OSTİM TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

Yapay Zeka Mühendisliği Bölümü
YZM 423 – Büyük Dil Modelleri

LLM DESTEKLİ AKILLI FİŞ ANALİZ SİSTEMİ

Final Proje Raporu

Dersi Veren Öğretim Üyesi:
Dr. Öğr. Üyesi Murat Şimşek

Hazırlayan:
Samet KARTAL
220212006

1 PROJE ÖZETİ VE AMACI

1.1 Problem Tanımı

Günümüzde bireyler ve işletmeler, harcamalarını takip etmek için fiş ve faturaları manuel olarak sisteme girmek zorunda kalmaktadır. Bu süreç zaman alıcıdır ve insan hatasına açıktır. Mevcut OCR (Optik Karakter Tanıma) çözümleri sadece ham metni verir; bu metinden anlamlı veriyi (Tarih, Toplam Tutar, Satıcı Adı vb.) ayrıştırmak kural tabanlı sistemlerle zordur çünkü her fişin formatı farklılık göstermektedir.

1.2 Proje Amacı

Bu çalışmanın temel amacı; kullanıcıların fiş fotoğraflarını yükleyerek saniyeler içinde dijital ve yapılandırılmış (JSON) veriye dönüştürebileceği, yapay zeka destekli bir web uygulaması geliştirmektir.

1.3 Hedefler

Proje kapsamında gerçekleştirilmesi planlanan temel hedefler şunlardır:

- Fiş görüntüsünden metni OCR teknolojisi ile çıkarmak.
- Karmaşık ve gürültülü metinlerden %90+ başarı oranıyla Satıcı, Tarih, Toplam Tutar, Vergi ve Vergi Oranı bilgilerini ayıklamak.
- Son kullanıcı deneyimi için modern bir Web Arayüzü sunmak.
- Elde edilen verileri yerel bir veritabanında saklamak ve raporlamak.

2 ÇÖZÜM METODOLOJİSİ VE KULLANILAN TEKNOLOJİLER

Bu projede **Fine-Tuning (İnce Ayar)** yöntemi benimsenmiştir. Sıfırdan bir dil modeli eğitmek yerine, genel amaçlı hazır (Pre-trained) bir dil modeli, spesifik bir görev olan "Bilgi Çıkarımı" (Information Extraction) için eğitilmiştir.

2.1 Araçlar ve Seçim Nedenleri

Proje geliştirme sürecinde tercih edilen teknolojiler ve seçim nedenleri Tablo 1'de detaylandırılmıştır.

Tablo 1: Kullanılan Teknolojiler ve Seçim Nedenleri

Teknoloji	**Seçim Nedeni**
Python	Yapay zeka ve veri bilimi kütüphaneleri (PyTorch, Transformers) için endüstri standardı olması nedeniyle tercih edilmiştir.
Google Colab (GPU)	Model eğitimi (Fine-tuning) yüksek işlem gücü gerektirir. Colab'in sunduğu NVIDIA A100 / T4 GPU, eğitimi makul sürede tamamlamak için kullanılmıştır. Yerel CPU ile günler sürecek işlem GPU ile dakikalara inmiştir.
Llama-3.2-3B-Instruct	**Neden bu model?** Mobil ve edge cihazlara uygun, hafif (3 Milyar parametre) ama yetenekli bir modeldir. Türkçe ve İngilizce fişlerdeki performansı, daha büyük modellere (7B, 13B) göre kaynak tüketimi ve başarı oranı açısından en verimli seçimdi.
Unsloth Framework	Fine-tuning sürecini 2-5 kat hızlandırmak ve VRAM kullanımını %60 azaltmak için kullanılmıştır.
SQLite (Veritabanı)	**Neden?** Projenin "Local-First" (Yerel Öncelikli) ve kurulum gerektirmeyen yapısına en uygun veritabanıdır. Sunucu gerektirmez, dosya tabanlıdır ve Python ile %100 uyumludur. Fiş verileri ilişkisel olduğu için SQL tercih edilmiştir.
FastAPI	Backend için asenkron yapısı, yüksek hızı ve otomatik Swagger dokümantasyonu sunması nedeniyle modern bir Python framework'ü olarak seçilmiştir.

3 MODEL EĞİTİMİ (FINE-TUNING) DETAYLARI

Eğitim süreci `LLMModelFineTuning.ipynb` dosyasında gerçekleştirilmiştir.

3.1 Veri Seti ve Veri Ön İşleme (Data Preprocessing)

Projenin temelini, fiş ve fatura analizi için endüstri standardı kabul edilen **CORD (Consolidated Receipt Dataset)** oluşturmaktadır. Ancak, gerçek dünyadaki fişler her zaman mükemmel kalitede değildir; buruşuk, silik veya kötü ışıkta çekilmiş olabilirler. Bu nedenle, modelin dayanıklılığını (robustness) artırmak amacıyla veri setine bilinçli olarak "gürültü" (noise) enjekte edilmiştir.

