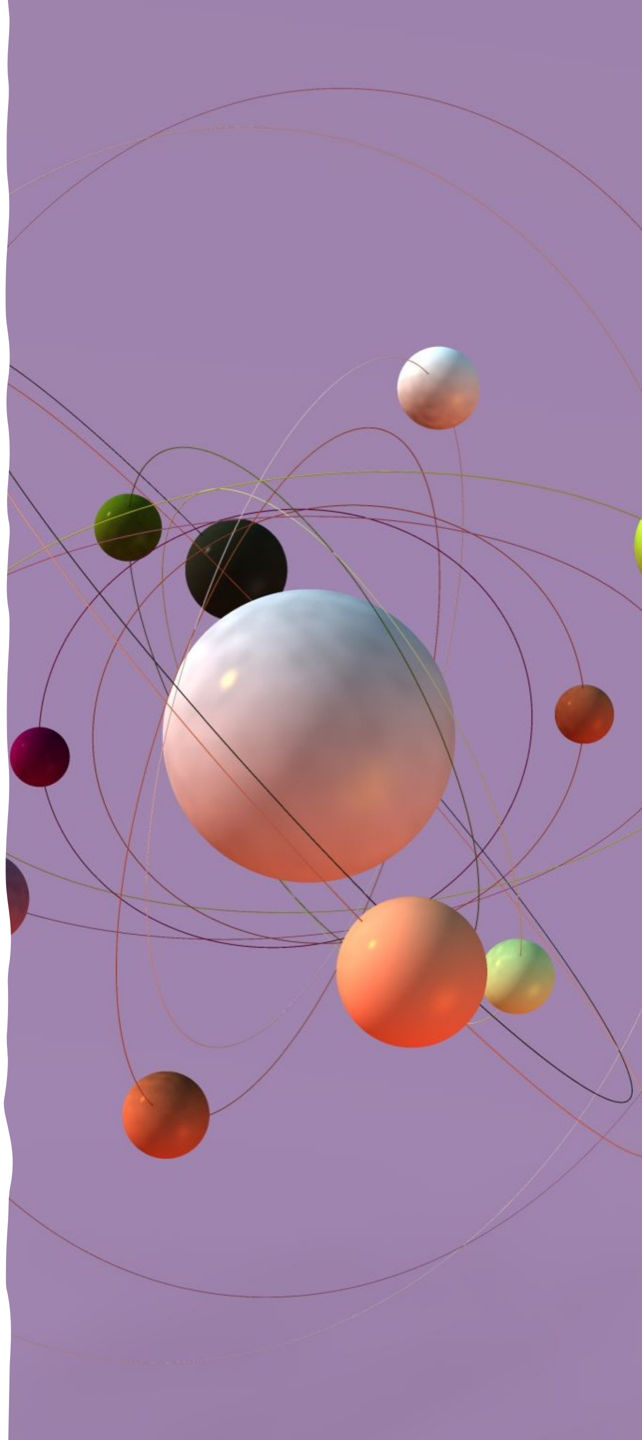
Vector Space Model- IR Project



Vishal Nigam
University of Trieste
SM3800014

Motivation & Objective

Why IR matters: Search engines, recommendation systems, academic search.

