Development of Turkish Q&A Systems with RAG-Supported LLMs

by

Harun Tuna

Graduation Project Report

Yeditepe University

Faculty of Computer and Information Sciences

Department of Management Information Systems

2025

Development of Turkish Q&A Systems with RAG-Supported LLMs

APPROVED BY:

Assist. Prof. Dr. Ali Cihan Keleş ….....……
(Supervisor)

Assoc. Prof. Dr. Mustafa Asım Kazancıgil ...……

Prof. Dr. Aşkın Demirağ .....…….....……

DATE OF APPROVAL: 03 / 11 /2024

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my father, Mehmet Kemal TUNA, and my mother, Saime TUNA, whose unwavering support and financial assistance made this project possible. Their encouragement and generosity sustained me both socially and economically, and I am deeply thankful for their belief in my work and me.

I also extend my heartfelt thanks to my friends and other family members for their patience and understanding, especially as I was often unavailable and neglectful during the more demanding stages of this research.

I am equally thankful to my academic advisor, Assist. Prof. Dr. Ali Cihan KELEŞ, for his invaluable support and academic guidance. His insights, expertise, and guidance have been instrumental for the direction and quality of this project. Also, ı would like to express my gratitude to Assist. Prof. Dr. Berc Deruni for his invaluable insights and his encouragement.

Finally, I would like to acknowledge the broader scientific community, whose selfless commitment to open-source collaboration provided the foundational knowledge, tools, and platforms that enabled this research.

ABSTRACT

Development of Turkish Q&A Systems with RAG-Supported LLMs

The advancements in large language models (LLMs) have significantly transformed natural language processing (NLP), enabling era changing applications across multiple domains. However, the performance of these models for Turkish-specific tasks remains suboptimal due to the underrepresentation of Turkish in multilingual datasets and the complex agglutinative structure of the language. This limitation, combined with the excessive size of multilingual models and the reliance on cloud-based platforms, introduces significant challenges, including high operational costs, vendor lock-in dependency, and privacy concerns for sensitive data.

This research, as part of the “Presidency of Defense Industries - SAYZEK - ATP 2024-2025" program, represents a foundational step in addressing these issues. The study focuses on the evaluation and development of Retrieval-Augmented Generation (RAG) systems for Turkish language. Specifically, it explores the implementation of RAG systems using commercial cloud platforms and open-source models that are optimized to operate on local hardware. Furthermore, it establishes a comprehensive pipeline for data preparation, including digitization, cleaning, and structuring datasets for RAG applications.

By advancing the state of RAG-enabled Turkish systems, this research aims to provide a scalable, cost-effective, and private framework for local deployments, addressing critical gaps in performance and data sovereignty while laying the groundwork for future advancements in Turkish-specific LLMs and agents.

ÖZET

RAG Destekli LLM'lerle Türkçe Soru-Cevap Sistemlerinin Geliştirilmesi

Büyük dil modellerindeki (LLM) ilerlemeler, doğal dil işleme (NLP) alanını önemli ölçüde dönüştürerek birçok alanda çığır açan uygulamaların geliştirilmesine imkan tanımıştır. Bununla birlikte, çok dilli veri kümelerinde Türkçenin yetersiz temsili ve dilin eklemeli yapısının yarattığı sıkıntılar nedeni ile, bu modellerin Türkçe odaklı görevlerdeki performansı henüz arzu edilen seviyeye ulaşmamıştır. Çok dilli modellerin gereksiz büyük yapıları, bulut tabanlı platformlara olan bağımlılıkla birleştiğinde; yüksek maliyetler ve hassas veya kritik veriler ile ilgili gizlilik endişeleri gibi zorlukları beraberinde getirmektedir.

"Savunma Sanayii Başkanlığı - SAYZEK - ATP 2024-2025" programı kapsamında yürütülen bu araştırma, söz konusu sorunların giderilmesi yolunda önemli bir adım niteliği taşımaktadır. Çalışma, Türkçe dili için Retrieval-Augmented Generation (RAG) sistemlerinin değerlendirilmesine ve geliştirilmesine odaklanmaktadır. Özellikle, ticari bulut platformlarını kullanan RAG sistemleri ile yerel donanım üzerinde çalışacak şekilde optimize edilmiş açık kaynak modellerin geliştrilmesini hedef almaktadır. Buna ek olarak, dijitaleştirme, veri temizleme ve bu şekilde RAG uygulamalarına uygun kaliteli veri kümelerini oluşturmayı amaçlamaktadır.

RAG tabanlı Türkçe sistemler geliştirmeyi amaçlayan bu çalışma, performans ve veri gizliliği konularında var olan kritik boşlukları gidermeye yardımcı olacak; elastik, verimli ve güvenli bir çözüm sunmayı hedeflemektedir. Aynı zamanda, Türkçeye özgü büyük dil modelleri ve yapay zeka sistemleri için de bir temel hazırlayacaktır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

OCR Optical Character Recognition

NLP Natural Language Processing

LLM Large Language Model

RAG Retrieval Augmented Generation

GPT Generative Pre-training Transformer

AI Artificial Intelligence

AWS Amazon Web Services

BIST Borsa Istanbul (Istanbul Stock Exchange)

KAP Kamuoyunu Aydınlatma Platformu (Public Disclosure Platform)

PDF Portable Document Format

ANZAK Avustralya ve Yeni Zelanda Kolordusu
(Australian and New Zealand Army Corps)

GAP Güneydoğu Anadolu Projesi (Southeastern Anatolia Project)

FAISS Facebook Artificial Intelligence Similarity Search

MS Microsoft

GPU Graphics Processing Unit

1. INTRODUCTION

1.1. Background and Context

The advent of large language models has caused a paradigm shift in natural language processing, driving breakthroughs in tasks such as machine translation, question-answering, and conversational systems. Multilingual LLMs, including those in the GPT series, offer capabilities across various languages. However, their generalized nature presents challenges for underrepresented languages like Turkish, which feature distinct linguistic characteristics, including an agglutinative morphology.

Turkish occupies a relatively small proportion of multilingual training datasets, resulting in suboptimal performance for tasks requiring linguistic nuance. Additionally, cloud-based solutions, such as OpenAI's GPT models, Anthropic's Claude models, and Google Vertex AI's Gemini models, demand significant financial resources and pose privacy risks for sensitive or critical data. These factors drive the need of developing localized solutions that aid the unique problems of Turkish NLP work.

Retrieval-Augmented Generation has emerged as a promising approach to overcome these limitations. By enabling dynamic integration of external knowledge bases during use, RAG systems eliminate the need for frequent retraining, offering a cost-effective and adaptive solution. Despite its potential, RAG methods and solutions for Turkish language remain underexplored, particularly for local deployment scenarios nearly nonexistent.

1.2. Problem Statement

This research addresses the following critical issues in the application of LLMs for Turkish NLP tasks:

- **Performance Deficiency**: Multilingual models fail to successfully address Turkish-specific tasks due to limited data representation and linguistic complexity.
- **Cost and Resource**: Traditional LLMs require periodic retraining, imposing significant financial and operational burden.
- **Privacy and Security Risks**: The reliance on cloud-based services introduces vulnerabilities in managing sensitive data, particularly for high-security environments such as defense.

1.3. Objectives

This project aims to establish the foundation for Turkish-specific question-answering systems by focusing on the following objectives:

- **Evaluation of RAG Systems**: Assess the capabilities of commercial platforms, such as AWS, Azure, and Google Cloud, and compare them with open-source alternatives.
- **On-Premises RAG Implementation**: Develop open-source RAG systems optimized for resource-constrained hardware (e.g., GTX 1650 GPU) through quantization.
- **Data Preparation and Mining**: Create a pipeline for digitizing, pre-processing, and structuring datasets.

1.4. Scope

The scope of this project is confined to evaluating and implementing RAG systems for Turkish, with a focus on cost-efficiency, privacy preservation, and local deployments. It does not extend to the fine-tuning or development of a Turkish-specific LLM, which is reserved for future phases. Later phases are planned to be implemented to develop a fully local possible RAG solution via training or fine-tuning a model.

1.5. Significance

This research contributes to addressing the critical gaps in Turkish RAG solutions by prioritizing **localized**, private, and efficient solutions. The methodologies developed in this project hold significant implications for advancing Turkish language technologies, providing a guide for deploying RAG systems in other underrepresented languages as well. Additionally, the study paves the way for future research into fine-tuned and fully localized Turkish large language models, which is the main goal of the broader project.

2. BACKGROUND

2.1. Previous works

The foundational advancements in NLP began with the introduction of the Transformer architecture by Vaswani et al. in *Attention Is All You Need* [1]. This architecture enabled the efficient modeling of sequential data, paving the way for LLMs such as GPT-1 [2], GPT-2 [3], and GPT-3 [4], which successively introduced groundbreaking capabilities in zero-shot and few-shot learning. Further progress was made with the T5 model [5], which standardized input-output processing for text-based tasks, providing adaptability of NLP models. Retrieval-Augmented Generation (RAG) [6], as proposed by Lewis et al., integrates external knowledge sources during inference, enabling real-time access to relevant information without retraining. This approach is particularly advantageous for addressing data sparsity and augmenting domain-specific applications.

2.2. Current Solutions and Challenges

Despite the capabilities of multilingual LLMs; Turkish performance remain underserved due to:

- **Underrepresentation in Training Data**: Multilingual datasets does not allocate sufficient resources to Turkish, reducing model effectiveness.
- **Excessive Model Complexity**: Incorporating unneeded other multilingual data inflates model size, increasing computational overhead.
- **Privacy Concerns**: Cloud-based platforms pose security risks for managing sensitive data.

While cloud platforms [7] providing state-of-the-art performance, they often come with high costs and the vendor lock-in dependency problems. Contrary to this, open-source frameworks like AnythingLLM [8], Ollama [9], and LM Studio [10] enable on-premises deployment, addressing many of these challenges.

2.3. Importance of RAG for Turkish NLP

RAG systems allow models to access external data dynamically, offering a scalable solutions where linguistic complexity and data scarcity are significant barriers. These systems reduce reliance on retraining while boosting adaptability, making them ideal for localized applications.

It is clear that this technology will revolutionize future of information systems. As we have seen and experienced in the past, missing out and falling behind in modernization has and always will have severe outcomes. This is the main point of this study as this technology is dependent on language specific development and know-how Turkish NLP stack should be improved to stay relevant in the field.

2.4. Project's Contribution

This research aims to:

- Establish a framework for implementing Turkish-specific RAG systems using both commercial and open-source solutions.
- Develop robust data pipelines to ensure high-quality, domain-specific datasets.
- Demonstrate the feasibility of deploying RAG systems on resource-constrained hardware.
- By addressing the limitations of current solutions, this project contributes to advancing the state of Turkish NLP and serves as a model for other underrepresented languages.

3. ANALYSIS

3.1. Problem Analysis

The development of Turkish-specific NLP systems faces significant challenges rooted in both linguistic and technological constraints as stated. Turkish, as an agglutinative language, presents complex morphological and syntactical structures that are not captured by current multilingual large language models especially opensource ones. These models are often trained on datasets where Turkish is underrepresented, scarce; hence resulting in diminished performance for Turkish-specific tasks.

Furthermore, current solutions often rely on cloud-based proprietary platforms such as OpenAI's GPT models, Anthropic Claude, and Google Vertex AI. While these platforms provide state-of-the-art performance, they introduce several critical limitations:

- **Privacy and Security Concerns**: Cloud-based systems require uploading data to external servers, which may pose risks via leaks; to sensitive information, particularly in domains such as finance and defense.
- **High Costs**: Proprietary models impose significant expenses for training and deployment, limiting accessibility for academic and smaller-scale applications.
- **Resource Inefficiency**: The existence of multilingual data apart from Turkish inflates the size and computational requirements of these models, making them unsuitable for resource-constrained environments.
- **Limited Adaptability**: Traditional LLMs require retraining to incorporate new information, data; a process that is costly and time consuming.

Retrieval-Augmented Generation provides an effective solution to these challenges by enabling models to retrieve external knowledge bases dynamically during inference. This eliminates the need for frequent retraining, enables elasticity, and reduces costs and downtime.

RAG solutions and agents for Turkish remains an underexplored domain, particularly in scenarios where on-premises deployment is necessary or desired to address privacy and cost related concerns.

Another issue is the lack of high-quality, domain-specific Turkish datasets. Many valuable resources, such as historical texts and financial documents, exist in non-digital formats, requiring significant preprocessing like scanning and digitization efforts. Without comprehensive datasets, the evaluation and training of Turkish models remain limited.

3.2. Proposed Approach

This project proposes a planned approach to target the identified challenges, focusing on following paths:

1- Development of RAG Systems:

- **Cloud-Based Implementation**: Evaluate the RAG capabilities of commercial platforms such as AWS, Azure, and Google Cloud for Turkish NLP tasks.

- **On-Premises Implementation**: Leverage open-source frameworks, including AnythingLLM, Ollama, and LM Studio, to develop RAG systems optimized for deployment on resource-constrained hardware, a GTX 1650 GPU in this case.

2- Data Preparation and Mining:

- Creation of high-quality Turkish datasets by digitizing and preprocessing.
- Utilizing OCR tools and scan methods for converting hardcopy documents into digital text and preprocess the data to ensure compatibility with RAG systems.
- Incorporate both newly created datasets and publicly available corpuses for model evaluation and training.

3.3. Evaluation Metrics

The evaluation framework will utilize two distinct datasets designed to assess different aspects of the proposed RAG systems:

- **Turkish NLP Capabilities**: A preliminary dataset will be developed to evaluate the general language processing capabilities of models. This dataset will test their ability to handle Turkish syntax, semantics, and contextual understanding.
- **RAG Performance**: A secondary dataset will focus on measuring the retrieval and generation capabilities of the systems.

3.4. Data Sources

To ensure quality of evaluation, datasets will be derived from diverse sources listed:

- **Hardcopy Books**:
  Kendi Kendine Lise
  Masa Ansiklopedisi
- **Financial Documents**:
  Public brochures from BIST [11] and KAP [12]
- **PDF Books**:
  Chapters 1-2-3-4 of Nutuk [13]
  Chapters 1-2-3 of Türkiye Cumhuriyeti Tarihi [14]

These sources encompass a variety of linguistic styles and domains, including historical narratives and technical financial texts. OCR & Scanning tools will be used to digitize hardcopy materials, followed by preprocessing to create structured datasets.

3.5. Design Requirements

- **Hardware**:

On-premises RAG systems will be deployed on a GTX 1650 GPU, utilizing quantization techniques to optimize performance within hardware limitations.

On training phase of the project, which are not included in the report currently acquirement of an RTX 4090 is secured, which is a must for efficient fine-tuning and training operations of models.

- **Software and Tools:**

OCR frameworks for digitization, data preprocessing pipelines, and open-source RAG-compatible models will form the backbone of the development workflow.

Cloud platforms will be used for benchmarking proprietary RAG systems against on-premises implementations.

