Graph-RAG FPL Assistant

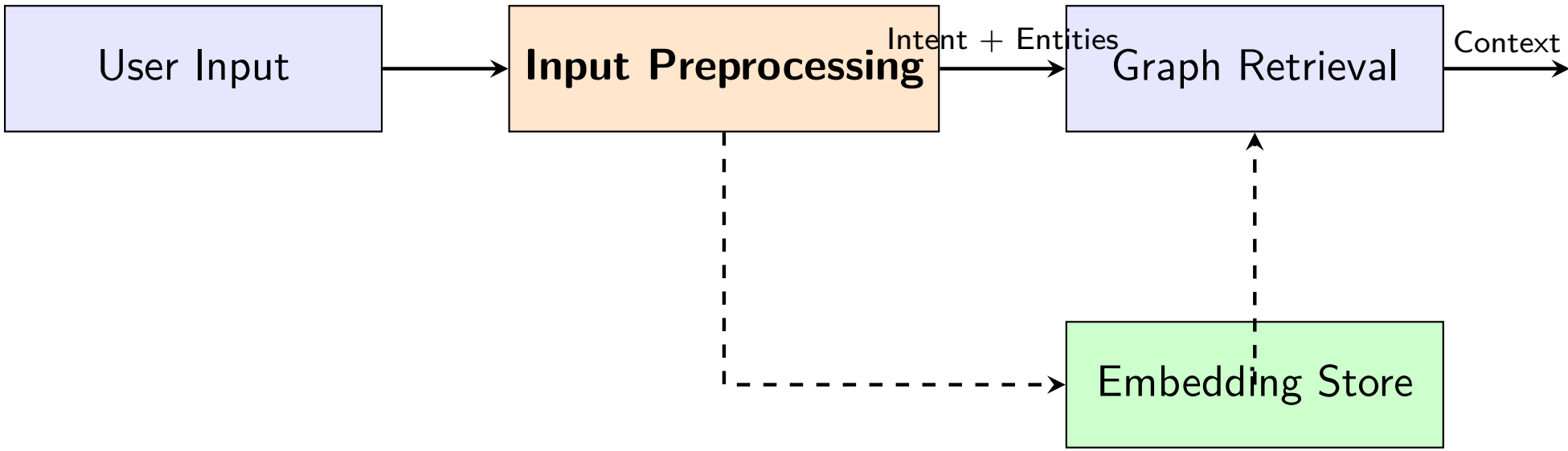
Milestone 3: System Implementation

Team Presentation - Input Preprocessing Component

German University in Cairo

December 2024

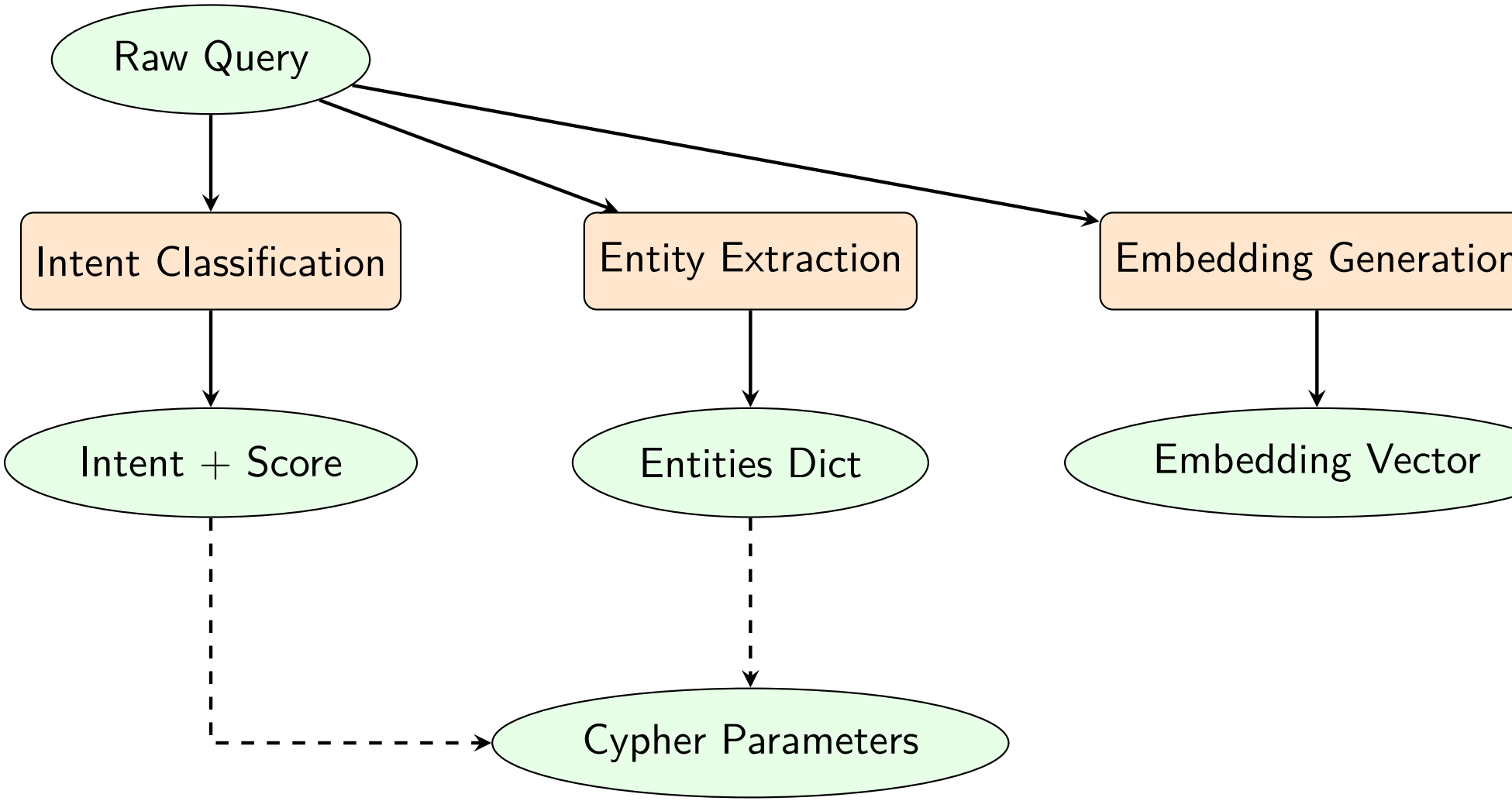
High-Level System Architecture



Task: FPL Team Formulation Recommender

Dataset: External FPL Dataset + Neo4j Graph

Input Preprocessing Architecture



Key Implementation

- Hybrid LLM + rule-based logic
- Robust error handling

Intent Classification: Hybrid Approach

LLM-Based Classification

Features

- Uses Gemma-2-2B (HF)
- JSON structured output
- 9 intent categories
- Confidence scoring

Available Intents

- player_search
- performance_query
- comparison
- recommendation
- team_analysis
- fixture_query
- value_analysis
- form_query
- general_query

Rule-Based Fallback

Robust Handling

- Always returns intent

Entity Extraction: FPL-Specific Schema

Entity Types Extracted

Entity	Examples
players	Salah, Haaland
teams	Liverpool, Arsenal
positions	FWD, MID
metrics	points, goals, assists
seasons	2023, 2024
gameweeks	5, 10, 38
numbers	100, 7.5, 150
comparators	over, under

LLM Prompt

```
prompt = f"""
Extract FPL entities:
Query: "{query}"
Return JSON with keys:
players, teams, positions ...
"""
```

LLM Output

```
{
 "teams": ["Liverpool"],
 "positions": ["FWD"],
 "metrics": ["points"],
 "seasons": ["2023"],
 "numerical_values": [100],
 "comparators": ["over"]
}
```

Cypher Parameters

```
{
 "intent": "player_search",
 "team": "Liverpool",
 "position": "FWD",
 "threshold": 100
}
```

