A TOOL FOR ALIGNING MANUSCRIPTS WITH SUITABLE JOURNALS

BY

CHIN HONG AN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2026

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Ts Dr Ku Chin Soon, who has given me this bright opportunity to engage in a recommendation system project. His insights, feedback, and patience helped shaping the direction and success of this project. It is my first step to establish a career in machine learning and deep learning. A million thanks to you.

My thanks also go to the Faculty of Information and Communication Technology (FICT) of Universiti Tunku Abdul Rahman for providing the course framework, facilities and resources that made this research possible. I am grateful to the lecturing staff who provided useful comments during the IIPSPW sessions.

I would like to acknowledge the maintainers and providers of the datasets and tools used in this study (for example, public preprint repositories and open-index directories) their open resources allowed rapid prototyping and evaluation of the proposed methods. I am also thankful to the authors of the open-source libraries and models that were indispensable to the implementation and experiments.

Special thanks to my coursemates and friends for their feedback and moral support during long coding and writing sessions. Finally, I wish to thank my family for their understanding and encouragement throughout this project.

ABSTRACT

This project is a research-based project focused on machine learning and deep learning for the evaluation of machine learning algorithms used in aligning manuscripts with suitable journals. Traditional "Journal Finder" tools are limited to single publishers' portfolios and rely on keyword-only matching (TF–IDF, BM25), which overlooks context and synonyms, resulting in poor recommendations and delayed publication. In this project, journal scope metadata from the Directory of Open Access Journals (DOAJ) has been assembled and normalised, covering 21,782 journals across 327 categories. Three retrieval models were implemented and compared — TF-IDF, Okapi BM25 and Sentence-BERT — with hyperparameters tuned using Bayesian optimisation (Optuna). Three distinct input strategies were tested — title only, title + scope, and a hybrid approach using title + scope + category — to determine the optimal method for content representation. For each pipeline, manuscripts and journal scopes were encoded into vector representations to retrieve the top candidate journals. Performance was rigorously evaluated on a held-out test set using HitRate@K, MAP@K and NDCG@K, under both random and equal distribution setups. The results demonstrate that TF-IDF with the hybrid input strategy achieves a HitRate@1 of 88.84%, outperforming BM25 (83.47%) and Sentence-BERT (68.53%). This finding contradicted the initial hypothesis that transformer-based models would yield superior performance, and was attributed to the standardised vocabulary and short document lengths in the journal metadata. The best-performing model was deployed in the FlareSearch web application as a practical journal recommendation tool.

Area of Study: Machine learning, Deep learning

Keywords: Recommender system, information retrieval, TF-IDF, Bayesian optimisation, journal finder

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

NLP	Natural Language Processing
CBF	Content-Based Filtering
TF-IDF	Term Frequency-Inverse Document Frequency
CF	Collaborative Filtering
IR	Information Retrieval
ELMo	Embeddings from Language Models
biLM	Two-layer Bidirectional Language Model
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
BERT	Bidirectional Encoder Representations from Transformers
SBERT	Sentence-BERT
NLTK	Natural Language Toolkit
MAP	Mean Average Precision
MRR	Mean Reciprocal Rank
NDCG	Normalised Discounted Cumulative Gain
DCG	Discounted Cumulative Gain
IDCG	Ideal Discounted Cumulative Gain
UI	User Interface
CV	Cross-Validation
TPE	Tree-structured Parzen Estimator
DOAJ	Directory of Open Access Journals
IEEE	Institute of Electrical and Electronics Engineers

CHAPTER 1
Introduction

1.1 Project Background

The process of publishing in a journal begins with selecting a research topic and thoroughly reviewing the existing literature. After that, the next step will be preparing the manuscript [1]. Selecting an appropriate journal for an academic manuscript is important for timely publication and maximizing research time, yet authors often rely on manual matching scopes and keywords [2]. In response, publishers and third parties came out with "Journal Finder" tools such as Elsevier Journal Finder and Springer—which analyze the manuscript (titles, abstracts, or keywords & etc.) to suggest a suitable journal for publication by text-matching and metadata filters [3].

Recent advances in deep learning, particularly transformer-based language models have demonstrated superior ability to capture document semantics for tasks like scientific information retrieval and text similarity [3]. Specifically, services like Elsevier Journal Finder and Springer use Natural Language Processing (NLP) techniques to mine the input text and generate all relevant annotations. Other than that, there's a lot more methods that can be used for recommendation system like TF-IDF (Term Frequency-Inverse Document Frequency), CBF (Content-based filtering), CF (Collaborative filtering) etc [4].

This research will explore the performance difference of aligning manuscripts that implements with different processing techniques, algorithms and different methods from other fields. To analyze the best alternative combination for the operation of aligning manuscripts with journals.

1.2 Problem Statement

Most journal recommender systems restrict suggestions to a single publisher's portfolio and rely on keyword-only matching, forcing authors to find through poor-fit recommendations and ultimately delaying the progress of their research. Systems like Elsevier's Journal Finder [4][5] only index in-house titles, means that it only recommends journals indexed in their own database, this resulted in limited coverage for some cases where authors from other domains like medical related manuscripts, could not find suitable journals on system that is limited to scientific journals that excluded medical domain, or vice versa. Hence, algorithms like Okapi BM25 [5] ignore word order, context and deeper semantic relationships, which lead to higher mismatches. Consequently, authors must manually search through low-relevance suggestions, delaying dissemination of research.

1.3 Motivation

This research is motivated by the desire to address these issues to contribute to a more efficient and coverage of aligning manuscripts to journals. By exploring alternative solutions and seeking hybrid approaches that may have good performance when applied in alignment with journals, this research aims to provide insight for researchers to use as a reference for future work.

