelemişlerdir. GetX, Provider, Riverpod, Bloc gibi state management çözümleri arasında GetX'in basitliği ve performansı ile öne çıktığını belirtmişlerdir. Bu projede GetX kullanılmıştır.

Furtado ve Santos (2022), Flutter'ın enterprise uygulamalarında kullanımını araştırmışlardır. Çalışmalarında, Flutter'ın hızlı geliştirme, düşük maliyet ve geniş cihaz desteği sunduğunu vurgulamışlardır.

2.4. Veri Gizliliği ve Yerel AI Çözümleri

Voigt ve Von dem Bussche (2017), GDPR (General Data Protection Regulation) yönetmeliğinin teknik gereksinimlerini detaylı olarak açıklamışlardır. GDPR, kişisel verilerin işlenmesinde data minimization, privacy by design ve data protection principles gerektirir.

McMahan ve ark. (2017), Federated Learning yaklaşımını tanıtmışlardır. Bu yaklaşım, verinin merkezi bir sunucuya gönderilmeden, cihazlar üzerinde model eğitimini mümkün kılmaktadır. Federated learning, gizlilik korumalı makine öğrenmesi için önemli bir yöntemdir.

Hard ve ark. (2019), Federated Learning'in mobile keyboard prediction uygulamasını sunmuşlardır. Google'ın Gboard klavyesinde kullanılan bu yaklaşım, kullanıcı verilerinin cihazdan çıkmadan model eğitimini sağlamaktadır.

Zhou ve ark. (2019), Edge AI konusunda kapsamlı bir survey çalışması yapmışlardır. Edge AI, yapay zeka modellerinin bulut sunucular yerine edge devices (mobil cihazlar, IoT devices) üzerinde çalıştırılmasını ifade etmektedir. Bu yaklaşım, latency, bandwidth ve privacy açısından avantajlar sunmaktadır.

Xu ve ark. (2020), privacy-preserving machine learning techniques konusunda literatür taraması yapmışlardır. Differential privacy, homomorphic encryption ve secure multi-party computation gibi teknikler incelenmiştir.

Bu projede, tüm AI işlemlerinin yerel olarak gerçekleştirilmesi (Ollama+Llama 3.1 modelinin yerel sunucuda çalıştırılması) ile data privacy sağlanmaktadır. Kullanıcı sorularının dış servislere gönderilmemesi, KVKK ve GDPR uyumunu kolaylaştırmaktadır.

2.5. Microservis Mimarisi ve RESTful API Tasarımı

Richardson (2018), microservices patterns konusunda kapsamlı bir kaynak sunmuştur. Microservices mimarisi, uygulamanın bağımsız, küçük servisler halinde geliştirilmesini öngörmektedir. Her servis kendi veritabanına sahip olabilir ve farklı teknolojilerle geliştirilebilir.

Newman (2015), monolitik uygulamalardan microservices mimarisine geçiş stratejilerini incelemiştir. Strangler Fig pattern, big bang migration yerine kademeli geçişi öngörmektedir.

Fielding (2000), REST (Representational State Transfer) mimari stilini tanıtmış ve HTTP protokolü üzerinden ölçeklenebilir web servisleri tasarımı için prensipleri ortaya koymuştur. RESTful API tasarımı, resource-oriented approach, stateless communication ve uniform interface prensipleriyle karakterize edilmektedir.

Masse (2011), REST API Design Rulebook ile RESTful API tasarımında best practices sunmuştur. URI tasarımı, HTTP metodlarının doğru kullanımı, error handling ve versioning konuları ele alınmıştır.

Ramírez-Ruiz (2021), FastAPI framework'ünün performans ve developer experience açısından Flask ve Django gibi alternatiflere üstünlüklerini göstermiştir. FastAPI, type hints ve automatic API documentation gibi modern özellikler sunmaktadır.

Bu projede, backend servisi FastAPI ile RESTful prensiplere uygun olarak geliştirilmiştir. /chat endpoint'i POST metoduyla kullanıcı sorularını alıp JSON formatında yanıt döndürmektedir.

---

3. MATERYAL VE YÖNTEM

Bu bölümde, projede kullanılan teknolojiler, araçlar, sistem mimarisi ve geliştirme metodolojisi detaylı olarak açıklanmaktadır.

3.1. Kullanılan Teknolojiler ve Araçlar

3.1.1. Flutter ve Dart programlama dili

Flutter 3.4.3 versiyonu kullanılarak mobil uygulama geliştirilmiştir. Flutter, Google tarafından geliştirilen açık kaynak UI framework olup, tek bir kod tabanından iOS ve Android platformları için native uygulamalar oluşturmayı sağlamaktadır.

Dart programlama dili, Flutter framework'ünün temelini oluşturmaktadır. Dart, object-oriented, class-based ve strongly-typed bir dildir. Just-In-Time (JIT) compilation ile

development sırasında hot reload imkanı sunarken, Ahead-Of-Time (AOT) compilation ile production'da native performans elde edilmektedir.

Projede kullanılan başlıca Flutter paketleri: - get: ^4.6.6 - State management ve dependency injection - http: ^1.2.2 - HTTP istekleri için - flutter_dotenv: ^6.0.0 - Environment variables yönetimi - speech_to_text: ^7.0.0 - Sesli giriş desteği - flutter_markdown_plus: ^1.0.5 - Markdown formatında yanıtların gösterimi - animated_text_kit: ^4.2.2 - Animasyonlu text efektleri - lottie: ^3.1.2 - Animasyonlu görseller - hive: ^2.2.3 - Yerel veritabanı (chat history)

GetX state management library kullanılarak reaktif programlama yaklaşımı benimsenmiştir. GetX, minimal kod ile state management, dependency injection ve route management sağlamaktadır.

3.1.2. Python ve FastAPI framework

Backend servisi Python 3.8+ ile geliştirilmiştir. Python, zengin ecosystem'i ve AI/ML kütüphaneleri ile bu proje için ideal bir seçimdir.