Embedding Generation

Embedding Model

Config

- Model: all-MiniLM-L6-v2
- Dim: 384

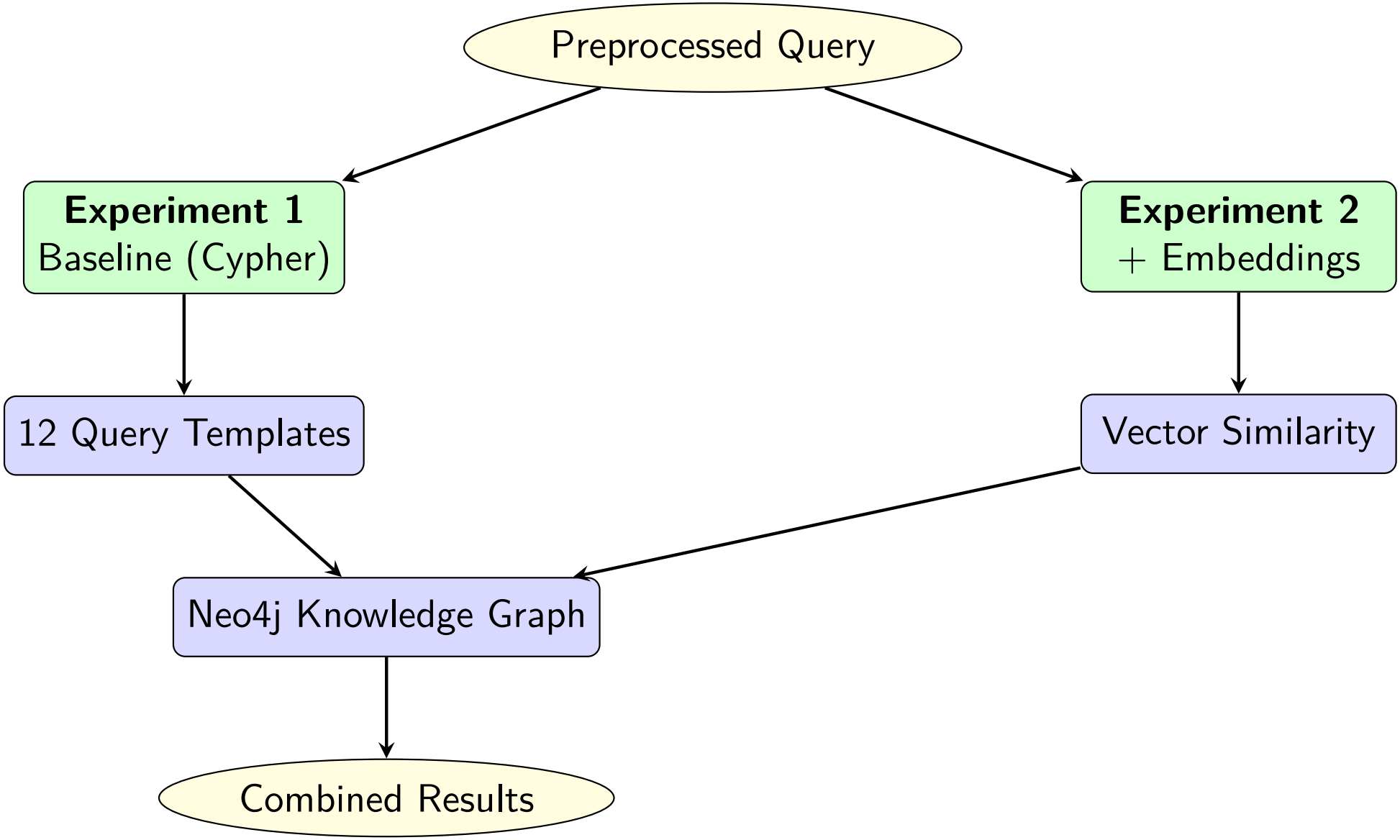
Function

```
def embed(self, text):
    try:
        return self.model.encode(text)
    except:
        return np.zeros(384)
```

