T.C.
KIRKLARELİ ÜNİVERSİTESİ
TEKNİK BİLİMLER MESLEK YÜKSEKOKULU
BİLGİSAYAR TEKNOLOJİLERİ BÖLÜMÜ
BİLGİSAYAR PROGRAMCILIĞI PROGRAMI

Flutter'da On-Device Makine Öğrenmesi:

TensorFlow Lite ve Gemini API İncelemesi

CEREN YALNIZ

1247008014

İLERİ WEB PROGRAMLAMA

NADİR SUBAŞI

KIRKLARELİ

13.12.2025

İÇİNDEKİLER

KISALTMALAR

AI – Artificial Intelligence (Yapay Zekâ)

API – Application Programming Interface (Uygulama Programlama Arayüzü)

CPU – Central Processing Unit (Merkezi İşlem Birimi)

GPU – Graphics Processing Unit (Grafik İşlem Birimi)

ML – Machine Learning (Makine Öğrenmesi)

NLP – Natural Language Processing (Doğal Dil İşleme)

NNAPI – Neural Networks Application Programming Interface

TFLite – TensorFlow Lite

UI – User Interface (Kullanıcı Arayüzü)

UX – User Experience (Kullanıcı Deneyimi)

SDK – Software Development Kit (Yazılım Geliştirme Kiti)

LLM – Large Language Model (Büyük Dil Modeli)

RAM – Random Access Memory (Rastgele Erişimli Bellek)

GİRİŞ

Mobil teknolojilerin hızlı gelişimi, yapay zekâ uygulamalarının yalnızca masaüstü veya sunucu tabanlı sistemlerle sınırlı kalmasını engelleyerek mobil cihazlar üzerinde çalışan düşük gecikmeli ve yüksek performanslı makine öğrenmesi çözümlerine olan ihtiyacı artırmıştır. Bu gelişmeler sonucunda mobil uygulamalarda yapay zekânın temelde iki farklı yaklaşım üzerinden ele alındığı görülmektedir: cihaz içi (on-device) makine öğrenmesi ve bulut tabanlı büyük dil modelleri. On-device makine öğrenmesi, hesaplamaların doğrudan mobil cihaz üzerinde gerçekleştirildiği optimize edilmiş modelleri ifade ederken; bulut tabanlı yapay zekâ çözümleri, özellikle Google Gemini API örneğinde olduğu gibi, yüksek işlem gücü gerektiren doğal dil işleme süreçlerini sunucu tarafında yürütmektedir.

Bu raporda , modern mobil uygulama geliştirme aracı olan Flutter çerçevesinde bu iki yaklaşımın yapısal özellikleri ve kullanım mantıkları ele alınmaktadır. Amaç, TensorFlow Lite'ın cihaz içi makine öğrenmesi bağlamındaki rolü ile Google Gemini API'nin bulut tabanlı yapay zekâ yeteneklerini akademik bir bakış açısıyla incelemek ve bu iki yaklaşımın mobil yapay zekâ ekosistemindeki konumunu açıklamaktır.

On-Device Makine Öğrenmesinin Kavramsal Temelleri

On-device makine öğrenmesi, makine öğrenmesi modellerinin hesaplama süreçlerinin tamamen mobil cihaz üzerinde gerçekleştirilmesini esas alan bir yaklaşımdır. Bu yaklaşımın tercih edilmesinde üç temel faktör öne çıkmaktadır: gizlilik, düşük gecikme süresi ve bağımsız çalışma yeteneği.

1. Gizlilik ve Veri Güvenliği

Bulut tabanlı çözümlerde kullanıcıya ait görüntü, ses veya metin verilerinin işlenebilmesi için sunucuya aktarılması gerekmektedir. Bu durum, veri güvenliği ve kişisel verilerin korunması açısından çeşitli riskler doğurabilmektedir. On-device makine öğrenmesi ise verinin cihaz dışına çıkmasını engelleyerek kullanıcı gizliliğini en üst düzeyde korur. Bu nedenle sağlık, finans ve kişisel görüntü işleme gibi hassas verilerin işlendiği alanlarda cihaz içi yaklaşım giderek yaygın bir standart hâline gelmektedir.

2. Gecikme Süresi ve Performans

Gerçek zamanlı kamera ve sensör tabanlı uygulamalarda milisaniyelik gecikmeler dahi kullanıcı deneyimini olumsuz etkileyebilmektedir. Hesaplamaların doğrudan cihaz üzerinde yapılması, sunucuya veri gönderme ve yanıt bekleme sürecini ortadan kaldırarak gecikmeyi minimum seviyeye indirir. Bu durum, özellikle nesne tanıma, yüz tespiti ve hareket izleme gibi görevlerde performansın belirgin şekilde artmasını sağlar.