1.4 Research Objectives

The objectives of this research are as follows:

1. Aims to gather and normalize journal scope metadata from **open-access directories** to reach over ~10,000 journals to overcome the coverage limitations.
2. To examine both keyword-based models (TF–IDF, BM25) and BERT—Bidirectional Encoder Representation with Transformers models in retrieving manuscripts against journal scopes.
3. To compare different processing pipelines, including raw title encoding versus title-abstract encoding, and experiment with title-abstract-category as input, to see how they affect retrieval performance, an HitRate@1 (top-1 accuracy) increase of 2% is considered as success by including category as input.

4. To measure and compare the effectiveness of each method using ranking metrics mainly using ranking metrics including **HitRate@K, MAP@K and NDCG@K**, determining which approach yields the highest relevance scores.

1.5 Project Scope

This project explores a range of document-retrieval methods for matching full manuscripts to journal scopes. **This project includes traditional keyword retrieval like TF-IDF and BM25 to compare with the transformer encoder** in current development, which is Sentence Transformer (Sentence-BERT) that extracts patterns or representations from the data or word embeddings by passing it through an encoder [6]. Similar texts will have nearby vector representation to allow retrieval by computing vector similarity instead of keyword matching. Hence, three types of inputting mode will be taken, which are title only encoding, title-abstract encoding and title-abstract-category encoding. The choice for including category as one of the inputs is because it can significantly improve the precision and accuracy of the system by eliminating unnecessary categories.

All the methods and inputs in this project are eventually to be benchmarked against each other. The performance of keyword-based and embedding-based retrieval is evaluated using standard ranking metrics. In the research, studies will be conducted to obtain suitable metrics for evaluating this type of journal recommendation system. To ensure optimal model configurations, a **Bayesian optimisation framework (Optuna)** was adopted to systematically search for the best hyperparameters for each model, replacing the originally proposed manual grid search. In the end come up with the best performing system.

1.6 Contributions

As mentioned in the problem statement, most journal finder tools tend to index only limited titles or limited number of journal articles, which results in low precision and many off-target recommendations.

In this research, aligning manuscripts with the current standard method, hybrid of two or more methods, the methods parameter and their performance will be evaluated and explored. Furthermore, learning methods that were previously not widely used in

aligning manuscripts will be explored and tested in this research. A Bayesian optimisation framework was implemented for **systematic hyperparameter tuning**, enabling efficient search over continuous parameter spaces rather than the limited discrete grid originally proposed. Hence, other researchers can make use of this research for refining their future work in journal recommendations system.

1.7 Chapter Summary

As a foundation for the research, Chapter 1 begins with an overview of the journal publication process and the role of existing "Journal Finder" tools. It highlights a critical gap in current systems, which is their reliance on **limited, single-publisher** databases and simplistic **keyword-matching** algorithms like BM25. This often results in **poor-fit recommendations**, forcing researchers to manually sift through **irrelevant suggestions** and delaying the dissemination of their work.

In response to these challenges, the project's motivation and objectives were established. The core aim is to conduct a comparative analysis of different manuscript-to-journal matching techniques. This research **evaluated** traditional **keyword-based** models (TF-IDF, BM25) alongside a modern **deep learning** approach (BERT) to determine their relative effectiveness. Furthermore, the project scope includes an investigation into how varying the input data. Which are from **title-only** to combinations including the **abstract and category**, impacts retrieval performance. The evaluation utilised **HitRate@K, MAP@K and NDCG@K** as the primary ranking metrics, with **Bayesian optimisation (Optuna)** adopted for hyperparameter tuning. The ultimate contribution of this work will be to provide a thorough performance benchmark of these methods, offering valuable insights for the development of more accurate and comprehensive journal recommender systems.

CHAPTER 2

Literature Reviews

2.1 Filtering and Retrieval in Recommender System

2.1.1 Filtering Method in Recommender Systems

For literature recommenders, models are usually classified into three main approaches -- which are **Content-based Filtering (CBF), Collaborative Filtering (CF) and Hybrid** [2][4][7]. Each of these approaches uses different information and algorithms to generate personalized suggestions.

[7] proposed that in CBF, it models a user's interests by analysing the features of items the user liked in the past. For example, in a document recommendation scenario, each item (document) is represented by a keyword-weight vector (often TF-IDF weighted), and the user's profile is the aggregate of features from items they rated positively. New items are scored by comparing their feature vector to the user profile (e.g. via cosine similarity of TF-IDF vectors), and the highest-scoring items are recommended. This method can recommend items even if no other users have rated them, if their content matches the user's profile. Therefore, it capable to handle new items that have rich content since it does not rely on other users' rating. It also provides interpretable recommendations based on explicit features like keywords, tags, etc.

CBF are limited by the features that it tends to overspecialize [7], meaning it only suggests items very similar to what the user has already seen. For instance, a user who has never liked Malay cuisine will never be recommended a Malay restaurant. Furthermore, content-based systems also suffer from the cold-start problem for new users, as they need enough initial ratings to build an accurate profile [7][8]. Therefore, content-based systems require good feature extraction, means that items must be described clearly by meaningful content like text, metadata, etc. If items have inadequate features, or features that are hard to extract, the system may have difficulties in discriminating them well [7].