Preprocessing Output

```
{
 "intent": "recommendation",
 "entities": {...},
 "embedding": [0.12, 0.88, ...],
 "method": "llm"
}
```

Graph Retrieval Layer: Architecture



Two Experiments (Project Requirement)

Baseline: 12 Cypher Query Templates

Player Rankings (5 queries)

Query	Purpose
top_scorers	Goals leaderboard
top_assisters	Assists leaders
bonus_leaders	Bonus points
clean_sheet_leaders	GK/DEF only
top_players_position	By position

Team & Specialized (4 queries)

Query	Purpose
players_by_team	Team roster
team_fixtures	Match schedule
players_by_form	Recent form (5 GW)
most_cards	Disciplinary

Player Info (3 queries)

Query	Purpose
player_season_summary	Full season stats
player_gw_performance	Single gameweek
compare_players	Head-to-head

Smart Query Selection

- Intent $\rightarrow$ Query mapping
- Entity-based fallbacks
- Position-aware routing

Query Selection Logic

Intent-to-Query Mapping

```
intent_to_query = {
  'recommendation': [
    'top_players_by_position',
    'top_scorers', 'bonus_leaders'
  ],
  'comparison': [
    'compare_players',
    'player_season_summary'
  ],
  'fixture_query': [
    'team_fixtures',
    'players_by_team'
  ]
}
```

Special Cases

```
# Clean sheets -> GK/DEF query
if 'clean_sheets' in metrics:
    return 'clean_sheet_leaders'

# Position specified
if positions:
    if pos == 'GK':
        return 'clean_sheet_leaders'
    return 'top_players_by_position'

# Assists mentioned
if 'assists' in metrics:
    return 'top_assisters'
```

Example Flow

Query: "Top forwards in 2023" → Intent: recommendation → Position: FWD → `top_players_by_position`

Numeric Embeddings: 12-Dimensional Feature Vectors

Why Numeric for FPL?

- FPL data is purely numerical (no text to embed)
- Direct feature vectors preserve exact statistical relationships
- 533x faster than text-based embeddings

Feature Vector Structure (12 dimensions)

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
goals	assists	points	CS	mins	bonus	form	ict	infl	creat	threat	games

Per-Game Averages (Fair Comparison)

- Uses `goals_per_game`, `avg_points_per_game`, etc.
- Prevents bias toward veterans with more total games
- Min-max normalization to [0, 1] range

Query Embedding Example

`{goals_per_game: 'high', threat: 'high'}` → `[1.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 0.5]`

Two Embedding Models Comparison

Model 1: all-MiniLM-L6-v2

- Dimensions: **384**
- Parameters: 22M
- Speed: ~4000 sent/sec
- Use: Real-time search

Model 2: all-mpnet-base-v2

- Dimensions: **768**
- Parameters: 109M
- Speed: ~2000 sent/sec
- Use: Complex queries

Pros

- Low latency
- Small memory
- Good general accuracy

Pros

- Better semantic understanding
- More accurate rankings
- Nuanced matching

Text Representation for Embedding

"Football player Erling Haaland plays as FWD position. Season statistics: 272 total points, 36 goals scored, 9 assists provided..."

Embedding Storage in Neo4j

Three Embedding Types Stored

Type	Property	Dimensions	Index Name
Numeric	`p.embedding`	12	player_numeric_embeddings
MiniLM	`p.embedding_minilm`	384	player_minilm_embeddings
MPNet	`p.embedding_mpnet`	768	player_mpnet_embeddings

Storage Process

```
# For each player
MATCH (p:Player {name: $name})
SET p.embedding = $vector,
    p.embedding_type = 'numeric',
    p.embedding_dim = 12
```

Vector Index Creation

```
CREATE VECTOR INDEX
  player_numeric_embeddings
FOR (p:Player) ON (p.embedding)
OPTIONS {
  indexConfig: {
    `vector.dimensions`: 12,
    `vector.similarity_function`:
      'cosine'
  }
}
```