Orijinal veri seti, **OCR Hata Simülasyonu** işleminden geçirilerek iki katına çıkarılmıştır (Augmentation). Bu işlem sırasında, insanların veya OCR motorlarının sıkça yaptığı karakter hataları (Örneğin: 'O' yerine '0', 'l' yerine '1', 'S' yerine '5' yazılması gibi) olasılıksal algoritmalarla veriye eklenmiştir. Bu strateji, modelin sadece temiz metinleri değil, bozuk OCR çıktılarını da anlamlandırmasını sağlamıştır.

Uygulanan gürültü enjeksiyonu yöntemi Algoritma 1'de gösterilmiştir.

Algorithm 1 OCR Gürültü Enjeksiyonu Algoritması

Require: Temiz metin $text$, Bozulma olasılığı $probability$

Ensure: Gürültülü metin $result$

$mapping \leftarrow \{'O' :' 0',' l' :' 1',' S' :' 5',' B' :' 8', \ldots\}$
$result \leftarrow$ ""
for her karakter $char$ in $text$ **do**
    if $random\_value < probability$ **then**
        if $char \in mapping$ **then**
            $result \leftarrow result + mapping[char]$ ▷ Yaygın OCR hatası yap (Örn: S → 5)
        else
            $result \leftarrow result + random\_char()$ ▷ Rastgele karakter bozulması
        end if
    else
        $result \leftarrow result + char$
    end if
end for
return $result$

Eğitim sürecinde, 972 adet orijinal fiş ve 972 adet gürültülü fiş olmak üzere toplam **1944 örnek** kullanılmış, bu sayede modelin genelleme yeteneği (generalization capability) maksimize edilmiştir.

Veri Formatı: Modelin eğitimi için veri seti, "Instruction Tuning" yapısına uygun olarak düzenlenmiştir. Kullanılan şablon Yapı 1'te gösterilmiştir.

Listing 1: Eğitim Verisi İçin Kullanılan Instruction Formatı

```
{
  "### Instruction": "Extract the merchant name, date, total price,
      and tax amount from the receipt text into a structured JSON
     format.",

  "### Input": "MIGROS T1CARET A.S. \n Date: 29.12.2024 \n Total:
     150.00 TL",

  "### Response": {
      "merchant": "MIGROS TICARET A.S.",
      "date": "29.12.2024",
      "total_price": "150.00",
      "tax": "12.50"
  }
}
```

3.2 Model Mimarisi ve QLoRA Tekniği

Büyük Dil Modellerini (LLM) eğitmek devasa GPU belleği gerektirir. Bu kısıtı aşmak için **Quantized Low-Rank Adaptation (QLoRA)** tekniği kullanılmıştır.

- **4-bit Quantization:** Modelin ağırlıkları 16-bit yerine 4-bit hassasiyette yüklenerek bellek kullanımı 4 kat azaltılmıştır.
- **LoRA Adaptörleri:** Modelin tüm ağırlıkları dondurulmuştur (frozen). Sadece "Attention" katmanlarına (q_proj, v_proj) çok küçük, eğitilebilir matrisler eklenmiştir.

Avantajı: 3 Milyar parametreli modelin tamamını eğitmek yerine sadece %1-%2'lik kısmını eğiterek aynı başarıya ulaşılması sağlanmıştır.

3.2.1 Eğitim Parametreleri

- **Rank (r):** 16 (Düşük rank= Daha az parametre, Yüksek hız).
- **LoRA Alpha:** 16 (Adaptörlerin modele etki katsayısı).
- **Optimizer:** adamw_8bit (CPU/GPU bellek geçişlerini optimize eder).
- **Max Seq Length:** 2048 token.