In CF, the system makes recommendations based on the pattern of the interaction between user and item. A collaborative system predicts a user's preference for an item by making use of the measure of similarity between users [8]. Formally, the utility $u(c , s)$ of item s for user c is estimated based on the utilities $u(c_j , s)$ assigned to

item s by those users $c_j \in C$ who are "similar" to user c [7]. For example, in movie recommendation, the system first identifies the "peers" of user c – other users with similar rating histories (often found via user–user or item–item similarity measures) – and then recommends movies highly rated by those peers. In practice, similarity can be computed by measures like Pearson correlation or cosine similarity on the user–item rating matrix [8]. Since CF relies on other users' judgments, it can recommend items irrespective of their content. A pure collaborative system "can deal with any kind of content" and suggest items that are different from those the user has seen before [7].

Just like CBF, CF also suffers from the cold-start issues [7][8]. A new user who has rated few items provides little data for matching, resulting in unreliable recommendations [7]. Similarly, a new item with few ratings cannot be recommended until it gathers enough feedback [7]. Furthermore, even with many users, in CF the rating matrix is usually sparse, so many items/users lack sufficient overlaps. When user tastes are unique or ratings are sparse, it may be impossible to find meaningful neighbours [7]. In addition, basic neighbourhood CF may not capture complex patterns in the data, and large-scale CF can be computationally expensive.

For hybrid approaches, it combines CBF and CF to make use of their advantages of both and mitigate their weaknesses. For example, run a content-based and a collaborative model in parallel and then fuse their outputs. The separate predictions can be combined by a weighted sum or voting scheme [7]. Alternatively, the system may dynamically switch between recommenders: for each query (or each user) it chooses the method that is expected to give better confidence [7]. Common hybrid mechanisms include [7]:

- Weighted Combination: merge CBF and CF scores.
- Switching: choose CBF or CF based on context.
- Unified Model: Integrate all features into one model.

By design, hybrid approach can overcome the limitations of CBF and CF method. They can recommend new items (via content features) while also bringing in collaborative signals for accuracy. For example, content-based overspecialization can be relieved by CF's ability to introduce novel items, and CF's cold-start can be mitigated by content features in the model. Hybrids often achieve better accuracy and coverage in practice.

The main downside is complexity. Combining models or data sources requires careful tuning (e.g. balancing weights), and unified models can be computationally intensive. Hybrid systems may also be harder to interpret. However, they provide a flexible framework that captures a broader range of information than pure approaches [7].

2.1.2 Retrieval Techniques in Recommender Systems

There are various types of techniques found used in recommendation systems, like Elsevier Journal Finder [5] that utilized Okapi BM25 as their ranking algorithm, TF-IDF mentioned by [7] earlier, etc.

2.1.2.1 TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure commonly used in NLP and information retrieval [9]. [4] stated that in CBF, most of the publications used TF-IDF to evaluate the similarities between text objects. TF-IDF negates the effect of high-frequency words while determining the importance of an item [4]. TF-IDF combines two components: TF and IDF.

Term Frequency (TF) measures how often a word appears in a document. A higher frequency suggests greater importance [9]. If a term appears frequently in a document, it is likely relevant to the document's content [2]. The function can be represented as:

$$TF(t, d) = \frac{t}{d} \tag{2.1}$$

Where:

t: Number of times term t appears in documents d

d: Total number of terms in documents d

Inverse Document Frequency (IDF) reduces the weight of common words across multiple documents while increasing the weight of rare words. If a term appears in fewer documents, it is more likely to be meaningful and specific [2][9]. The function can be represented as:

$$IDF(t, D) = log\ \frac{D}{t}$$

(2.2)

Where:

D: Total number of documents in corpus D

t: Number of documents containing term t

TF-IDF is simple, interpretable, and effective at capturing keyword relevance with very little training data. But it relies on exact word matching and produces very high-dimensional sparse profiles, so it cannot capture synonyms or latent semantics. As a result, pure TF–IDF models often suffer from limited coverage.

2.1.2.2 Okapi BM25

Okapi BM25 is a probabilistic IR ranking function used by search engines to evaluate how well a document matches a specific search query [10]. It belongs to the family of probabilistic information retrieval models, which aim to calculate the likelihood that a document is relevant to a user's query based on the statistical properties of the text [10].

BM25 ranks documents based on how well they match a query [5][10], considering factors mentioned previously such as TF, document length, and IDF. In TF, BM25 introduces a saturation effect, which beyond a certain point, additional occurrences of a term contribute less to the score [10]. This prevents long documents from having anomaly weight. While in IDF, is computed similarly to TF–IDF. The formula of BM25 score for a document *d* with respect to a query *q* is computed as [10]:

$$Score(q, d) = \sum_{teq} IDF(t).TF(t, d)$$

(2.3)

This sums up the contributions of all query terms *t* in the document *d*.

BM25 remains efficient and is well-founded statistically, it was widely used in search engines and content recommendation like Elsevier Journal Finder. But like TF-IDF, BM25 still uses only keyword counts and ignores word order or semantics. It

cannot by itself incorporate user preference data or learn from ratings, so its use in recommenders is limited to content-scoring.

2.2 Previous Work in Recommendation System

In recent years, deep learning has made significant progress in many fields, which includes research in recommendation systems. A significant advancement in the field of journal recommendation has been the shift from traditional, context-free word embeddings to more sophisticated language models capable of capturing **deep, context-sensitive semantics**. Older models, relying on techniques like word2vec, would assign a single, static vector to each word regardless of its usage. This approach struggles with polysemy meaning of words in different contexts.

2.2.1 ELMo

The introduction of Embeddings from Language Models (ELMo) in 2018 February [11] marked a key step forward, as it provides a dynamic word representation that changes based on the word's surrounding context. The primary feature of ELMo, as mentioned in [12], is the ability to handle semantics by capturing the context of each word. ELMo provides a dynamic, word-level vector representation, which allows it to generate different vectors for the same word when it appears in different contexts. This capability is derived from its underlying architecture, which is a **two-layer bidirectional language model** (biLM) pre-trained on **vast amounts of texts** [12].