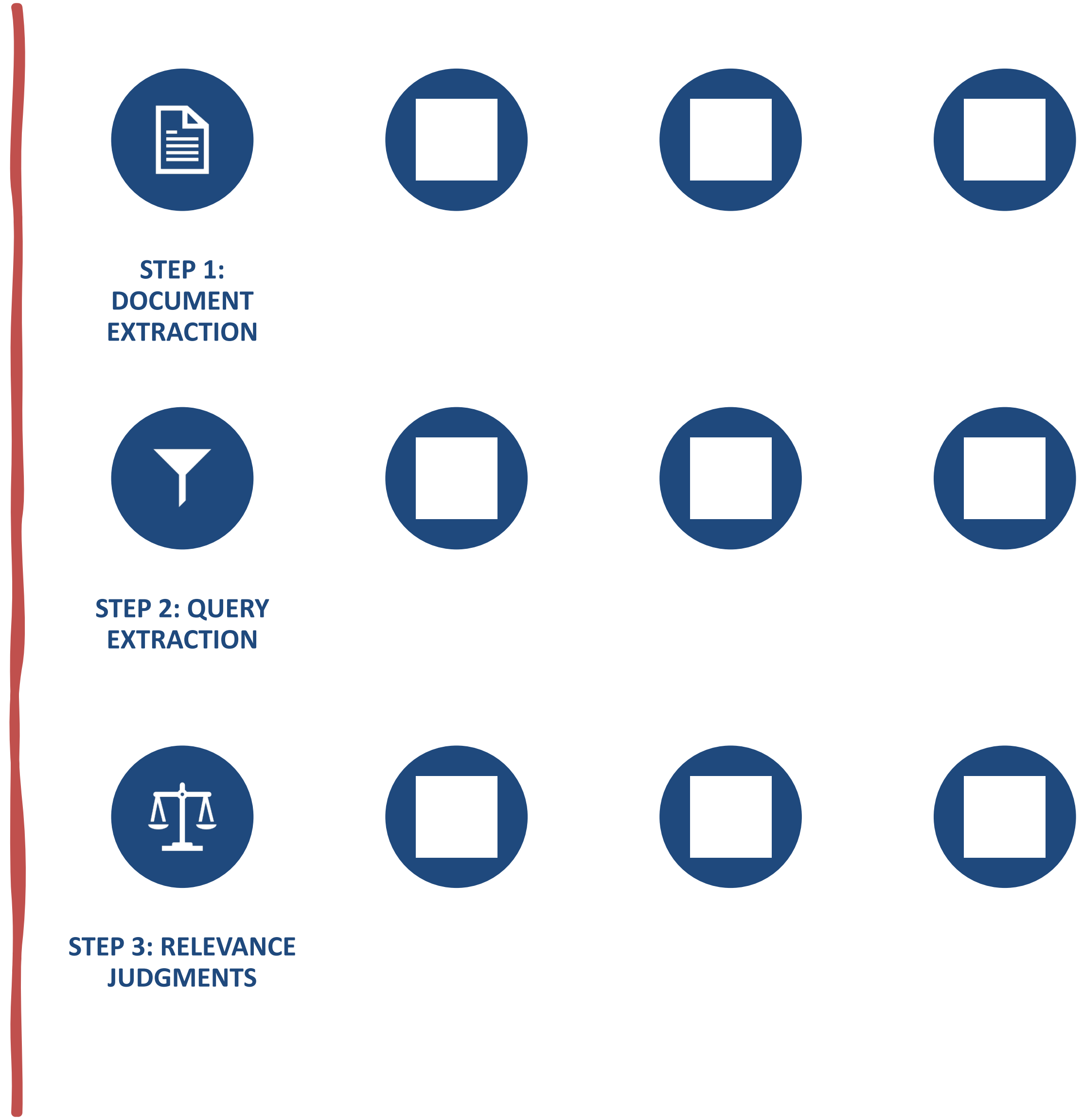
My goal: Build a functional IR system using VSM from scratch.

Techniques tested: Multiple retrieval strategies including feedback mechanisms.

Pipeline Overview

- Load Raw Artifacts
- Indexing
- Query + Pre-processing
- Retrieval
- Feedback
- Evaluation

Load Raw Artifacts



Indexing

- **Document Pre-processing**: Tokenize & clean 400 Cranfield docs
- **TF–IDF Computation**: TF, DF, IDF, and TF-IDF vectors
- **Inverted Index**: Term → [doc, weight] mappings
- **Champion Lists**: Top-5 docs per term by TF-IDF score
- **Cluster Pruning**: √N leaders + follower assignment (cosine similarity)
- **Static Quality Scores**: Higher score for lower doc numbers
- **Impact-Ordered Index**: Posting lists sorted by term weight
- **Index Files Saved**: All JSON files stored in /index folder

Query + Pre-Processing

1. User Query:

- Input query read from .txt file (e.g., query1.txt)

2. Pre-processing Pipeline:

- **Lowercasing** (to normalize casing)
- **Tokenization** (extract alphabetic tokens)
- **Stop-word Removal** (remove common filler words)
- **Stemming** (reduce words to their base/root form)

3. Query Vector Construction:

- After pre-processing, each token is mapped to its **TF-IDF weight** using the idf.json index
- Final **query vector** is formed as a sparse weighted vector aligned with document vector space
- Used for **cosine similarity** in retrieval

4. Final Output:

- Cleaned, weighted query vector
- Ensures **alignment** with document vectors for effective matching