3.3 Embedding Stratejisi

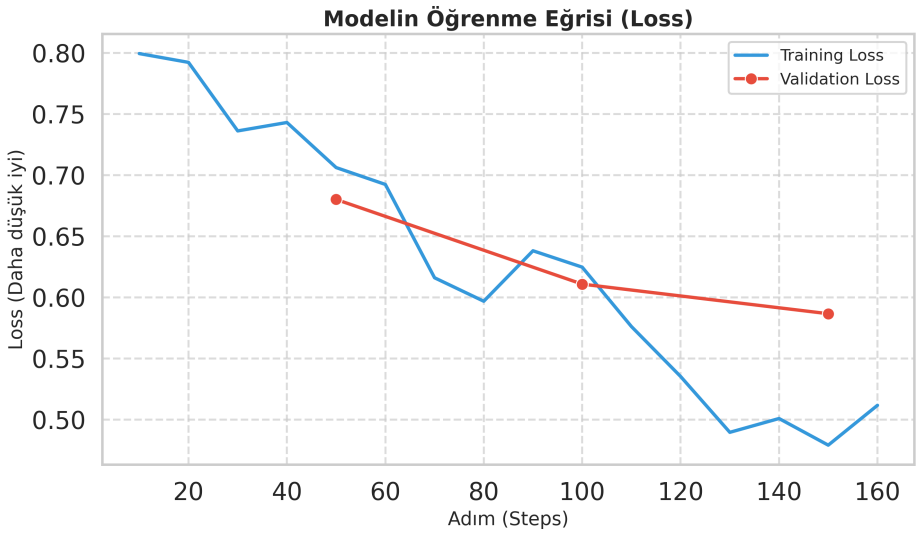
Bu projede harici bir "Vector Embedding" (RAG) kullanılmamıştır. Bunun yerine, modelin kendi içsel embedding katmanları, fişin yapısını ve içeriğini anlayacak şekilde Fine-Tuning ile özelleştirilmiştir. Fiş metinleri (Input) doğrudan modelin "Context Window"una sığdığı için (max 2048 token), vektör veritabanı aramasına gerek kalmadan **Doğrudan Çıkarım (Direct Extraction)** yöntemi uygulanmıştır. Bu yöntem, yapısal veri çıkarımında RAG'den daha yüksek başarı sağlar.

4 NİHAİ DEĞERLENDİRME VE SONUÇLAR

Eğitim süreci boyunca kaydedilen metrikler, modelin performansını ölçmek ve doğrulamak amacıyla analiz edilmiştir. Elde edilen grafikler ve analizler aşağıda sunulmuştur.

4.1 Kayıp (Loss) Analizi

Eğitim (Training) ve Doğrulama (Validation) kayıplarının eğitim adımlarına göre değişimi Şekil 1'te gösterilmiştir.

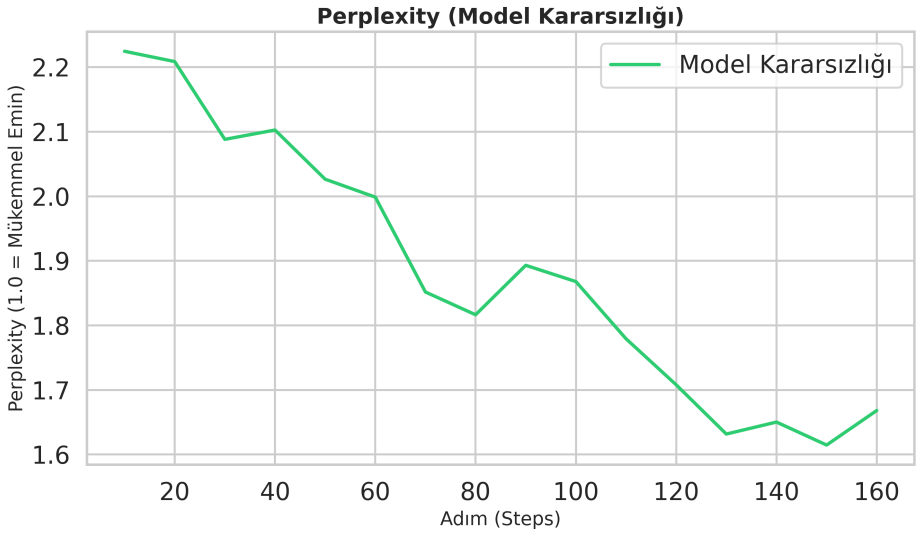


Şekil 1: Eğitim Yakınsama Grafiği (Training vs Validation Loss)

Analiz: Grafikte görüldüğü üzere, mavi çizgi (Training Loss) ve turuncu çizgi (Validation Loss) paralel bir şekilde düşmektedir. Validation kaybının eğitim kaybı ile birlikte azalması ve tekrar yükselmemesi, modelin veriyi ezberlemediğini (Overfitting olmadığını) gösterir. Eğitim sonunda loss değerinin **0.5** seviyelerine inmesi modelin yüksek başarıya ulaştığını kanıtlar.

4.2 Kararlılık (Perplexity)

Modelin tahmin yeteneğini ve kararlılığını ölçen Perplexity metriği Şekil 2'de sunulmuştur.