Based on [12], which focused on a recommendation system for journals in physics, chemistry, and biology, ELMo as the primary feature engineering mechanism to analyze the abstracts of scientific papers. In this paper [12], a pre-trained ELMo model was incorporated as an embedding layer directly following the input layer in both the Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) architectures. Its main role was to convert the text from the paper abstracts into numeric values, specifically high-dimensional vectors, so they could be processed by the deep learning models [12]. The ELMo model takes text as input and generates a tensor. For each word in the text, it produces a vector containing 1024 values [12]. The highest accuracy was achieved with the RNN model using data from the physics (PH) discipline, reaching 83% when considering the top-20 recommended journals.

2.2.2 BERT

Bidirectional Encoder Representations from Transformers (BERT) is consistently presented as a state-of-the-art, pre-trained language model that fundamentally enhances how machines understand academic text for journal recommendation [13][14][15]. Its primary specialty, and a significant leap over previous methods, is its ability to **process text bidirectionally** [13][15]. Unlike earlier models that read text in one direction (left-to-right or right-to-left), BERT's Transformer architecture uses a **multi-head self-attention mechanism** to consider the **entire sequence of words at once** [13][15]. This allows it to grasp the full context of a word by examining the words that appear both before and after it, which is crucial for accurately resolving issues of polysemy—where a single word can have multiple meanings depending on its usage [13][15]. This deep, contextual understanding results in more precise and semantically rich vector representations of scholarly texts, forming a robust foundation for the recommendation task [14][15].

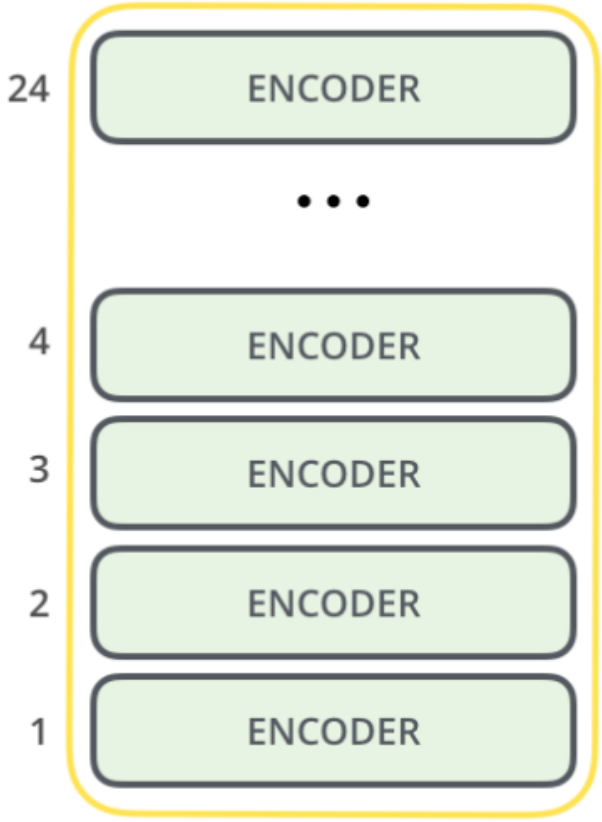

Figure 2.2.2.1: Structure of BERT, a Transformer Encoder stack [11]

In the context of journal recommendation, BERT is deployed as a powerful **feature extractor** that converts unstructured text into meaningful numerical vectors [13][14][15]. The models detailed in the papers use various combinations of a manuscript's explicit features as input, most commonly the title and abstract [13][14][15], but also keywords [15] and, in an innovative approach, the journal name itself [13]. This text is fed into the BERT model, which then generates contextualized embeddings for each word. A key component of this process is the special [CLS] token, which is prepended to every input sequence [14]. The final hidden state corresponding to this token is used as an aggregated, fixed-dimensional representation for the entire text, capturing its global semantic meaning and serving as a direct input for downstream classification layers [13][14]. The models often integrate BERT as an initial embedding layer within more complex architectures, such as two-tower neural networks or in conjunction with other models like TextRNN, to further process these embeddings and extract higher-order semantic features [13][15].

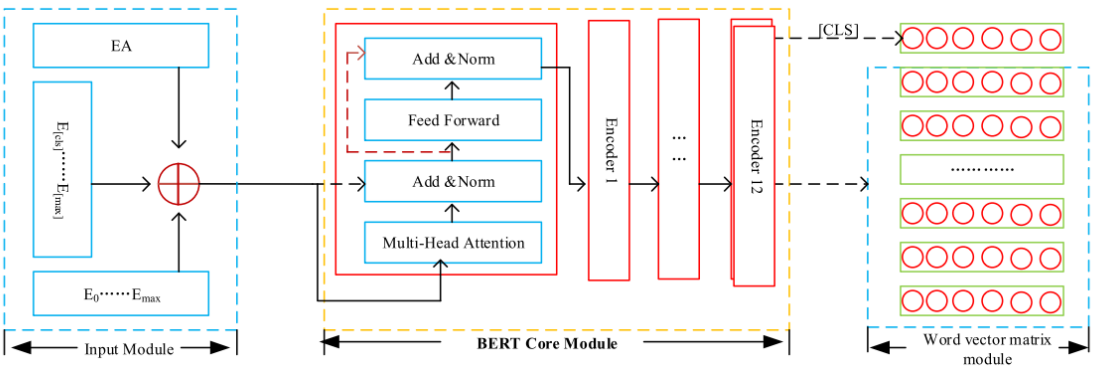

Figure 2.2.2.2: BERT language representation model structure [14]