3. Bağımsız Çalışma Yeteneği

On-device makine öğrenmesi sistemlerinin internet bağlantısına ihtiyaç duymadan çalışabilmesi, mobil uygulamaların farklı koşullarda güvenilirliğini artırır. Kırsal bölgeler, saha çalışmaları veya bağlantının kesildiği durumlarda bu özellik büyük bir avantaj sağlar. Bu yönüyle on-device ML, endüstriyel uygulamalar ve acil durum senaryoları için uygun bir çözüm sunmaktadır.

On-Device Makine Öğrenmesi Mimarisi

On-device makine öğrenmesi mimarisi, makine öğrenmesi modellerinin tüm hesaplama aşamalarının mobil cihaz üzerinde yürütülmesi prensibine dayanmaktadır. Bu mimaride veri toplama, ön işleme, modelin çalıştırılması ve çıktı üretimi adımları cihazın kendi donanım kaynakları kullanılarak gerçekleştirilir. Sunucuya veri aktarımının olmaması, bu mimariyi klasik bulut tabanlı makine öğrenmesi sistemlerinden ayıran temel özelliktir.

Cihaz içi mimaride veri akışı genellikle mobil cihazın donanım bileşenlerinden başlar. Kamera, mikrofon veya hareket sensörleri gibi kaynaklardan elde edilen veriler uygulama içerisinde işlenerek makine öğrenmesi modeline girdi olarak sunulur. Model tarafından üretilen çıktılar ise yine aynı uygulama içinde kullanıcı arayüzüne aktarılır. Bu kapalı döngü yapı, veri aktarımına bağlı gecikmeleri ortadan kaldırarak daha hızlı ve kararlı bir çalışma ortamı sağlar.

Bu mimarinin önemli bileşenlerinden biri, donanım kaynaklarının etkin biçimde kullanılmasıdır. Mobil cihazlarda bulunan CPU, GPU ve bazı durumlarda özel yapay zekâ hızlandırıcıları, modelin çalıştırılmasında aktif rol oynar. Mimari tasarım, bu kaynakların dengeli kullanılmasını hedefleyerek hem performansın korunmasını hem de enerji tüketiminin kontrol altında tutulmasını amaçlar. Bu durum, uzun süreli kullanım gerektiren mobil uygulamalarda kullanıcı deneyimi açısından büyük önem taşımaktadır.

Gerçek zamanlı işlem gerektiren nesne tanıma, yüz tespiti ve hareket analizi gibi senaryolarda on-device ML mimarisi önemli avantajlar sunar. Ayrıca internet bağlantısına bağımlı olmaması, bu mimarinin farklı kullanım koşullarında güvenilir bir şekilde çalışmasını mümkün kılar.

Sonuç olarak on-device makine öğrenmesi mimarisi, veri gizliliği, düşük gecikme süresi ve donanım verimliliği gibi avantajlarıyla mobil yapay zekâ uygulamalarının temel yapı taşlarından biri hâline gelmiştir. Bu mimari yaklaşım, TensorFlow Lite gibi mobil odaklı makine öğrenmesi çözümlerinin gelişmesini sağlamış ve Flutter gibi modern mobil geliştirme platformlarıyla birlikte etkin bir kullanım alanı kazanmıştır.

On-Device ML’de Performans

On-device makine öğrenmesi yaklaşımında performans, mobil uygulamanın kullanıcı deneyimini doğrudan etkileyen temel unsurlardan biridir. Performans değerlendirmesi

yapılırken yalnızca modelin doğruluğu değil; gecikme süresi, enerji tüketimi ve donanım kaynaklarının verimli kullanımı gibi faktörler de dikkate alınmaktadır. Mobil cihazların sınırlı donanım kapasitesi göz önünde bulundurulduğunda, on-device ML çözümlerinde performans optimizasyonu kritik bir gereklilik hâline gelmiştir.

Performans açısından en önemli avantajlardan biri **düşük gecikme süresidir**. Hesaplamaların doğrudan cihaz üzerinde gerçekleştirilmesi, verinin sunucuya gönderilmesi ve yanıt beklenmesi sürecini ortadan kaldırır. Bu durum, özellikle kamera tabanlı nesne tanıma, yüz tespiti ve hareket analizi gibi gerçek zamanlı uygulamalarda daha hızlı ve tutarlı sonuçlar elde edilmesini sağlar. Kullanıcı etkileşimine anında yanıt verebilen uygulamalar, daha akıcı bir deneyim sunar.