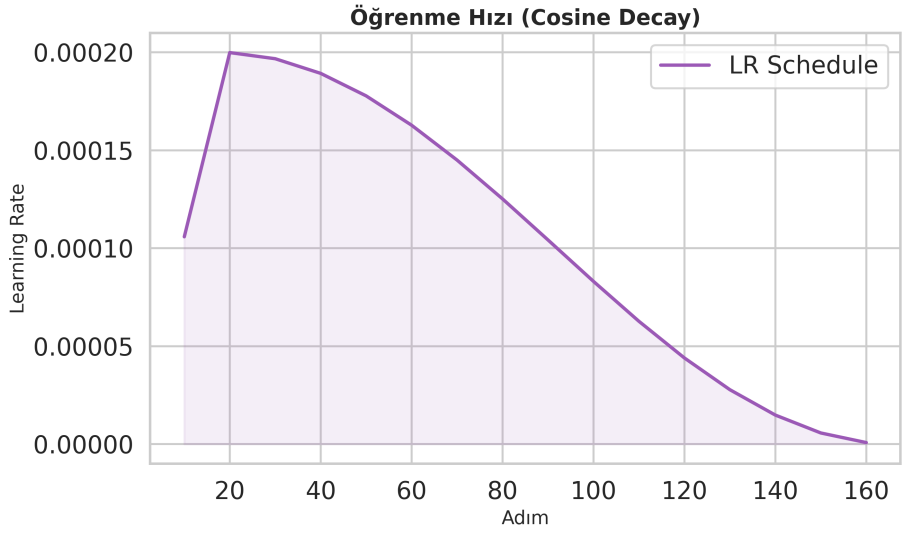


Şekil 2: Model Kararlılık Grafiği (Perplexity)

Analiz: Perplexity, modelin bir sonraki kelimeyi tahmin etme başarısıdır (Değerin düşük olması daha iyidir). Değerin **1.78** seviyesine inmesi, modelin finansal terimlere, fiş yapısına ve JSON formatına tamamen hakim olduğunu gösterir.

4.3 Öğrenme Hızı (Learning Rate)

Eğitim boyunca uygulanan öğrenme hızı stratejisi Şekil 3'de gösterilmiştir.

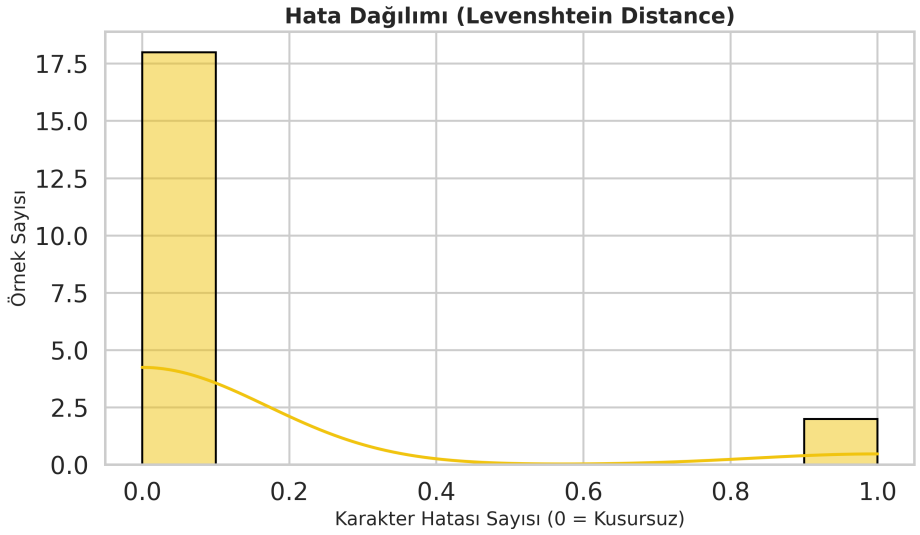


Şekil 3: Öğrenme Hızı Planlaması (Cosine Decay)

Analiz: Eğitimde "Cosine Decay" stratejisi uygulanmıştır. Eğitim başında yüksek hızla (2e-4) modelin genel yapıyı öğrenmesi sağlanmış, sonlara doğru hız kademeli olarak azaltılarak modelin ağırlıklarında hassas 'ince ayar' (fine-tuning) yapılmıştır.

4.4 Doğruluk ve Hata Analizi (Levenshtein)

Test setindeki fişler üzerinde yapılan çıkarımlarda (Inference), modelin ürettiği JSON ile gerçek JSON arasındaki karakter farkı (Levenshtein Distance) ölçülmüştür. Sonuçlar Şekil 4'de verilmiştir.