The effectiveness of using BERT is demonstrated through significant performance gains over a wide range of baseline methods, including traditional techniques and other deep learning models like LSTM, Bi-LSTM, and CLAVER [13][14][15]. Experimental results across different datasets show that BERT-based frameworks consistently achieve superior results in key evaluation metrics. For instance, models incorporating BERT show substantial improvements in Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), and Top-N Recall [13][15]. [13] noted that its proposed model, which uses BERT, improved MAP by up to 19.42% and MRR by up to 59.79% over benchmarks. [15] found that a BERT-based model outperformed the next-best baseline in MAP by 4.14% and in MRR by 7.98%. This

consistent outperformance is attributed to the high-quality, context-sensitive vector representations that BERT produces, which enable the recommendation models to make more accurate and relevant journal predictions [14][15].

2.3 Comparison of Related Works

To conclude findings and information for the research, a range of journal finder and journal recommendation systems have been studied, each of them differs in **algorithm approach**, **input options** applicable to our project, and their **evaluation metrics**. Their options of input, metrics and the way of algorithm application are to be compared and referred to ensure the correct direction of this research. Table 2.3.1 summarizes key features of these systems.

Table 2.3.1: Comparison of various recommendation systems

Reference ID	System	Algorithm	Input	Evaluation Metric
[4]	**Springer Journal Suggester**	TF–IDF + SVM classification	Text (Titles, Abstracts, Keyword)	Accuracy, Precision
[5]	**Elsevier Journal Finder**	TF–IDF + BM25	Text (Titles, Abstracts, Keyword)	Top-N recall
[12]	**ELMo & Deep Learning**	CNNs, RNNs, ELMo	Text (Abstracts)	Accuracy
[13]	**PTMFFJRec**	Hybrid (Transfer Learning, BERT, and GCN)	Text (Titles, Abstracts)	MAP, MRR, Recall
[14]	**JNSMFFJRec**	Hybrid (BERT, two-tower neural network)	Text (Titles, Abstracts)	MAP, MRR, Recall
[15]	**CBGJRS**	Hybrid (BERT + Bipartite Graph)	Text (Abstracts)	Accuracy, MRR, F1-score

[16]	**Cross-loop Contrast**	Graph Contrastive Learning	Multi-granularity Text (Words, Sentences, Topics)	Accuracy, MRR

2.4 Research Gaps and Key Findings

2.4.1 Publisher Scope

Most of the current journal finder tools like Elsevier and Springer, index only their in-house titles, excluding some domain from their recommendations. According to [5], Elsevier's Journal Finder covers around **2900 journals**, but it is limited to Elsevier titles only.

2.4.2 Semantic Matching

Keyword-based algorithms like TF-IDF and BM25 are both **ignoring deeper context**, which may lead to mismatches when manuscripts use **phrases that have similar meaning** but different word from the possible match. According to [10][11], both BM25 and TF-IDF algorithms rely heavily on term frequencies and inversed document frequencies, which does not capture semantics beyond exact overlaps.

2.4.3 Evaluation Metrics

Throughout the studies stated in Section 2.3 and Table 2.3.1, several metrics are identified as the common metrics used for evaluating a recommendation system, which are: **Accuracy, Recall and MAP**. Therefore, a recommendation system should be focused on comparing and aiming for good number in these metrics.

2.4.4 Input Options

Recommendation system like [4] [5] [13] [14] usually takes **Title** and **Abstracts** as their primary input for recommending a journal and sometimes may take **keyword** as an additional input if appropriate for their algorithm. In [16], the **topics** of the paper are also considered as their input.

2.5 Limitation of Current Studies

As stated in the problem statement, most journal recommendations system [4][5] largely rely on **shallow scope, term-based matching techniques** that inadequately capture document semantics. Traditional approaches like TF–IDF and BM25 treat documents as bags of words, matching keywords without understanding context or synonyms. Such heuristics often fail to recognize semantically similar content that uses different terminology, limiting relevance when aligning manuscripts to journals. In addition, many published tools are confined to single publishers' portfolios – for example, the Elsevier Journal Finder only suggests Elsevier titles – which restricts the recommendation scope and may overlook more suitable outlets.

While recent advancements in journal recommendation systems, particularly those leveraging deep learning, have shown impressive results, the current body of work is not without its limitations. A primary limitation is the heavy reliance on semantic similarity derived from textual content, such as abstracts and titles. Although models like BERT and ELMo excel at capturing deep contextual meaning, their focus on content matching overlooks other critical factors that influence a researcher's submission decision. Real-world choices are often guided by a journal's impact factor, review speed, acceptance rate, publication costs, and specific scope, none of which are typically contained within the manuscript's text. This is particularly problematic for interdisciplinary research, where papers span multiple fields, making simple semantic matching insufficient and leading to excessive intraclass variation that models struggle to handle.

2.6 Summary

This chapter provided a comprehensive review of the literature on recommendation systems, establishing the theoretical and practical context for this research. The review began by exploring fundamental filtering methodologies, detailing the respective strengths and weaknesses of **Content-Based Filtering (CBF)**, which risks overspecialization; **Collaborative Filtering (CF)**, which struggles with the cold-start problem for new users and items; and **hybrid** approaches designed to mitigate these issues. It then examined traditional retrieval techniques like **TF-IDF and Okapi BM25**, acknowledging their effectiveness in keyword-based matching while highlighting their inherent limitation: by treating documents as a "bag of words," they fail to capture the actual meaning of syntax, word order, and synonyms, thus preventing a deep understanding of semantic context.