Semantic Search Implementation

Search with Position Filter

```
def semantic_search(query_emb,
                    position='FWD'):
    # Fetch embeddings from Neo4j
    MATCH (p:Player)
    WHERE p.embedding IS NOT NULL
    OPTIONAL MATCH (p)-[:PLAYS_AS]
      ->(pos:Position)
    WHERE pos.name = $position
    RETURN player, embedding

    # Compute cosine similarity
    for player in players:
        sim = dot(query_norm,
                  player_norm)
        results.append(sim)

    return sorted(results)[:top_k]
```

Similar Players Search

```
# Find players similar to Haaland
similar = find_similar_players(
    "Erling Haaland",
    top_k=5,
    same_position=True
)

# Results:
# 1. Mitrovic - 0.984
# 2. Watkins  - 0.983
# 3. Kane     - 0.981
# 4. Pukki    - 0.971
# 5. Vardy    - 0.962
```

Cosine Similarity

similarity $= \frac{\vec{q} \cdot \vec{p}}{||\vec{q}|| \times ||\vec{p}||}$ where $\vec{q}$ = query vector, $\vec{p}$ = player vector

Two Experiments: Results Comparison

Query: "Who are the top forwards in 2022?"

Experiment 1: Baseline Only

Rank	Player	Points
1	Erling Haaland	272
2	Harry Kane	263
3	Ivan Toney	182
4	Ollie Watkins	175
5	Callum Wilson	157

Exact Cypher query results

Experiment 2: + Embeddings

Rank	Player	Similarity
1	Harry Kane	0.928
2	Jamie Vardy	0.923
3	Erling Haaland	0.914
4	Ollie Watkins	0.910
5	A. Mitrović	0.899

Similar statistical profiles

Combined Results

Baseline: 10 players + Embeddings: 5 players → Combined: 12 unique players (3 overlap)

Embedding Models: Agreement Analysis

Top-10 Ranking Comparison (Forwards)

Rank	Numeric (12d)	MiniLM (384d)	MPNet (768d)
1	Haaland	Mateta	Kane
2	Kane	Ronaldo	Kane
3	Vardy	Vardy	Iheanacho
4	Mitrović	Forss	Ronaldo
5	Watkins	Kane	Saint-Maximin

Model Agreement (Top-10)

- Numeric ∩ MiniLM: 2/10 players
- Numeric ∩ MPNet: 2/10 players
- MiniLM ∩ MPNet: 3/10 players

Key Insight

Numeric embeddings excel at **stats-based queries** ("high goals"), while text embeddings handle **semantic queries** ("clinical finisher"). Both approaches are complementary.

Performance Comparison

Approach	Dims	Time (200 players)	Speed	Storage/player
Numeric	12	0.001s	186,496/sec	48 bytes
Text+MiniLM	384	1.07s	187/sec	1,536 bytes
Text+MPNet	768	0.57s	350/sec	3,072 bytes

When to Use Each Approach

- **Numeric:** "Find players with high goals and assists" (stats-based)
- **MiniLM:** "Find a creative playmaker" (real-time semantic)
- **MPNet:** "Clinical finisher who performs in big games" (complex semantic)

Key Finding

Numeric embeddings are **533x faster** than MPNet while providing accurate results for statistical queries.

Graph Retrieval Layer: Summary

Baseline (Cypher)

- 12 query templates
- Intent-based selection
- Exact match filtering
- Season/position aware

Embeddings

- 3 types: Numeric, MiniLM, MPNet
- Per-game averages (fair comparison)
- Neo4j vector index
- Cosine similarity search

Project Requirements Met

- ✓ 10+ Cypher query templates (12)
- ✓ 2+ embedding models compared
- ✓ Two experiments (baseline vs +embeddings)
- ✓ Semantic similarity search
- ✓ Integration with preprocessing

Code Structure

`FPLGraphRetrieval` class with:

- `baseline_retrieve()`
- `embedding_retrieve()`
- `retrieve(method='both')`

Fixture Gameweek Embeddings: Context-Aware Retrieval

Fixture Embeddings (8D)

