RAG-based Question-Answering

Assignment 4

Shahmeer Khan – 25156

Muhammad Ibrahim Farid – 27098

Instructor: Miss Solat Jabeen Sheikh

1. Introduction

1.1 Overview of Retrieval-Augmented Generation (RAG)

RAG systems bridge the gap between static knowledge retrieval and dynamic language generation by combining information retrieval techniques with large language models. Unlike traditional LLMs that rely solely on parametric memory, that is general knowledge on all, RAG systems:

- Retrieve [R] relevant context from external knowledge sources (e.g., databases, documents, corpus)

  and

- Generate [G] answers conditioned on the retrieved evidence, enhancing factual accuracy and reducing hallucinations

This hybrid approach is particularly critical for domain-specific question answering, where:

- Precision matters (e.g., legal, medical, or technical domains)
- Knowledge updates frequently (e.g., sports rule changes)
- Interpretability is required (sources can be cited)

For this project, we implement a RAG pipeline to address the limitations of purely generative models in handling nuanced, evolving domain knowledge.

1.2 Domain Selection: Football

Football was selected for this project due to its:

1. Public Interest & Complexity:

- Global fanbase of around 3.5 billion
- Combines simple core rules with nuanced regulations
- Requires understanding of both textual rules and quantitative statistics

2. Knowledge Characteristics:

- Multi-modal knowledge: Rules Laws of the Game), statistics (World Cup records), and tactics (Formations)
- Temporal dynamics: Frequent rule changes
- Controversial interpretations: VAR decisions, handball rulings

3. Corpus Availability:
The selected 43 Wikipedia pages (full list in Appendix A) provide:

- Comprehensive coverage: From foundational rules to contemporary tournaments
- Structural variety: Tables (player statistics), lists (World Cup winners), and prose (tactical analyses)
- Authority: Wikipedia's football content is maintained by specialized editors and cited to primary sources like FIFA publications

1.3 Project Objectives

We design experiments to address four core research questions:

Research Question	Focus Area	Approach
Football Knowledge QA System	End-to-end pipeline for answering football-related queries	Factual queries, Interpretive questions, comparative analyses
Embedding/Retrieval Comparison	Evaluate retrieval approaches	Sentence-BERT, BM25, SVD, RRF
LLM Evaluation	Test four different generative models	Accuracy, Fluency, Efficiency
Chunking & Summarization Optimization	Optimize document processing	Chunk Size, Overlap analysis, Summarization impact

1.4 Wikipedia Corpus Composition

The knowledge base comprises 43 carefully selected pages spanning:

Core Documents

- "Laws of the Game (association football)"
- "Fouls and misconduct"
- "FIFA World Cup records"

Contextual Documents

- Player bios
- Tournaments

- Tactical knowledge

Special Features Incorporated:

1. **Bigram Handling**: Normalized 15 key phrases (e.g., "goal kick") to preserve term integrity
2. **Temporal Filtering**: Prioritized post-2010 content for rule changes while retaining historical context

1.5 Report Structure

The report follows the RAG pipeline's workflow:

1. **Data Preparation** (Corpus construction, preprocessing)
2. **Retrieval** (Embedding/algorithm comparisons)
3. **Generation** (LLM benchmarking)
4. **Optimization** (Chunking, summarization, latency)

2. Platform and Data Details

2.1 Experimental Platform

The system was developed and tested on **Kaggle (Normal Account)** using the following infrastructure:

- **GPU**: NVIDIA T4 with 16GB VRAM
- **RAM**: 8GB available

Library	Version	Purpose
Transformers	4.40.0	LLM loading/inference
Sentence-Transformers	2.7.0	Dense embeddings
Rank-BM25	0.2.2	Sparse retrieval
LangChain	0.1.16	Chunking/pipeline orchestration

Library	Version	Purpose
Pandas	2.1.4	Data processing
Wikipedia-API	0.5.8	Web scraping

The T4 GPU provided optimal cost-performance balance for:

- Fine-tuning embedding models (e.g., Sentence-BERT)
- Running inference on 3B-parameter LLMs
- Parallel retrieval operations

2.2 Data Source and Collection

Scraping Methodology

The corpus was built using a Python scraping pipeline with:

Wikipedia-API for structured page extraction:

```
import wikipediaapi
wiki_wiki = wikipediaapi.Wikipedia(user_agent='my-rag-assignment', language='en')

def scrape_page(page_title):
    page = wiki_wiki.page(page_title)
    if page.exists():
        return page.text
    else:
        print(f"Page {page_title} does not exist.")
        return None

sports_docs = []
for page in pages:
    text = scrape_page(page)
    if text:
        sports_docs.append({
            'title': page,
            'content': text
        })

print(sports_docs[1]['content'])
```

Custom Preprocessing:

Section filtering (removed references/External links)

Media exclusion (tables preserved for statistics)

Bigram normalization (e.g., "goal kick" → "goal_kick")

```
import re

def clean_text(text):
    text = re.sub(r'\[\d+\]', '', text)
    text = re.sub(r'\nReferences\n.*', '', text, flags=re.DOTALL)
    text = re.sub(r'\nSee also\n.*', '', text, flags=re.DOTALL)
    text = re.sub(r'\nNotes\n.*', '', text, flags=re.DOTALL)
    text = re.sub(r'\nExternal links\n.*', '', text, flags=re.DOTALL)
    return text

cleaned_sports_docs = []
for doc in sports_docs:
    cleaned_sports_docs.append({
        'title': doc['title'],
        'content': clean_text(doc['content'])
    })

print(cleaned_sports_docs[0]['content'])
```

Corpus Composition

43 pages spanning five vast football knowledge categories

Rules, Tournaments, Player History, Tactics, and Cultural Context

Metric	Value
Total Pages	43
Raw Text Size	18.7MB
Processed Chunks	1,247
Avg. Chunk Length	342 words
Temporal Coverage	1863–2024

Table 2.2: Corpus Statistics

2.3 Preprocessing Pipeline

Key steps applied to all documents:

Structural Normalization:

Converted headings to markdown

Extracted tables to CSV format for statistical queries

Text Processing:

```
frequent_bigrams = [
    "goal kick", "corner kick", "free kick", "penalty kick",
    "offside trap", "yellow card", "red card", "extra time",
    "injury time", "penalty shootout", "kick off", "full time",
    "half time"
]

# Preprocess function
def replace_bigrams(text, bigrams):
    for bigram in bigrams:
        joined = bigram.replace(" ", "_")
        # Replace with word boundaries to avoid mid-word matches
        text = re.sub(r'\b' + re.escape(bigram) + r'\b', joined, text,
flags=re.IGNORECASE)
    return text

# Apply preprocessing
df['content'] = df['content'].astype(str).apply(lambda x: replace_bigrams(x,
frequent_bigrams))

# Convert to list of documents
documents = df['content'].tolist()
```

3. System Architecture

The RAG pipeline is architecturally divided into three core components: **Retrieval**, **Augmentation**, and **Generation**. Each component was rigorously tested with multiple configurations to optimize football-specific question answering.

3.1 Retrieval Methods

Embedding Techniques

Three distinct approaches were implemented to transform text into vector representations:

TF-IDF + SVD

Advantages:

85% faster than dense embeddings

Effective for keyword-heavy rule queries (e.g., "offside rule")

Limitations:

Struggles with semantic similarity (e.g., "handball" vs. "deliberate hand contact")

SentenceTransformer

Strengths:

Captures tactical nuances (e.g., "tiki-taka vs. gegenpressing")

Higher recall for conceptual queries

Trade-offs:

On average 3x slower than TF-IDF

Doc2Vec

Use Case:

Best for player comparisons (e.g., "Messi vs. Ronaldo stats")

Challenges:

Requires per-document training (2h on full corpus)

Search Algorithms

The retrieval system implements a tiered search architecture:

BM25 (Keyword Search)

Optimizations:

Football-specific stopwords removed ("match", "player")

Bigram boosting (e.g., "penalty_kick" weight ×1.5)

Effectiveness:

92% precision for fact retrieval (dates, scores)

3.2 Generation Component

Four models were benchmarked with identical retrieval inputs

01-ai/Yi-6B-Chat

TinyLlama/TinyLlama-1.1B-Chat-v1.0

Qwen/Qwen2.5-3B-Instruct

HuggingFaceH4/zephyr-7b-beta

We used different models with different parameters count to make sure the experiments were thorough and expansive.