Enerji tüketimi de on-device ML performansının değerlendirilmesinde önemli bir kriterdir. Mobil cihazlarda uzun süreli makine öğrenmesi işlemleri batarya tüketimini artırabilmektedir. Bu nedenle on-device ML mimarilerinde, modelin mümkün olan en kısa sürede çalıştırılması ve donanım kaynaklarının dengeli kullanılması hedeflenir. Optimize edilmiş modeller ve donanım hızlandırıcılarının kullanımı, enerji verimliliğinin artırılmasına katkı sağlar.

Donanım kaynaklarının etkin kullanımı, performansın sürdürülebilirliği açısından belirleyici bir faktördür. Mobil cihazlarda bulunan CPU ve GPU'nun yanı sıra, bazı platformlarda özel yapay zekâ hızlandırıcıları da makine öğrenmesi işlemlerinde kullanılabilmektedir. On-device ML çözümleri, bu donanım bileşenlerinden uygun olanları kullanarak hem işlem süresini kısaltmakta hem de sistem yükünü azaltmaktadır. Bu yaklaşım, mobil uygulamaların uzun süre stabil bir şekilde çalışmasını mümkün kılar.

Sonuç olarak, on-device makine öğrenmesinde performans; düşük gecikme, enerji verimliliği ve donanım kaynaklarının etkin kullanımı gibi unsurların dengeli bir biçimde yönetilmesine bağlıdır. Bu performans gereksinimleri, mobil ortama özel olarak geliştirilen TensorFlow Lite gibi çözümlerin önemini artırmakta ve on-device ML uygulamalarının yaygınlaşmasında belirleyici bir rol oynamaktadır.

Bu performans gereksinimleri, mobil cihazlara özel olarak geliştirilen makine öğrenmesi çözümlerinin önemini ortaya koymaktadır. Bu noktada TensorFlow Lite, on-device machine learning yaklaşımını destekleyen ve mobil donanımlar için optimize edilmiş yapısıyla öne çıkan temel teknolojilerden biridir. TensorFlow Lite, performans, enerji verimliliği ve düşük gecikme gibi ihtiyaçlara doğrudan cevap vererek mobil yapay zekâ uygulamalarının yaygınlaşmasında önemli bir rol üstlenmektedir.

TensorFlow Lite: Mobil Cihazlar için Optimize Edilmiş Makine Öğrenmesi

TensorFlow Lite, Google'ın TensorFlow modellerini mobil cihazlarda çalıştırmak için geliştirdiği hafif, optimize edilmiş bir sistemdir. Geleneksel TensorFlow modelleri yüksek işlem gücü gerektirdiği için mobil cihazlarda verimli çalışamazken, TFLite mimarisi bu sorunu çözmek üzere tasarlanmıştır.

1. Model Optimizasyonu ve Dönüştürme Süreci

TensorFlow modellerinin mobil cihazlarda verimli bir şekilde çalışabilmesi için TFLite Converter aracılığıyla kapsamlı bir dönüştürme ve optimizasyon sürecinden geçmesi gerekir. Bu sürecin ilk aşamasını oluşturan **kantizasyon**, model ağırlıklarının 32-bit kayan noktalı gösterimler yerine 16-bit ya da 8-bit gibi daha düşük çözünürlüklü biçimlere dönüştürülmesini sağlar. Bu dönüşüm, modelin boyutunu önemli ölçüde küçültürken, aynı zamanda hesaplamaların daha hızlı gerçekleştirilmesine olanak tanıyarak enerji tüketimini de azaltır. Kantizasyonun ardından uygulanan **pruning** tekniği ise modelde öğrenmeye anlamlı katkı sağlamayan, düşük öneme sahip bağlantıların sistematik biçimde kaldırılması esasına dayanır. Böylece modelin yapısal karmaşıklığı azaltılır ve performans kaybı yaşanmadan daha verimli bir mimari elde edilir.

Bu adımları tamamlayan **hızlandırıcı uyumlaştırma** aşamasında, dönüştürülen modelin mobil cihazlarda bulunan donanımsal hızlandırıcılarla uyumlu hâle getirilmesi hedeflenir. Android Neural Networks API (NNAPI), GPU Delegate ya da iOS platformundaki Metal benzeri hızlandırıcıların kullanımına olanak tanınması, modelin yalnızca CPU üzerinde değil, ilgili özel donanımlar üzerinde de çalışabilmesini sağlayarak performansı kayda değer ölçüde artırır. Bütün bu optimizasyon teknikleri bir araya geldiğinde, model hem çalışma hızı hem bellek kullanımı hem de enerji verimliliği açısından mobil cihazlara uygun, kompakt ve yüksek performanslı bir hâle dönüştürülmüş olur.