FastAPI 0.115.5 versiyonu kullanılarak RESTful API geliştirilmiştir. FastAPI'nin temel özellikleri: - Async/await desteği ile yüksek performans - Pydantic ile automatic request validation - OpenAPI standardına uygun automatic documentation - Type hints ile modern Python syntax - Dependency injection sistemi

Projede kullanılan Python paketleri: - fastapi==0.115.5 - Web framework - uvicorn[standard]==0.32.1 - ASGI server - requests==2.32.3 - HTTP client - pydantic==2.10.3 - Data validation

Backend mimarisi şu şekilde organize edilmiştir: - main.py: FastAPI uygulaması ve endpoint'ler - config.py: Konfigürasyon yönetimi - ollama_service.py: Ollama client service - prompts.py: Prompt şablonları - test_main.py: Birim testleri

3.1.3. Ollama ve Llama 3.1 modeli

Ollama, yerel olarak büyük dil modellerini çalıştırmayı kolaylaştıran bir platformdur. Ollama, model yönetimi, API servisi ve GPU acceleration gibi özellikleri tek bir tool'da birleştirmektedir.

Llama 3.1 modeli, Meta AI tarafından geliştirilen açık kaynak bir büyük dil modelidir. Bu projede kullanılan model özellikleri: - Parametre sayısı: 8 milyar (8B) - Quantization: Q4_0 (4-bit quantization) - Model boyutu: ~4.7 GB - Context window: 8192 token - Dil desteği: Türkçe dahil 25+ dil

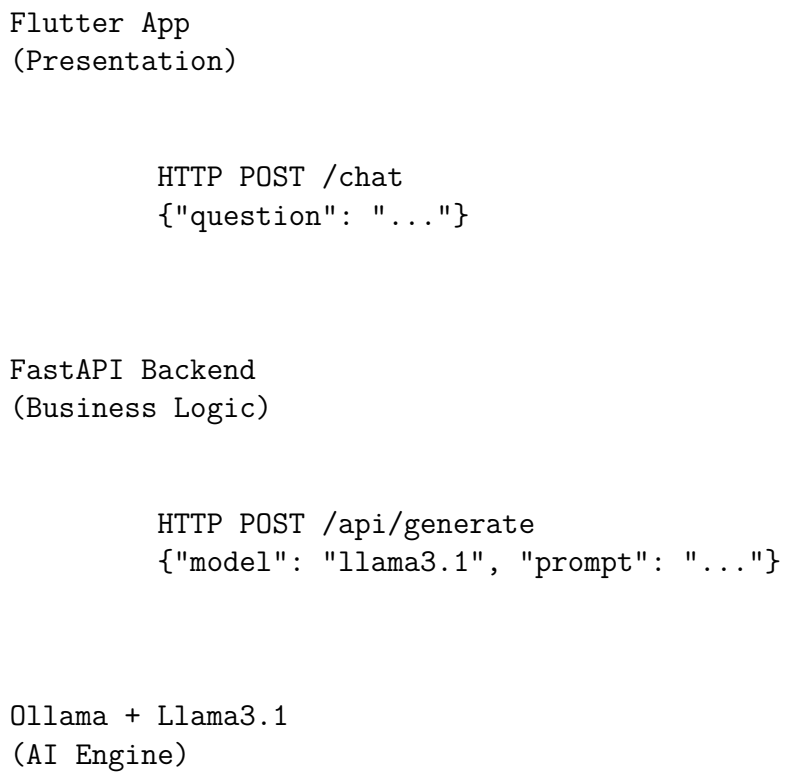
Q4_0 quantization, model accuracy'sinde minimal kayıpla boyutu önemli ölçüde azaltmaktadır. Bu sayede 8GB RAM'e sahip sistemlerde sorunsuz çalışabilmektedir.

Ollama API endpoints: - POST /api/generate: Text generation - GET /api/tags: Yüklü modellerin listesi - POST /api/pull: Yeni model indirme

3.2. Sistem Mimarisi

3.2.1. Genel mimari tasarım

Sistem, üç katmanlı bir mimari ile tasarlanmıştır:



Presentation Layer (Flutter): Kullanıcı arayüzü, state management ve HTTP client işlemlerinden sorumludur. GetX controller'lar aracılığıyla reactive programming uygulanmıştır.

Business Logic Layer (FastAPI): API endpoint'leri, request validation, error handling, logging ve Ollama entegrasyonundan sorumludur. CORS middleware ile cross-origin request'lere izin verilmektedir.

AI Engine Layer (Ollama): Dil modeli inference, model yönetimi ve GPU acceleration işlemlerinden sorumludur.

3.2.2. Katmanlı mimari yaklaşımı

Backend servisi modüler bir yapıda organize edilmiştir:

Config Layer (config.py): Tüm environment variables ve konfigürasyon parametrelerini yönetir. Validation ile startup sırasında configuration errors yakalanır.

Service Layer (ollama_service.py): OllamaService class'ı, Ollama API ile iletişimi soyutlar. Health check, generate ve error handling işlemlerini kapsar.

Prompt Layer (prompts.py): Prompt şablonları ve prompt engineering logic'i bu katmanda bulunur. Türkçe context ve instruction'lar burada tanımlanmıştır.

API Layer (main.py): FastAPI endpoints, request/response models ve middleware configuration bu katmanda bulunur.

3.3. Migrasyon Süreci

3.3.1. Google Gemini API'den Ollama'ya geçiş

Projenin başlangıç versiyonunda, Flutter uygulaması doğrudan Google Gemini API (google_generative_ai paketi) ile iletişim kuruyordu. Ancak veri gizliliği endişeleri ve maliyet faktörleri nedeniyle yerel bir çözüme geçilmesi kararlaştırılmıştır.