The discussion then transitioned to the evolution of recommendation systems, highlighting the significant advancements brought by deep learning, which marked a paradigm shift from static word embeddings to dynamic, context-aware models. The introduction of **ELMo** and the state-of-the-art **Bidirectional Encoder Representations from Transformers (BERT)**, were presented as a major leap forward. The review emphasized how BERT's ability to process an entire text sequence at once, which considers both preceding and succeeding words that allows for a profoundly more accurate understanding of academic language, effectively resolving ambiguity and grasping complex semantic relationships. This capability has led to substantial and consistent performance gains in journal recommendation tasks.

Through a comparative analysis of related works, key research gaps were identified, including the previously mentioned issue of **single-publisher scope** and the shortcomings of relying purely on semantic matching. The chapter concluded by acknowledging that even these advanced models have limitations, as they often overlook the practical, non-textual factors that heavily influence researchers' submission decisions, such as a journal's impact factor, review speed, acceptance rate, and publication costs. This oversight reinforces the need for the different angles of research proposed in this project.

CHAPTER 3
System Model

3.1 Overview

This research implements and evaluates methods to align manuscripts with suitable journals by comparing traditional keyword retrieval (TF–IDF, BM25) with modern dense transformer embeddings (Sentence-BERT). The work is experimental: produce a benchmark (data + code), a retrieval pipeline (several processing variants), and a prototype (web UI) that authors can use to get journal suggestions. The design emphasises reproducibility (saved artifacts, deterministic splits) and a clear verification plan using standard information retrieval metrics.

In FYP1, the system was proposed with a verification plan based on Recall@K and Accuracy@K. However, upon **further review of the B!SON journal recommendation framework** by Entrup et al. [19] and the broader information retrieval literature, the evaluation metrics were revised to HitRate@K, MAP@K (Mean Average Precision at K) and NDCG@K (Normalised Discounted Cumulative Gain at K). These metrics are more **widely adopted in recommendation system research** and provide a more comprehensive assessment of **ranking quality beyond simple accuracy**. The justification for this change is detailed in Section 3.3.

Similarly, the hyperparameter tuning strategy was revised from a manual fixed grid search to Bayesian optimisation using the Optuna framework. The original proposal specified a small number of discrete values per parameter (e.g., ngram_range $\in$ {(1,1), (1,2)}). This approach was limited in coverage and could not explore continuous ranges efficiently. The adoption of Optuna enabled automated, data-driven search over broader parameter spaces. The details of this change are described in Section 3.4.

3.2 System Design

Below is a top-down description of the system with functional blocks and responsibilities.

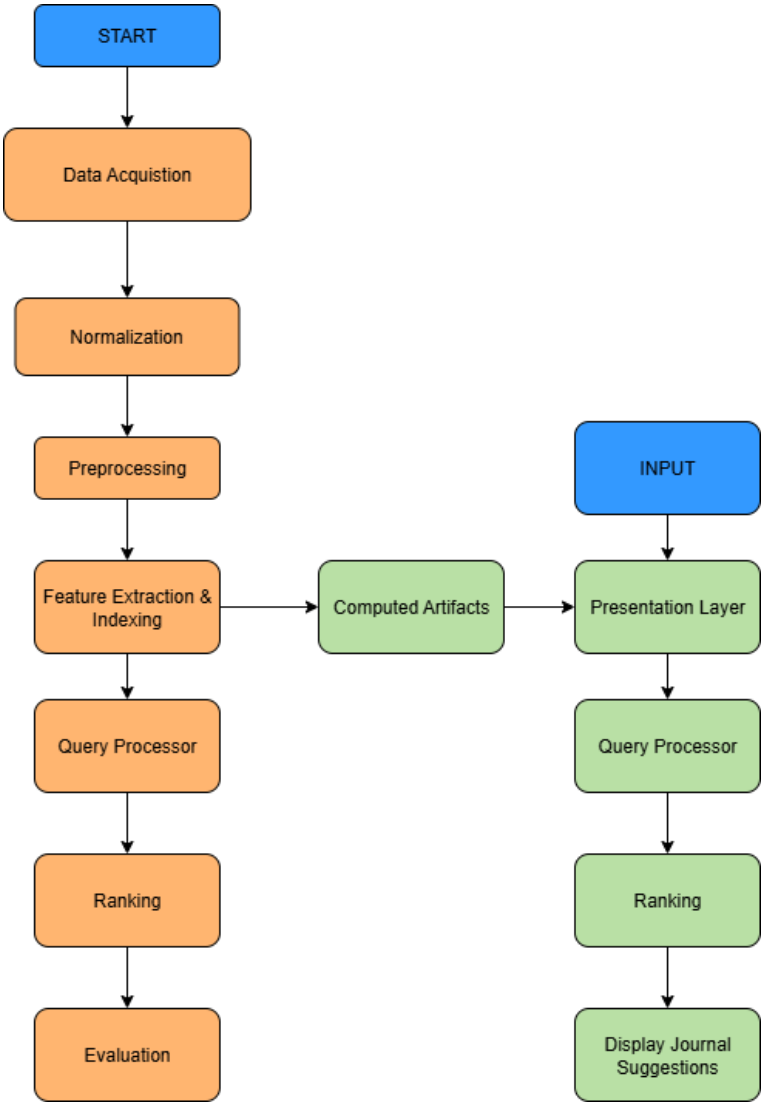

Figure 3.2.1: Overall flowchart for Journal Finder, FlareSearch

The system is split into two parts, data preprocessing layer and presentation layer. From here, it starts with these main steps:

3.2.1 Data Acquisition

In this project, dataset options are considered based on the available columns that are useful for the feature extraction. Therefore, dataset should have at least a Journal Title column, and subject column, any additional column that are useful such

as keywords is a plus. The dataset used in this project is journalcsv_doaj.csv [17], containing 21,782 journal entries across multiple publishers and domains.