Retrieval

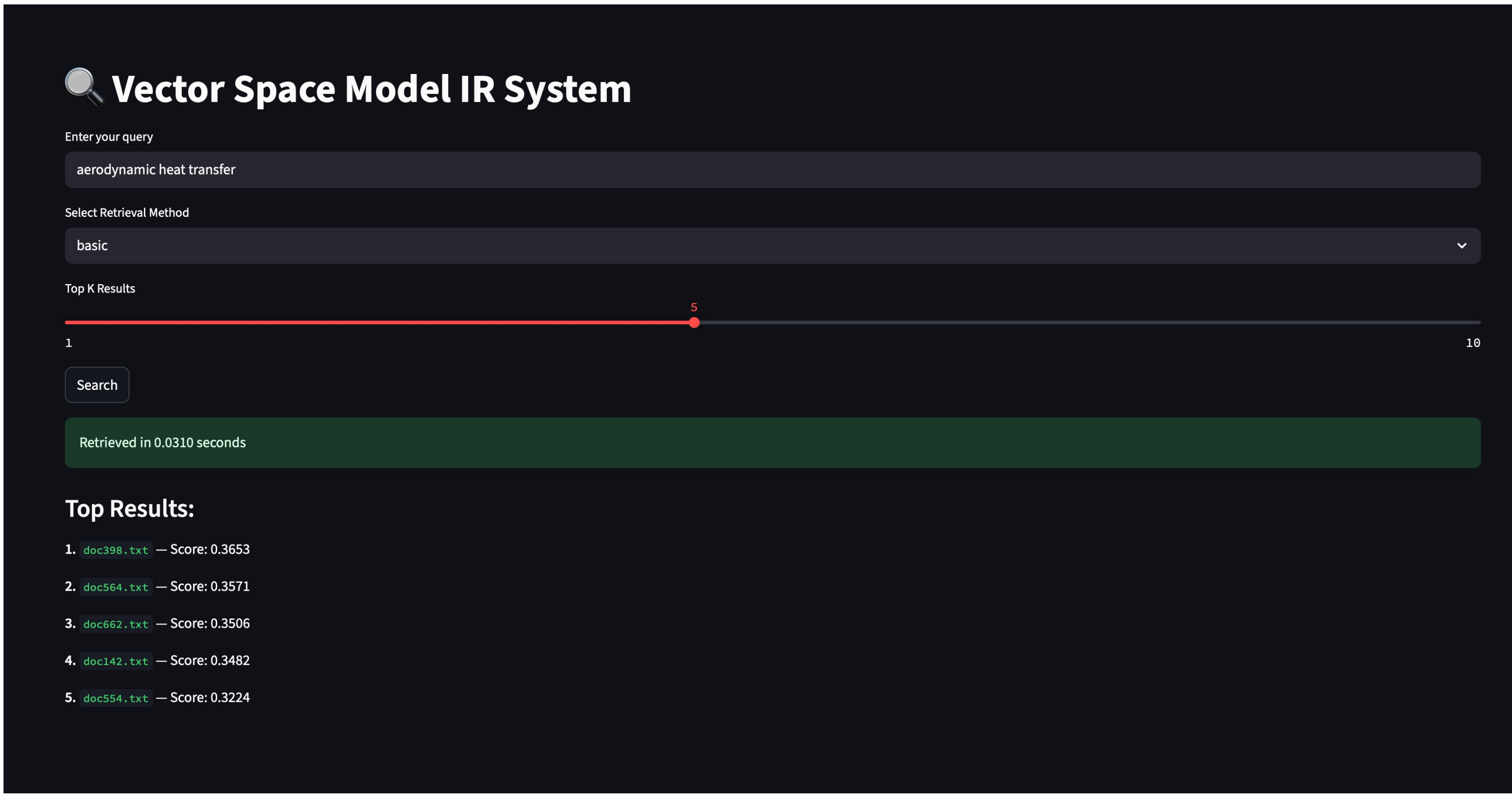
Workflow

1. **Choose Retrieval Strategy** (via method argument)

2. **Ranking**
- Documents are scored and ranked based on cosine similarity or combined scores
- Top-k results returned

3. **Output:**
- List of top-k documents with scores
- Supports **modular evaluation** and **performance comparison**

Streamlit App



Feedback

Manual Relevance Feedback using Rocchio Algorithm

Pseudo-Relevance Feedback (Blind Rocchio)

Feedback @Rocchio Algorithm

Objective:
Improve retrieval by modifying the query using user-labled relevant and non-relevant documents.

How It Works:

1. **Original Query Vector**
2. **User selects**:
    1. Relevant docs
    2. (Optional) Non-relevant docs
3. **Rocchio Formula**: with parameters

- α=1.0 (original query weight)
- β=0.75 (relevant doc boost)
- γ=0.25 (non-relevant doc penalty)

Used In:

- search_with_feedback() in search.py
- Uses rocchio_feedback() from relevance_feedback.py

Key Advantage:

- Interactive and **user-controlled** improvement of query focus

Pseudo Feedback

Objective:

- Enhance query automatically by **assuming** top-ranked documents are relevant.