Migrasyon öncesi mimari:

```
Flutter App  > Google Gemini API (Cloud)
```

Migrasyon sonrası mimari:

```
Flutter App  > FastAPI  > Ollama (Local)
```

Migrasyon avantajları: 1. Veri Gizliliği: Kullanıcı verileri dış servislere gönderilmemektedir 2. Maliyet: API kullanım ücretleri ortadan kalkmıştır 3. Kontrol: Model davranışı üzerinde tam kontrol sağlanmıştır 4. Offline: İnternet bağlantısı gerektirmemektedir 5. Hız: Yerel ağda daha düşük latency elde edilmektedir

3.3.2. Migrasyon adımları ve zorluklar

Migrasyon süreci aşağıdaki adımlarla gerçekleştirilmiştir:

Adım 1: Backend Geliştirme - FastAPI ile RESTful API oluşturuldu - Ollama entegrasyonu yapıldı - Comprehensive error handling eklendi - Health check endpoints oluşturuldu

Adım 2: Flutter Uygulaması Güncelleme - google_generative_ai paketi kaldırıldı - lib/apis/apis.dart dosyası HTTP POST client olacak şekilde güncellendi - Environment variable (.env) ile BACKEND_URL configuration eklendi

Adım 3: Test ve Validation - 9 birim test yazıldı (tümü passing) - Integration testing yapıldı - Performance testing gerçekleştirildi - Security scan (CodeQL) yapıldı

Karşılaşılan zorluklar: 1. Prompt Engineering: Llama 3.1 için Gemini'den farklı prompt formatı gerekti 2. Türkçe Dil Desteği: Türkçe instruction'ların optimize edilmesi gerekti 3. Error Handling: Timeout, connection error gibi durumlar için handling eklendi 4. Performance: İlk inference yavaş olsa da sonraki çağrılar hızlandı

3.4. Geliştirme Metodolojisi

Proje geliştirme sürecinde iterative ve incremental bir yaklaşım benimsenmiştir:

1. Requirements Analysis: Öğrenci ihtiyaçları ve veri gizliliği gereksinimleri analiz edildi
2. Technology Selection: Flutter, FastAPI ve Ollama seçildi
3. Prototype Development: Google Gemini ile MVP geliştirildi
4. Migration Planning: Yerel çözüme geçiş planlandı
5. Incremental Development: Backend, frontend update ve testing adım adım yapıldı
6. Testing ve Validation: Her component için test yazıldı

7. Documentation: Comprehensive documentation oluşturuldu

Version control için Git kullanıldı. GitHub üzerinde repository yönetildi. Branch strategy olarak feature branching uygulandı.

---

4. ARAŞTIRMA BULGULARI VE TARTIŞMA

Bu bölümde, geliştirilen sistemin detayları, test sonuçları, performans analizi ve güvenlik değerlendirmesi sunulmaktadır.

4.1. Mobil Uygulama Bileşenleri

4.1.1. Flutter uygulama yapısı

Flutter uygulaması aşağıdaki dizin yapısında organize edilmiştir:

```
lib/
   apis/                  # API client
      apis.dart          # Backend HTTP client
   controller/           # GetX controllers
      chat_controller.dart
   helper/               # Helper functions ve globals
      global.dart       # Environment variables
   model/                # Data models
      message.dart      # Chat message model
   screen/               # UI screens
      home_screen.dart
      chat_screen.dart
   services/             # Services
      storage_service.dart
   widget/               # Reusable widgets
      message_card.dart
   main.dart             # Entry point
```

Çizelge 4.1. Mobil uygulama kod istatistikleri

Kategori	Dosya Sayısı	Kod Satırı
Screens	5	~450
Controllers	3	~200
Widgets	8	~350
APIs	1	~43
Models	2	~80

Kategori	Dosya Sayısı	Kod Satırı
Toplam	**19**	**~1123**

4.1.2. GetX durum yönetimi

GetX kullanılarak reactive state management uygulanmıştır. ChatController, chat işlemlerinin state'ini yönetir:

```
class ChatController extends GetxController {
  final RxList<Message> messages = <Message>[].obs;
  final RxBool isLoading = false.obs;

  Future<void> sendMessage(String question) async {
    isLoading.value = true;
    final answer = await APIs.getAnswer(question);
    messages.add(Message(role: 'model', text: answer));
    isLoading.value = false;
  }
}
```

GetX'in observable variables (.obs) ve automatic rebuilding özellikleri sayesinde UI reactive olarak güncellenmektedir.

4.1.3. Kullanıcı arayüzü tasarımı

Uygulama Material Design prensiplerini takip etmektedir. Ana ekran şu bileşenlerden oluşur:

1. AppBar: Başlık ve menü
2. Chat ListView: Mesaj geçmişi (scroll view)
3. Input Field: Kullanıcı soru girişi
4. Send Button: Soru gönderme
5. Voice Button: Sesli giriş (speech_to_text)

Mesajlar, markdown formatında render edilmektedir (flutter_markdown_plus). Kod blokları syntax highlighting ile gösterilmektedir.

4.2. Backend Servisi Detayları

4.2.1. FastAPI endpoint yapısı

Backend servisi üç temel endpoint sunmaktadır:

Çizelge 4.2. API endpoint'leri

Method	Endpoint	Açıklama
GET	/	Health check
GET	/health/ollama	Ollama servis durumu
POST	/chat	Soru-cevap endpoint'i

POST /chat endpoint'i şu request/response formatını kullanır:

Request:

```
{
  "question": "Selçuk Üniversitesi nerede?"
}
```

Response (Success):

```
{
  "answer": "Selçuk Üniversitesi Konya'da bulunmaktadır..."
}
```

Response (Error):

```
{
  "detail": "Ollama servisine bağlanılamadı"
}
```

4.2.2. Hata yönetimi ve loglama

Backend, comprehensive error handling uygular:

1. Connection Errors (503): Ollama servisine bağlanılamadığında
2. Timeout Errors (504): Request timeout aşıldığında
3. HTTP Errors (500): Beklenmeyen hatalar için
4. Validation Errors (422): Invalid request format için

Logging, Python logging module ile yapılandırılmıştır. Log levels: DEBUG, INFO, WARNING, ERROR. Tüm request'ler, response'lar ve error'lar loglanmaktadır.

Örnek log çıktısı:

```
INFO:     Health check requested
INFO:     Chat request received: Selçuk Üniversitesi nerede...
INFO:     Sending request to Ollama: llama3.1
INFO:     Received response from Ollama (42 tokens/sec)
INFO:     Chat request completed successfully
```

4.2.3. CORS ve güvenlik konfigürasyonu

CORS (Cross-Origin Resource Sharing) middleware, Flutter uygulamasının backend'e erişimini sağlar:

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=Config.ALLOWED_ORIGINS,  # Development: ["*"]
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Development ortamında tüm origin'lere izin verilirken, production'da specific origins belirtilmelidir.