3.2.2 Normalization & Preprocessing

This process includes identifying the Title, Subjects, Keywords columns, and then cleaning text by utilising the **NLTK toolkit**. The dataset is split into training and test set in the ratio of 80:20 with a fixed random seed (random_state=42) throughout all models in this experiment. The output of this process includes: doaj_train.csv (17,426 journals), doaj_test.csv (4,356 journals) and doaj_normalized.csv.

3.2.3 Feature Extraction & Encoding

For TF–IDF, TF–IDF vectors are computed for journal scopes using scikit-learn's TfidfVectorizer [18]. **For BM25**, documents are tokenised and indexed using the rank_bm25 library. **For Sentence-BERT**, Title and Scope are encoded into dense vector embeddings using pretrained sentence-transformer models and stored as NumPy arrays.

3.2.4 Query Processor

This process accepts the input from the user, which includes title, abstract and optionally a category selection, and preprocess the user input to **compute cosine similarity or BM25** against indexes and/or embeddings. The values will then pass for ranking.

3.2.5 Ranking

This process combines scores from indexes, title similarity or scope similarity with optimised weights obtained through the hyperparameter tuning process described in Section 3.4. After that, it records ranked lists of top K journals per manuscript.

3.2.6 Evaluation

Evaluation is based on the training and test set split from the dataset, to give insights for the system performance. It produces per-query metrics including HitRate@K, MAP@K and NDCG@K. Top K values in this experiment are set to {1,

3, 5, 10}, and this applies to all models in this system. The details of these metrics are described in Section 3.3.

3.2.7 Presentation Layer

The option for presentation layer in this project is mainly considered with the ease of development and user-friendly approach. Therefore, **streamlit** from the Python library is utilized in this project to develop a simple web application of FlareSearch, a journal finder that recommends journal based on the paper's information.

The web app will accept input such as **Title, Abstract, and an optional category selection** which is displayed in the web app in the form of drop-down list for user to enter their paper information. After entering, it will display top journal suggestions with the journal's title, scope and link if available.

3.3 Evaluation Metrics

In FYP1, the proposed evaluation metrics were Accuracy@K and Recall@K, as these were the most commonly identified metrics in the literature. However, upon further investigation into the B!SON journal recommendation system by Entrup et al. [19] and the information retrieval literature, three revised metrics were adopted for FYP2. These metrics provide a more comprehensive evaluation of ranking quality and are more widely used in recommendation system benchmarks.

3.3.1 HitRate@K

HitRate@K measures the proportion of queries for which at least one document sharing the same category as the query appeared within the top-K ranked results. This is functionally **equivalent to the Accuracy@K** metric used in FYP1 and the **top@N accuracy** used in B!SON [19]. The formula is:

$$HitRate@K = \left(\frac{1}{N}\right) \times \sum hit(q, K) \tag{3.1}$$

Where N is the total number of queries, and hit(q, K) = 1 if any document in the top-K results shares the same primary category as query q, and 0 otherwise. HitRate@1 is equivalent to the top-1 accuracy reported in FYP1.

3.3.2 MAP@K (Mean Average Precision at K)

MAP@K addresses a limitation of HitRate@K: it does not distinguish between a relevant result at rank 1 versus rank 10. MAP@K computes the **average precision for each query**, which rewards systems that place relevant documents at higher ranks:

$$AP@K(q) = (\frac{1}{\min(R,K)} \times \sum_{\{k=1\}}^{\{K\}} Precision@k * rel(k)$$

(3.2)

Where R is the total number of relevant documents for query q, Precision(k) is the proportion of relevant documents among the top-k results, and rel(k) = 1 if the document at rank k is relevant. MAP@K is the mean of AP@K across all queries. This metric was adopted because it captures **ranking quality rather than just presence** in the top-K.

3.3.3 NDCG@K (Normalised Discounted Cumulative Gain at K)

NDCG@K accounts for the position of relevant documents in the ranked list by applying a logarithmic discount. Documents **ranked higher contribute more to the score** than those ranked lower:

$$DCG@K = \sum_{\{k=1\}}^{\{K\}} rel(k) \,/log_2(k+1)$$

(3.3)

The DCG is normalised by dividing by the ideal DCG (IDCG), which is the DCG achieved if all relevant documents were ranked at the top positions. NDCG@K = DCG@K / IDCG@K, yielding a value between 0 and 1.

3.3.4 Justification for Metric Change

The change from Accuracy/Recall to HitRate@K/MAP@K/NDCG@K was motivated by three factors. First, alignment with the reference B!SON framework [19],

which used **top@N accuracy** (equivalent to HitRate@K) as its primary metric. Second, the need to **evaluate ranking quality** (MAP@K, NDCG@K) rather than **just hit/miss accuracy**, as a recommender system that **consistently places the correct journal at rank 1** is more useful than one that places it at rank 10. Third, HitRate@1 is **mathematically equivalent to Accuracy@1**, ensuring backward compatibility with the FYP1 results.

3.4 Hyperparameter Tuning Methodology

3.4.1 Limitation of the Original Grid Search

In FYP1, the hyperparameter tuning strategy **was a manual fixed grid search**, where a small number of predetermined values were tested for each parameter (Table 3.3.4.1 in FYP1). For example, TF-IDF was limited to ngram_range ∈ {(1,1), (1,2)} and max_features ∈ {5000, 10000, 20000}, totalling only 6 combinations. This approach had two key limitations: it could not explore continuous parameter spaces (e.g., BM25's k1 and b), and it was **restricted to the researcher's intuition** about which values to test.