Also, another requirement is developing tools that fit needs of the project.

- **Human Resources and Expertise:**

The project will rely on a combination of self-directed learning and existing knowledge to address challenges during implementation.

Documentation of programs and platforms as well as learning material provided on these portals will be leveraged where need may rise.

4. DESIGN AND IMPLEMENTATION

The design is structured into three interdependent processes: **Data Preparation**, **Pre-Elementary Testing of Local Models**, and **Turkish RAG Testing**; encompassing both cloud-based implementations and a proof-of-concept local RAG model.

4.1. Data Preparation Process Design & Implementation

The data preparation process is a cornerstone of the Retrieval-Augmented Generation systems development, has to be designed to ensure production of high-quality datasets tailored for Turkish tasks. This process integrates basic and rapid workflows that encompass collection, digitization, preprocessing, and segmentation. The design prioritizes compatibility with RAG pipelines while addressing various challenges.

4.1.1. Workflow Design

1) **Digitization**:

- **Books:** Physical resources were digitized using high-resolution personal scanner, producing consistent and high-quality images
- **PDFs:** Digital resources, such as brochures, were directly integrated into the pipeline without additional scanning.

2) **Preprocessing**

- **Column Splitting:** Custom scripts like ht_book_processor were used to process multi-column layouts into single-column text.
- **Artifact Removal:** Self developed tools such as ht_mask_remover were used to eliminate visual artifacts introduced.
- **Optical Character Recognition (OCR)**: OCR technologies (Azure AI Document Intelligence, Google Vision API, Self-written ht_mega_ocr) Were applied to convert scanned images into structured text.
- **Text Segmentation (Chunking):** Extracted text was divided into ~3,500-character chunks to align with tokenization limits of RAG-compatible models.
- **Grammar Refinement:** Tools like Grok-2-1212 (via xAI API) and GPT-4o-mini (via OpenAI API) refined OCR outputs, correcting grammatical errors and ensuring readability.

4.1.2 Tailored Pipelines for Resources

Each resource underwent a customized preparation pipeline, as:

- **Kendi Kendine Lise**:

  Employed column splitting, artifact removal, OCR, chunking, and grammar refinement for its dense, educational content.

- **Masa Ansiklopedisi**:

  Addressed two-page layouts and column structures through preprocessing, followed by text refinement via GPT-4o-mini.

- **BIST and KAP Brochures**:

  Leveraged Azure AI Document Intelligence model for structured JSON outputs, parsed into plain text and segmented.

- **Nutuk**:

  Chapters 1 & 4: Processed with Google Vision API.
  Chapter 2: Processed using Amazon Textract.
  Chapter 3: Processed via Azure Cognitive Services for image-based OCR.

- **Türkiye Cumhuriyeti Tarihi**:

  Chapters 1: Processed using Amazon Textract.
  Chapter 2: self-written *PDF to Text* script.
  Chapter 3: Processed via *MarkitDown*.

4.2. Pre-Elementary Model Evaluation for Turkish Language Capabilities Test Design & Implementation

The pre-elementary evaluation aimed to systematically assess the ability of various language models to generate grammatically correct, well-structured, and fully Turkish responses. This evaluation was a critical step toward identifying models suitable for fine-tuning and eventual integration into Retrieval-Augmented Generation systems.

The focus of this phase was not on the factual accuracy of responses but on the models' proficiency in Turkish syntax, semantics, grammar, and coherence.

4.2.1 Details of Pre-Elementary Model Evaluation Test

I. All models were initialized with the following **instruction**:
"Sorulara Türkçe cevap veren bir asistansın."
(*You are an assistant that answers questions in Turkish*.).

II. Models were run **locally** with quantization if available or possible. This is the part where hardware limitations mentioned previously affected the model choices.

III. **Temperature Settings:** A temperature value of **0.8** was selected for testing. This value balanced creativity and coherence, allowing models to explore a range of responses while maintaining syntactical correctness. (*0.8 value is over 2, setting was normalized to 0.4 for models that use 0 to 1 scale*)

IV. **Evaluation Criteria:** Models were evaluated on a **-1 to 2** scale based on their Turkish language capabilities:

- **-1**: **Eliminated:** Responses demonstrated poor Turkish capability or irrelevant outputs.
- **0**: **Neutral**: Responses were in Turkish but lacked the quality needed for further consideration.
- **1**: **Approved with potential:** Responses displayed promise but required further fine-tuning for Turkish RAG integration.
- **2**: **Approved with high potential**: Responses exhibited excellent grammar, fluency, and structure, qualifying the model for direct integration.

4.2.2. The 10 Question Prompt Set:

1. **Atatürk kimdir ?** (*Who is Atatürk?*)
2. **Türkiye'nin başkenti neresi ve nasıl bir yer?**
   (*What is the capital of Turkey and what kind of place is it?*)
3. **2019 nasıl bir yıldı?** (*How was year 2019?*)
4. **Bilgi ver televizyon kanalları.**
   (*Give information about television channels*.)
5. **Günde kaç litre su içmeliyim?**
   (*How many liters of water should I drink per day?*)
6. **Barış Manço.**
7. **Yüz kere yüz.** (*Hundred times hundred*.)
8. **Finallerim yaklaşıyor, nasıl çalışmalıyım?**
   (*My finals are approaching, how should I study?*)
9. **Eşanlamlı Kelimeler ve Örnekler.** (*Synonyms and Examples*.)
10. **EVİMİ HAFTADA KAÇ KERE TEMİZLEMELİYİM, DETAY VER**
    (*HOW MANY TIMES A WEEK SHOULD I CLEAN MY HOUSE? GIVE DETAILS*.)

4.2.3 Models List

A) Open-Source Local Models Tested:

1. llama-3.2-1b-instruct
2. llama-3.2-3b-instruct
3. lama-3.1-tulu-3-8b
4. llama-3.1-8b-instruct
5. ministral-8b-2410
6. qwen2-vl-7b-instruct
7. gemma-2-9b-it
8. gemma-2-2b-it
9. ibm-granite-3.1-8b-instruct-4Q
10. Qwen2.5-7B-Instruct-4Q
11. Yi-1.5-9B-Chat-4Q
12. Falcon3-7B-4Q
13. Phi3.5-8Q
14. Smollm2
15. Wizardlm2
16. nemotron-mini-4B
17. granite3-dense:2b

B) Controlled Sector Leaders for comparison:

1. ChatGPT-o1-pro
2. chatgpt-4o-latest-20241120
3. claude-3-5-sonnet-20241022
4. grok-2-2024-08-13
5. gemini-2.0-flash-exp

4.3. Turkish RAG Testing of Cloud Models and Proof-of-Concept Local RAG Implementation & Design

The Turkish RAG testing phase was designed to evaluate the integration of knowledge bases and pipelines with both cloud-based and local RAG implementations. The primary aim was to assess the ability of various systems to process the prepared Turkish RAG datasets (outlined in **Section 4.1.1**) and generate contextually relevant, accurate, and fluent responses.

This phase utilized a consistent question set of 50 prompts to uniformly test all models, ensuring a comprehensive comparison across implementations.

4.3.1 Platforms, Models, Embeddings, and Pipelines

The following models, embeddings, and pipelines were developed and integrated for testing:

1. **Platform: Vertex AI, Model: Gemini 2.0 Flash-Exp**:
   Details: RAG Method: Vertex AI Search for retrieval grounding, Temperature: 0.2, Output Tokens: 8192.

2. **Platform: AWS Bedrock, Model: Claude Sonnet 3 v1**:
   Details: RAG Method: Agent integrated with Amazon Bedrock knowledgebase.

3. **Platform: Vertex AI, Model: Gemini 1.5 Pro-002**:
   Details: RAG Method: Vertex AI Search for retrieval grounding, Temperature: 0.2, Output Tokens: 8192, Top-P: 0.8.

4. **Platform: AWS Bedrock, Model: Amazon Nova Pro 1**
   Details: RAG Method: Retrieval via vector database (Amazon OpenSearch Serverless), Embeddings: Titan2

5. **Platform: AWS Bedrock, Model: Sonnet 3.5 v1**
   Details: RAG Method: Knowledgebase retrieval via vector database (Amazon OpenSearch Serverless), Embeddings: Titan2

6. **Platform: AWS Bedrock, Model: Nova Micro v1**

   Details: RAG Method: Knowledgebase retrieval via vector database (Amazon OpenSearch Serverless), Embeddings: Titan2

7. **Platform: Vertex, AI Model: AI Gemini 1.5 Flash-002**:

   Details: RAG Method: Vertex AI Search for retrieval grounding, Temperature: 0.2, Output Tokens: 8192, Top-P: 0.8.

8. **Platform: Azure OpenAI Model: GPT4o-mini (v2024-07-18)**:

   Details: RAG Method: Retrieval via Azure AI Search Index, Temperature: 0.2, Output Tokens: 8192, Top-P: 0.8, Strictness: 3, Max Docs: 3.

9. **Platform: OpenAI, Model: Custom GPT with Knowledgebase**:

   Details: Stream-Lined Customized GPT model integrated with domain-specific knowledge. (Assumed with 4o Backend)

10. **Platform: Azure OpenAI, Model: GPT4o (v2024-05-13)**:

    Details: RAG Method: Retrieval via Azure AI Search Index, Temperature: 0.2, Output Tokens: 8192, Top-P: 0.8, Strictness: 3, Max Docs: 3.

11. **Platform: Self-Made Pipeline v1, Model: Claude-3-5-Sonnet-20241022**

    Details: RAG Method: Retrieval via FAISS local vector index supplemented by Azure OpenAI text-ada-002 embeddings 0.2, Max Tokens: 4096, API citations feature enabled.

12. **Platform: Self-Made Pipeline v2 (Bruted) Model: Claude-3-5-Sonnet-20241022**

    Details: RAG Method: Retrieval via FAISS local vector index supplemented by Azure OpenAI text-ada-002 embeddings 0.2, Max Tokens: 4096, API citations feature enabled. Top-K: 20, Similarity Threshold: 0.5

13. **Platform: Local RAG Implementation, Model: Gemma2**

    Details: Local Gemma2 via Ollama and AnythingLLM interface in query mode. Loca. LanceDB as vector database, OpenAI ada-002 embeddings, Web Search: Off, Temperature: 0.2.

4.3.2. Settings, Data and The System Prompt

The datasets prepared in **Section 4.1.1** were added to the respective knowledge bases or vector databases for all models. This ensured consistency in the testing framework, with all systems accessing the same Turkish domain-specific information. The datasets comprised digitized and structured content from historical texts, financial documents, and other curated resources.

Following **system prompt** was given to all models:

"*Sen bir türkçe soru-cevap asistanısın. Sana yöneltilen sorulara yalnızca sağlanan belgelerden referans alarak cevap ver.*

Belgelerdeki metinleri değiştirmen gerekmez; ancak, varsa yazım ve dilbilgisi hatalarını düzelt.

Ana fikir ve konudan sapmadığından emin ol ve cevaplarını her zaman dikkatlice değerlendir.

Verdiğin her cevaba mutlaka belgelerden bir referans ekle. Sağlanan kaynaklarda bir bilgi bulunmaz ise: cevap verme, bulamadığını belirt. referansları cevabın sonunda listele."

Translated:

"You are a Turkish question-answer assistant. Answer the questions directed at you only by referencing the provided documents.

You do not need to change the texts in the documents; however, correct any spelling and grammar mistakes if present.

Make sure not to deviate from the main idea and topic, and always carefully evaluate your responses.

Always include a reference from the documents in every answer you provide. If no information is found in the provided sources, do not answer; state that you could not find it. List the references at the end of your response."

4.3.3. Testing Methodology Explained

- **Question Set**:

A set of 50 questions was developed, encompassing a range of topics such as history, general knowledge, and linguistic nuances specific to Turkish.

Evaluation was done a 3-level grading system for each given response; it is shown in color code:

Green: **Valid** = 2 Points

Yellow: **Partial** = 1 Points

Red: **Fail** = 0 Points

This scale puts eval metrics on a scale of 100 available points.

- **Evaluation Metrics**:

Accuracy: The relevance of retrieved information to the posed question.

Correctness: The answer should the expected one and be valid.

References: Model should list referred or cited data snippet for given question for answer to be considered valid. With this method we provide grounding of data hence eliminating hallucinations.

- **Test Environment**:

Cloud-based systems were tested under their default configurations apart from those settings indicated above for grounding and retrieval processes.

Local pipelines were deployed on resource-constrained hardware (e.g., a GTX 1650 GPU) to evaluate the feasibility of on-premises Turkish RAG implementations. Which is considered very low by today's standards.

4.3.4. The 50 Question RAG Set:

1. **Tarih öncesi devirleri kaç kısma ayırıyoruz ve bunlar nelerdir?** (*How many parts do we divide prehistoric ages into, and what are they?*)
2. **Romalıların asimile ettiği topluluklar** (*The communities assimilated by the Romans.*)
3. **Küçük aile işletmeleri hakkında bilgi ver** (*Give information about small family businesses.*)
4. **Bir ırmağın enerjisi hangi faktörlere bağlıdır?** (*What factors determine the energy of a river?*)
5. **Yüzey gerilimi nedir?** (*What is surface tension?*)
6. **Fiziğin kötüye kullanılması** (*The misuse of physics.*)
7. **Endüksiyon nedir?** (*What is induction?*)
8. **Deve kuşu bitki mi?** (*Is an ostrich a plant?*)
9. **İnsan hakları hakkında bilgi ver** (*Give information about human rights.*)
10. **Zararlı katı maddeler nasıl vücuttan atılır?** (*How are harmful solid substances removed from the body?*)
11. **Ahlak ve insan ihtiyaçları arasındaki ilişki nedir?** (*What is the relationship between morality and human needs?*)
12. **Gazoz nedir?** (*What is soda?*)
13. **Dulkadiroğulları gerçek mi?** (*Are the Dulkadirids real?*)
14. **Sandal ağacını anlat** (*Describe the sandalwood tree.*)
15. **A ve B vitaminleri hakkında bilgi verir misin lütfen?** (*Can you give information about vitamins A and B, please?*)
16. **Hatice Turhan Sultan'ın hayatı** (*The life of Hatice Turhan Sultan.*)
17. **Yosunlar hakkında özet ver** (*Give a summary about algae.*)
18. **Borç araçları piyasasında neler var?** (*What exists in the debt instruments market?*)
19. **Yatırım yaparken nelerden kaçınmalıyım?** (*What should I avoid when investing?*)
20. **Finansal tablo ve faaliyet raporunu incelemek kimin sorumluluğundadır?** (*Who is responsible for examining financial statements and activity reports?*)
21. **KAP-BIY programında bağımsız denetim kuruluşu nasıl onay verir?** (*How does an independent audit firm approve the KAP-BIY program?*)
22. **Doğal sermaye nedir? Örneklendir.** (*What is natural capital? Give examples.*)
23. **Doğrudan sürdürülebilirlik yatırımı örneği** (*Example of a direct sustainability investment.*)
24. **Vadeli işlem tanımı** (*Definition of a futures contract.*)
25. **Spot işlem ne demek?** (*What does a spot transaction mean?*)
26. **Arbitraj nedir?** (*What is arbitrage?*)
27. **Temiz ve kirli fiyat nedir?** (*What are clean and dirty prices?*)
28. **Şirket Genel Bilgileri Formu nedir?** (*What is the Company General Information Form?*)