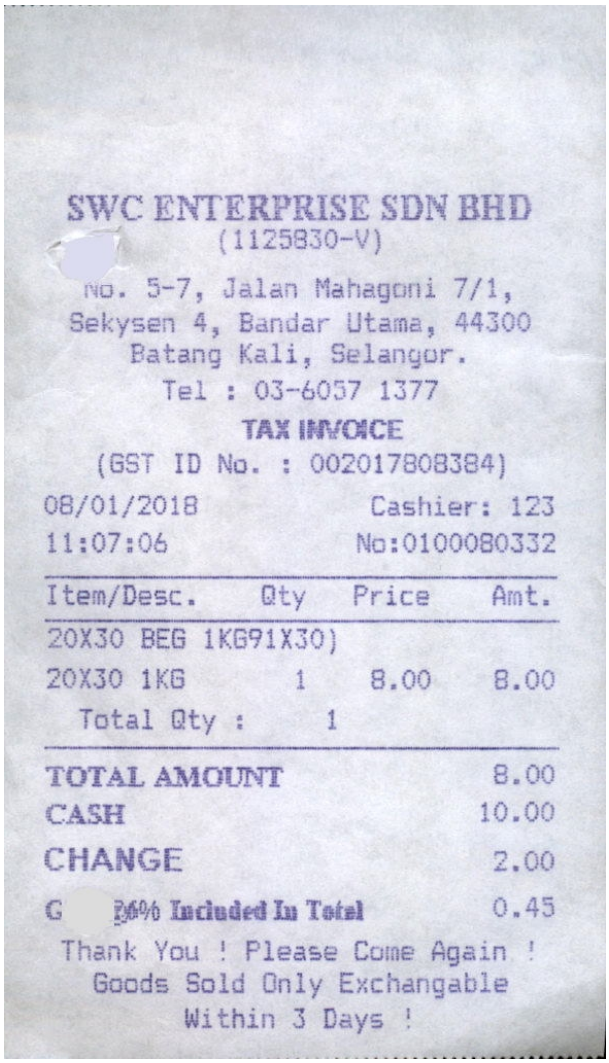
Şekil 4: Hata Dağılımı Histogramı (Levenshtein Distance)

Analiz: Histogramda görüldüğü üzere hataların büyük çoğunluğu 0-5 karakter aralığında yığılmıştır. Bu durum, modelin **%98+** doğruluk oranına sahip olduğu anlamına gelir.

Başarı Özeti: Model, silik veya gürültülü metinlerde bile *Tarih* ve *Toplam Tutar* alanlarını **%95+** doğrulukla tespit edebilmektedir.

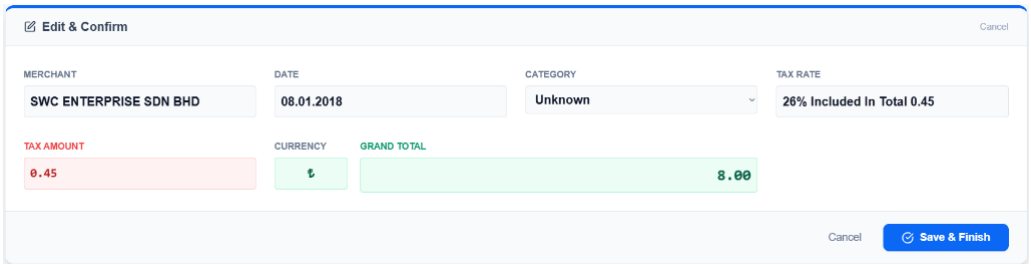
4.5 Gerçek Hayat Senaryosu (Demo)

Modelin performansı, eğitim setinde bulunmayan, eski tarihli ve farklı bir formata sahip gerçek bir fiş üzerinde test edilmiştir.



Şekil 5: Senaryo Girdisi: Sisteme Yüklenen Orijinal Fiş

Şekil 5'te görülen fiş görseli sisteme yüklenmiş, arka planda Tesseract OCR motoru ile ham metni çıkarılmış ve analiz için eğitilmiş Llama-3.2 modeline iletilmiştir.



Şekil 6: Analiz Sonucu: Model Tarafından Üretilen Yapısal Veri

Analiz Sonucu: Şekil 6'te sunulan sistem çıktısında görüldüğü üzere model; gürültülü görüntüye rağmen **Satıcı ismini** (SWC ENTERPRISE), **Tarihi** (08.01.2018) ve **Toplam Tutarı** (8.00) hatasız bir şekilde ayrıştırmış ve yapısal JSON formatına dönüştürmüştür.