2 Çalışma Zamanı Yapısı

TensorFlow Lite'ın çalışma zamanı bileşeni olan **TFLite Interpreter**, modelin yürütülmesinden sorumlu temel mekanizmadır ve dönüştürülmüş TFLite modelinin belleğe alınmasından, giriş tensörlerinin işlenmesine ve nihai çıktıların üretilmesine kadar tüm süreci yönetir. Interpreter, modeli adım adım çalıştırırken hesaplamaları doğrudan cihazın donanımsal kaynakları üzerinde gerçekleştirir. Bu noktada işlem yalnızca CPU ile sınırlı kalmaz; cihazın sunduğu imkânlara bağlı olarak **GPU**, **NNAPI**, hatta özel amaçlı hızlandırıcı birimleri de devreye alınabilir. Böylece model, mümkün olan en yüksek verimle çalıştırılır.

Bu çalışma zamanı yapısının en dikkat çekici yönü, tüm işlemlerin cihaz üzerinde lokal olarak yürütülmesi sayesinde **düşük gecikme** ve **yüksek tepki hızı** sağlamasıdır. Özellikle gerçek zamanlı görüntü işleme, nesne takibi veya sensör verilerinin anlık analizini gerektiren uygulamalarda bu hız avantajı kritik önem taşır. Bunun yanında, modelin sürekli bir sunucu bağlantısına ihtiyaç duymadan çalışması, sistemin hem istikrarlı hem de kesintisiz bir performans sergilemesine imkân tanır. TFLite Interpreter'ın bu yapıdaki rolü, mobil cihazların sınırlı işlem gücüne rağmen makine öğrenmesi modellerini etkili bir biçimde çalıştırabilmesini sağlayarak on-device AI yaklaşımının temelini oluşturmaktadır.

TensorFlow Lite'ın bu çalışma zamanı yapısı, Flutter ile geliştirilen mobil uygulamalar açısından önemli avantajlar sunmaktadır. Flutter'ın kamera, sensör ve kullanıcı etkileşimine doğrudan erişim sağlayan yapısı, TFLite Interpreter tarafından işlenen verilerin hızlı bir şekilde kullanıcı arayüzüne aktarılmasına olanak tanır. Bu sayede Flutter tabanlı

uygulamalarda makine öğrenmesi çıktıları düşük gecikme ile sunulabilmekte ve gerçek zamanlı etkileşim sağlanabilmektedir. Bu özellikler, TensorFlow Lite'ı Flutter ekosisteminde cihaz içi makine öğrenmesinin temel bileşenlerinden biri hâline getirmektedir.

3 Kullanım Alanları

TensorFlow Lite, sahip olduğu hafif ve optimize edilmiş yapısı sayesinde mobil yapay zekâ ekosisteminin birçok farklı alanında etkin olarak kullanılmaktadır. En yaygın kullanım alanlarından biri **görüntü işleme** olup; nesne sınıflandırma, yüz tespiti ve insan poz tahmini gibi görevlerde yüksek doğruluk ve düşük gecikme sunması nedeniyle tercih edilmektedir. Bunun yanında, mobil cihazların mikrofon verilerini işleyebilme kapasitesi sayesinde **ses işleme** uygulamalarında da önemli bir rol üstlenmektedir. Uyandırma kelimesi algılama ("wake word detection") veya temel ses sınıflandırma modelleri bu alana örnek gösterilebilir.

TFLite, yalnızca kamera ve ses verileriyle sınırlı kalmayıp, modern akıllı telefonlarda yaygın olarak bulunan ivmeölçer, jiroskop gibi sensörlerden gelen verilerin analiz edilmesinde de kullanılmaktadır. Bu kapsamda **sensör verisi analizi**, adım sayma, hareket tespiti ya da titreşim örüntülerinin değerlendirilmesi gibi uygulamalarda sıkça karşımıza çıkar. Ayrıca TFLite, düşük boyutlu modelleri desteklemesi sayesinde **hafif doğal dil işleme (NLP)** görevlerinde de kullanılabilmektedir. Mobil ortama uygun küçük ölçekli metin sınıflandırma veya duygu analizi modelleri bu kategorinin tipik örnekleridir.