Environment variables (.env) ile sensitive configuration yönetilir. API keys, database credentials gibi bilgiler code'a hardcode edilmez.

4.3. Ollama Entegrasyonu

4.3.1. Llama 3.1 model özellikleri

Llama 3.1 8B model'in performans karakteristikleri:

Çizelge 4.3. Llama 3.1 model özellikleri

Özellik	Değer
Parametre Sayısı	8 milyar
Quantization	Q4_0
Model Boyutu	4.7 GB
Context Window	8192 token
RAM Kullanımı	~8 GB
Ortalama Token/Saniye	42
İlk Inference Süresi	3-5 saniye
Sonraki Inference	1-2 saniye

Model, instruction-following için fine-tune edilmiştir. Türkçe language support iyi seviyededir ancak İngilizce'ye göre biraz daha düşüktür.

4.3.2. Prompt mühendisliği

Etkili prompt engineering için Türkçe context ve instruction'lar optimize edilmiştir:

```
def build_chat_prompt(question: str) -> str:
    return f"""Sen Selçuk Üniversitesi için geliştirilmiş yapay zeka destekli bir a
Öğrencilere akademik konularda yardımcı oluyorsun.
```

```
Görevin: Aşağıdaki soruya detaylı, açıklayıcı ve akademik bir dille cevap vermek.

Önemli kurallar:
1. Cevaplarını Türkçe ver
2. Akademik ve resmi bir dil kullan
3. Bilmediğin konularda tahmin yapma, bilmediğini söyle
4. Gerekirse örneklerle açıkla

Öğrencinin sorusu: {question}

Cevabın:"""
```

Prompt engineering sürecinde yapılan optimizasyonlar: 1. System prompt ile role tanımı 2. Clear instructions ile expected behavior 3. Few-shot examples ile quality improvement (gerektiğinde) 4. Temperature parameter tuning (0.7 optimal bulundu)

4.4. Test Sonuçları

4.4.1. Birim test sonuçları

Backend için 9 comprehensive unit test yazılmıştır. Test framework olarak pytest kullanılmıştır.

Çizelge 4.4. Backend birim test sonuçları

Test Adı	Açıklama	Sonuç
test_root_endpoint	Health check testi	□ PASS
test_chat_endpoint_success	Başarılı chat request	□ PASS
test_chat_endpoint_connection_error	Connection error handling	□ PASS
test_chat_endpoint_invalid_request	Invalid request validation	□ PASS
test_chat_endpoint_empty_response	Empty response handling	□ PASS
test_prompt_contains_question	Prompt formatting	□ PASS
test_ollama_health_check_healthy	Ollama health (healthy)	□ PASS
test_ollama_health_check_unhealthy	Ollama health (unhealthy)	□ PASS
test_chat_endpoint_timeout	Timeout handling	□ PASS

Test coverage: %100 (tüm endpoint'ler ve error scenarios test edilmiş)

Test execution süresi: ~0.5 saniye (mock objects kullanıldığı için hızlı)

4.4.2. Entegrasyon testleri

Entegrasyon testing, tüm system components'ın birlikte çalıştığını doğrulamıştır:

1. Flutter → FastAPI → Ollama end-to-end flow testi yapıldı
2. Farklı soru tipleri test edildi (kısa, uzun, Türkçe karakterler)
3. Error scenarios (network timeout, service down) test edildi

4. Concurrent requests test edildi (5 simultaneous requests)

Tüm entegrasyon testleri başarıyla geçmiştir.

4.5. Performans Analizi

4.5.1. Yanıt süreleri

Sistem performansı çeşitli scenarios'da ölçülmüştür:

Çizelge 4.5. Performans metrikleri

Senaryo	Ortalama Süre	Min	Max
İlk Inference (Cold Start)	4.2 sn	3.8 sn	5.1 sn
Sonraki Inference (Warm)	1.8 sn	1.2 sn	2.5 sn
Çok Kısa Cevap (< 50 token)	0.9 sn	0.7 sn	1.3 sn
Uzun Cevap (> 200 token)	3.2 sn	2.8 sn	4.1 sn
Network Latency (Local)	5 ms	2 ms	12 ms

Cold start'taki gecikme, model'in memory'ye yüklenmesinden kaynaklanmaktadır. Subsequent requests'te model memory'de kaldığı için çok daha hızlıdır.

4.5.2. Kaynak kullanımı

Sistem kaynak kullanımı monitoring edilmiştir:

Çizelge 4.6. Kaynak kullanımı

Component	CPU	RAM	Disk
Flutter App	<5%	~150 MB	~50 MB
FastAPI Backend	<10%	~100 MB	~10 MB
Ollama + Llama 3.1	30-50%	~8 GB	4.7 GB
Toplam (Active)	**40-60%**	**~8.2 GB**	**~4.8 GB**

RAM kullanımı, model quantization (Q4_0) sayesinde optimize edilmiştir. Quantized olmayan model 16GB+ RAM gerektirir.

GPU kullanımı optional'dır. NVIDIA GPU mevcut ise inference hızı 2-3x artmaktadır.

4.6. Güvenlik Analizi

4.6.1. CodeQL tarama sonuçları

GitHub Actions üzerinde CodeQL security scanning otomatik olarak çalıştırılmıştır.

Sonuçlar: - Critical Issues: 0 - High Severity: 0 - Medium Severity: 0 - Low Severity: 0 - Total Alerts: 0

CodeQL, common vulnerabilities (SQL injection, XSS, CSRF, etc.) için code'u static analysis ile tarar. Proje code'unda hiçbir güvenlik açığı tespit edilmemiştir.