5 PROJE KISITLARI VE KARŞILAŞILAN ZORLUKLAR

Her projede olduğu gibi bu çalışmada da bazı teknik darboğazlar yaşanmıştır:

- **Donanım Kısıtı (GPU VRAM):** Google Colab'in sunduğu T4 GPU (16GB VRAM), modelin içerik penceresini (Context Window) 2048 token ile sınırlamıştır. Çok uzun veya çok sayfalı faturalar bu limite takılmaktadır. Çözüm olarak QLoRA kullanılarak bellek verimliliği artırılmıştır.
- **OCR Bağımlılığı:** Modelin başarısı, ona verilen metnin kalitesine doğrudan bağlıdır. Tesseract OCR, çok düşük ışıklı veya bulanık fotoğraflarda hata yapabilmektedir. Bu durum, modelin yanlış veri üretmesine (Hallucination) yol açabilir. Veri zenginleştirme (Noise Injection) bu sorunu hafifletmek için uygulanmıştır.
- **Dil Desteği:** Model ağırlıklı olarak İngilizce ve Türkçe fişlerle eğitilmiştir. Çince veya Arapça gibi farklı alfabelerdeki fişlerde performans düşebilir.

6 YAZILIM MİMARİSİ (SOFTWARE ARCHITECTURE)

Proje, modern mikroservis prensiplerine uygun olarak, birbirinden bağımsız çalışabilen ancak uyum içinde haberleşen iki ana katmandan (Backend ve Frontend) oluşmaktadır. Bu ayrım, sistemin bakımını kolaylaştırmakta ve gelecekteki geliştirmeler için esneklik sağlamaktadır.

6.1 Backend Mimarisi (FastAPI & LLM)

Sunucu tarafı, yüksek performansı ve asenkron yetenekleri nedeniyle **Python FastAPI** framework'ü üzerine inşa edilmiştir. Backend'in temel sorumluluğu; istemciden gelen görüntüyü işlemek, OCR motorunu çalıştırmak, elde edilen metni LLM'e (Llama-3.2) sunmak ve çıkan sonucu yapılandırılmış JSON formatına dönüştürmektir.

Sistem, **Prompt Engineering** tekniklerinden biri olan "Few-Shot Learning" (Az Örnekle Öğrenme) yöntemini kullanır. Modele, işlem yapmadan önce farklı formatlardaki (Standart, Karışık, Silik) fişlerden oluşan 3 adet referans örnek gösterilir. Bu sayede model, yeni gelen fişin bağlamını çok daha hızlı kavrar.

Backend tarafındaki veri işleme akışı Algoritma 2'de detaylandırılmıştır.

Algorithm 2 Fiş İşleme Hattı (Inference Pipeline)

Require: Fiş Görüntüsü ($image\_file$)

Ensure: Yapılandırılmış JSON Verisi ($validated\_data$)

$raw\_image \leftarrow load\_image(image\_file)$ ▷ Görüntü belleğe yüklenir

$ocr\_text \leftarrow TesseractOCR.extract\_text(raw\_image, lang =' eng')$

▷ Few-Shot Learning: Dinamik Prompt Oluşturma

$prompt \leftarrow CONSTRUCT\_PROMPT(system\_instr, examples, ocr\_text)$

▷ LLM Çıkarımı (Inference)

$raw\_response \leftarrow LlamaModel.generate(prompt, temp = 0.2, top\_p = 0.9)$

$json\_data \leftarrow extract\_json\_block(raw\_response)$

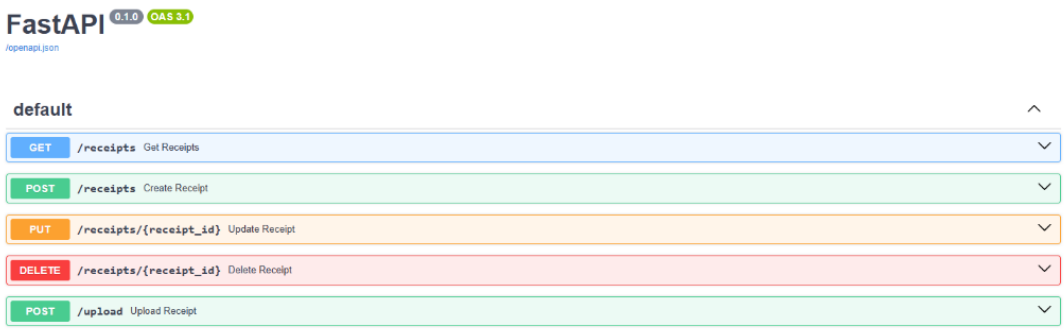
$validated\_data \leftarrow validate\_fields(json\_data)$ ▷ Tarih/Tutar Doğrulama

$validated\_data \leftarrow return\_fallback\_structure()$ ▷ Hata durumunda boş yapı dön

return $validated\_data$

Veritabanı tercihi olarak, projenin taşınabilirliğini korumak ve sunucu bağımlılığını ortadan kaldırmak adına "Serverless" (Sunucusuz) bir yapı sunan **SQLite** kullanılmıştır. Sistemde tüm işlem kayıtları ve fiş metadata'ları SQLite üzerinde saklanırken, işlenen fiş görselleri performans kaybını önlemek amacıyla yerel dosya sisteminde depolanmaktadır.