Tüm bu farklı alanlarda sağladığı esneklik ve performans, TensorFlow Lite'ın mobil yapay zekâ çözümlerinin geliştirilmesinde neden temel bir yapı taşına dönüştüğünü açık bir biçimde ortaya koymaktadır.

Flutter ile geliştirilen mobil uygulamalarda TensorFlow Lite, özellikle kamera tabanlı görüntü işleme, ses algılama ve sensör verisi analizi gibi görevlerde yaygın olarak kullanılmaktadır. Flutter'ın platformlar arası yapısı sayesinde aynı TensorFlow Lite modeli Android ve iOS cihazlarda çalıştırılabilmekte, bu da geliştirme sürecini daha verimli ve sürdürülebilir hâle getirmektedir. Bu yönüyle TensorFlow Lite, Flutter tabanlı mobil yapay zekâ uygulamalarında pratik ve ölçeklenebilir bir çözüm sunmaktadır.

TensorFlow Lite'ın Flutter'daki Entegrasyon Mantığı

Flutter ekosisteminde TensorFlow Lite entegrasyonu, genel olarak "modelin uygulamaya dahil edilmesi – çalışma zamanı bileşeninin (Interpreter) kurulması – giriş verisinin hazırlanması – çıkarım (inference) – çıktıların arayüze aktarılması" adımlarından oluşan bir iş akışıyla açıklanabilir. Uygulama tarafında kamera veya sensörlerden elde edilen veriler uygun formata dönüştürülerek modele giriş olarak verilir; Interpreter modeli çalıştırdıktan sonra üretilen sonuçlar Flutter arayüz bileşenlerine aktarılır. Bu yaklaşım, cihaz üzerinde düşük gecikmeli çıkarım yapılmasını mümkün kılarken, Flutter'ın hızlı arayüz oluşturma kapasitesi sayesinde sonuçların kullanıcıya anlık olarak yansıtılmasını sağlar.

Google Gemini API: Büyük Dil Modellerinin Mobil Uygulamalara Entegrasyonu

Google Gemini, geniş ölçekli parametre yapısı ve çok katmanlı mimarisi sayesinde metin üretimi, mantıksal akıl yürütme, kod üretimi, içerik çözümleme ve çok modlu (metin, görüntü vb.) veri işleme gibi karmaşık görevlerde yüksek performans sunan modern bir yapay zekâ modelidir. Ancak bu tür büyük dil modelleri, yüksek hesaplama yükü ve bellek gereksinimleri nedeniyle mobil cihazların donanım kapasitesini aşmaktadır. Bu nedenle Gemini, doğrudan cihaz üzerinde çalıştırılmak yerine bulut tabanlı bir API aracılığıyla mobil uygulamalara entegre edilmektedir. Bu mimaride hesaplama süreçleri güçlü sunucu altyapılarında gerçekleştirilirken, mobil cihaz temel olarak kullanıcı girdisini iletmek ve üretilen çıktıyı kullanıcıya sunmakla görevli hâle gelir. Böylece cihazın işlem gücü ve batarya kapasitesi zorlanmadan gelişmiş yapay zekâ yeteneklerinin mobil uygulamalarda kullanılması mümkün olur.

1. Büyük Dil Modellerinin Temel Yapısı

Büyük dil modelleri, milyarlarca parametre içeren derin sinir ağı mimarilerine dayanan ve dilin yapısını farklı soyutlama düzeylerinde öğrenebilen gelişmiş yapay zekâ sistemleridir. Geniş veri kümeleri üzerinde eğitilmeleri sayesinde dildeki anlam ilişkilerini, sözdizimsel yapıları ve bağlamdan kaynaklanan nüansları temsil edebilirler. Bu temsil gücü, modellerin yalnızca metni analiz etmesini değil; aynı zamanda bağlama uygun yanıt üretmesini, tutarlı açıklamalar yapmasını ve yeni içerikler oluşturmasını mümkün kılar.

Bu modellerin önemli bir özelliği, çok adımlı muhakeme gerektiren görevlerde de etkili olabilmeleridir. Problemi alt parçalara ayırarak çıkarım yapma, metin içinde örtük ilişkileri ortaya çıkarma veya birden fazla adım içeren akıl yürütme süreçleri, büyük dil modellerinin mobil uygulamalarda daha “yardımcı” ve “etkileşimli” sistemler olarak kullanılmasına katkı sağlamaktadır. Bu yönüyle büyük dil modelleri, metin odaklı yapay zekâ uygulamalarının temel teknolojilerinden biri olarak kabul edilmektedir.