Procedure:

1. Run initial search using basic cosine similarity.
2. Select top k results as **pseudo-relevant**.
3. Apply Rocchio update with:
    1. Only relevant component (no user input)
    2. Same formula as manual feedback, but:

Used In:

•search_with_pseudo_feedback() in search.py

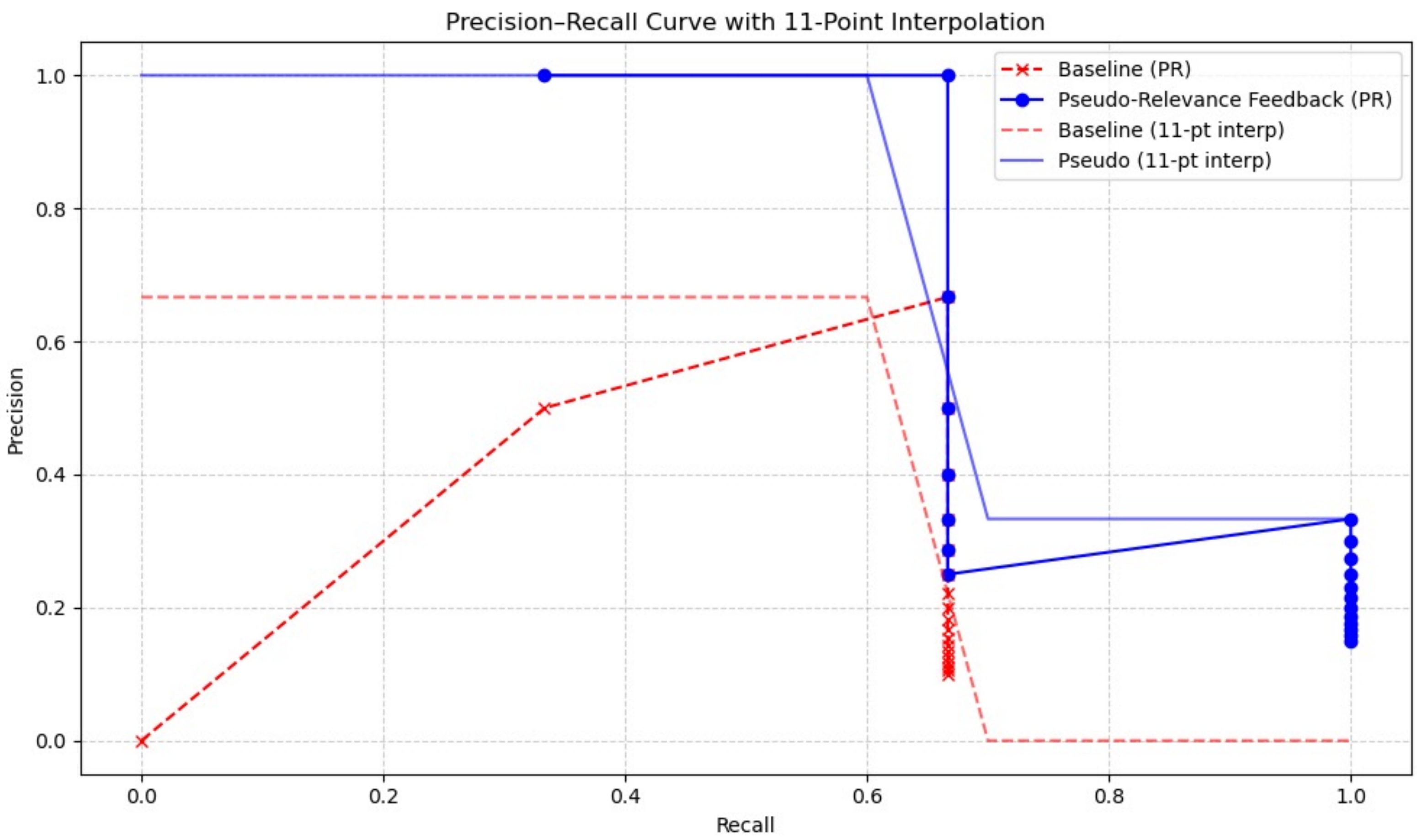
•Internally calls rocchio_feedback() with non_relevant_docs=None

Key Benefit:

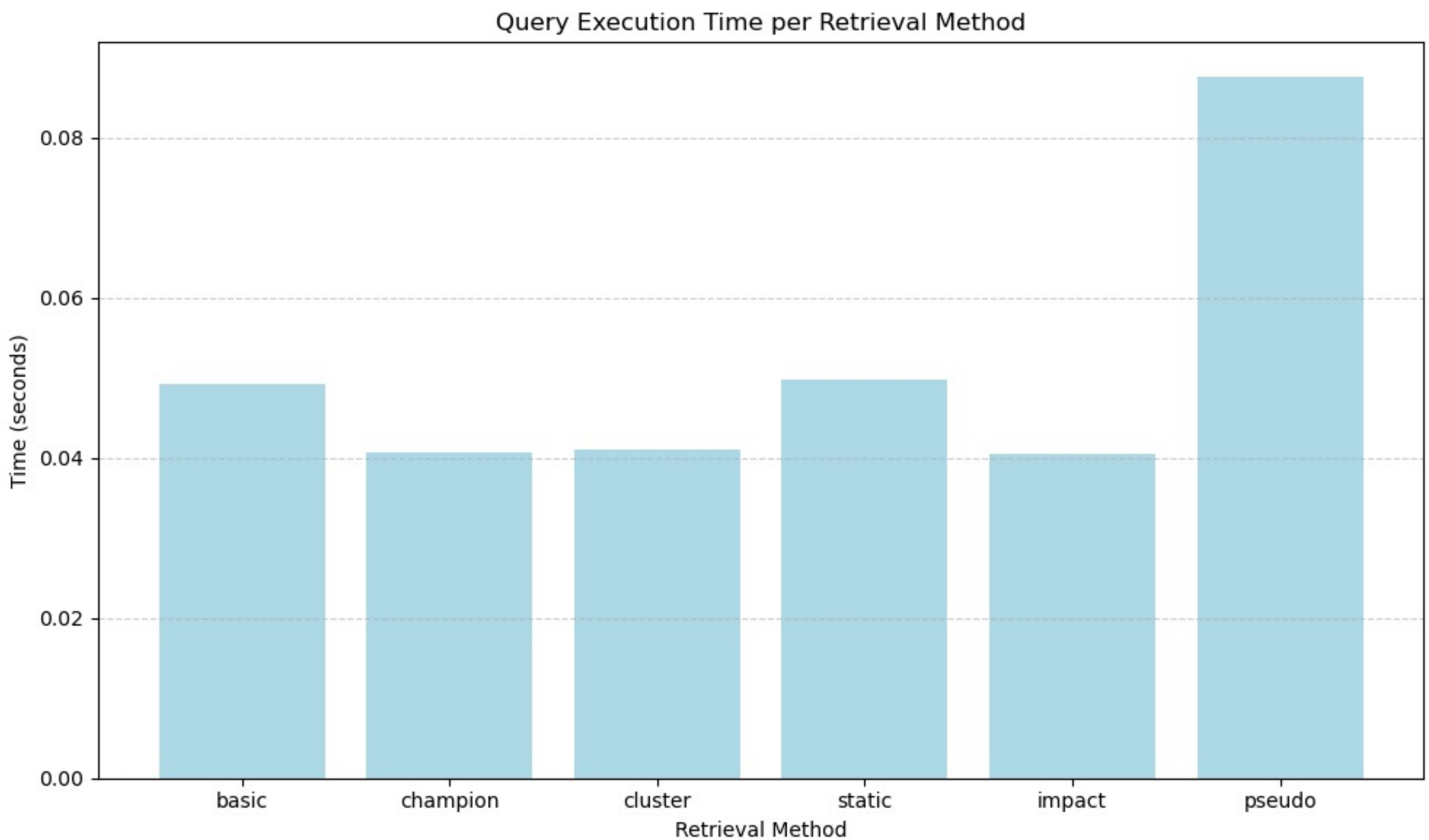
•No human input required, **fully automatic** refinement.

PR curve Analysis & Evaluation

Precision-Recall Curve (Query1) with 11-Point Interpolation



Query Execution Time Bar Chart



Conclusion

Implemented a **Vector Space Model (VSM)** based Information Retrieval system with multiple retrieval strategies

Designed custom index structures and retrieval methods from scratch for full control and explainability.

Future Work

- Selecting better relevance documents
- Integrate more robust evaluation metrics: eg: Mean Average Precision (MAP)

Extend system to support:

- Boolean and phrase queries using biword or positional indexes
- Real-time query correction using k-gram and edit distance

Question Time

Thank you