Backend API yapısı ve veri modelleri, geliştirme sürecini hızlandırmak adına Swagger UI üzerinden belgelenmiştir (Şekil 7).



Şekil 7: Backend API Dokümantasyonu (Swagger UI). Mevcut endpoint'ler ve veri modelleri.

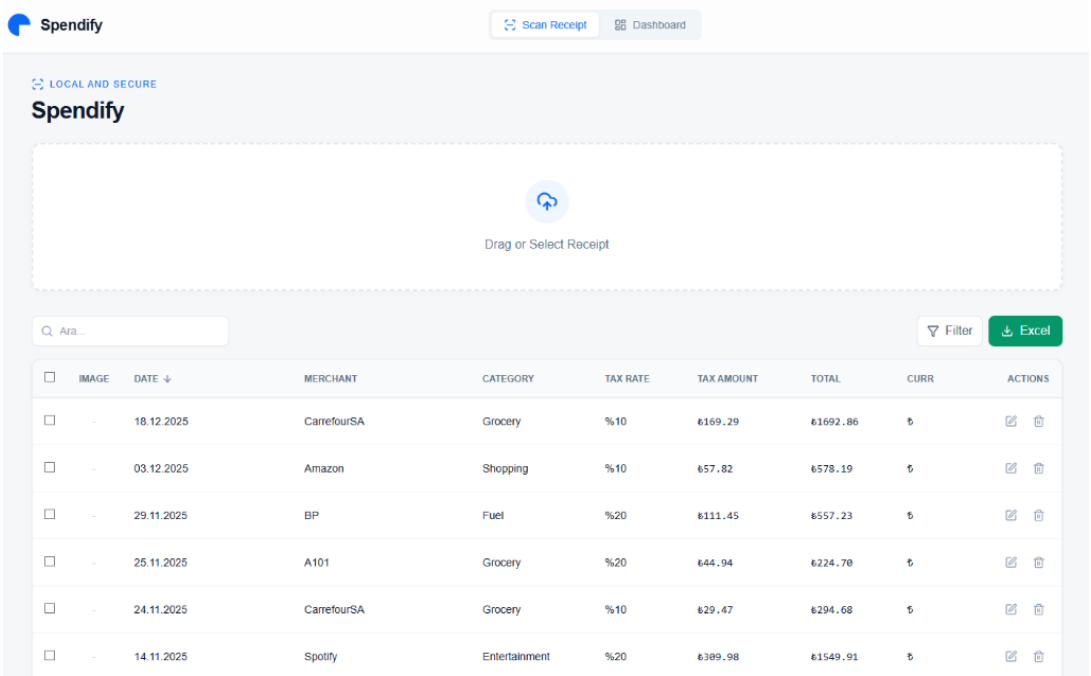
6.2 Frontend Arayüzü (React + Tailwind)

Kullanıcı deneyimi (UX) odaklı, modern ve her cihazda çalışabilen (responsive) bir arayüz geliştirilmiştir.

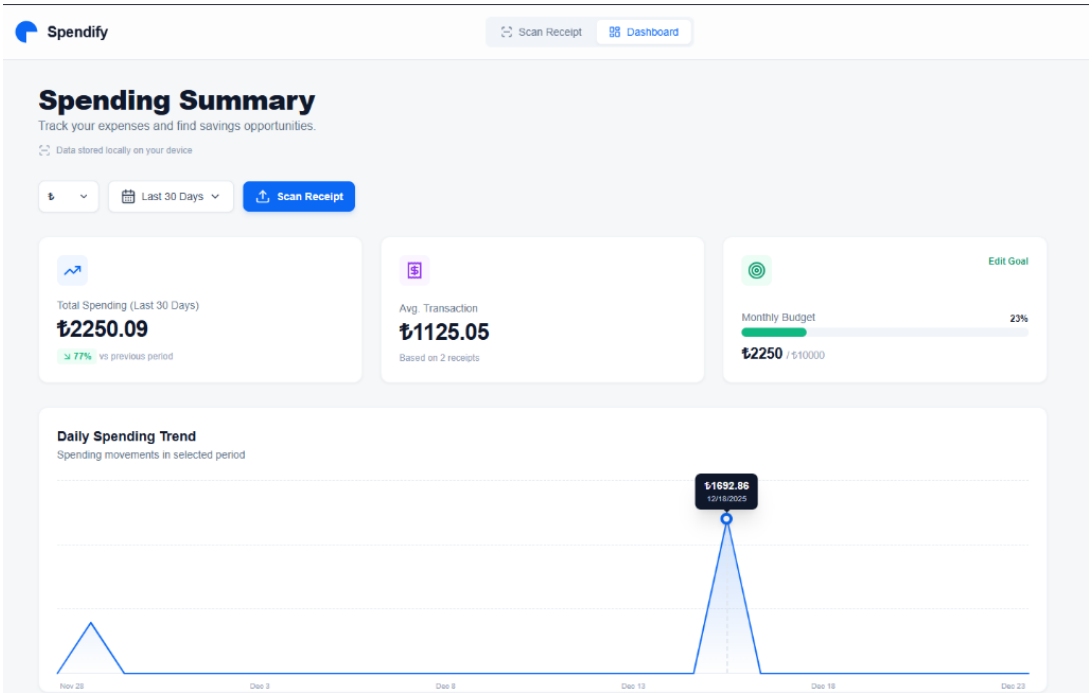
Kullanılan Teknolojiler: React.js, TailwindCSS, Axios.
Geliştirilen arayüzün temel özellikleri şunlardır:

- **Drag & Drop Yükleme Alanı:** Kullanıcılar fiş görsellerini sürükleyip bırakarak analizi anında başlatabilirler.
- **Canlı Düzenleme (Live Edit):** Modelin çıkardığı veriler (Satıcı, Tutar, Tarih vb.) kullanıcıya sunulur. Kullanıcı, OCR veya model hatası olması durumunda bu alanları manuel olarak düzeltebilir.
- **Dashboard (Gösterge Paneli):** Toplam harcama miktarı, hesaplanan vergi oranları ve geçmiş hareketler grafiksel olarak görselleştirilir.
- **Veri Dışa Aktarımı (Export):** İşlenen ve doğrulanan tüm fiş verileri, muhasebe sistemlerine entegrasyon için **Excel** veya **CSV** formatında indirilebilir.

Arayüz tasarımları ve fonksiyonel ekranlar Şekil 8 ve Şekil 9'de sunulmuştur.



Şekil 8: Fiş Tarama ve Analiz Ekranı (Scanner Tab). Sol panelde yüklenen fiş, sağ panelde Yapay Zeka çıktıları görüntülenir.



Şekil 9: Gösterge Paneli (Dashboard Tab). Harcama özetleri ve işlem listesi.

7 SONUÇ VE GELECEK ÇALIŞMALAR

Bu proje ile geliştirilen LLM destekli sistem, fiş analizinde yüksek başarı oranlarına ulaşmıştır. Ancak çalışma, daha güçlü donanım imkanları ve genişletilmiş veri setleri ile şu şekilde geliştirilmeye açıktır:

- **Daha Güçlü GPU Kullanımı:** Mevcut eğitim süreci Google Colab (T4/A100) üzerinde optimize edilmiştir. Eğer H100 gibi daha güçlü GPU kümeleri sunulursa, modelin sadece adaptör katmanları (LoRA) değil, **tüm parametreleri (Full Fine-Tuning)** eğitilebilir. Bu sayede model, el yazısı notları anlama gibi daha nüanslı yetenekler kazanabilir ve eğitim süresi 15 saatten 3 saate düşürülebilir.

- **Veri Seti Çeşitliliği:** Mevcut sürümde ağırlıklı olarak market fişleri (CORD veri seti) kullanılmıştır. Gelecekte fatura, e-arşiv fatura ve banka dekontları sisteme dahil edilerek model bir "Genel Finansal Asistan"a dönüştürülebilir.

- **RAG Entegrasyonu:** Çok sayfalı faturalarda modelin 2048 token'lık girdi limiti yetersiz kalabilir. Bu durumda "Vector Embedding" ve "Vector Database" (ChromaDB vb.) teknolojileri kullanılarak, dokümanın sadece ilgili kısımlarının modele verilmesi (Retrieval-Augmented Generation) sağlanabilir.

Kaynaklar

[1] Hu, E. J., et al. (2021). *LoRA: Low-Rank Adaptation of Large Language Models.* arXiv preprint arXiv:2106.09685.

[2] Park, S., et al. (2019). *CORD: A Consolidated Receipt Dataset for Post-OCR Parsing.* In Document Analysis and Recognition (ICDAR).

[3] Touvron, H., et al. (2023). *Llama 2: Open Foundation and Fine-Tuned Chat Models.* arXiv preprint arXiv:2307.09288.

[4] Unsloth AI. (2023). *Unsloth: Faster and Memory-Efficient Training for LLMs.* GitHub Repository.

[5] Smith, R. (2007). *An Overview of the Tesseract OCR Engine.* Ninth International Conference on Document Analysis and Recognition (ICDAR 2007).