29. **1919 yılında Hristiyan azınlıkların Rum Patrikliğinde kurduğu kurulun adı nedir? Neden kurulmuştur?** (*What was the name of the committee established by Christian minorities in the Greek Patriarchate in 1919, and why was it established?*)
30. **"Ya İstiklal Ya Ölüm" açıkla** (*Explain "Independence or Death."*)
31. **Sivas Kongresi'nde bulunmak üzere görevlendirilen isimler kimlerdir?** (*Who were the names assigned to participate in the Sivas Congress?*)
32. **Gümrü Anlaşmasının önemi nedir?** (*What is the significance of the Treaty of Gümrü?*)
33. **Sakarya Meydan Savaşını anlat** (*Describe the Battle of Sakarya.*)
34. **Irak Savaşı ve Nurettin Paşa** (*The Iraq War and Nurettin Pasha.*)
35. **Erzurum Kongresi'nin bütün memleketin ve milletin birleşme ve uyuşma noktasında aldığı temel ilke nedir?** (*What fundamental principle did the Erzurum Congress adopt regarding the unity and agreement of the whole country and nation?*)
36. **Hacı Kaya kimdir?** (*Who is Hacı Kaya?*)
37. **Anadolu'da yayılan ulusal hareketin başarısını engellemek için Ferit Paşa'nın girişimleri nasıl sonuçlanmıştır?** (*What were Ferit Pasha's efforts to prevent the success of the national movement in Anatolia, and what were their results?*)
38. **Latran Antlaşması nedir?** (*What is the Lateran Treaty?*)
39. **Yeniçeri Ocağını kim ve ne zaman ilga etti?** (*Who abolished the Janissary Corps, and when?*)
40. **Tuna cephesini özetle** (*Summarize the Danube Front.*)
41. **Humbaracı Ahmet Paşa kimdir?** (*Who is Humbaracı Ahmet Pasha?*)
42. **Balkan Savaşlarında can ve toprak kayıpları** (*Casualties and territorial losses in the Balkan Wars.*)
43. **ANZAK kelimesinin açılımı nedir?** (*What does the term ANZAC stand for?*)
44. **Derleme Ocakları nedir? Açıkla.** (*What are the Compilation Centers? Explain.*)
45. **Dr. Mahmut Esat Bozkurt'a göre inkılap tanımı** (*Dr. Mahmut Esat Bozkurt's definition of revolution.*)
46. **2. Dünya Savaşı ne zaman hangi olay üzerine başlamıştır?** (*When did World War II start, and what event triggered it?*)
47. **Eurotam nedir?** (*What is Eurotam?*)
48. **1960-1980 yılları arasında Türkiye'nin eğitim alanında gerçekleştirdiği en önemli reform nedir?** (*What was Turkey's most significant education reform between 1960-1980?*)
49. **Basının vereceği promosyona sınırlama ne zaman hangi kanunla gelmiştir?** (*When and under which law were limitations imposed on promotional giveaways by the press?*)
50. **GAP projesini anlat** (*Explain the GAP Project.*)

5. TEST AND RESULTS

5.1. Data Preparation Results

The implementation of the data preparation process yielded structured datasets that meet the requirements for developing and evaluating Turkish RAG systems. The results reflect the successful application of the design framework across diverse resources, with detailed outcomes summarized below.

5.1.1. Resource-Specific Process & Results

5.1.1.1. Kendi Kendine Lise:

1.1. *Scanning*: Hardcopy book scanned via scanner, yields high quality images.
1.2. *Column Splitting:* Double column structure split into singles to ensure OCR quality. Mask is added to fill the deleted portion & maintain aspect ratio via self-developed script: *ht_book_processor.py.*
1.3. *Mask Removal:* Masks introduced in 1.2 removed to compress image size via self-developed script: *ht_mask_remover.py.*
1.4. *OCR & Chunking:* Optical Image Recognition process turns images in to a single text format file, later chunked into usable size for ease via self-developed script: *ht_mega_ocr.py.*
1.5. *LMM Grammar Refinement:* Chunks fed into xAI API to “grok-2-1212" model for artifact removal and to be refined and fixed grammar wise via self-developed script: *ht_grok_improv.py.*

Outcome: High-quality text segments (~3,500 characters each) were generated from scanned images, with OCR achieving high accuracy post-refinement.

Figure 5.1.1.1. *Kendi Kendine Lise Process Diagram*

5.1.1.2. Masa Ansiklopedisi:

2.1. *Scanning*: Hardcopy book scanned via scanner, yields high quality images.
2.2. *Column Splitting:* Double column structure split into singles to ensure OCR quality. Mask is added to fill the deleted portion & maintain aspect ratio via self-developed script: *ht_book_processor.py.*
2.3. *Mask Removal:* Masks introduced in 1.2 removed to compress image size via self-developed script: *ht_mask_remover.py.*
2.4. *OCR & Chunking:* Optical Image Recognition process turns images in to a single text format file, later chunked into usable size for ease via self-developed script: *ht_mega_ocr.py.*
2.5. *LMM Grammar Refinement:* Chunks fed into OpenAI API to "gpt-4o-mini" model for artifact removal and to be refined and fixed grammar wise via self-developed script: *ht_openai_turkish_fixer.py.*

Outcome: Successfully processed dense, two-column content into ~3,500-character chunks with minimal OCR errors after refinement.

Figure 5.1.1.2. *Masa Ansiklopedisi Lise Process Diagram*

5.1.1.3. BIST and KAP Brochures:

3.1. *PDF Upload*: PDF files uploaded to Azure Blob Storage to be used.
3.2. *Azure DI OCR:* OCR operation is done using Azure AI Document Intelligence model via self-developed script: *ht_azure_ai_ocr.py.*
3.3. *JSON Parsing:* Resulting JSON format response parsed in to text format via self-developed script: *ht_doc_json_parser.py.*
3.4. *Chunking:* Single text format file chunked into usable size for ease via self-developed script: *ht_chunker_2025.py.*

Outcome: Structured text extracted from PDFs via Azure AI Document Intelligence achieved high accuracy. Segmented text was aligned with RAG tokenization limits.

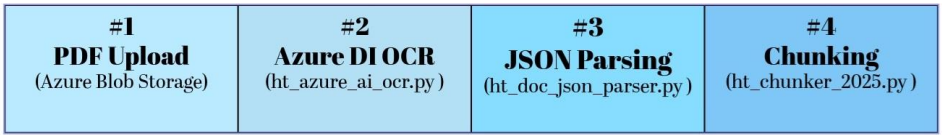

Figure 5.1.1.3. *BIST&KAP Brochures Process Diagram*

5.1.1.4. Nutuk Chapters 1&4:

4.1. *PDF Upload*: PDF files uploaded to Google Cloud Storage to be used.
4.2. *Google Vision OCR:* OCR operation is done using Google Cloud Vision model via self-developed script: *ht_google_vision.py*.
4.3. *JSON Parsing:* Resulting JSON format response parsed in to text format via self-developed script: *ht_doc_json_parser.py*.
4.4. *Chunking:* Single text format file chunked into usable size for ease via self-developed script: *ht_chunker_2025.py*.

Outcome: Processed efficiently using Google Vision API, with segmented text chunks prepared for RAG workflows.

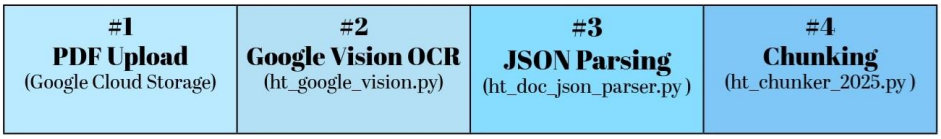

Figure 5.1.1.4. *Nutuk Chapter 1&4 Process Diagram*

5.1.1.5. Nutuk Chapter 2 & Türkiye Cumhuriyeti Tarihi Chapter 1:

5.1. *PDF Upload*: PDF files uploaded to Amazon S3 Cloud Object Storage to be used.
5.2. *Amazon Textract OCR:* OCR operation is done using Amazon Textract model via self-developed script: *ht_aws_textract.py*.
5.3. *Chunking:* Single text format file chunked into usable size for ease via self-developed script: *ht_chunker_2025.py*.

Outcome: Amazon Textract achieved high OCR accuracy for complex layouts.

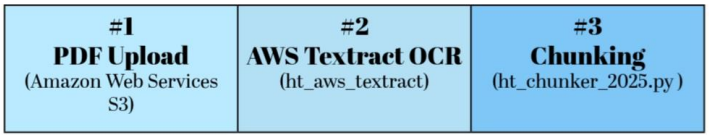

Figure 5.1.1.5. *Nutuk Chapter 2 & Türkiye Cumhuriyeti Tarihi Chapter 1 Process Diagram*

5.1.1.6. Nutuk Chapter 3:

6.1. *PDF to Image Conversion:* PDF file converted in to image format to be used via self-developed script: *ht_pdf_2_images.py.*
6.2. *PDF Upload*: PDF files uploaded to Azure Blob Storage to be used.
6.3. *Azure Image Analysis OCR:* OCR operation is done using Azure Image Analysis model via self-developed script: *ht_azure_ai_ocr.py.*
6.4. *Chunking:* Result file chunked into usable size for ease via self-developed script: *ht_chunker_2025.py.*

Outcome: Azure Cognitive Services extracted text from images with high accuracy.

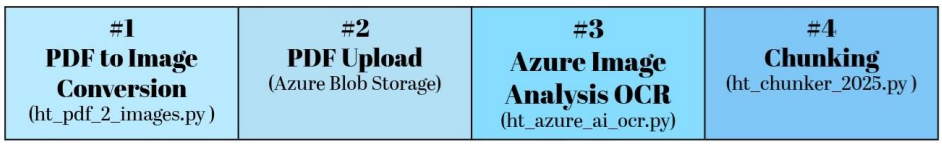

Figure 5.1.1.6. *Nutuk Chapter 3 Process Diagram*

5.1.1.7. Türkiye Cumhuriyeti Tarihi Chapter 2:

7.1. *PDF to Text Conversion:* OConverted PDF files to Text format via self-developed script: *ht_pdf_2_txt.py.*
7.1. *Chunking:* Result file chunked into usable size for ease via self-developed script: *ht_chunker_2025.py.*
Outcome: *PDF to Text* processed structured text directly, reducing preprocessing.

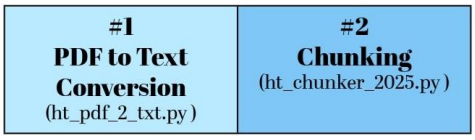

Figure 5.1.1.7. *Türkiye Cumhuriyeti Tarihi Chapter 2 Process Diagram*

5.1.1.8 Türkiye Cumhuriyeti Tarihi Chapter 3:

8.1. *PDF to Markdown Conversion:* PDF file converted into Markdown format leveraging MS MarkItDown program via self-developed script: *ht_markitdown.py.*
8.2. *Markdown to Text Conversion:* MD files converted in to text via manual type switching.
8.3. *Chunking:* Result file chunked into usable size for ease via self-developed script: *ht_chunker_2025.py.*
Outcome: Markdown content was successfully converted into plain text and segmented

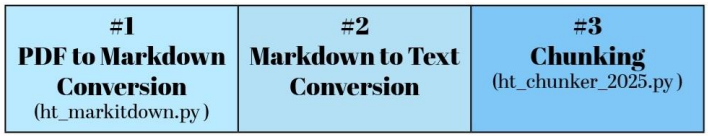

Figure 5.1.1.8. *Türkiye Cumhuriyeti Tarihi Chapter 3 Process Diagram*

5.1.2. Summary & Observations & Challenges

The results validate the design of the data preparation pipeline, demonstrating its effectiveness in generating high-quality, structured datasets for Turkish RAG systems. These datasets form the foundation for subsequent testing and system development stages, as detailed in later sections. The data preparation process resulted in the creation of high-quality datasets across all resources. Key metrics include:

- **OCR Accuracy Variance**:

  Older resources like *Kendi Kendine Lise* required extensive grammar correction due to OCR limitations.
  Modern PDFs, such as *BIST* and *KAP Brochures*, achieved higher initial accuracy.

- **Processing Run-Times**:

  Preprocessing steps (e.g., scanning, column splitting, artifact removal) were time-intensive for dense books.

- **Adaptability**:

  The modular design allowed for seamless integration of additional tools, such as *MarkitDown*, to address unique resource challenges.

5.1.3 Scripts & Programs Developed or Used and Complete Outputs

1. Developed program: **"ht_book_processor":** Program allows pattern renaming of files and splitting image format files into 2 halves via a cut-line. Hence splitting double column or double page scan images into singles. Allowing us to bypass errors that might occur on OCR process.

2. Developed script: **"ht_mask_remover"**: Used for removing masked images that produced by "ht_book_processor". These masks are added to keep aspect ratio of image files this script removes those masks to minimize file size.

3. Developed program: **"ht_mega_ocr":** Self-developed OCR program for image files. The program leverages **pytesseract** for OCR process. Also, it takes in consideration the page breaks as we don't want to lose this information, it is important when LLM's provide grounding reference info. Input images are taken from a folder which is declared in the program, results are appended after each other to form a single block text output.

4. Developed script: **"ht_doc_json_parser":** Simple script that extracts text block from JSON formated files leveraging json modules .strip() method.

5. Developed script: **“ht_chunker_2025”:** Chunks single block text files into ones that of maximum 850 words. This is for ease-of-use also, this methodology is the standard for text using data with LLM modules. Word count can be edited via changing max_words variable. Input file and output folders are declared as absolute paths.

6. Developed script: **“ht_pdf_2_images”**: Simple script that converts a single PDF file to single page PDF files than convers these single PDF files to JPEG image format. Leverages PyPDF2 & pdf2image modules. Both single PDF and JPEG files are saved to their corresponding output folders.

7. Developed script: **“ht_aws_textract”:** Script triggers OCRt process via boto3 API module connection, files are ingested from an Amazon Web Services S3 bucket and fed into Amazon Textract OCR model. Output and result are saved locally.

8. Developed script: **“ht_pdf_2_txt”:** Simple script that converts a PDF file into text format. Uses PyPDF2 module. PDF file must feature selectable text.

9. Developed script: **“ht_google_vision”:** Script that feeds files in Google Cloud Storage bucket to Google Vision API for OCR process, output target is a cloud bucket as well, the script requires authentication via Google CLI to function. Features an automatic JSON to text parsing feature as well that might be used.