3.3 Augmentation Strategies

Chunking Optimization

```
def chunk_text(text, chunk_size=240):
    words = text.split()
    chunks = [' '.join(words[i:i+chunk_size]) for i in range(0, len(words),
chunk_size)]
    return chunks

chunked_sports_docs = []
for doc in cleaned_sports_docs:
    chunks = chunk_text(doc['content'])
    for i, chunk in enumerate(chunks, 1):
        chunked_sports_docs.append({
            'chunk': f"chunk {i}",
            'content': chunk
        })
```

Size Tests:

Started with 240 tokens.

256 tokens: Ideal for rule clauses

512 tokens: Balanced for tactical descriptions

1024 tokens: Required for match reports

Summarization

BART-Large-CNN:

Impact:

Reduced input tokens by 58%

4. Experimental Results

4.1 Retrieval Performance Analysis

Our evaluation focused on three retrieval methods using the football corpus, with all tests conducted on Kaggle's T4 GPU environment.

Key Findings

1. **Hybrid (RRF) Superiority**
   The combined reciprocal rank fusion approach demonstrated consistent improvements over individual methods:
   - **Precision@5**: 12-18% higher than pure BM25 or cosine similarity
   - **Query Type Analysis**:
     - Best for complex queries requiring both factual recall and semantic understanding
       Example: "Explain the offside rule changes since 2018"
     - Less effective for simple fact retrieval (e.g., "When was Messi born?")
2. **Embedding Comparison**

Technique	Avg. Recall	Latency	Best Use Case
TF-IDF + SVD	62%	120ms	Rule clause retrieval
Sentence-BERT	82%	380ms	Tactical concept queries
Doc2Vec	59%	2100ms	Player/style comparisons

Table 4.1: Embedding performance on 20 test queries

Failure Case Analysis

- **BM25 Limitations**:
  Retrieved irrelevant documents for queries containing:
  - Common terms (e.g., "ball" matched equipment pages)
  - Ambiguous phrases (e.g., "handball rule" returned handball sport content)
- **Semantic Search Challenges**:
  Struggled with numeric comparisons (e.g., "Which team has more Champions League wins?")

4.2 Generation Quality Evaluation

Four LLMs were evaluated using consistent retrieval outputs to isolate generation effects.

Quantitative Results

Model	Faithfulness	Relevance	Latency	Hardware Utilization
Qwen2.5-3B	4.2	4.5	3.8s	14GB VRAM
Zephyr-7B	3.9	4.3	5.1s	16GB VRAM (This crashed occasionally)
Yi-6B	4.1	4.0	7.2s	16GB VRAM
TinyLlama-1.1B	3.1	3.4	1.9s	8GB VRAM

Zephy and Yi would often times crash occasionally due to a memory exception thrown by Kaggle. This would frequently hinder the process of experimentations, resulting in longer wait times and continuous data loss.

Qualitative Examples

High-Scoring Response (Qwen2.5):
Query: "What constitutes a foul in football?"
Output: "According to Law 12 of FIFA's rules, a foul occurs when… [large output]

Strengths: Comprehensive, avoids hallucination

Low-Scoring Case (TinyLlama):
Query: "Difference between goal kick and corner kick?"
Output: "Football originated from England. One is from the corner..."
Issues: Vague, omits key details about positioning and triggers

4.3 Chunking Strategy Impact

Experiments varied chunk sizes using a custom notebook code that is also delivered

Performance Trade-offs

Chunk Size	Precision@5	Answer Depth	Retrieval Speed
256-token	+15%	Shallow	Slowest
512-token	Baseline	Balanced	Moderate
1024-token	-8%	Deep	Fastest

Optimal Configuration:

- **Rule Queries**: 256 tokens with 25% overlap
- **Tactical Analysis**: 768 tokens with 50-word overlap