2. API Tabanlı Mimari

Gemini API, büyük dil modellerinin mobil uygulamalarda etkin biçimde kullanılmasını sağlayan bulut tabanlı bir mimariye sahiptir. Süreç, mobil uygulamanın kullanıcıdan aldığı girdiyi API aracılığıyla sunucuya iletmesiyle başlar. Sunucu tarafında model girdiyi işler, bağlamı analiz eder ve talebe uygun bir çıktı üretir. Üretilen yanıt daha sonra mobil cihaza geri gönderilir ve uygulama tarafından kullanıcıya sunulur.

Bu yaklaşımın temel avantajı, hesaplama yükünün cihazdan alınarak bulut altyapısına aktarılmasıdır. Böylece mobil cihazın işlemcisi ve belleği zorlanmadan karmaşık yapay zekâ işlemleri gerçekleştirilebilir. Ayrıca model sunucu tarafında bulunduğu için güncellemeler merkezi olarak yapılabilir ve kullanıcılar uygulama güncellemesi gerekmeksizin daha güncel model sürümlerinden yararlanabilir. Bu yönüyle API tabanlı mimari, ölçeklenebilir ve sürdürülebilir bir çözüm sunar.

Flutter tabanlı mobil uygulamalarda Gemini API entegrasyonu, genellikle istemci–sunucu temelli bir istek–yanıt modeli üzerinden gerçekleştirilir. Kullanıcıdan alınan metin girdileri Flutter uygulaması tarafından API'ye gönderilir; sunucu tarafında Gemini modeli bu girdiyi işleyerek bağlama uygun bir çıktı üretir. Üretilen yanıt tekrar mobil cihaza iletilir ve Flutter arayüz bileşenleri aracılığıyla kullanıcıya sunulur. Bu yaklaşım, mobil cihaz üzerinde yüksek hesaplama gerektirmeden gelişmiş dil modeli yeteneklerinin Flutter uygulamalarında kullanılmasını mümkün kılar.

3. Uygulama Alanları

Gemini API'nın sunduğu yetenekler, mobil uygulamalarda geniş bir kullanım alanı oluşturur. Metin üretimi ve yaratıcı yazım, uzun içeriklerden özet çıkarma ve kullanıcıyla doğal diyalog kurabilen sohbet botları bu alanların başında gelmektedir. Eğitim uygulamalarında ise açıklama üretme, örnek problem çözümü sunma veya kişiselleştirilmiş öğrenme desteği sağlama gibi işlevlerle kullanıcı deneyimini zenginleştirebilir.

Buna ek olarak, çok adımlı muhakeme becerisi gerektiren mantıksal akıl yürütme görevlerinde de kullanılabilmektedir. Yazılım geliştirme alanında ise kod üretimi, kodun açıklanması veya hata tespitine yardımcı olma gibi süreçlerde destekleyici bir bileşen olarak değerlendirilebilir. Bu çeşitlilik, Gemini API'yı modern mobil uygulamalarda çok yönlü ve güçlü bir bulut tabanlı yapay zekâ bileşeni hâline getirmektedir.

Mobil uygulama tasarımında Gemini API çoğu zaman cihaz içi makine öğrenmesi çözümleriyle birlikte tamamlayıcı bir rol üstlenmektedir. Gerçek zamanlı ve gizlilik odaklı görevler TensorFlow Lite gibi on-device modellerle gerçekleştirilirken; metin üretimi, özetleme veya mantıksal muhakeme gibi daha karmaşık görevler Gemini API üzerinden yürütülebilmektedir. Bu hibrit yaklaşım, mobil uygulamaların hem performans hem de işlevsellik açısından daha dengeli bir yapı kazanmasını sağlamaktadır.

On-Device ML ve Bulut Tabanlı Modellerin Karşılaştırmalı Analizi

Cihaz içi makine öğrenmesi ile bulut tabanlı yapay zekâ modelleri, mimari olarak farklı gereksinimlere yanıt veren iki ayrı yaklaşımı temsil etmektedir. Bu nedenle her iki sistemin güçlü yönleri, sınırlılıkları ve kullanım alanları da birbirinden ayrışmaktadır. On-device makine öğrenmesi, verilerin yerel olarak işlenmesini temel alarak gizlilik, düşük gecikme süresi ve çevrimdışı çalışma gibi avantajlar sunarken; bulut tabanlı modeller, yüksek hesaplama kapasitesi sayesinde karmaşık dil işleme ve analiz görevlerinde üstün performans sağlamaktadır. Bu karşılaştırmanın amacı, tek bir yaklaşımın mutlak olarak daha iyi olduğunu göstermekten ziyade, hangi yöntemin hangi kullanım senaryosu için daha uygun olduğunu ortaya koymaktır.