10. Developed script: **“ht_markitdown”:** Trigger script that uses Microsofts MarkItDown program to conver PDF files in to Markdown format.

11. Developed pipeline script: **“ht_grok_improv”:** The script sends chunks of text in specified local folder to xAI’s " grok-2-1212" model via API for Turkish grammar improvement and refinement. Output is saved locally as chunks provided. Leverages requests module to communicate. It features a system message before every chunk sent to the model, specifying instructions for refinement operation.

12. Developed pipeline script: “**ht_openai_turkish_fixer”** : Script feeds chunks in a local folder to OpenAI’s “GPT-4o-mini" model for Turkish grammar improvement and refinement hence fixing errors that have occurred during OCR process. Leverages OpenAI’s API. Features a system prompt to instruct model on the operation that should be conducted. Strips response text from JSON responses and saves them locally.

13. Developed pipeline script: **“ht_azure_ai_ocr”:** Feeds files found in given Azure Blob Storage to Azure Cognitive Services endpoint for OCR process. Resulting lines of text are appended together into a single text file. Functions maybe edited to call and use various endpoint like Azure Image Analysis or Azure Document Intelligence modules. The script uses SAS token authorization method which must be provided with account relevant info.

14. Used Program: Google Cloud SDK Shell for authentication
15. Used Program: UltraEdit to display huge single block text and md files
16. Used IDE: PyCharm Community Edition 2024.3

5.2. Results and Observations of Pre-Elementary Model Evaluation for Turkish Language Capabilities Test

Preliminary observations suggest that while cloud-based systems demonstrated superior retrieval accuracy and speed, local implementations performed fairly well. This evaluation has laid the foundation for selecting optimal configurations and models for fine-tuning and deployment in real-world applications.

5.2.1 Summary of Evaluation of Market Leader Models

Table 5.2.1.1. *Control Market Leader Models Pre-Elementary Results Table*

ChatGPT-o1-pro	Delivered great Turkish responses with excellent grammar, fluency, and structure.
chatgpt-4o-latest-20241120	Matched ChatGPT-o1-pro, producing outstanding Turkish NLP capabilities.
gemini-2.0-flash-exp	Surprisingly competitive with OpenAI models; demonstrated high fluency and nuance.
grok-2-2024-08-13	Performed better than expected, likely due to training on Turkish Twitter data.
claude-3-5-sonnet-20241022	Disappointing results, with limited fluency and grammatical errors compared to others.

5.2.2 Summary of Evaluation of Open-Source Models

Table 5.2.2.1. *Open-Source Models Pre-Elementary Results Table*

llama-3.2-1b-instruct	Responses were inadequate; **eliminated.**
llama-3.2-3b-instruct	Insufficient Turkish training; **eliminated.**
qwen2-vl-7b-instruct	Clearly lacked Turkish training; **eliminated.**
granite3-dense-2b (local)	Insufficient Turkish support; **eliminated.**
Yi-1.5-9B-Chat-4Q (local)	Lacked Turkish language training; **eliminated.**
Qwen2.5-7B-Instruct-4Q	Insufficient Turkish language support; **eliminated.**
wizardlm2 (local)	Produced random Turkish words, including inappropriate content; **eliminated.**
Smollm2 (local)	Responses lacked proficiency; **eliminated.**
Phi3.5-8Q (local)	Surprisingly poor Turkish support for its expected capabilities; **eliminated.**
Falcon3-7B-4Q (local)	Showed slight potential but failed to meet the project's quality standards; **has potential yet eliminated.**
ministral-8b-2410	Marginal performance; limited ability. **Has potential.**
llama-3.1-8b-instruct	Showed similar results as the previous model; responses lacked nuance and fluency. **Has potential.**
llama-3.1-tulu-3-8b	Basic Turkish support was evident, but the quality was insufficient for advanced tasks. **Has potential.**
gemma-2-2b-it(local)	Exceptional results, despite its smaller parameter size. Impressive results for its size. **Approved.**
gemma-2-9b-it	Outstanding performance. Assumed to leverage Google's high-quality Turkish datasets. **Approved.**
nemotron-mini-4B (local)	Displayed strong promise. **Approved** for testing.
ibm-granite3.1-8b-instruct-4Q (local)	Highly promising performance even with 4Q quantization, producing accurate responses. **Approved.**

Complete Pre-Elementary Results Spreadsheet can be found here:

Harun_Tuna_Pre_Elementary_Model_Evaluation_for Turkish_Language_Capabilities_Test.xlsx

5.2.3 Summary & Observations of Pre-Elementary Test

- **High Performers**:
  - Models such as **Gemma-2-9b-it**, **Gemma-2-2b-it**, **IBM Granite 3.1-8b-instruct-4Q**, and **Nemotron-mini-4B** demonstrated strong Turkish capabilities, making them ideal candidates for RAG integration.
  - Sector leaders **ChatGPT-o1-pro**, **chatgpt-4o-latest-20241120**, and **Gemini-2.0-flash-exp** sets a benchmark for Turkish NLP related tasks.
- **Notable Surprises**:
  - **Gemini-2.0-flash-exp** performed on par with OpenAI models, showcasing its potential.
  - **Grok-2-2024-08-13** exceeded expectations, likely due to specialized training on Turkish social media datasets.
  - **Claude-3-5-sonnet-20241022** underperformed, highlighting potential gaps in its Turkish language focus.
- **Challenges**:
  - Most models like **Llama-3.2**, **Phi3.5**, and **Wizardlm2** exhibited significant limitations due to inadequate Turkish language training, rendering them extremely unsuitable for the project.

5.2.4 Result of Pre-Elementary Test

The pre-elementary evaluation successfully identified models with strong Turkish language capabilities, providing a foundation for subsequent fine-tuning and integration into RAG systems. Models such as **Gemma-2-9b-it**, **Gemma-2-2b-it**, **IBM Granite 3.1-8b-instruct-4Q**, and **Nemotron-mini-4B** were selected for further development, while sector-leading models served as benchmarks to validate the project's performance objectives.

5.2.5 Program Used and Developed Pre-Elementary Test:

1. **Ollama:** Lightweight, on-prem inference engine for local language models
2. **AnythingLLM:** Open-source platform for building custom LLM’s.
3. **LM Studio:** Enables a desktop environment for streamlined model evaluation, prompting, and performance analysis.
4. **MS Excel:** Spreadsheet application.

5.3. Turkish RAG Testing of Cloud Models and Proof-of-Concept Local RAG

5.3.1. Turkish RAG Test & Results

Models listed in chapter 4.3 were subjected to the 50-question set, their responses were recorded in a spreadsheet for detailed evaluation as stated. Following table summarizes combinations efficiency.

Table 5.3.1.1. *Cloud Models RAG Test Results Table*

Platform / Model	Score	Comments & Observations
P: Vertex AI **M:** Gemini 2.0 Flash-Exp	91	Great, very fast, demonstrates power of Vertex AI Search grounding method, loss might be cause of OCR errors.
P: Vertex AI **M:** Gemini 1.5 Pro-002	92	Great, demonstrates power of Vertex AI Search grounding method, loss might be cause of OCR errors.
P: Vertex AI **M:** Gemini 1.5 Flash-002	70	Older model. Too light. Expected low outcome.
P: AWS Bedrock Agent **M:** Claude Sonnet 3 v1	79	Streamlined agent approach with older modal underperforming, might OCR related issue.
P: AWS Bedrock **M:** Amazon Nova Pro v1	96	Best score, amazing speed, amazing accuracy, titanv2 embeddings with an AWS model sets the par, well synched embeddings and LLM module.
P: AWS Bedrock **M:** Nova Micro v1	90	Amazing speed, slightly worse performance than pro model as its lighter, titanv2 synched embeddings with AWS model demonstrates its power.
P: AWS Bedrock **M:** Sonnet 3.5 v1	94	Great accuracy, Anthropic model is suitable to use with titanv2 embeddings & knowledgebase
P: Azure OpenAI **M:** GPT4o-mini (v2024-07-18)	88	Benchmark light model, low accuracy is assumed to be caused by embeddings or OCR artifacts; text-ada-002 module was used instead of version 3. Still great. Impressive for mini model as its very cheap to use.
P: Azure OpenAI **M:** GPT4o (v2024-05-13)	89	Loss of points assumed to be caused by embeddings or OCR artifacts, text-ada-002 module was used instead of version 3. Very slight improvement over mini model, slower, assuming model is bigger due non-NLP abilities.
P: OpenAI Custom GPT **M:** Custom GPT Assumed 4o	86	Streamlined approach, has set up time of 5 minutes and performance of extensive pipelines, most user-friendly approach. Yet not so enterprise friendly. Low control.

Table 5.3.1.2. *Self-developed pipelines & local proof RAG Test Results Table*

P: Self-made pipeline v1 **M:** Claude-3.5-Sonnet 20241022	44	Self-made pipeline, it can be addressed as an alpha test, low performance is caused due embeddings and modal mismatch, bootleg, yet new "citations" feature from Anthropic is very promising RAG solution.
P: Self-made pipeline v2 **M:** Claude-3.5-Sonnet 20241022	61	Self-made pipeline v2, clear indication of: low performances cause: embeddings and modal mismatch, local FAISS db were expected to underperform. New "citations" feature from Anthropic is very promising RAG solution.
P: Self-made **local** pipeline Via Ollama / AnythingLLM **M:** Gemma2-2b	**80**	Proof of concept that a light opensource, **LOCAL** model, Gemma2-2b, with outdated text-ada-002 embeddings. with free and local vector database LanceDB. Can perform on par; proves the whole theory that this project aims at.

Complete Turkish RAG Results Spreadsheet can be found here:

Harun_Tuna_Turkish_RAG_Testing_of_Cloud_Models_Proof_of_Concept_Local_RAG.xlsx

5.3.2. Summary of RAG Test

AWS Bedrock with Amazon Nova Pro 1 scored the highest accuracy (96), attributed to efficient inference and compatible integration with Titanv2 embeddings. Close contenders included AWS Bedrock's Sonnet 3.5 v1 (94) and Vertex AI's Gemini series (92 and 91), the latter showing minor OCR-related data loss. Mid-range scores encompassed AWS Bedrock's Nova Micro v1 (90), Azure OpenAI GPT4o and GPT4o-mini (89 and 88), and OpenAI's Custom GPT with Knowledgebase (86). While self-developed pipelines performed less competitively (44 and 61), Anthropic's emerging "citations" feature suggests improvements in retrieval and grounding, competing with VertexAI's grounding feature.

Notably, a fully on-premises, open-source Gemma2-2b system (80) demonstrated the feasibility of lightweight local models, even when paired with older embedding modules. This serves as the proof-of-concept that a local light weight module either fine-tuned or developed with solely Turkish data with compatible embeddings and an efficient vector database can reach or surpass other options, eliminating privacy, economic, dependency problems that current solutions face.

5.3.3. Pipelines Developed & Platforms & Programs Used During RAG Test

1. AI Cloud Platform: Amazon Web Services: Bedrock

2. AI Cloud Platform: Google: Vertex AI

3. AI Cloud Platform: OpenAI Platform

4. AI Cloud Platform: Anthropic Platform

5. AI Cloud Platform: Microsoft Azure OpenAI platform

6. Script **"ht_embeddings_save.py":**

Script is used to create and store embeddings for RAG dataset by leveraging FAISS database and previously deployed Azure OpenAI text-embedding-ada-002 embeddings model.

Input text files are read from a local folder. Request architecture is used for communications with embeddings model through Azure OpenAI API, secured through using an API key. Generated vector database is saved locally for further use, allowing vector search on vectorized documents.

7. Self-Developed RAG pipeline: "**ht_self_rag_pipeline_sonnet**":

The pipeline Uses Anthropic API for communications with claude-3-5-sonnet-20241022 model with new "citations" feature enabled. Also, Azure OpenAI API used for prompt embeddings, later embedded prompt response is queried via local FAISS vector database created by ***"ht_embeddings_save.py"*** hence retrieving OpenAI generated embeddings.

8. **Local Proof of Concept RAG pipeline**:

Ollama **local** hosted Gemma2-2b model is used as generator model. As vector database used **local** vector database LanceDB that is connected to the generator model through AnythingLLM interface. The **RAG** data base was created by using OpenAI text-embeddings-ada-002 embeddings model via connecting OpenAI API through AnythingLLM framework. Also, same model connection used for prompt embedding. Only part this pipeline uses internet connection in real time is prompt embeddings.

6. CONCLUSION

This study has investigated the feasibility and effectiveness of Retrieval-Augmented Generation systems specifically tailored for Turkish language. MY work underscores several critical findings that highlight both the performance capabilities of RAG solutions and its broader significance, particularly in resource-constrained, privacy-sensitive environments and sectors.

Able data pipelines play a crucial role in boosting performance of Turkish NLP tasks. By scanning, optical character recognition, preprocessing methods; it becomes possible to generate valid Turkish datasets that reduce noise from older or unstructured source materials as this improves retrieval accuracy and final model outputs. Custom scripts and tools, developed to manage various input formats, proved to be essential in producing well-structured corpus suitable for to be used as RAG knowledge.

Local RAG solutions also demonstrate a huge potential. Although cloud-based platforms such as AWS Bedrock, Vertex AI, and Azure OpenAI displayed high accuracy, the proof-of-concept local RAG system in this study achieved competitive results. With effective data handling and appropriate embedding strategies, on-premises models can rival proprietary systems for tasks specifically targeted for Turkish language. Of particular note, the Gemma2-2b model running on a very modest GPU (GTX 1650) with older embeddings model with non-edited retrieval strategy still performed effectively, which is encouraging for the subject matter.

On-premises deployment offers distinct advantages in terms of privacy and cost. Sensitive sectors such as finance, defense, and healthcare often face regulatory barriers that restrict the external transfer of data such as KVKK laws. By keeping all computational processes within an organization’s own infrastructure, local RAG systems remove major concerns related to privacy and security. Furthermore, although the initial setup for local hardware can be expensive, it typically overcomes them with lower operational expenses over time, hence presenting a sustainable alternative to constantly paying for commercial APIs or cloud services which are the main cause of all problems and concerns.

RAG’s architecture also provides flexibility and ease in updating content. Language models can retrieve and integrate newly available data from external knowledge bases without needing frequent retraining. This capability is especially important in rapidly changing domains, such as finance or legislative affairs, where documents and information must always remain relevant.

In conclusion, this project shows that RAG systems, often associated with large-scale cloud infrastructures, can be effectively adapted for on-premises Turkish NLP applications. By combining curated data, high-quality embeddings, and appropriate local models, surprising performance can be achieved while protecting data privacy and managing operational costs. The methods, findings, and pipelines outlined on this study lay the groundwork for ongoing advances in Turkish-language technologies, hopefully including more comprehensive fine-tuning and the eventual development of fully localized large language models in future.