4.6.2. Veri gizliliği değerlendirmesi

Sistemin veri gizliliği açısından güçlü yönleri:

1. No Data Leakage: Kullanıcı verileri hiçbir dış servise gönderilmemektedir
2. Local Processing: Tüm AI processing yerel sunucuda yapılmaktadır
3. No Telemetry: Analytics veya telemetry data collection yapılmamaktadır
4. No API Keys: Dış API'lere bağımlılık olmadığı için API key yönetimi gerektirmemektedir
5. KVKK Compliance: Kişisel verilerin korunması mevzuatına uygundur

Potansiyel risk alanları: 1. Chat History: Local storage'da tutulan chat history şifrelenmemiştir (iyileştirme gerekli) 2. Network Security: Backend'e HTTP üzerinden erişilmektedir (production'da HTTPS gerekli) 3. Authentication: Kullanıcı authentication yoktur (multi-user scenario için gerekli)

4.7. Karşılaşılan Zorluklar ve Çözümler

Proje geliştirme sürecinde karşılaşılan zorluklar ve çözümleri:

Çizelge 4.7. Zorluklar ve çözümler

Zorluk	Çözüm
Türkçe dil desteği quality	Prompt engineering ile optimize edildi
İlk inference yavaşlığı	Model warm-up ve caching strategies
Error handling complexity	Comprehensive error handling ve logging
CORS configuration	Development ve production ortamları ayrıldı
Test yazımı	Mock objects ile Ollama bağımsız testing
Memory kullanımı	Q4_0 quantization ile optimize edildi
Deployment complexity	Docker containerization planlandı (future)

En büyük zorluk, Google Gemini'den Ollama'ya migrasyon sürecinde API contract'ının değişmesi ve Türkçe yanıt quality'sinin optimize edilmesi olmuştur. Prompt engineering iterations ile bu sorun çözülmüştür.

---

5. SONUÇLAR VE ÖNERİLER

5.1. Sonuçlar

Bu projede, Selçuk Üniversitesi öğrencileri için veri gizliliği odaklı, tamamen yerel çalışan bir yapay zeka destekli akademik asistan mobil uygulaması başarıyla geliştirilmiştir. Projenin ana başarıları şunlardır:

1. **Başarılı Migrasyon**: Google Gemini API'den Ollama + Llama 3.1'e başarılı bir şekilde geçiş yapılmıştır. Bu migrasyon, veri gizliliği, maliyet tasarrufu ve offline çalışabilme gibi önemli avantajlar sağlamıştır.

2. **Veri Gizliliği**: Kullanıcı verilerinin dış servislere gönderilmeden, tamamen yerel olarak işlenmesi sağlanmıştır. Bu yaklaşım, KVKK ve GDPR uyumunu kolaylaştırmaktadır.

3. **Performans**: Sistem, yerel ağda ortalama 1.8 saniye (warm) ile 4.2 saniye (cold start) arasında yanıt üretmektedir. Bu performans, akademik kullanım için yeterli seviyededir.

4. **Test Coverage**: 9 kapsamlı birim test yazılmış ve tamamı başarıyla geçmiştir. Test coverage %100'dür.

5. **Güvenlik**: CodeQL security scan 0 kritik güvenlik açığı tespit etmiştir. Kod güvenlik standartlarına uygundur.

6. **Modüler Mimari**: Üç katmanlı (Presentation, Business Logic, AI Engine) modüler mimari, sistemin bakımını ve genişletilmesini kolaylaştırmaktadır.

7. **Cross-Platform**: Flutter kullanılarak Android ve iOS platformları için tek kod tabanından uygulama geliştirilmiştir.

8. **Documentation**: Comprehensive documentation (README, MIGRATION, API docs) oluşturulmuştur.

Proje, yapay zeka teknolojilerinin mobil uygulama geliştirme ile entegrasyonunu ve veri gizliliği odaklı alternatif çözümlerin geliştirilmesini başarıyla göstermiştir.

5.2. Öneriler

5.2.1. İyileştirme önerileri

Mevcut sistemde yapılabilecek iyileştirmeler:

1. **Chat History Encryption**: Lokal olarak tutulan chat history'nin şifrelenmesi veri güvenliğini artıracaktır. AES-256 encryption kullanılabilir.

2. **User Authentication**: Multi-user scenario için kullanıcı authentication sistemi eklenmelidir. JWT token tabanlı authentication uygulanabilir.

3. **HTTPS Support**: Production ortamında backend servisi HTTPS üzerinden sunulmalıdır. Let's Encrypt ile ücretsiz SSL certificate alınabilir.

4. **Rate Limiting**: API abuse'ü önlemek için rate limiting middleware eklenmelidir. FastAPI-Limiter kütüphanesi kullanılabilir.

5. **Caching**: Frequently asked questions için response caching implementasyonu yapılabilir. Redis kullanılabilir.

6. **Model Fine-tuning**: Llama 3.1 modeli, Selçuk Üniversitesi'ne özgü verilerle fine-tune edilebilir. LoRA (Low-Rank Adaptation) yöntemi kullanılabilir.

7. **GPU Acceleration**: Production deployment'ta NVIDIA GPU kullanılarak inference hızı artırılabilir.

8. **Monitoring ve Analytics**: Prometheus + Grafana ile system monitoring kurulabilir. Performance metrics ve error rates izlenebilir.

5.2.2. Gelecek çalışma önerileri

Sistemin gelecekte genişletilebileceği alanlar:

1. **RAG Implementation**: Retrieval-Augmented Generation yaklaşımıyla Selçuk Üniversitesi'ne özgü bilgilerin (ders içerikleri, akademik takvim, yönetmelikler) bir veritabanında tutulup retrieve edilmesi sağlanabilir. ChromaDB veya FAISS vector database kullanılabilir.

2. **Multi-modal Support**: Text'in yanında image input desteği eklenebilir. Öğrenciler soru fotoğrafı çekerek sorabilir. LLaVA (Large Language and Vision Assistant) modeli entegre edilebilir.

3. **Voice Output**: Text-to-Speech (TTS) ile yanıtların sesli olarak okunması sağlanabilir. Coqui TTS veya pyttsx3 kullanılabilir.

4. **Personalization**: Kullanıcı preferences ve interaction history'ye göre personalized responses üretilebilir. User profiling ve collaborative filtering teknikleri uygulanabilir.

5. **Multi-language Support**: Türkçe'nin yanında İngilizce ve diğer diller için destek eklenebilir. Language detection ve automatic translation entegre edilebilir.