1. Hesaplama Kapasitesi

On-device makine öğrenmesi, mobil cihazların sınırlı işlem gücü, bellek kapasitesi ve enerji tüketimi gibi kısıtları nedeniyle genellikle optimize edilmiş ve daha küçük ölçekli modellerle

çalışmayı gerektirir. Bu nedenle kantizasyon ve pruning gibi teknikler kullanılarak model boyutu küçültülür ve hesaplama maliyeti azaltılır. Buna karşılık, Google Gemini API gibi bulut tabanlı yapay zekâ sistemleri, mobil cihazlarda çalıştırılması mümkün olmayan büyük ve karmaşık model mimarilerini barındırabilir. Sunucu altyapılarında çalışan bu modeller, çok daha fazla parametreye sahip olup dil anlama, içerik üretimi ve çok adımlı muhakeme gibi görevlerde yüksek doğruluk oranları sunabilmektedir. Bu açıdan iki yaklaşım arasındaki temel fark, kullanılabilir hesaplama kapasitesinin ölçeği ve buna bağlı olarak gerçekleştirilebilecek görevlerin karmaşıklığıdır.

2. Gizlilik Düzeyi

Gizlilik açısından değerlendirildiğinde, on-device makine öğrenmesi belirgin bir avantaja sahiptir. Tüm veri işleme süreçlerinin cihaz üzerinde gerçekleştirilmesi, kullanıcıya ait görüntü, ses veya konum verilerinin herhangi bir sunucuya aktarılmasını engeller. Bu durum, özellikle hassas kişisel verilerin işlendiği uygulamalarda güvenlik risklerini önemli ölçüde azaltır. Buna karşılık, bulut tabanlı modellerde verinin sunucuya iletilmesi zorunlu olduğundan aktarım ve depolama süreçlerinde ek güvenlik riskleri ortaya çıkabilmektedir. Her ne kadar bu sistemler gelişmiş şifreleme ve güvenlik protokolleri kullansa da, verinin cihaz dışına çıkması başlı başına bir risk faktörü olarak değerlendirilmektedir. Bu nedenle gizlilik gereksinimi yüksek olan uygulamalarda on-device yaklaşım daha uygun bir çözüm sunmaktadır.

3. Bağımlılık Yapısı

Bağımlılık yapısı bakımından iki yaklaşım arasında belirgin bir ayrım bulunmaktadır. On-device makine öğrenmesi çözümleri, tüm hesaplamaları yerel donanım üzerinde gerçekleştirdiği için internet bağlantısına ihtiyaç duymaz. Bu özellik, uygulamaların çevrimdışı veya ağ bağlantısının zayıf olduğu ortamlarda dahi kesintisiz çalışabilmesini sağlar. Buna karşılık, bulut tabanlı yapay zekâ modelleri tüm işlem sürecini sunucu tarafında yürüttüğünden sürekli ve kararlı bir internet bağlantısına bağımlıdır. Kullanıcı girdisinin sunucuya iletilmesi ve yanıtın geri alınması, uygulamanın erişilebilirliğini doğrudan ağ koşullarına bağlamaktadır. Dolayısıyla iki yaklaşım arasındaki bu fark, mobil uygulamanın hangi koşullarda kullanılacağına göre yöntem seçimini doğrudan etkilemektedir.

Uygulamada bu iki yaklaşım çoğu zaman birbirinin alternatifi olmaktan ziyade tamamlayıcısı olarak konumlandırılmaktadır. Gerçek zamanlı, gizlilik odaklı ve çevrimdışı çalışması gereken görevler cihaz içi makine öğrenmesi çözümleriyle gerçekleştirilirken; metin üretimi, özetleme ve mantıksal muhakeme gibi daha karmaşık işlemler bulut tabanlı büyük dil modelleri aracılığıyla yürütülmektedir. Bu hibrit yaklaşım, mobil uygulamaların hem performans hem de işlevsellik açısından dengeli bir yapı kazanmasını sağlamaktadır.

Bu farklılıklar, mobil yapay zekâ uygulamalarının tasarımında hangi yöntemin tercih edileceğini belirlemede akademik bir değerlendirme çerçevesi sunmaktadır.