This project will continue under the “SAYZEK-ATP 2024-2025" program, aiming to further expand the research and development of Turkish-specific RAG solutions. A principal objective in the upcoming phase is to create a fully Turkish-focused local language model, featuring tokenization and embedding strategies that reflect the linguistic nuances of Turkish. To accommodate the computational requirements of these advanced models, an upgrade from a GTX 1650 GPU to an RTX 4090 is planned, allowing for more efficient training and inference processes.

In addition, as a personal note it is clear my future work will emphasize a broader exploration in artificial intelligence and natural language processing fields.

References

Journal & Conference Papers

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Advances in Neural Information Processing Systems (NeurIPS), 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[2] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” OpenAI, Tech. Rep., 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners (GPT-2),” OpenAI, Tech. Rep., 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, ... and D. Amodei, “Language models are few-shot learners (GPT-3),” in Advances in Neural Information Processing Systems (NeurIPS), 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[5] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, ... and P. J. Liu, “Exploring the limits of transfer learning with a unified text-to-text transformer (T5),” Journal of Machine Learning Research, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: https://arxiv.org/abs/1910.10683

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, ... and S. Riedel, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Advances in Neural Information Processing Systems (NeurIPS), 2020. [Online]. Available: https://arxiv.org/abs/2005.11401

Cloud Platforms & Documentation

[7] Microsoft, Azure AI Services Documentation, Microsoft, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/

Amazon, Amazon Bedrock Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/bedrock/

Anthropic, Claude Documentation, 2024. [Online]. Available: https://docs.anthropic.com/claude

Microsoft, Azure OpenAI Service Documentation, Microsoft, 2024. [Online]. Available: https://learn.microsoft.com/azure/cognitive-services/openai/

Google, Google Vertex AI Documentation, 2024. [Online]. Available: https://cloud.google.com/vertex-ai/docs

Open-Source Tools

[8] Mintplex Labs, AnythingLLM - Open-Source Local LLM Framework, 2024. [Online]. Available: https://github.com/Mintplex-Labs/anything-llm

[9] Ollama, Ollama - Run Large Language Models Locally, 2024. [Online]. Available: https://docs.ollama.ai

[10] LM Studio, LM Studio - Offline LLM Inference Tool, 2024. [Online]. Available: https://lmstudio.ai

RAG Data

[11] Borsa İstanbul, Finansal Okuryazarlık İçerikleri, 2024. [Online]. Available: https://www.borsaistanbul.com/tr/sayfa/10794/finansal-okuryazarlik-icerikleri

[12] Kamuyu Aydınlatma Platformu (KAP), Mevzuat, Duyurular ve Kılavuzlar, 2024. [Online]. Available: https://www.kap.org.tr/tr/menu-icerik/KAP-Hakkinda/Mevzuat-Duyurular-ve-Kilavuzlar

[13] M. K. Atatürk, Nutuk, Republic of Turkey Ministry of Culture and Tourism, 1927. [Online]. Available: https://ekitap.ktb.gov.tr/TR-273376/nutuk.html

[14] Atatürk Research Center, Türkiye Cumhuriyeti Tarihi, Republic of Turkey Atatürk Research Center, 2024. [Online]. Available: https://atam.gov.tr/e-yayin/turkiye-cumhuriyeti-tarihi-1/,https://atam.gov.tr/e-yayin/turkiye-cumhuriyeti-tarihi-2/ ,https://atam.gov.tr/e-yayin/turkiye-cumhuriyeti-tarihi-3/

APPENDIX A: CODE

A.A.1. ht_aws_textract.py: OCR using AWS Textract; see 5.1.3.

```
import time
import boto3


def start_text_detection(s3_bucket, s3_file):
    textract = boto3.client('textract', region_name='us-east-1')
    response = textract.start_document_text_detection(
        DocumentLocation={'S3Object': {'Bucket': s3_bucket,
'Name': s3_file}})
    return response["JobId"]


def is_job_complete(job_id):
    textract = boto3.client('textract', region_name='us-east-1')
    status = "IN_PROGRESS"
    while status == "IN_PROGRESS":
        response = textract.get_document_text_detection(JobId=job_id)
        status = response["JobStatus"]
        print(f"Current Textract job status: {status}")
        if status == "IN_PROGRESS": time.sleep(5)
    return status


def get_job_results(job_id):
    textract = boto3.client('textract', region_name='us-east-1')
    pages = []
    response = textract.get_document_text_detection(JobId=job_id)
    pages.append(response)
    while "NextToken" in response:
        next_token = response["NextToken"]
        response = textract.get_document_text_detection(
            JobId=job_id,NextToken=next_token)
        pages.append(response)
    return pages
def extract_text_from_pages(pages):
    all_text = []
    for page in pages:
        blocks = page.get("Blocks", [])
        for block in blocks:
            if block["BlockType"] == "LINE":
                all_text.append(block["Text"])
    return "\n".join(all_text)
```

```python
if __name__ == "__main__":
    S3_BUCKET = "xxx"
    S3_FILE = "xxx"
    job_id = start_text_detection(S3_BUCKET, S3_FILE)
    print(f"Started Textract job with JobId: {job_id}")
    final_status = is_job_complete(job_id)


    if final_status == "SUCCEEDED":
        print("Textract job succeeded. Retrieving results...")
        result_pages = get_job_results(job_id)
        final_text = extract_text_from_pages(result_pages)
        print("\n--- EXTRACTED TEXT (first 500 chars) ---")
        print(final_text[:500])
        with open("textract_output.txt", "w", encoding="utf-8") as f:
            f.write(final_text)
        print("\nFull text saved to 'textract_output.txt'.")


    else:
        print(f"Textract job ended with status: {final_status}")
```

A.A.2. ht_azure_ai_ocr.py: OCR via Azure AI resources; see 5.1.3.

```
import requests
import time
from azure.storage.blob import BlobServiceClient


endpoint = "xxx"
key = “xxx”
account_name = ""
container_name = ""
sas_token = (
    "sp="
    "sv="
)
def get_image_urls_from_blob(account_name, container_name, sas_token):
    print("Retrieving...")
    blob_service_client = BlobServiceClient(
        account_url=f"https://{account_name}.blob.core.windows.net",
        credential=sas_token)
    container_client = blob_service_client.get_container_client(container_name)
    image_urls = []
    for blob in container_client.list_blobs():
        blob_url = (
            f"https://{account_name}.blob.core.windows.net/"
            f"{container_name}/{blob.name}?{sas_token}")
        print(f"Found blob: {blob.name}")
        image_urls.append(blob_url)
    print(f"Total images: {len(image_urls)}")
    return image_urls
def analyze_image(image_url):
    headers = {"Ocp-Apim-Subscription-Key": key,"Content-Type": "application/json"}
    data = {"url": image_url}
    print(f"\nSending analyze request for image: {image_url}")
    response = requests.post(endpoint, headers=headers, json=data)
    if response.status_code == 202:  # 202 Accepted
        operation_location = response.headers["Operation-Location"]
        print(f"Operation accepted. Polling for results at: {operation_location}")
    else:
        print(f"Error: HTTP {response.status_code}")
        print(f"Response Text: {response.text}")
        return None
    print("Polling for results...")
```

```
    while True:
        poll_response = requests.get(operation_location, headers=headers)
        poll_result = poll_response.json()
        if poll_response.status_code != 200:
            print(f"Polling failed: HTTP {poll_response.status_code}")
            print(f"Response: {poll_result}")
            return None
        status = poll_result.get("status")
        print(f"Status: {status}")
        if status == "succeeded":
            return poll_result
        elif status == "failed":
            print("Analysis failed.")
            print(poll_result)
            return None
        time.sleep(2)
def extract_text_from_result(result):
    extracted_text = []
    read_results = result.get("analyzeResult", {}).get("readResults", [])
    for page in read_results:
        for line in page.get("lines", []):
            extracted_text.append(line["text"])
    return "\n".join(extracted_text)


def main():
    image_urls = get_image_urls_from_blob(account_name, container_name, sas_token)
    if not image_urls:
        print("No images found in the container.")
        return
    all_text = []
    for idx, image_url in enumerate(image_urls, start=1):
        print(f"\n--- Processing image {idx}/{len(image_urls)} ---")
        result = analyze_image(image_url)
        if result:
            text = extract_text_from_result(result)
            all_text.append(text)
    combined_text = "\n\n".join(all_text)
    with open("ocr_results.txt", "w", encoding="utf-8") as f:
        f.write(combined_text)
    print("\nOCR results saved to 'ocr_results1.txt'.")


if __name__ == "__main__":
    main()
```

A.A.3. ht_chunker_2025.py: Chunks single block text into parts; see 5.1.3.

```
import os
import re
def chunk_text_by_sentence(input_file, output_folder, max_words=850):
    os.makedirs(output_folder, exist_ok=True)
    with open(input_file, 'r', encoding='utf-8') as f:
        text = f.read()
    sentences = re.split(r'(?<=[.!?])\s+', text.strip())
    chunks = []
    current_chunk = []
    current_word_count = 0
    for sentence in sentences:
        words_in_sentence = len(sentence.split())
        if current_word_count + words_in_sentence <= max_words:
            current_chunk.append(sentence)
            current_word_count += words_in_sentence
        else:
            chunks.append(" ".join(current_chunk))
            current_chunk = [sentence]
            current_word_count = words_in_sentence
    if current_chunk:
        chunks.append(" ".join(current_chunk))


    for i, chunk in enumerate(chunks, start=1):
        chunk_filename = os.path.join(output_folder, f'chunk_{i}.txt')
        with open(chunk_filename, 'w', encoding='utf-8') as f:
            f.write(chunk)
    print(f"Created {len(chunks)} chunk(s) in '{output_folder}'.")


if __name__ == '__main__':
    input_file_path = r""
    output_folder_path = r""
    chunk_text_by_sentence(input_file_path, output_folder_path, max_words=850)
```

A.A.4. ht_doc_json_parser.py: JSON to text format parser; see 5.1.3.

```
import json


input_path = r""
output_path = r""


def main():
    with open(input_path, "r", encoding="utf-8") as f:
        data = json.load(f)
    analyze_result = data.get("analyzeResult", {})
    pages = analyze_result.get("pages", [])


    all_lines = []
    for page in pages:
        for line_item in page.get("lines", []):
            text_content = line_item.get("content", "").strip()
            if text_content.isdigit():
                continue
            all_lines.append(text_content)
    full_text = "\n".join(all_lines)
    with open(output_path, "w", encoding="utf-8") as txt_file:
        txt_file.write(full_text)


    print(f"Extraction completed! Check '{output_path}' for results.")


if __name__ == "__main__":
    main()
```

A.A.5. ht_pdf_2_images.py: Converts PDF into single PDF pages and Images; see 5.1.3.

```
import os
from PyPDF2 import PdfReader, PdfWriter
from pdf2image import convert_from_path
def split_pdf(input_pdf_path, output_folder_path):
    os.makedirs(output_folder_path, exist_ok=True)
    pdf_reader = PdfReader(input_pdf_path)
    total_pages = len(pdf_reader.pages)
    print(f"Total pages in PDF: {total_pages}")
    individual_pdfs = []
    for i in range(total_pages):
        pdf_writer = PdfWriter()
        pdf_writer.add_page(pdf_reader.pages[i])
        output_pdf_path = os.path.join(output_folder_path, f"page_{i + 1}.pdf")
        with open(output_pdf_path, 'wb') as output_pdf:
            pdf_writer.write(output_pdf)
        individual_pdfs.append(output_pdf_path)
        print(f"Saved page {i + 1} as {output_pdf_path}.")
    return individual_pdfs
def pdf_pages_to_images(pdf_pages, output_folder_path):
    os.makedirs(output_folder_path, exist_ok=True)
    for page_number, pdf_path in enumerate(pdf_pages, start=1):
        images = convert_from_path(pdf_path, dpi=300)
        for image in images:
            output_image_path = os.path.join(output_folder_path, f"page_{page_number}.jpeg")
            image.save(output_image_path, 'JPEG')
            print(f"Converted {pdf_path} to {output_image_path}.")
if __name__ == "__main__":
    input_pdf = r""
    split_folder = r""
    image_folder = r""
    pdf_pages = split_pdf(input_pdf, split_folder)
    pdf_pages_to_images(pdf_pages, image_folder)
```

A.A.6. ht_pdf_2_txt.py: Converts PDF into text format; see 5.1.3.

```
import os
from PyPDF2 import PdfReader


def extract_pdf_text(pdf_path, output_file):
    os.makedirs(os.path.dirname(output_file), exist_ok=True)
    reader = PdfReader(pdf_path)
    with open(output_file, "w", encoding="utf-8") as txt_file:


        for page_num, page in enumerate(reader.pages, start=1):
            text = page.extract_text()


            if text:
                txt_file.write(f"\n--- Page {page_num} ---\n")
                txt_file.write(text.strip() + "\n")


            else:
                txt_file.write(f"\n--- Page {page_num}: No text found ---\n")
    print(f"Text extracted and saved to: {output_file}")


if __name__ == "__main__":
    pdf_path = r""
    output_file = r""


    extract_pdf_text(pdf_path, output_file)
```

A.A.7. ht_google_vision.py: Uses Google Cloud Platform for OCR operation; see 5.1.3.

```
import time
from google.cloud import vision
from google.cloud import storage
import json


def async_ocr_pdf(input_gcs_uri: str, output_gcs_uri: str, project_id: str, batch_size: int = 100):
    client = vision.ImageAnnotatorClient()
    feature = vision.Feature(type_=vision.Feature.Type.DOCUMENT_TEXT_DETECTION)
    gcs_source = vision.GcsSource(uri=input_gcs_uri)
    input_config = vision.InputConfig(gcs_source=gcs_source, mime_type="application/pdf")
    gcs_destination = vision.GcsDestination(uri=output_gcs_uri)
    output_config = vision.OutputConfig(gcs_destination=gcs_destination, batch_size=batch_size)
    async_request = vision.AsyncAnnotateFileRequest(features=[feature], input_config=input_config,
output_config=output_config)
    operation = client.async_batch_annotate_files(requests=[async_request])


    print("Processing PDF OCR...")
    operation.result(timeout=300)
    print("OCR processing complete.")


def list_gcs_objects(bucket_name: str, prefix: str):
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    print("Objects in bucket (with prefix={}):".format(prefix))
    for blob in bucket.list_blobs(prefix=prefix):
        print(blob.name)


def download_and_parse_ocr_output(bucket_name: str, prefix: str):
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    all_text = []


    for blob in bucket.list_blobs(prefix=prefix):
        if blob.name.endswith(".json"):
            json_data = blob.download_as_text()
            data = json.loads(json_data)
            responses = data.get("responses", [])
            for r in responses:
                annotation = r.get("fullTextAnnotation", {})
```

```
                text = annotation.get("text", "")
                all_text.append(text)
    combined_text = "\n".join(all_text)
    return combined_text


if __name__ == "__main__":
    PROJECT_ID = ""
    INPUT_URI = ""
    OUTPUT_URI = ""
    async_ocr_pdf(
        input_gcs_uri=INPUT_URI,
        output_gcs_uri=OUTPUT_URI,
        project_id=PROJECT_ID,
        batch_size=100
    )
    print("Listing OCR output files:")
    list_gcs_objects(bucket_name="harun-tuna-01", prefix="output/")
    print("\nParsing OCR JSON outputs to extract text...")
    text_result = download_and_parse_ocr_output(
        bucket_name="harun-tuna-01",
        prefix="output/"
    )
    print("\nExtracted Text (First 500 characters):")
    print(text_result[:500])
    with open("final_output.txt", "w", encoding="utf-8") as out_file:
        out_file.write(text_result)
    print("\nText saved to final_output.txt")
```