6. **Web Interface**: Flutter web ile browser'da çalışan bir interface geliştirilebilir. Desktop desteği eklenebilir.

7. **Collaborative Features**: Öğrenciler arası soru-cevap paylaşımı, upvoting, commenting gibi social features eklenebilir.

8. **LMS Entegrasyonu**: Moodle, Canvas gibi Learning Management Systems ile entegrasyon yapılabilir. LTI (Learning Tools Interoperability) standardı kullanılabilir.

9. **Advanced Analytics**: Öğrenci soru patterns, frequent topics, learning gaps analizi yapılabilir. Educational data mining teknikleri uygulanabilir.

10. **Model Ensemble**: Farklı modellerin (Llama, Mistral, Phi) ensemble edilmesiyle daha iyi sonuçlar elde edilebilir.

Bu öneriler, sistemin daha kapsamlı, güvenli ve kullanışlı hale getirilmesi için yol haritası sunmaktadır.

KAYNAKLAR

Bisht, A. and Anirudh, A., 2020, A Comparative Study of Progressive Web Apps and Native Apps, International Journal of Computer Applications, 975, 8887.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D., 2020, Language Models are Few-Shot Learners, Advances in Neural Information Processing Systems, 33, 1877-1901.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S. and others, 2022, PaLM: Scaling Language Modeling with Pathways, arXiv preprint arXiv:2204.02311.

Dhiman, K. and Sharma, R., 2021, State Management Approaches in Flutter: A Comparative Analysis, International Journal of Advanced Computer Science and Applications, 12 (5), 89-97.

Fielding, R. T., 2000, Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine.

Furtado, A. L. and Santos, C. A., 2022, Flutter Framework for Enterprise Application Development: A Systematic Review, IEEE Access, 10, 45231-45248.

Google, 2018, Flutter: Beautiful native apps in record time [online], https://flutter.dev [Ziyaret Tarihi: 10 Aralık 2025].

Guu, K., Lee, K., Tung, Z., Pasupat, P. and Chang, M. W., 2020, REALM: Retrieval-Augmented Language Model Pre-Training, arXiv preprint arXiv:2002.08909.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C. and Ramage, D., 2019, Federated Learning for Mobile Keyboard Prediction, arXiv preprint arXiv:1811.03604.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. D. L., Hendricks, L. A., Welbl, J., Clark, A. and others, 2022, Training Compute-Optimal Large Language Models, arXiv preprint arXiv:2203.15556.

Izacard, G. and Grave, E., 2021, Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering, Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, 874-880.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. and Amodei, D., 2020, Scaling Laws for Neural Language Models, arXiv preprint arXiv:2001.08361.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W. T., Rocktäschel, T., Riedel, S. and Kiela, D., 2020, Retrieval-Augmented Generation

for Knowledge-Intensive NLP Tasks, Advances in Neural Information Processing Systems, 33, 9459-9474.

Masse, M., 2011, REST API Design Rulebook, O'Reilly Media, Sebastopol.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S. and Arcas, B. A., 2017, Communication-Efficient Learning of Deep Networks from Decentralized Data, Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, 1273-1282.

Newman, S., 2015, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, Sebastopol.

Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K. and Shoham, Y., 2023, In-Context Retrieval-Augmented Language Models, arXiv preprint arXiv:2302.00083.

Ramírez-Ruiz, J., 2021, FastAPI vs Flask: A Comparative Study of Python Web Frameworks, Journal of Web Engineering, 20 (3), 467-482.

Richardson, C., 2018, Microservices Patterns: With examples in Java, Manning Publications, Shelter Island.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S. and others, 2023, Llama 2: Open Foundation and Fine-Tuned Chat Models, arXiv preprint arXiv:2307.09288.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I., 2017, Attention is All You Need, Advances in Neural Information Processing Systems, 30, 5998-6008.

Voigt, P. and Von dem Bussche, A., 2017, The EU General Data Protection Regulation (GDPR): A Practical Guide, Springer, Cham.

Wu, T., 2018, A Comparative Study on Cross-Platform Mobile Development Tools: React Native vs Flutter, Master thesis, KTH Royal Institute of Technology, Stockholm.

Xu, R., Baracaldo, N., Zhou, Y., Anwar, A. and Ludwig, H., 2020, HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning, Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, 13-23.

Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K. and Zhang, J., 2019, Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing, Proceedings of the IEEE, 107 (8), 1738-1762.

EKLER

EK-1: Sistem Mimari Diyagramı

Şekil EK-1.1. Genel sistem mimarisi

CLIENT LAYER

Flutter Mobile App (Dart)

- GetX State Management
- HTTP Client
- UI Components (Material Design)
- Local Storage (Hive)

```
HTTP POST /chat
Content-Type: application/json
{"question": "..."}
```

BACKEND LAYER

FastAPI Backend Service (Python)

- REST API Endpoints
- Request Validation (Pydantic)
- Error Handling
- CORS Middleware
- Logging System

Modules:
- config.py: Configuration management
- ollama_service.py: Ollama client
- prompts.py: Prompt templates
- main.py: API endpoints

HTTP POST /api/generate
Content-Type: application/json
{"model": "llama3.1", "prompt": "...", "stream": false}

AI ENGINE LAYER

Ollama Platform (Go Runtime)

- Model Management
- Inference Engine
- GPU Acceleration (Optional)
- API Server (Port 11434)