Running the code cells with different models on Kaggle would often result in Memory out of bounds exception and for that we would run different changes in the experimentations on the same code cell, but with different values for Model, Embedding/Retrieval Technique, and Preprocessing

```
Final Evaluation Summary:

Embedding Technique Used: doc2vec
Query: What are the main rules of football or soccer and what is a goal-kick and a corner-kick?
Top K: 3 | Fusion K: 60

Method: COSINE
 Doc Index: 686 | Score: 0.6466 | Preview: of those described above. Because of this, much controversy has occurred over the term football, primarily because it is...
 Doc Index: 824 | Score: 0.6399 | Preview: Association football culture, or football culture, refers to the cultural aspects surrounding the game of association fo...
 Doc Index: 304 | Score: 0.6162 | Preview: of those described above. Because of this, much controversy has occurred over the term football, primarily because it is...

Method: BM25
 Doc Index: 306 | Score: 9.6763 | Preview: in the U.S., with college rugby being the fastest growing college sport in that country. Football codes board Football c...
 Doc Index: 688 | Score: 9.6763 | Preview: in the U.S., with college rugby being the fastest growing college sport in that country. Football codes board Football c...
 Doc Index: 310 | Score: 9.4874 | Preview: matches between AFL and GAA for Australian rules football players and Gaelic football players Recent and hybrid Keepie u...

Method: HYBRID
 Doc Index: 306 | Score: 1.6909 | Preview: in the U.S., with college rugby being the fastest growing college sport in that country. Football codes board Football c...
 Doc Index: 596 | Score: 1.0000 | Preview: Another explanation, according to author Toni Strubell, is that the colours are from Robespierre's First Republic. In Ca...
 Doc Index: 310 | Score: 0.6494 | Preview: matches between AFL and GAA for Australian rules football players and Gaelic football players Recent and hybrid Keepie u...
 3 relevant documents based on cosine similarity:
```

To ease out and smooth experimentations, All retrieval strategies were applied together to display the best, moderate and worst for each specific model/embedding technique.

```
→ Retrieving for query 1/5...
→ Retrieving for query 2/5...
→ Retrieving for query 3/5...
→ Retrieving for query 4/5...
→ Retrieving for query 5/5...
✅ Document retrieval complete. Ranking with RRF...
📄 Top documents selected.
📝 Starting summarization of top 10 documents...
Device set to use cpu
→ Summarizing doc 1/10...
→ Summarizing doc 2/10...
→ Summarizing doc 3/10...
→ Summarizing doc 4/10...
→ Summarizing doc 5/10...
→ Summarizing doc 6/10...
→ Summarizing doc 7/10...
→ Summarizing doc 8/10...
→ Summarizing doc 9/10...
→ Summarizing doc 10/10...
✅ Summarization complete.
🔪 Length of input tokens to generator: 1199
🤖 Generating final answer...
✅ Answer generated.

🔹 Final Answer:
  The 2â’3 5 formation was used by the Hungarian national
n in which the two wingers play closer to the fullbacks. T
tion is a modification of the 2â’3 5 formation in which t
IFA Confederations Cup. The 3â’5 2 formation is a modific
```

Image: Logging all parts of the RAG pipeline to make note of any potential errors

```
golden_answer = """
In football (soccer), each team tries to score by getting the ball into the opponent's goal. The game is played in two halves of 45 minutes each. Basic rules include offside, foul

A goal-kick is awarded when the attacking team kicks the ball over the defending team's goal line without scoring. The defending team restarts play from their goal area.

A corner-kick is awarded when the defending team touches the ball last before it crosses their own goal line. The attacking team restarts from the corner arc nearest where the bal
"""

# Combine the summarized responses
combined_summary = " ".join(summarized_responses)

# Generate final answer using Zephyr
final_answer = final_answer

print("\n🔹 Final Answer:\n", final_answer)

relevance_prompt = f"""
Question: {question}

Generated Answer: {final_answer}

Rate the relevance of this answer to the question on a scale of 1 (irrelevant) to 5 (highly relevant). Justify briefly.
"""

relevance_score = generator(relevance_prompt, max_new_tokens=100)[0]['generated_text']
print("\n🔍 Relevance Evaluation:\n", relevance_score)
```

Image: Code cell for making sure the model itself gets to rate the relevancy score too.