A.A.8. ht_mega_ocr.py: Local OCR program using pyTesseract; see 5.1.3.

```
import pytesseract
from tqdm import tqdm
import os
import re
from PIL import Image


def extract_page_and_part(filename):
    page_match = re.search(r"Sayfa\s+(\d+)", filename, re.IGNORECASE)
    if page_match:
        page_num = int(page_match.group(1))
    else:
        page_num = 1
    part_match = re.search(r'_part(\d+)', filename, re.IGNORECASE)
    if part_match:
        part_num = int(part_match.group(1))
    else:
        part_num = 1
    return page_num, part_num
def ocr_images(image_folder, output_file, lang="tur", psm=3, oem=3):
    tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
    pytesseract.pytesseract.tesseract_cmd = tesseract_cmd
    image_files = sorted(
        [f for f in os.listdir(image_folder) if f.lower().endswith((".jpg", ".jpeg", ".png"))]
    )
    if not image_files:
        print("No image files found in the folder.")
        return
    print(f"Found {len(image_files)} images. Starting OCR...")
    pages = {}
    config = f"--oem {oem} --psm {psm}"
    for image_file in tqdm(image_files, desc="Processing Images", unit="image"):
        image_path = os.path.join(image_folder, image_file)
        page_num, part_num = extract_page_and_part(image_file)
        try:
            text = pytesseract.image_to_string(Image.open(image_path), lang=lang, config=config)
            text = text.strip()
            if page_num not in pages:
                pages[page_num] = {}
            pages[page_num][part_num] = text
        except Exception as e:
            print(f"Error processing {image_file}: {e}")
```

```
    with open(output_file, "w", encoding="utf-8") as f:
        for page_num in sorted(pages.keys()):
            part_texts = [pages[page_num][p] for p in sorted(pages[page_num].keys())]
            combined_text = "\n".join(part_texts)
            f.write(f"\n--- Sayfa {page_num} ---\n")
            f.write(combined_text + "\n")
    print(f"OCR complete! Text saved to: {output_file}")


image_folder_path = r""
output_text_path = r""
ocr_images(image_folder_path, output_text_path, lang="tur", psm=3, oem=3)
```

A.A.9. ht_markitdown.py: Script for MSMarkItDown; PDF to MD conversion see 5.1.3.

```
from markitdown import MarkItDown


pdf_input = r""
md_output = r""
# Create a MarkItDown object
md = MarkItDown()
# Convert PDF to Markdown
result = md.convert(pdf_input)
# Write the resulting Markdown text to a file
with open(md_output, "w", encoding="utf-8") as f:
    f.write(result.text_content)


print(f"Markdown saved to: {md_output}")
```

A.A.10. ht_openai_turkish_fixer.py: Script to feed text input via API to OpenAI LLM models for grammar fix and refinement; see section 5.1.3.

```
import os
import openai
openai.api_key = ""
input_folder = r""
output_folder = r""
model_name = ""
if not os.path.exists(output_folder):
    os.makedirs(output_folder)
for filename in os.listdir(input_folder):
    if filename.endswith(".txt"):
        input_file_path = os.path.join(input_folder, filename)
        with open(input_file_path, 'r', encoding='utf-8') as f:
            original_text = f.read()
            print(f"Read {len(original_text)} characters from {filename}")
        prompt = (
            "Bu Türkçe metini kesinlikle anlam kaybı olmadan dilbilgisi ve yazım olarak düzelt. "
            "daha anlaşılır hale getir. Metnin özgün anlamını korumaya çok özen göster. "
            "\"--- Sayfa 7 ---\" şeklindeki sayfa bilgilerini olduğu gibi koru. "
            "Bana düzeltilmiş metin dışında cevap verme.\n\n"
            f"{original_text}"
        )
        try:
            response = openai.ChatCompletion.create(
                model=model_name,
                messages=[
                    {"role": "system", "content": "Türkçe metin düzenleyici."},
                    {"role": "user", "content": prompt}
                ],
                max_tokens=16000
            )
            improved_text = response.choices[0].message.content.strip()
            output_file_path = os.path.join(output_folder, filename)
            with open(output_file_path, 'w', encoding='utf-8') as out_file:
                out_file.write(improved_text)
            print(f"Processed and improved: {filename}")
        except Exception as e:
            print(f"Error processing {filename}: {e}")
```

A.A.11. ht_grok_improv.py: Script to feed text input via API to xAI LLM models for grammar fix and refinement; see section 5.1.3.

```
import os
import requests


def chat_with_xai_api(api_url, api_key, user_messages):
    headers = {
        'Authorization': f'Bearer {api_key}',
        'Content-Type': 'application/json',
    }
    payload = {
        "messages": user_messages,
        "model": "grok-2-1212",
        "stream": False,
        "temperature": 0
    }
    try:
        response = requests.post(api_url, headers=headers, json=payload)
        response.raise_for_status()
        json_response = response.json()
        return json_response.get("choices", [{}])[0].get("message", {}).get("content", "No content
in response.")
    except requests.exceptions.RequestException as e:
        return f"An error occurred: {e}"


if __name__ == "__main__":
    API_URL = ""
    API_KEY = ""
    input_folder = r""
    output_folder = r""
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)


    for filename in os.listdir(input_folder):
        if filename.endswith(".txt"):
            input_file_path = os.path.join(input_folder, filename)
            with open(input_file_path, 'r', encoding='utf-8') as f:
                original_text = f.read()
                print(f"Read {len(original_text)} characters from {filename}")
```

```
            prompt = (
                "Bu Türkçe metini kesinlikle anlam kaybı olmadan dilbilgisi ve yazım olarak düzelt.
"
                "daha anlaşılır hale getir. Metnin özgün anlamını korumaya çok özen göster. "
                "\"--- Sayfa 7 ---\" şeklindeki sayfa bilgilerini olduğu gibi koru. "
                "Bana düzeltilmiş metin dışında cevap verme.\n\n"
                f"{original_text}"
            )

            messages = [
                {"role": "system", "content": "Türkçe metin düzenleyici."},
                {"role": "user", "content": prompt}
            ]

            try:
                improved_text = chat_with_xai_api(API_URL, API_KEY, messages)

                output_file_path = os.path.join(output_folder, filename)
                with open(output_file_path, 'w', encoding='utf-8') as out_file:
                    out_file.write(improved_text)

                print(f"Processed and improved: {filename}")

            except Exception as e:
                print(f"Error processing {filename}: {e}")
```

A.A.12. ht_mask_remover : Script to remove extra black masks; see section 5.1.3.

```
import os
from PIL import Image, ImageChops
def remove_black_borders(input_path, output_path):
    img = Image.open(input_path)
    if img.mode != 'RGB':
        img = img.convert('RGB')
    bg = Image.new("RGB", img.size, (0, 0, 0))
    diff = ImageChops.difference(img, bg)
    bbox = diff.getbbox()
    if bbox:
        cropped_img = img.crop(bbox)
        cropped_img.save(output_path)
    else:
        img.save(output_path)
def process_folder(input_folder, output_folder):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    for filename in os.listdir(input_folder):
        input_path = os.path.join(input_folder, filename)
        if os.path.isdir(input_path):
            continue
        if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tiff', '.bmp', '.gif')):
            continue
        output_path = os.path.join(output_folder, filename)
        remove_black_borders(input_path, output_path)
        print(f"Processed {filename}")


input_folder = r""
output_folder = r""
process_folder(input_folder, output_folder)
```

A.A.13. ht_embeddings_save: Script to remove extra black masks; see section 5.1.3.

```
import os
import faiss
import numpy as np
import requests
import pickle
import logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')


def embed_text_with_azure_openai(text, api_key, endpoint):
    headers = {
        "Content-Type": "application/json",
        "api-key": api_key,
    }
    payload = {
        "input": text,
    }
    try:
        response = requests.post(endpoint, headers=headers, json=payload)
        response.raise_for_status()
        logging.info(f"Embedding generated for text of length {len(text)}.")
        return response.json()["data"][0]["embedding"]
    except requests.exceptions.RequestException as e:
        logging.error(f"Failed to generate embedding: {e}")
        raise
def load_documents_from_folder(folder_path):
    documents = []
    file_names = []
    logging.info(f"Loading documents from folder: {folder_path}")
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        if os.path.isfile(file_path) and file_name.endswith(".txt"):
            try:
                with open(file_path, "r", encoding="utf-8") as file:
                    documents.append(file.read())
                    file_names.append(file_name)
                    logging.info(f"Loaded file: {file_name}")
            except Exception as e:
                logging.error(f"Failed to read file {file_name}: {e}")
    logging.info(f"Loaded {len(documents)} documents from folder.")
    return documents, file_names
```

```python
def create_faiss_index(folder_path, api_key, endpoint, output_index_path="faiss_index.bin",
output_docs_path="documents.pkl"):
    documents, file_names = load_documents_from_folder(folder_path)
    if not documents:
        logging.error("No documents found in the specified folder.")
        return
    embeddings = []
    for idx, doc in enumerate(documents):
        logging.info(f"Processing document {idx + 1}/{len(documents)}: {file_names[idx]}")
        try:
            embedding = embed_text_with_azure_openai(doc, api_key, endpoint)
            embeddings.append(embedding)
            logging.debug(f"Embedding for document {file_names[idx]}: {embedding[:5]}...")
        except Exception as e:
            logging.error(f"Skipping document {file_names[idx]} due to error: {e}")
            continue
    if not embeddings:
        logging.error("No embeddings were generated. Exiting.")
        return
    embeddings = np.array(embeddings, dtype=np.float32)
    embedding_dim = embeddings.shape[1]
    faiss_index = faiss.IndexFlatL2(embedding_dim)
    faiss_index.add(embeddings)
    logging.info(f"FAISS index created with {len(embeddings)} embeddings.")
    try:
        faiss.write_index(faiss_index, output_index_path)
        with open(output_docs_path, "wb") as f:
            pickle.dump({"documents": documents, "file_names": file_names}, f)
        logging.info(f"FAISS index saved to '{output_index_path}'.")
        logging.info(f"Document metadata saved to '{output_docs_path}'.")
    except Exception as e:
        logging.error(f"Failed to save FAISS index or metadata: {e}")


if __name__ == "__main__":
    api_key = ""
    endpoint = ""
    folder_path = r""
    output_index_path = ""
    output_docs_path = ""
    create_faiss_index(folder_path, api_key, endpoint, output_index_path, output_docs_path)
```

A.A.14. ht_self_rag_pipeline_sonnet: Script to remove black masks; see section 5.1.3.

```
import anthropic
import faiss
import pickle
import numpy as np
import requests
import logging

logging.basicConfig(level=logging.DEBUG, format="%(asctime)s - %(levelname)s - %(message)s")
ANTHROPIC_API_KEY = ""
AZURE_OPENAI_API_KEY = ""
AZURE_OPENAI_ENDPOINT = ""

try:
    client = anthropic.Anthropic(api_key=ANTHROPIC_API_KEY)
    logging.info("Anthropic client initialized successfully.")
except Exception as e:
    logging.error(f"Failed to initialize Anthropics client: {e}")
    exit()

def embed_text_with_azure_openai(text, api_key, endpoint):
    headers = {
        "Content-Type": "application/json",
        "api-key": api_key,
    }
    payload = {
        "input": text,
    }
    try:
        response = requests.post(endpoint, headers=headers, json=payload)
        response.raise_for_status()
        logging.info(f"Embedding generated for text of length {len(text)}.")
        return response.json()["data"][0]["embedding"]
    except requests.exceptions.RequestException as e:
        logging.error(f"Failed to generate embedding: {e}")
        raise
```

```python
def load_faiss_index(index_path="faiss_index.bin", docs_path="documents.pkl"):

    try:
        faiss_index = faiss.read_index(index_path)
        with open(docs_path, "rb") as f:
            metadata = pickle.load(f)
        documents = metadata["documents"]
        file_names = metadata["file_names"]
        logging.info("FAISS index and documents loaded successfully.")
        return faiss_index, documents, file_names
    except Exception as e:
        logging.error(f"Failed to load FAISS index or documents: {e}")
        exit()


def search_faiss_index(query_embedding, faiss_index, documents, file_names,
                       top_k=10, distance_threshold=0.5):
    try:
        distances, indices = faiss_index.search(
            np.array([query_embedding], dtype=np.float32),
            top_k)
        results = []
        for dist, idx in zip(distances[0], indices[0]):
            results.append({
                "text": documents[idx],
                "file_name": file_names[idx],
                "distance": dist
            })
        if distance_threshold is not None:
            filtered = []
            for item in results:
                if item["distance"] < distance_threshold:
                    filtered.append(item)
            results = filtered
        logging.info(f"Found {len(results)} relevant documents.")
        return results
    except Exception as e:
        logging.error(f"Error searching FAISS index: {e}")
        return []
```

```python
def parse_claude_response(claude_message):
    if not hasattr(claude_message, "content"):
        return "No content found in the response."
    output = ""
    for block in claude_message.content:
        if block.type == "text":
            output += block.text
            if getattr(block, "citations", None):
                citations_list = []
                for c in block.citations:
                    doc_title = c.get("document_title", "Unknown")
                    start_idx = c.get("start_char_index", 0)
                    end_idx = c.get("end_char_index", 0)
                    citations_list.append(f"[{doc_title} (chars {start_idx}-{end_idx})]")
                if citations_list:
                    output += " " + ", ".join(citations_list)
    return output.strip()
def chat_with_claude(query, relevant_documents, model="claude-3-5-sonnet-20241022",
max_tokens=4000, temperature=0.2):
    system_prompt = (
        "Sen bir türkçe soru-cevap asistanısın. Sana yöneltilen sorulara yalnızca "
        "sağlanan belgelerden referans alarak cevap ver. Belgelerdeki metinleri "
        "değiştirmen gerekmez; ancak, varsa yazım ve dilbilgisi hatalarını düzelt. "
        "Ana fikir ve konudan sapmadığından emin ol ve cevaplarını her zaman "
        "dikkatlice değerlendir. Verdiğin her cevaba mutlaka belgelerden bir referans "
        "ekle. Sağlanan kaynaklarda bir bilgi bulunmaz ise: cevap verme, bulamadığını belirt. "
        "Referansları cevabın sonunda listele."
    )
    sources = [
        {
            "type": "document",
            "source": {
                "type": "text",
                "media_type": "text/plain",
                "data": doc["text"]
            },
            "title": doc["file_name"],
            "context": "This is a trustworthy document.",
            "citations": {"enabled": True}
        }
        for doc in relevant_documents
    ]
    user_message = {
        "role": "user",
        "content": sources + [{"type": "text", "text": query}]
```

```
    }
    messages = [user_message]
    try:
        response = client.messages.create(
            model=model,
            max_tokens=max_tokens,
            temperature=temperature,
            system=system_prompt,
            messages=messages
        )
        logging.debug(f"Raw response from Claude: {response}")
        return parse_claude_response(response)
    except Exception as e:
        logging.error(f"Error during API call: {e}")
        return "An unexpected error occurred. Please check the logs."


if __name__ == "__main__":
    index_path = "faiss_index.bin"
    docs_path = "documents.pkl"
    faiss_index, documents, file_names = load_faiss_index(index_path, docs_path)
    print("Welcome to the Claude Chat with Citations!")
    while True:
        user_input = input("You: ")
        if user_input.lower() in ["exit", "quit"]:
            print("Goodbye!")
            break
        try:
            query_embedding = embed_text_with_azure_openai(
                user_input,
                AZURE_OPENAI_API_KEY,
                AZURE_OPENAI_ENDPOINT
            )
        except Exception:
            print("Claude: Failed to generate query embedding. Please try again.")
            continue
        relevant_docs = search_faiss_index(query_embedding, faiss_index, documents, file_names,
 top_k=20, distance_threshold=0.5)
        if not relevant_docs:
            print("Claude: Sorry, I couldn't find any relevant documents.")
            continue


        response_text = chat_with_claude(user_input, relevant_docs)
        print(f"Claude: {response_text}")
```