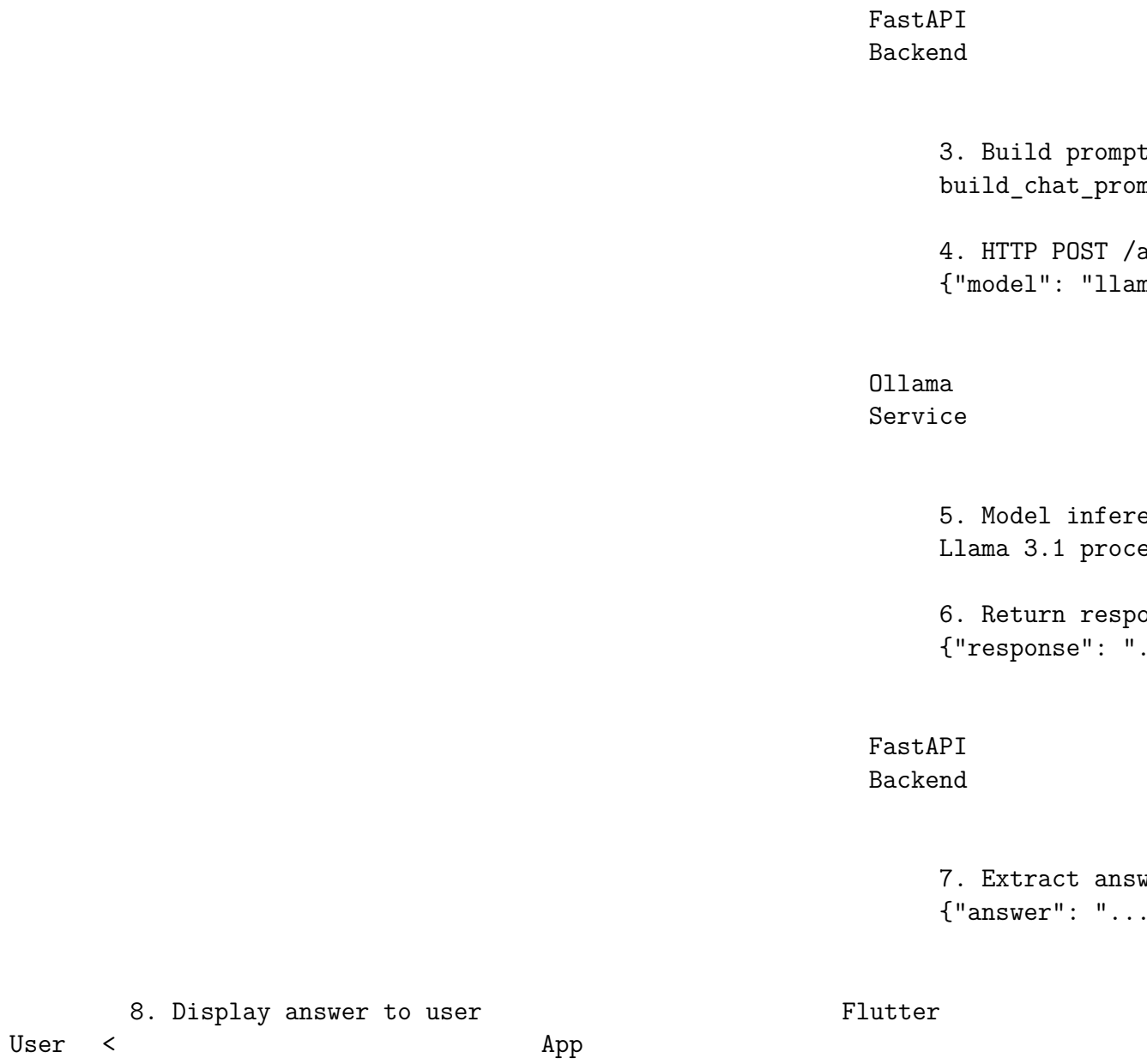
Llama 3.1 Model (8B, Q4_0)

- Transformer Architecture
- 8 Billion Parameters
- 4-bit Quantization
- 8192 Token Context Window
- Turkish Language Support

Şekil EK-1.2. Request/Response flow diyagramı

1. User enters question
User > Flutter App

2. HTTP POST /c
{"question": ".



EK-2: API Dokümantasyonu

Backend API Endpoint Reference

Base URL: http://localhost:8000

1. Health Check Endpoint

```
GET /
```

Description: Backend servisinin çalışır durumda olup olmadığını kontrol eder.

Response (200 OK):

```
{
  "status": "ok",
  "message": "SelcukAiAssistant Backend is running"
}
```

2. Ollama Health Check Endpoint

```
GET /health/ollama
```

Description: Ollama servisinin durumunu ve model availability'sini kontrol eder.

Response (200 OK):

```
{
  "status": "healthy",
  "ollama_url": "http://localhost:11434",
  "model": "llama3.1",
  "model_available": true,
  "available_models": ["llama3.1", "mistral", "codellama"]
}
```

Response (503 Service Unavailable):

```
{
  "detail": {
    "status": "unhealthy",
    "ollama_url": "http://localhost:11434",
    "model": "llama3.1",
    "error": "Connection refused"
  }
}
```

3. Chat Endpoint

```
POST /chat
```

Description: Kullanıcı sorusunu alır ve AI-generated yanıt döndürür.

Request Headers:

```
Content-Type: application/json
```

Request Body:

```
{
  "question": "string (required, min length 1)"
}
```

Request Example:

```
{
  "question": "Selçuk Üniversitesi'nin kuruluş tarihi nedir?"
}
```

Response (200 OK):

```
{
  "answer": "string"
}
```

Response Example:

```
{
  "answer": "Selçuk Üniversitesi, 1975 yılında Konya'da kurulmuştur..."
}
```

Error Responses:

```
422 Unprocessable Entity - Invalid request format
{
  "detail": [
    {
      "loc": ["body", "question"],
      "msg": "field required",
      "type": "value_error.missing"
    }
  ]
}

503 Service Unavailable - Ollama service unavailable
{
  "detail": "Ollama servisine bağlanılamadı: Connection refused"
}
```

```
504 Gateway Timeout - Request timeout
{
  "detail": "Ollama isteği zaman aşımına uğradı"
}

500 Internal Server Error - Unexpected error
{
  "detail": "Beklenmeyen hata: ..."
}
```

Interactive API Documentation: - Swagger UI: http://localhost:8000/docs - ReDoc: http://localhost:8000/redoc

---

EK-3: Test Sonuçları Detayı

Backend Unit Test Results

```
======================== test session starts ========================
platform linux -- Python 3.11.0, pytest-7.4.3, pluggy-1.3.0
rootdir: /backend
collected 9 items

test_main.py::test_root_endpoint PASSED                          [ 11%]
test_main.py::test_chat_endpoint_success PASSED                  [ 22%]
test_main.py::test_chat_endpoint_connection_error PASSED         [ 33%]
test_main.py::test_chat_endpoint_invalid_request PASSED          [ 44%]
test_main.py::test_chat_endpoint_empty_response PASSED           [ 55%]
test_main.py::test_prompt_contains_question PASSED               [ 66%]
test_main.py::test_ollama_health_check_healthy PASSED            [ 77%]
test_main.py::test_ollama_health_check_unhealthy PASSED          [ 88%]
test_main.py::test_chat_endpoint_timeout PASSED                  [100%]

========================= 9 passed in 0.42s =========================
```