Flutter’ın Yapay Zekâ Teknolojilerini Destekleme Kapasitesi

Flutter, Dart programlama diline dayanan modern ve yüksek performanslı bir mobil geliştirme çerçevesi olarak yapay zekâ tabanlı teknolojilerin entegrasyonuna son derece elverişli bir mimari sunmaktadır. Hem TensorFlow Lite’ın cihaz içi makine öğrenmesi modelleriyle hem de Google Gemini API’nin bulut tabanlı yapay zekâ hizmetleriyle uyumlu çalışabilmesi, Flutter’ı akıllı mobil uygulama geliştirme süreçlerinde önemli bir konuma taşımaktadır. Bu özellikleri sayesinde Flutter, cihaz içi ve bulut tabanlı yapay zekâ yaklaşımlarının tek bir uygulama mimarisi içinde birlikte kullanılmasına olanak tanımaktadır.

Flutter’ın bu alandaki gücü, sunduğu teknik olanaklarla desteklenmektedir. Platformlar arası uyumluluk sayesinde geliştiriciler, Android ve iOS için tek bir kod tabanı üzerinden uygulama geliştirebilmekte; bu durum geliştirme süresini ve bakım maliyetlerini önemli ölçüde azaltmaktadır. Ayrıca Flutter, kamera ve sensör verilerine erişimi kolaylaştıran güçlü eklenti desteği sunarak görüntü işleme ve sensör tabanlı makine öğrenmesi uygulamalarının geliştirilmesini kolaylaştırmaktadır.

Bunun yanı sıra, Flutter’ın GPU hızlandırmalı grafik motoru, yoğun hesaplama gerektiren işlemlerin ve kullanıcı arayüzü animasyonlarının akıcı biçimde çalışmasını mümkün kılarak kullanıcı deneyimini iyileştirir. Zengin paket ekosistemi, TensorFlow Lite ve Gemini API gibi teknolojilerin uygulamalara hızlı ve sorunsuz biçimde entegre edilmesine imkân tanır. Flutter’ın düşük gecikmeli arayüz işleme kapasitesi ise yapay zekâ tabanlı çıktılarla kullanıcı etkileşiminin anlık olarak yansıtılmasını sağlar.

Bu yönleriyle Flutter, hem akademik çalışmalar hem de endüstriyel uygulamalar için akıllı mobil sistemlerin geliştirilmesinde güçlü, esnek ve geleceğe yönelik bir platform sunmaktadır.

Sonuç

Mobil yapay zekâ uygulamalarında hem cihaz içi hem de bulut tabanlı yaklaşımların önemli roller üstlendiği görülmektedir. TensorFlow Lite, mobil cihazların sınırlı donanım kaynaklarını verimli biçimde kullanarak düşük gecikmeli ve gizlilik odaklı işlemleri mümkün kılarken; Google Gemini API, gelişmiş dil işleme ve muhakeme yetenekleri sayesinde cihaz üzerinde gerçekleştirilemeyecek kadar karmaşık görevleri yüksek doğrulukla yerine getirebilmektedir. Flutter’ın bu iki teknolojiyi aynı ekosistem içerisinde bir araya getirebilmesi, mobil uygulamaların hem performans hem de işlevsellik açısından daha üst düzey çözümler sunmasına olanak tanımaktadır.

Bu çerçevede mobil yapay zekânın geleceği, cihaz üzerinde çalışan hızlı ve optimize edilmiş modeller ile bulut destekli güçlü büyük dil modellerinin birlikte kullanıldığı hibrit mimariler doğrultusunda şekillenmektedir. Böyle bir yaklaşım, kullanıcı deneyimini zenginleştirirken geliştiricilere de esnek, ölçeklenebilir ve sürdürülebilir uygulamalar tasarlama imkânı sunmaktadır.

KAYNAKÇA

Google. (2024). *TensorFlow Lite documentation*.
https://www.tensorflow.org/lite

Google. (2024). *TensorFlow Lite performance and optimization*.
https://www.tensorflow.org/lite/performance

Google. (2024). *Flutter documentation*.
https://docs.flutter.dev

Google. (2024). *Using TensorFlow Lite with Flutter*.
https://pub.dev/packages/tflite_flutter

Google. (2024). *Gemini API documentation*.
https://ai.google.dev

Brown, T. B., Mann, B., Ryder, N., et al. (2020). *Language models are few-shot learners*. Advances in Neural Information Processing Systems (NeurIPS).

Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). *Attention is all you need*. Advances in Neural Information Processing Systems (NeurIPS).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. (2017). *Efficient processing of deep neural networks: A tutorial and survey*. Proceedings of the IEEE.