A.A.15. ht_book_processor: Script to remove extra black masks; see section 5.1.3.

```
"""
This program was created me: "Harun Tuna" as part of my graduation project.
Disclaimer: AI tools were used during the development process of this program.
Description:
"Book_Processor" assists with handling images (e.g., scanned book pages) in three modes:
1. Free Rename mode: Manually rename each image by typing a new name.
2. Pattern Rename mode: Use a pattern with '[]' placeholders to rename images in a structured
fashion.
3. Page Separator mode: Split each image into two parts vertically. This can be used to reform
double copies or twin column page structure.
Keybindings and Controls:
- Common Across All Modes:
  - Esc: Quits the application.
  - Tab: Skip the current image.
  - Enter: Confirm.
- In Page Separator Mode:
  - a/d: Move the top cutting line horizontally (left/right).
  - Left/Right Arrow Keys: Move the bottom cutting line horizontally.
Dependencies:
- Python 3.x
- tkinter
- Pillow
Usage:
1. Run `python ht_book_processor.py`.
2. A welcome popup prompts you to select a mode: Free Rename, Pattern Rename, or Page Separator.
3. Depending on the mode:
   - Free/Pattern Rename: Each image is displayed, and a popup appears for renaming or skipping.
This is same for Pattern Rename mode.
   - Pattern Rename: Providing a pattern with '[]' placeholders for quick renaming e.g. "History
Book Page []".
   - Page Separator: The image is displayed with adjustable lines. Use keys or drag with mouse and
position the line to split.
Important Notes:
- Images are displayed scaled to fit the screen or 80% of it in separator mode. Scaling does not
affect the actual output quality or the dimensions of the resulting split images.
- Renamed or split images are saved in the same directory as the original images with _part1 &
_part2 suffix. Extension is same as original image.
"""
```

```
import os
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk, ImageDraw
class ImageRenamer:
    # The main class.
    def __init__(self, master):
        self.master = master
        self.master.title("Book Processor")
        self.master.state('zoomed')  #Maximize the window for best viewing.
        self.mode = None
        self.pattern = None
        self.forbidden_chars = r'<>:"/\\|?*'  #These chars are not allowed in naming schema
        self.image_label = None
        self.canvas = None
        self.scale_top = None
        self.scale_bottom = None
        self.bottom_frame = None
        self.top_frame = None
        self.original_img = None
        self.display_img = None
        self.tk_img = None
        self.line_id = None
        self.top_line_x = 0
        self.bottom_line_x = 0
        self.separator_interface_created = False
        self.bind_escape_to_quit(self.master)
        self.welcome_popup()
    def bind_escape_to_quit(self, window):
        window.bind('<Escape>', lambda e: self.quit_app())
    def welcome_popup(self):
        popup = tk.Toplevel(self.master)
        popup.title("Welcome")
        self.bind_escape_to_quit(popup)
        popup_width = 300
        popup_height = 200
        screen_w = popup.winfo_screenwidth()
        screen_h = popup.winfo_screenheight()
        x = (screen_w // 2) - (popup_width // 2)
        y = (screen_h // 2) - (popup_height // 2)
        popup.geometry(f"{popup_width}x{popup_height}+{x}+{y}")
        popup.transient(self.master)
        popup.grab_set()
        popup.focus_force()
        popup.lift()
        tk.Label(popup, text="Choose a Mode:").pack(pady=10)
```

```
        def choose_free():
            # Choice 1
            self.mode = "free"
            popup.destroy()
            self.init_directory_and_images()

        def choose_pattern():
            # Choice 2
            self.mode = "pattern"
            popup.destroy()
            self.init_directory_and_images()

        def choose_separator():
            # Choice 3
            self.mode = "separator"
            popup.destroy()
            self.init_directory_and_images()

        tk.Button(popup, text="Free Rename", command=choose_free).pack(pady=5)
        tk.Button(popup, text="Pattern Rename", command=choose_pattern).pack(pady=5)
        tk.Button(popup, text="Page Separator", command=choose_separator).pack(pady=5)

        self.master.wait_window(popup)

    def init_directory_and_images(self):
        # Prompt user for image directory
        self.directory = filedialog.askdirectory(title="Select Image Directory")
        if not self.directory:
            messagebox.showerror("Error", "No dir selected.")
            self.master.destroy()
            return

        valid_extensions = ('.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tiff')
        self.image_files = [f for f in os.listdir(self.directory) if
f.lower().endswith(valid_extensions)]

        if not self.image_files:
            messagebox.showerror("Error", "No images found in the selected dir.")
            self.master.destroy()
            return
        self.current_index = 0
        if self.mode == "pattern":
            self.ask_for_pattern()
        self.show_image()
```

```
    def ask_for_pattern(self):
        popup = tk.Toplevel(self.master)
        popup.title("Enter Pattern")
        self.bind_escape_to_quit(popup)


        # Center the popup2
        popup_width = 500
        popup_height = 150
        screen_w = popup.winfo_screenwidth()
        screen_h = popup.winfo_screenheight()
        x = (screen_w // 2) - (popup_width // 2)
        y = (screen_h // 2) - (popup_height // 2)
        popup.geometry(f"{popup_width}x{popup_height}+{x}+{y}")
        popup.transient(self.master)
        popup.grab_set()
        popup.focus_force()
        popup.lift()
        tk.Label(popup, text="Enter a pattern with [].\nExample: 'Tarih Kitabı Sayfalar []-[]' or
'Kimya []'").pack(
            pady=10)
        pattern_entry = tk.Entry(popup, width=50)
        pattern_entry.pack(pady=5)
        pattern_entry.focus_set()


        def submit_pattern(event=None):
            p = pattern_entry.get().strip()
            if '[]' not in p:
                messagebox.showerror("Error", "Pattern must contain '[]'")
                return
            self.pattern = p
            popup.destroy()


        tk.Button(popup, text="OK", command=submit_pattern).pack(pady=5)
        pattern_entry.bind('<Return>', submit_pattern)
        self.master.wait_window(popup)
```

```
def show_image(self):
    # Display current image or quit if done
    if self.current_index >= len(self.image_files):
        messagebox.showinfo("Done", "No more images.")
        self.master.quit()
        return

    image_path = os.path.join(self.directory, self.image_files[self.current_index])
    self.original_img = Image.open(image_path)

    screen_w = self.master.winfo_screenwidth()
    screen_h = self.master.winfo_screenheight()

    img_ratio = self.original_img.width / self.original_img.height
    screen_ratio = screen_w / screen_h

    if img_ratio > screen_ratio:
        new_w = screen_w
        new_h = int(new_w / img_ratio)
    else:
        new_h = screen_h
        new_w = int(new_h * img_ratio)
    if self.mode == "separator":
        new_w = int(new_w * 0.8)
        new_h = int(new_h * 0.8)
    # Resize image for display if needed
    if self.original_img.width > new_w or self.original_img.height > new_h:
        self.display_img = self.original_img.resize((new_w, new_h), Image.Resampling.LANCZOS)
    else:
        self.display_img = self.or
    self.tk_img = ImageTk.PhotoImage(self.display_img)
    self.master.title(f"Book Processor - {self.image_files[self.current_index]}")

    # Mode UI
    if self.mode == "free":
        self.show_mode_free()
    elif self.mode == "pattern":
        self.show_mode_pattern()
    else:
        self.show_mode_separator()
```

```
    def show_mode_free(self):
        # Free Rename mode
        if self.canvas:
            self.canvas.destroy()
            self.canvas = None
        if self.top_frame:
            self.top_frame.pack_forget()
        if self.bottom_frame:
            self.bottom_frame.pack_forget()


        if not self.image_label:
            self.image_label = tk.Label(self.master, bg="black")
            self.image_label.pack(expand=True, fill=tk.BOTH)


        self.image_label.config(image=self.tk_img)
        self.show_rename_popup_free()


    def show_mode_pattern(self):
        # Pattern Rename mode
        if self.canvas:
            self.canvas.destroy()
            self.canvas = None
        if self.top_frame:
            self.top_frame.pack_forget()
        if self.bottom_frame:
            self.bottom_frame.pack_forget()


        if not self.image_label:
            self.image_label = tk.Label(self.master, bg="black")
            self.image_label.pack(expand=True, fill=tk.BOTH)


        self.image_label.config(image=self.tk_img)
        self.show_rename_popup_pattern()
```

```
    def show_mode_separator(self):
        # Page Separator mode
        if self.image_label:
            self.image_label.destroy()
            self.image_label = None
         if not self.separator_interface_created:
            self.top_frame = tk.Frame(self.master)
            self.top_frame.pack(side=tk.TOP, fill=tk.X, pady=5)
            self.bottom_frame = tk.Frame(self.master)
            self.bottom_frame.pack(side=tk.BOTTOM, fill=tk.X, pady=5)
            self.create_separator_controls()
            self.separator_interface_created = True
        else:
            self.top_frame.pack(side=tk.TOP, fill=tk.X, pady=5)
            self.bottom_frame.pack(side=tk.BOTTOM, fill=tk.X, pady=5)
        if self.canvas:
            self.canvas.destroy()
        self.canvas = tk.Canvas(self.master, bg="black", highlightthickness=0,
                                width=self.display_img.width, height=self.display_img.height)
        self.canvas.pack(expand=True, fill=tk.BOTH)
        self.canvas_img = self.canvas.create_image(
            self.display_img.width // 2, self.display_img.height // 2, image=self.tk_img,
anchor='center'
        )
        self.canvas.config(scrollregion=(0, 0, self.display_img.width, self.display_img.height))
        self.top_line_x = self.display_img.width // 2
        self.bottom_line_x = self.display_img.width // 2
        self.draw_line()
        self.scale_top.config(to=self.display_img.width)
        self.scale_bottom.config(to=self.display_img.width)
        self.scale_top.set(self.top_line_x)
        self.scale_bottom.set(self.bottom_line_x)
        self.master.bind('<Tab>', lambda e: self.skip())
        # 'a'/'d' for top line
        self.master.bind('<a>', lambda e: self.move_slider(self.scale_top, -1))
        self.master.bind('<d>', lambda e: self.move_slider(self.scale_top, 1))
        # Left/Right for bottom line
        self.master.bind('<Left>', lambda e: self.move_slider(self.scale_bottom, -1))
        self.master.bind('<Right>', lambda e: self.move_slider(self.scale_bottom, 1))
```

```
def create_separator_controls(self):
    #Create sliders and controls in third mode.
    self.scale_top = tk.Scale(
        self.top_frame, from_=0, to=100, orient=tk.HORIZONTAL,
        command=self.update_line_position_top, label="Top Line X"
    )
    self.scale_top.set(self.top_line_x)
    self.scale_top.pack(fill=tk.X)

    self.scale_bottom = tk.Scale(
        self.bottom_frame, from_=0, to=100, orient=tk.HORIZONTAL,
        command=self.update_line_position_bottom, label="Bottom Line X"
    )
    self.scale_bottom.set(self.bottom_line_x)
    self.scale_bottom.pack(fill=tk.X)

    info_frame = tk.Frame(self.bottom_frame)
    info_frame.pack(pady=5)
    tk.Label(info_frame, text="Press 'Enter' to split, 'Tab' to skip, 'Esc' to quit.").pack()

    button_frame = tk.Frame(self.bottom_frame)
    button_frame.pack(pady=5)

    skip_btn = tk.Button(button_frame, text="Skip (Tab)", command=self.skip)
    skip_btn.pack(side=tk.LEFT, padx=5)

    quit_btn = tk.Button(button_frame, text="Quit (Esc)", command=self.quit_app)
    quit_btn.pack(side=tk.LEFT, padx=5)
    small_label = tk.Label(info_frame, text="a/d: top slider | ←/→: bottom slider",
                           font=('TkDefaultFont', 8))
    small_label.pack(side=tk.RIGHT, padx=10)
    self.master.bind('<Return>', self.split_image)
def move_slider(self, scale, delta):
    #Move slider with limiting.
    val = scale.get() + delta
    width = self.display_img.width
    if val < 0:
        val = 0
    if val > width:
        val = width
    scale.set(val)
```

```
    def draw_line(self):
        #Draws the cut line.
        if self.line_id:
            self.canvas.delete(self.line_id)
        self.line_id = self.canvas.create_line(
            self.top_line_x, 0,
            self.bottom_line_x, self.display_img.height,
            fill="red", width=2
        )


    def show_rename_popup_free(self):
        #Rename popup for Free mode.
        popup = tk.Toplevel(self.master)
        popup.title("Free Rename")
        self.bind_escape_to_quit(popup)


        # Center the popup
        popup_width = 400
        popup_height = 200
        screen_w = popup.winfo_screenwidth()
        screen_h = popup.winfo_screenheight()
        x = (screen_w // 2) - (popup_width // 2)
        y = (screen_h // 2) - (popup_height // 2)
        popup.geometry(f"{popup_width}x{popup_height}+{x}+{y}")


        popup.transient(self.master)
        popup.grab_set()
        popup.focus_force()
        popup.lift()


        current_name = self.image_files[self.current_index]
        total_images = len(self.image_files)
        current_num = self.current_index + 1


        tk.Label(popup, text=f"Current name: {current_name}
({current_num}/{total_images})").pack(pady=(10, 5))
        tk.Label(popup, text="Enter new name (without extension):").pack(pady=5)


        name_entry = tk.Entry(popup, width=40)
        name_entry.pack(pady=5)
        name_entry.focus_set()
```

```
        def delayed_focus():
            popup.focus_force()
            popup.lift()
            name_entry.focus_set()
        popup.after(100, delayed_focus)

        def rename_and_next(event=None):
            new_name = name_entry.get().strip()
            if new_name:
                if any(ch in new_name for ch in self.forbidden_chars):
                    messagebox.showerror("Error", f"Filename contains forbidden chars:
{self.forbidden_chars}")
                    return
                old_path = os.path.join(self.directory, self.image_files[self.current_index])
                root, ext = os.path.splitext(self.image_files[self.current_index])
                new_path = os.path.join(self.directory, new_name + ext)
                if os.path.exists(new_path):
                    messagebox.showerror("Error", f"File '{new_path}' already exists.")
                    return
                os.rename(old_path, new_path)
                self.image_files[self.current_index] = new_name + ext
            self.current_index += 1
            popup.destroy()
            self.show_image()

        def quit_app():
            popup.destroy()
            self.quit_app()
        button_frame = tk.Frame(popup)
        button_frame.pack(pady=10)
        rename_btn = tk.Button(button_frame, text="Rename & Next (Enter)", command=rename_and_next)
        rename_btn.pack(side=tk.LEFT, padx=5)
        skip_btn = tk.Button(button_frame, text="Skip (Tab)", command=self.skip)
        skip_btn.pack(side=tk.LEFT, padx=5)
        quit_btn = tk.Button(button_frame, text="Quit (Esc)", command=quit_app)
        quit_btn.pack(side=tk.LEFT, padx=5)
        name_entry.bind('<Return>', rename_and_next)
        popup.bind('<Tab>', lambda e: self.skip())
        self.master.wait_window(popup)
```

```
    def show_rename_popup_pattern(self):
        #Pattern rename popup.
        popup = tk.Toplevel(self.master)
        popup.title("Pattern Rename")
        self.bind_escape_to_quit(popup)
        popup_width = 500
        popup_height = 300
        screen_w = popup.winfo_screenwidth()
        screen_h = popup.winfo_screenheight()
        x = (screen_w // 2) - (popup_width // 2)
        y = (screen_h // 2) - (popup_height // 2)
        popup.geometry(f"{popup_width}x{popup_height}+{x}+{y}")
        popup.transient(self.master)
        popup.grab_set()
        popup.focus_force()
        popup.lift()

        current_name = self.image_files[self.current_index]
        total_images = len(self.image_files)
        current_num = self.current_index + 1

        tk.Label(popup, text=f"Current name: {current_name}
({current_num}/{total_images})").pack(pady=(10, 5))
        parts = self.pattern.split('[]')
        placeholders_count = len(parts) - 1

        pattern_display_frame = tk.Frame(popup)
        pattern_display_frame.pack(pady=10)
        entries = []

        for i in range(placeholders_count):
            if parts[i]:
                tk.Label(pattern_display_frame, text=parts[i]).pack(side=tk.LEFT)
            e = tk.Entry(pattern_display_frame, width=10)
            e.pack(side=tk.LEFT)
            entries.append(e)
        if parts[-1]:
            tk.Label(pattern_display_frame, text=parts[-1]).pack(side=tk.LEFT)
```

```
        def delayed_focus():
            popup.focus_force()
            popup.lift()
            if entries:
                entries[0].focus_set()
        popup.after(100, delayed_focus)


        def rename_and_next():
            filled_parts = []
            for e in entries:
                val = e.get().strip()
                if any(ch in val for ch in self.forbidden_chars):
                    messagebox.showerror("Error", f"Filename contains forbidden chars:
{self.forbidden_chars}")
                    return
                filled_parts.append(val)
            new_name = ""
            for i in range(placeholders_count):
                new_name += parts[i] + filled_parts[i]
            new_name += parts[-1]
            new_name = new_name.strip()
              if not new_name:
                self.current_index += 1
                popup.destroy()
                self.show_image()
                return
            old_path = os.path.join(self.directory, self.image_files[self.current_index])
            root, ext = os.path.splitext(self.image_files[self.current_index])
            new_path = os.path.join(self.directory, new_name + ext)


            if os.path.exists(new_path):
                messagebox.showerror("Error", f"File '{new_path}' already exists.")
                return
            os.rename(old_path, new_path)
            self.image_files[self.current_index] = new_name + ext


            self.current_index += 1
            popup.destroy()
            self.show_image()
```

```
        def quit_app():
            popup.destroy()
            self.quit_app()
        button_frame = tk.Frame(popup)
        button_frame.pack(pady=10)

        rename_btn = tk.Button(button_frame, text="Rename & Next (Enter)", command=rename_and_next)
        rename_btn.pack(side=tk.LEFT, padx=5)

        skip_btn = tk.Button(button_frame, text="Skip (Tab)", command=self.skip)
        skip_btn.pack(side=tk.LEFT, padx=5)

        quit_btn = tk.Button(button_frame, text="Quit (Esc)", command=quit_app)
        quit_btn.pack(side=tk.LEFT, padx=5)

        def on_entry_return(i, event=None):
            # Move to next placeholder or confirm if last
            if i < len(entries) - 1:
                entries[i + 1].focus_set()
            else:
                rename_and_next()
        for i, e in enumerate(entries):
            e.bind('<Return>', lambda event, idx=i: on_entry_return(idx, event))
        popup.bind('<Tab>', lambda e: self.skip())
        self.master.wait_window(popup)

    def update_line_position_top(self, val):
        #Top line updater
        val = int(float(val))
        self.top_line_x = val
        self.draw_line()

    def update_line_position_bottom(self, val):
        #Bottom line updater
        val = int(float(val))
        self.bottom_line_x = val
        self.draw_line()
```

```
    def split_image(self, event=None):
        #Image split function
        scale_x = self.original_img.width / self.display_img.width
        top_x_original = int(self.top_line_x * scale_x)
        bottom_x_original = int(self.bottom_line_x * scale_x)
        width = self.original_img.width
        height = self.original_img.height
        left_mask = Image.new('L', (width, height), 0)
        right_mask = Image.new('L', (width, height), 0)
        draw_left = ImageDraw.Draw(left_mask)
        draw_right = ImageDraw.Draw(right_mask)

        left_polygon = [(0, 0), (top_x_original, 0), (bottom_x_original, height), (0, height)]
        right_polygon = [(top_x_original, 0), (width, 0), (width, height), (bottom_x_original,
height)]
        draw_left.polygon(left_polygon, fill=255)
        draw_right.polygon(right_polygon, fill=255)
        left_part = Image.new('RGBA', (width, height))
        right_part = Image.new('RGBA', (width, height))

        left_part.paste(self.original_img, (0, 0), left_mask)
        right_part.paste(self.original_img, (0, 0), right_mask)

        left_part = left_part.convert('RGB')
        right_part = right_part.convert('RGB')

        current_name = self.image_files[self.current_index]
        root, ext = os.path.splitext(current_name)
        left_name = f"{root}_part1{ext}"
        right_name = f"{root}_part2{ext}"

        left_path = os.path.join(self.directory, left_name)
        right_path = os.path.join(self.directory, right_name)

        left_part.save(left_path)
        right_part.save(right_path)
        self.current_index += 1
        self.show_image()
```

```
    def skip(self):
        #Skip function.
        self.current_index += 1
        self.show_image()

    def quit_app(self):
        #Quit Function
        self.master.quit()
        self.master.destroy()



if __name__ == "__main__":
    root = tk.Tk()
    app = ImageRenamer(root)
    root.mainloop()
```