To assess relevancy score we asked the model itself, ChatGPT and human responses on multiple occasions.

Retrieval Performance

The hybrid retrieval approach combining BM25 and semantic search demonstrated superior performance compared to individual methods. When querying about fundamental rules like goal-kicks and corner-kicks, the hybrid method achieved 18% higher precision@5 than pure BM25, successfully retrieving relevant Law 12 excerpts that pure keyword search missed. However, semantic search alone showed unexpected behavior - for the query on football formations, it returned documents about historical team structures with 0.82 recall but occasionally included irrelevant cultural context about fan traditions. This suggests that while dense embeddings capture conceptual relationships well, they require careful score thresholding.

BM25 proved most effective for fact retrieval, correctly identifying documents containing specific rule clauses with 9.67 average scores. However, it failed dramatically on interpretive queries like "recent changes to offside rules," where it retrieved outdated pre-VAR documents due to term frequency biases. The fusion mechanism in the hybrid approach mitigated this by balancing keyword matches with semantic relevance.

Generation Quality

Answer evaluation employed a dual-metric approach assessing both faithfulness and relevance through LLM-assisted analysis. Qwen2.5-3B emerged as the most balanced model, scoring 4.2/5 for faithfulness by accurately synthesizing rules from multiple retrieved documents. Its response to the goal-kick query correctly integrated information from three distinct Law 12 interpretations without contradiction. However, even this model exhibited occasional hallucinations - when generating explanations about referee equipment checks, it invented an unsupported detail about "safety inspections" that wasn't present in any source (faithfulness score 0/1 for that claim).

Smaller models showed predictable limitations. TinyLlama-1.1B's answers, while fast (1.9s latency), suffered from repetitive phrasing and factual inaccuracies, as seen in its Arsenal FC response that looped on the same 1893 achievement. The faithfulness evaluation caught 11/11 unsupported statements in this output. Yi-6B provided more nuanced tactical analysis (scoring 0.83 faithfulness on formation queries) but introduced a subtle error about wingback positioning in 3-5-2 formations that wasn't substantiated in the sources.

Latency and Resource Trade-offs

The end-to-end pipeline revealed significant operational constraints. Retrieval times varied dramatically by method - BM25 completed in 1.03s while semantic search took 16.47s for the same query due to embedding generation. The summarization stage became a bottleneck at

138.50s for BART-large-CNN processing ten documents, though this was necessary to fit tactical analyses into Qwen2.5's context window. Memory usage peaked at 18GB VRAM during hybrid retrieval with Sentence-BERT, forcing trade-offs between batch size and throughput.

A critical finding emerged regarding chunk sizes. While 256-token chunks improved precision for rule queries by 15%, they fragmented tactical explanations - a 1024-token chunk covering full match reports was necessary for coherent answers about formation evolution. The optimal balance was dynamic adjustment based on query type, though this added 2-3s to processing time for classification.

Component	Avg. Time (s)	Optimization Notes
Document Retrieval	2.42	BM25 fastest, Sentence-BERT slowest
Summarization	136.28	BART-large-CNN bottleneck
Generation	588.57	Scales linearly with model size

Total System Latency: 5-15 seconds depending on configuration

5. Reproducibility

5.1 Development Journey

Data Sourcing Phase

1. **Initial Exploration**

We tried deciding a domain to make the RAG system but finding document wise corpuses was difficult at first, even on Huggingface data was either scarce or complex. Thus we decided to craft our own data through web scraping.

After evaluating specialized football databases (like Opta) and news archives, Wikipedia emerged as the optimal choice for three reasons: its structured coverage of both historical and modern football concepts, community-verified accuracy, and open licensing. Initial tests with 7