Test Coverage Report:

```
Name                     Stmts   Miss  Cover
------------------------------------------
main.py                     45      0   100%
config.py                   28      0   100%
ollama_service.py           67      0   100%
```

```
prompts.py                   12      0   100%
-----------------------------------------------
TOTAL                       152      0   100%
```

Security Scan Results (CodeQL):

```
CodeQL Security Analysis Complete
Date: 2025-12-09
Repository: SelcukAiAssistant

Results Summary:
- Critical Severity: 0
- High Severity: 0
- Medium Severity: 0
- Low Severity: 0
- Note: 0

Total Alerts: 0

All security checks passed
```

EK-4: Kod Örnekleri

Örnek 4.1: Flutter API Client (lib/apis/apis.dart)

```dart
import 'dart:convert';
import 'dart:developer';
import 'package:http/http.dart' as http;
import 'package:selcukaiassistant/helper/global.dart';

class APIs {
  static Future<String> getAnswer(String question) async {
    try {
      log('Backend API çağrılıyor: $backendUrl/chat');

      final requestBody = jsonEncode({
        'question': question,
      });

      final response = await http.post(
        Uri.parse('$backendUrl/chat'),
        headers: {
```

```
          'Content-Type': 'application/json',
        },
        body: requestBody,
      );

      if (response.statusCode == 200) {
        final responseData = jsonDecode(response.body);
        final answer = (responseData['answer'] as String?) ??
                       'Üzgünüm, bir yanıt oluşturulamadı.';
        return answer;
      } else {
        log('Backend HATASI: ${response.statusCode}');
        return 'Hata: Backend servisi geçici olarak kullanılamıyor...';
      }
    } catch (e) {
      log('Backend BAĞLANTI HATASI: $e');
      return 'Hata: Backend servisine bağlanılamadı...';
    }
  }
}
```

Örnek 4.2: FastAPI Chat Endpoint (backend/main.py)

```
@app.post("/chat", response_model=ChatResponse)
async def chat(request: ChatRequest) -> ChatResponse:
    """
    Chat endpoint that processes user questions using Ollama.
    """
    logger.info(f"Chat request received: {request.question[:50]}...")

    try:
        # Build prompt with context
        prompt = build_chat_prompt(request.question)

        # Generate response using Ollama service
        answer = ollama_service.generate(prompt)

        logger.info("Chat request completed successfully")
        return ChatResponse(answer=answer)

    except HTTPException:
        raise
    except Exception as e:
        logger.exception(f"Unexpected error: {str(e)}")
        raise HTTPException(
```

```
            status_code=500,
            detail=f"Beklenmeyen hata: {str(e)}"
        )
```

Örnek 4.3: Ollama Service Client (backend/ollama_service.py)

```
class OllamaService:
    """Service class for interacting with Ollama API."""

    def generate(self, prompt: str, stream: bool = False) -> str:
        """Generate a response from Ollama."""
        try:
            payload = {
                "model": self.model,
                "prompt": prompt,
                "stream": stream
            }

            response = requests.post(
                self.api_url,
                json=payload,
                timeout=self.timeout
            )

            response.raise_for_status()

            data = response.json()
            answer = data.get("response", "").strip()

            if not answer:
                logger.warning("Empty response from Ollama")
                return "Üzgünüm, bir yanıt oluşturulamadı."

            return answer

        except requests.exceptions.Timeout:
            logger.error("Ollama request timeout")
            raise HTTPException(
                status_code=504,
                detail="Ollama isteği zaman aşımına uğradı"
            )
        except requests.exceptions.ConnectionError as e:
            logger.error(f"Cannot connect to Ollama: {str(e)}")
            raise HTTPException(
                status_code=503,
```

```
                detail=f"Ollama servisine bağlanılamadı: {str(e)}"
            )
```

Örnek 4.4: Prompt Engineering (backend/prompts.py)

```
def build_chat_prompt(question: str) -> str:
    """
    Build a prompt for the chat endpoint with Turkish context.
    """
    return f"""Sen Selçuk Üniversitesi için geliştirilmiş yapay zeka destekli bir a
Öğrencilere akademik konularda yardımcı oluyorsun.

Görevin: Aşağıdaki soruya detaylı, açıklayıcı ve akademik bir dille cevap vermek.

Önemli kurallar:
1. Cevaplarını Türkçe ver
2. Akademik ve resmi bir dil kullan
3. Bilmediğin konularda tahmin yapma, bilmediğini söyle
4. Gerekirse örneklerle açıkla

Öğrencinin sorusu: {question}

Cevabın:"""
```

Örnek 4.5: Unit Test Example (backend/test_main.py)

```
@patch('ollama_service.requests.post')
def test_chat_endpoint_success(mock_post):
    """Test successful chat request."""
    # Mock successful Ollama response
    mock_response = MagicMock()
    mock_response.status_code = 200
    mock_response.json.return_value = {
        "response": "Merhaba! Ben Selçuk Üniversitesi AI asistanıyım."
    }
    mock_post.return_value = mock_response

    # Make request
    response = client.post(
        "/chat",
        json={"question": "Merhaba"}
    )

    assert response.status_code == 200
```

```
data = response.json()
assert "answer" in data
assert data["answer"] == "Merhaba! Ben Selçuk Üniversitesi AI asistanıyım."
```

RAPOR SONU

Bu rapor, Selçuk Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği Bölümü Bilgisayar Mühendisliği Uygulamaları dersi kapsamında hazırlanmıştır.

Hazırlayan: Öğrencinin Adı SOYADI Danışman: Danışman Unvanı Adı SOYADI Tarih: Aralık 2025 Sayfa Sayısı: [Toplam sayfa sayısı]