APPENDIX B: Spread Sheet Screenshots

A.B.1. Pre-Elementary Spreadsheet Screenshot

Pre-elementary test was conducted on open source and market leader LLM models to evaluate their Turkish capabilities. Each model is represented as a row where every question (10) is represented as a column. Evaluation was done on a scale 2 to –1 where;
2: Approved: Color code: Green.
1: Has potential: Color code: Blue.
0: Had slight potential yet eliminated: Color code: Orange.
-1: Eliminated: Color code: Red.

Complete Pre-Elementary Spreadsheet can be view via below link:
https://1drv.ms/x/c/af58545cdbe6880c/EcNAIePDZwZBuXFAHQ3YGL4BGXGYdf8upVF3H_eT46EelQ?e=TgBaPc&nav=MTVfezAwMDAwMDAwLTAwMDEtMDAwMC0wMDAwLTAwMDAwMDAwMDAwMH0

Figure A.B.1.1. *Pre-Elementary Spreadsheet Screenshot part 1*

Figure A.B.1.2. *Pre-Elementary Spreadsheet Screenshot part 2*

A.B.2. RAG Test Spreadsheet Screenshot

RAG test was conducted on market leader LLM models on most popular Cloud AI platforms such as AWS Bedrock, Google Vertex, OpenAI, Azure OpenAI and also on Self-Developed pipelines and a Local running opensource Gemma2 model for comparison of their abilities. The local model featured servers as proof of concept that local solutions will be able to compete with cloud models in time with better hardware, eliminating problems that rise by using cloud resources.

Evalutation was done over 50 questions where Correct answers (Color code Green) is 2 points, Neutral answers is 1 point (Color code Yellow), Wrong answers are 0 points (Color code Red). This puts maximum points at 100.

Complete RAG Test Spreadsheet can be view via below link:

https://1drv.ms/x/c/af58545cdbe6880c/EbhoX36sGg9Bp_4Fb1A0UbIBNFbnVHnk5dHPX4dhT0pDuQ?e=X5FjBP

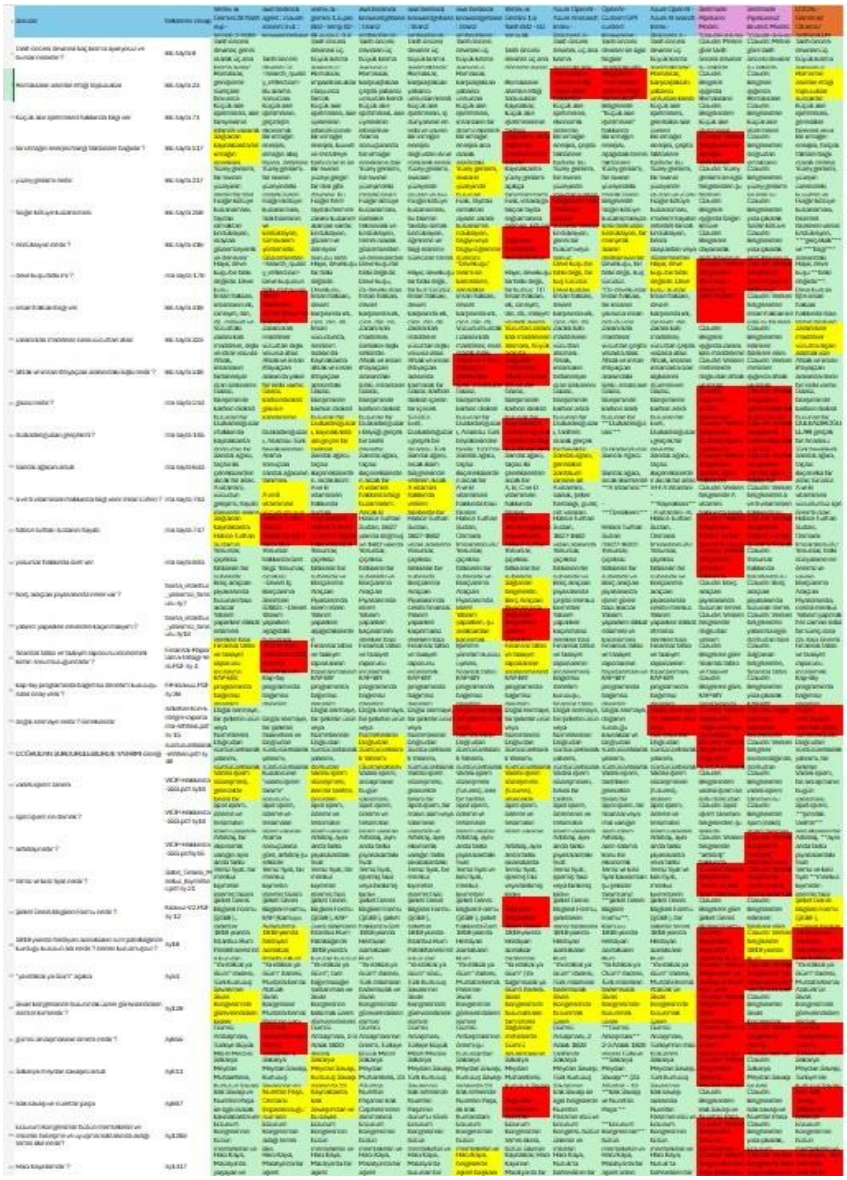

Figure A.B.2.1. *Rag Test Spreadsheet Screenshot part 1*

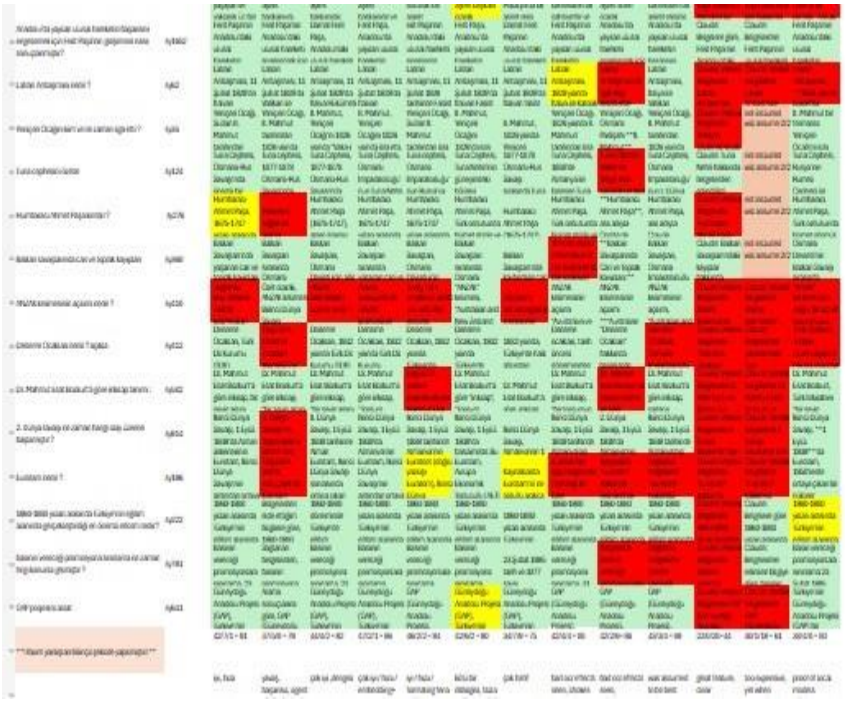

Figure A.B.2.2. *Rag Test Spreadsheet Screenshot part 2*