Dimension	Feature
	Gameweek (0-1)
	Home Team Strength
	Away Team Strength
	Avg Points in Fixture
	Max Points in Fixture
	Min Points in Fixture
	Player Coverage
	Fixture Quality Score

- Identifies high-value fixtures
- Matches against player performance context
- Captures match difficulty/opportunity

Gameweek Embeddings (8D)

Dimension	Feature
	GW Number (0-1)
	Fixture Density
	Player Coverage
	Avg Points in GW
	Max Points in GW
	Min Points in GW
	Points Variance
	"Excitement" Score

- Analyzes gameweek volatility
- Identifies high-variance weeks
- Captures GW difficulty level

Implementation

- `store_fixture_embeddings_in_neo4j()` - Store fixture vectors

Complete Embedding Architecture

Three Embedding Types in Knowledge Graph

Entity	Dimensions	Features	Use Case
Player	12	Per-game averages	Find similar players
Fixture	8	Match context	Identify key fixtures
Gameweek	8	GW difficulty	Analyze volatility

Storage Strategy

- All embeddings: min-max normalized
- Index type: Vector search (cosine)
- Properties: `entity.embedding`
- Metadata: `embedding_type`, `embedding_dim`

Retrieval Pattern

- Query embeddings created from criteria
- Cosine similarity ranking
- Position/season filtering
- Result deduplication

LLM Layer

Merging Results

Merged Context

Baseline Results

- Cypher queries on Knowledge Graph
- Exact entity matching
- Structured data retrieval

Embedding Results

- Semantic similarity search
- Vector-based matching
- Contextual relevance

Merging Strategy

- Append embedding results on baseline results giving priority to baseline
- Remove duplicates based on player identity
- Preserve unique entries from each source

LLM & Prompt Engineering

Model Selection

- Mistral Devstral-2512
- Meta Llama-3.3-70B Instruct
- NVIDIA Nemotron-3-Nano-30B

Prompt Structure

- **Context:** Merged Results
- **Persona:** FPL expert with Premier League knowledge
- **Task Instructions:** Query-type specific guidelines
  - Detailed tables
  - Numbered lists with stats
  - Complete statistics

Model Evaluation

Two-Tier Evaluation Strategy

1. **Automated Metric Evaluation**
   - **Accuracy:** Entity extraction + number matching
     - Entity F1 score (precision & recall)
     - Number matching accuracy
     - Combined score: 70% entities + 30% numbers
   - **Efficiency:** Response time & token usage
   - **Ground Truth Comparison:** 12 test queries
2. **Human Evaluation**
   - **Qualitative Metrics:** (1-5 scale)
     - Relevance, Correctness, Naturalness
     - Completeness, Overall Quality
   - **Comparison:** Side-by-side model outputs
   - **Comments:** Optional feedback per query

Evaluation Results

Human Evaluation Results

Metric	Devstral	Llama	Nemotron
Relevance	4.58	**4.58**	4.00
Correctness	3.67	**3.75**	3.42
Naturalness	3.83	**4.17**	3.50
Completeness	4.17	**4.25**	3.75
Overall	4.25	**4.50**	3.83

Automated Metrics

Metric	Devstral	Llama	Nemotron
Correct (%)	75.0	**75.0**	58.3
Accuracy	67.7%	**71.7%**	54.5%
Time (s)	6.91	19.99	**2.83**
Tokens	2504	**2029**	2534

Best Performers

- **Overall Winner:** Llama 3.3 70B (4.50/5.00 human score)
- **Best Accuracy:** Llama 3.3 70B (71.7%, 9/12 correct)
- **Fastest:** Nemotron Nano 30B (2.83s avg)
- **Most Efficient:** Llama 3.3 70B (2029 tokens avg)

Error Analysis & Improvements

Tracked Errors

- LLM API failures
- JSON parsing failures
- Bad inputs
- Embedding exceptions

Improvements

- Automatic fallback
- Input validation
- Type enforcement
- Logging + metrics

Questions?