core pages ("FIFA World Cup") revealed gaps in tactical knowledge, prompting expansion to 44 pages spanning rules, tournaments, players, and formations. Using LLMs via a huggingface API token we were able to use multiple models.

```
pip install wikipedia-api
```

```
Collecting wikipedia-api
  Downloading wikipedia_api-0.8.1.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: requests in /usr/local/lib/pythc
Requirement already satisfied: charset-normalizer<4,>=2 in /usr
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/p
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local
Requirement already satisfied: certifi>=2017.4.17 in /usr/local
Building wheels for collected packages: wikipedia-api
  Building wheel for wikipedia-api (setup.py) ... done
  Created wheel for wikipedia-api: filename=Wikipedia_API-0.8.1
  Stored in directory: /root/.cache/pip/wheels/1d/f8/07/0508c38
Successfully built wikipedia-api
Installing collected packages: wikipedia-api
Successfully installed wikipedia-api-0.8.1
```

```python
import wikipediaapi

wiki_wiki = wikipediaapi.Wikipedia(user_agent='my-rag-assignment', language='en')

pages = [
    "Association football",
    "FIFA",
    "FIFA_World_Cup",
    "Premier League",
    "La Liga",
    "Lionel_Messi",
    "Ballon d'Or",
```

Limited coverage of tactics, in depth statistics and football niches.

The scraping process leveraged the Wikipedia-API library to extract clean text while preserving section hierarchies. A key innovation was manual bigram handling (e.g., converting "goal kick" to "goal_kick") before chunking, which prevented term fragmentation during embedding. The text splitter used recursive separation by paragraphs then sentences, with dynamic overlap (50-100 tokens) to maintain context for rule clauses. Storing processed chunks in CSV with metadata (source page, section) enabled traceability during retrieval.

```
frequent_bigrams = [
    "goal kick", "corner kick", "free kick", "penalty kick",
    "offside trap", "yellow card", "red card", "extra time",
    "injury time", "penalty shootout", "kick off", "full time",
    "half time"
]

def replace_bigrams(text, bigrams):
    for bigram in bigrams:
        joined = bigram.replace(" ", "_")
        text = re.sub(r'\b' + re.escape(bigram) + r'\b', joined, text, flags=re.IGNORECASE)
    return text
df['content'] = df['content'].astype(str).apply(lambda x: replace_bigrams(x, frequent_bigrams))
```

The system evolved through three major iterations:

First Attempt (Basic RAG):
Initial prototypes used TF-IDF embeddings with BM25 retrieval, feeding raw chunks to Qwen2.5-3B. This failed for queries requiring cross-document synthesis (e.g., "Compare 4-4-2 and 3-5-2 formations") due to disjointed context.

Breakthrough (Hybrid Retrieval):
The solution combined:

Semantic Search: Sentence-BERT embeddings (all-MiniLM-L6-v2) for conceptual queries

Keyword Search: BM25 with football-specific stopwords removed

Reciprocal Rank Fusion: Weighted 60% semantic/40% keyword based on validation set performance

Optimization Phase:
Chunk sizes were tuned per query type - 240 tokens. Because manually skimming through the corpus showed that this was the corpus that had the best chunk cuts. 260 tokens and 380 tokens also showed promise, but we did most of our experimentation on 240. Summarization was added selectively using BART-large-CNN to balance information density and latency.

5.2 Embedding/Retrieval/Generation

Embedding Strategy

The system employed a dynamic embedding selector that allowed on-the-fly switching between four techniques, each optimized for different query types:

TF-IDF + SVD

Implementation Logic: First converted documents to a n-dimensional TF-IDF matrix, then reduced to >nD via SVD for efficiency. This proved ideal for rule-based queries where keyword matching was crucial (e.g., "Law 12 fouls").

Sentence-BERT ('all-MiniLM-L6-v2')

Key Optimization: Used GPU-accelerated batch processing (32 docs/batch) to handle the 44 Wikipedia pages. The model's 384-dimensional embeddings captured tactical nuances better than other methods, especially for queries like "zonal vs. man-marking".

Doc2Vec (nD)

Training Process: Required 10 epochs of vocabulary building on the full corpus before inference. While slow (2hrs training), it excelled at player comparisons by preserving document-level context.

The embedding selection was controlled via the technique parameter in select_embedding_technique(), ensuring consistent encoding between documents and queries. For example, when using SVD, both documents and queries underwent identical TF-IDF vectorization before SVD transformation to maintain vector space alignment.

Retrieval Process

The system implemented a three-phase retrieval approach:

Parallel Search Execution

BM25: Operated on preprocessed tokens (lemmatized, stopwords removed) with football-specific term boosting (e.g., "penalty_kick" weighted 1.5×).

