**Final Project Report: The Reviewer Puzzle**

**1. Project Introduction and Objectives**

The "Reviewer Puzzle" project addressed the difficulty of finding suitable reviewers for research papers amid increasing research complexity. The goal was to build an NLP system taking a paper (PDF) and recommending the "Top k authors" from a dataset most qualified to review it, presented via an intuitive UI.

**2. Phase 1: Dataset Creation and Preprocessing**

The project used a dataset of publications from 70 authors. Initial estimates suggested 700+ files, but processing confirmed 639 PDFs, with 637 successfully read. Data quality varied significantly, necessitating a robust extraction strategy.

**Step 1: Author Mapping**

Author names were assigned based on the dataset's folder structure.

* Python's os.walk recursively scanned all folders.
* Author names were extracted from the relative folder path using os.path.relpath. PDFs in the root directory were manually moved to appropriate author folders.

**Step 2: Robust Text Extraction**

Standard PDF parsing failed frequently. A 3-stage fallback pipeline was implemented:

1. Method 1 (Fast): PyMuPDF (fitz): Attempted fast text extraction, checked for encryption, and fell back on failure or empty results.
2. Method 2 (Robust): Tika: If PyMuPDF failed, Apache Tika (requiring a Java JDK) was used as a fallback. Tika handled many complex or mildly corrupted files that the first method could not.
3. Method 3 (OCR): Tesseract: If both PyMuPDF and Tika yielded no text (indicating an image-based PDF), pdf2image (using Poppler) converted pages to images, and Pytesseract performed OCR.

**Step 3: Text Cleaning**

Raw text was cleaned: converted to lowercase, removed punctuation/numbers, and removed English "stop words". The output was a pandas DataFrame with author\_name and processed\_text for 637 papers.

**3. Phase 2: Recommender System Building**

**Two NLP models were built for comparison:**

**Model 1: TF-IDF with Cosine Similarity (Keyword-Based)**

Matches literal keywords.

* How it Works: TfidfVectorizer (scikit-learn) created a word importance matrix.
* Recommendation: Cosine Similarity found papers with high keyword overlap.

**Model 2: Sentence Transformers (Semantic-Based)**

Understands underlying concepts.

* How it Works: The all-MiniLM-L6-v2 model encoded papers into semantic vectors.
* Recommendation: Cosine Similarity found papers with the closest meaning.

**4. Phase 3: Evaluation and Analysis**

Evaluation used sanity checks and qualitative review:

* Accuracy Check: Both models correctly ranked the paper's actual author #1 when tested with dataset papers, validating the core logic.
* Relevance Comparison: The Sentence Transformer (Embedding) model was superior, providing more conceptually relevant suggestions for ranks 2-k compared to TF-IDF.
* Practical Filtering: The system automatically filters out the paper's original author (if identified) from the final list in the UI, preventing self-recommendation.

**5. Phase 4: UI, Optimization, and Deployment**

This phase focused on usability and sharing:

1. Streamlit User Interface: An interactive UI using Streamlit was built, featuring PDF upload, selectable 'k', persistent state (st.session\_state), and side-by-side model comparison.
2. "Reviewer-Reviewer Similarity" Feature: Implemented by creating "author profile vectors" (averaging paper embeddings). The UI includes a dropdown to find authors with similar overall expertise.
3. Optimization for Speed: Addressed slow startup by:
   * Pre-processing: A one-time Jupyter Notebook script (using the full 3-stage extraction including Tika) generated and saved the final author\_df (.parquet), embeddings (.npy), and TF-IDF models (.joblib). The Sentence Transformer model was also saved locally.
   * Fast Loading: The app.py script loads these pre-computed files directly, reducing startup to seconds. ⚡️ *(Note: The deployed app itself does not need Tika/Java)*.
4. Deployment (Streamlit Cloud): Deployed using Streamlit Community Cloud for easy public access.
   * requirements.txt specified Python libraries.
   * packages.txt installed system dependencies (poppler-utils, tesseract-ocr, tesseract-ocr-eng - *Java/Tika is NOT needed here*).
   * The app code, requirements files, and pre-processed data/model files were uploaded to a GitHub repository linked to Streamlit Cloud, generating a public URL. ☁️<https://reviewer-app-vkb2005.streamlit.app/>

**6. Pros and Cons of the Final Project**

**Pros**

* Highly Robust Data Handling: The 3-stage extraction pipeline (PyMuPDF -> Tika -> OCR) successfully processed ~99.7% of PDFs, managing varied formats effectively.
* Semantically Strong: Sentence Transformer model provides relevant conceptual recommendations.
* User-Friendly & Fast UI: Optimized Streamlit app offers a good experience and quick loading.
* Accessible Deployment: Streamlit Cloud provides a free, public URL for easy sharing.

**Cons**

* Complex Pre-processing Dependencies: The *offline pre-processing script* relies on Java (for Tika), Tesseract, and Poppler, adding setup complexity for regenerating the core data files. (The deployed app itself is simpler).
* Closed-World System: Limited to recommending from the initial 70 authors.
* Limited Evaluation: Relied on qualitative checks due to the absence of a ground truth dataset.
* Basic Author Mapping: Uses folder names only; doesn't identify co-authors (though primary author self-review is filtered).