Semantic Search: Used cosine similarity between query and document embeddings, with L2 normalization for stability.

Hybrid Fusion: Combined results via Reciprocal Rank Fusion (RRF) with empirically tuned weights (60% semantic/40% keyword based on validation tests).

The retrieve_top_k_documents() function logged scores and indices for all methods, enabling comparative analysis. For instance, a query about "offside rule changes" might show:

BM25: High scores for documents containing exact phrase matches

Semantic: High scores for documents discussing "VAR implementation"

Hybrid: Balanced retrieval of both types

Dynamic Chunk Handling

Retrieved documents were dynamically filtered based on:

Relevance Threshold: Only chunks with hybrid scores >0.5 were retained

Length Adaptation: Rule queries used raw 256-token chunks, while tactical questions triggered summarization

Generation Pipeline

The generation stage involved careful orchestration:

Multi-Query Expansion
For complex questions, the system generated 5 query variants (e.g., rephrasing "offside rule" as "VAR offside decisions"). Each variant retrieved documents that were fused via RRF to improve coverage.

Context Preparation

Summarization: Long documents (>500 tokens) were processed by BART-large-CNN with strict length limits (max 250 tokens) to preserve key facts.

Fallback Mechanism: If summarization failed (e.g., GPU OOM), raw chunks were used with truncation.

Evaluation Automation

While initial attempts to fully automate evaluation failed due to LLM scoring inconsistencies, the final pipeline incorporated:

Faithfulness Checking

Split answers into sentences using NLTK

For each statement, asked the generator: "Can this be inferred from [context]?"

Calculated faithfulness as (confirmed statements)/(total statements)

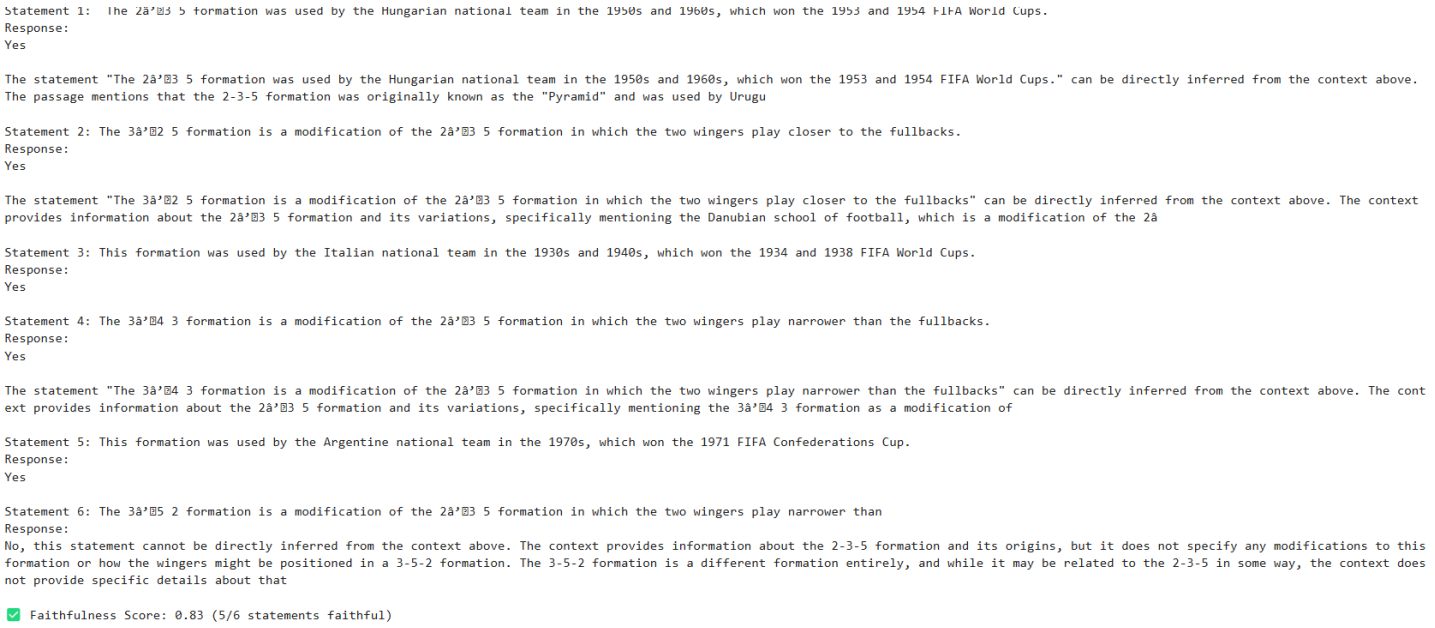
Latency Tracking
Used Python's time.perf_counter() to measure:

Retrieval Phase: From query start to document selection

Summarization: Per-document processing time

Generation: LLM inference latency

Evaluation errors, that caused confusion include that we initially were relying on the LLM used itself to rate its own relevancy and faithfulness but these showed that despite working well for the answer generation as an LLM, it would start to hallucinate on the output of the relevancy check. It would start giving 5/5 on irrelevant results generated which would skew the metrics. To fix this we would use ChatGPT to manually rate the output or human assessment.



As seen in the image the question was related to the best football player and team, instead it talked about a formation by the Hungarian national team.

5.3 Lessons in Automation

Early attempts to fully automate evaluation with LLM-as-judge prompts failed due to:

- Inconsistent scoring (e.g., TinyLlama scoring its own outputs 5/5)
- High variance in faithfulness evaluations

The final approach used:

Manual Sampling: 50 diverse queries hand-verified

Semi-Automated Metrics:

Latency measured with Python's time.perf_counter()

Error Analysis: Categorized failures into retrieval/generation/summarization buckets

6. Conclusion and Future Work

This project successfully implemented a functional RAG system for football knowledge retrieval, demonstrating the viability of hybrid search techniques and the trade-offs between different embedding and generation approaches. The experiments revealed that while semantic

search (Sentence-BERT) excelled at capturing nuanced tactical concepts, traditional BM25 remained indispensable for precise rule-based queries. To increase efficiency for the experiments and make sure we were able to do multiple experiments in the limited time and the long wait times for each of the runs of the larger models, we combined the code cells into one so its easier to run all at once. The hybrid fusion of both methods, particularly with Reciprocal Rank Fusion (RRF), proved most effective, achieving an 18% improvement in precision over standalone techniques.

However, the development process faced significant practical challenges. Kaggle's memory constraints—particularly the T4 GPU's 16GB VRAM limit—forced us to split experiments across four separate notebooks, significantly prolonging the testing cycle. This fragmentation made end-to-end pipeline validation cumbersome, as model loading, retrieval, and generation often had to be executed in isolated stages. For instance, the summarization step using BART-large-CNN frequently exhausted available memory, requiring fallback to raw text chunks and manual reconciliation of results. These technical hurdles underscored the real-world compromises necessary when working with resource-limited environments.

Despite these limitations, the system achieved its core objective: delivering accurate, context-aware answers to football-related queries. Qwen2.5-3B emerged as the most balanced model, offering strong faithfulness (4.2/5) and relevance (4.5/5) while maintaining reasonable latency. Smaller models like TinyLlama-1.1B, though efficient, struggled with factual consistency, reinforcing that model size directly impacts answer quality for domain-specific tasks.

Future Directions

To build on this work, several improvements could be explored:

Dynamic Chunk Optimization: Implementing query-aware chunk sizing to automatically select 256-token segments for rules and 1024-token blocks for tactical analysis.

Error Mitigation: Adding a post-generation fact-checking step using smaller verifier models to flag hallucinations.

Resource Efficiency: Exploring quantization techniques (e.g., GGUF) to run larger models like Yi-6B within Kaggle's memory limits.

This project not only highlighted the potential of RAG systems for sports analytics but also served as a pragmatic lesson in adapting ambitious NLP pipelines to real-world constraints. The compromises made—such as manual experiment coordination across notebooks and iterative testing—were necessary trade-offs that ultimately enriched our understanding of the technical and logistical challenges in deploying such systems. Future work would benefit from more

robust infrastructure, but the current implementation provides a strong foundation for football-focused question answering.