3.4.2 Bayesian Optimisation with Optuna

To address these limitations, Bayesian optimisation was adopted using the Optuna framework [20]. Optuna is an **open-source hyperparameter optimisation framework** that uses a Tree-structured Parzen Estimator (TPE) as its default sampling algorithm.

Unlike grid search, which evaluates every combination exhaustively, or random search, which samples combinations uniformly, Bayesian optimisation **learns from past evaluations** to focus the search on the **most promising regions** of the parameter space. The TPE sampler works by modelling two probability distributions: one for parameter values that produced good objective values, and one for values that produced poor results. At each iteration, it selects the next set of parameters by maximising the expected improvement ratio between these two distributions.

This approach offers several advantages over the original grid search. It can efficiently search continuous parameter spaces (e.g., BM25's k1 ∈ [0.5, 3.0] rather than

{1.2, 1.5, 1.8}). It requires fewer total evaluations to find good configurations because each trial is informed by previous results. It also supports mixed parameter types, including categorical (model variant), integer (ngram_max), and floating-point (weight values) parameters in a single search.

3.4.3 Cross-Validation Strategy

For each model, the objective function to be maximised was the mean HitRate@1 evaluated using stratified K-fold cross-validation on the training set. Stratified K-fold was chosen to ensure that each fold preserved the category distribution of the full training set, preventing evaluation bias from imbalanced category sizes.

For TF-IDF and SBERT, **5-fold cross-validation** was used. For BM25, 3-fold cross-validation was used instead due to the significantly **longer computation time** of the pure Python implementation in the rank_bm25 library, which lacked the vectorised matrix operations available to TF-IDF (scikit-learn) and SBERT (PyTorch with CUDA).

3.4.4 Search Spaces

The search spaces and configurations were as follows:

Table 3.4.4.1: Hyperparameter search configuration per model

Model	Hyperparameters	Search Range	Trials	CV Folds
TF-IDF	ngram_max	{1, 2, 3}	50	5-fold
	max_features	[3000, 30000]		
	sublinear_tf	{True, False}		
	min_df	{1, 2, 3, 4, 5}		
BM25	k1	[0.5, 3.0] continuous	30	3-fold
	b	[0.0, 1.0] continuous		

SBERT	w_title	[0.1, 0.8] continuous	100	5-fold
	w_scope	[0.1, 0.8] continuous		
	w_cat	1.0 − w_title − w_scope		
	model variant	{all-MiniLM-L6-v2, all-mpnet-base-v2}		

Stratified K-fold cross-validation was used to ensure each fold preserved the category distribution of the full training set. For BM25, 3-fold cross-validation was used instead of 5-fold due to the significantly longer computation time of the pure Python implementation.

3.5 Experimental Design

Two evaluation setups were designed to assess the robustness of each model:

1. **Random distribution** — The standard train/test split (17,426 training journals, 4,356 test journals) produced by the preprocessing pipeline with random_state=42.

2. **Equal distribution** — Following the methodology of B!SON [ref], a balanced subset was created by sampling an equal number of journals per category (20 for training, 5 for testing) from categories with sufficient members. This setup controlled for popularity bias, as the original dataset was heavily skewed towards categories such as Medicine (604 journals) while some categories contained as few as 1 journal.

Additionally, three input strategies were evaluated for each model:

- **Title only** — only the journal title was used as the query text.
- **Title + Scope** — the journal title, subjects and keywords were concatenated.
- **Title + Scope + Category** — the primary category label was appended to the above.

For TF-IDF and BM25, the category was appended as a text string to the query. For SBERT, the category was incorporated through a weighted category centroid similarity score.

3.6 System Requirement

3.6.1 Software

The software involved in this research is Visual Studio Code as the IDE, with Python 3.13.7 used as the environment. Key libraries include scikit-learn (TF-IDF), rank_bm25 (BM25), sentence-transformers (SBERT), Optuna (hyperparameter tuning), and Streamlit (web UI).

3.6.2 Hardware

The hardware involved in this research is a desktop device. The specification of hardware is illustrated in Table 3.4.2.1, including the desktop model (Motherboard), Processor, Operating System, Graphic cards, Memory capacity and Storage.

Table 3.6.2.1 Specifications of desktop

Description	**Specifications**
Motherboard	Gigabyte B760M Power
Processor	Intel(R) Core (TM) i5-14400F
Operating System	Windows 11 Pro
Graphic	NVIDIA GeForce RTX 4060 8GB GDDR6
Memory	32GB DDR5 5600MHz RAM
Storage	1.5TB NVMe SSD

3.7 Novelty aspects

This project is built on these aspects of novelty that may provide new insights into the future development of the recommendation system specifically in journal finder, which has a multi-publisher scope. From this approach, this research assembles and normalizes journal scope across publishers to create a **wider domain of recommendations**.

Hence, the **hybrid evaluation of input strategies** that is taken in this research. In which, obtaining another option of input strategies which is **category**, to narrow and **assist** the title-abstract input, which may help in increasing the overall precision of the system.

3.8 Project Timeline

The project is divided across FYP 1 & 2. In FYP 1 will do **partial data acquisition**, **preprocessing**, **implement Sentence-BERT pipeline**, **generate embeddings**, **implement evaluation framework** and produce initial results as well as a Streamlit prototype. In FYP2, will finalize **data acquisition**, **preprocessing**, **refine** Sentence-BERT model, **introduce TF-IDF/BM25** and another embedding models, conduct more extensive experiments, and prepare the final report and demonstration.

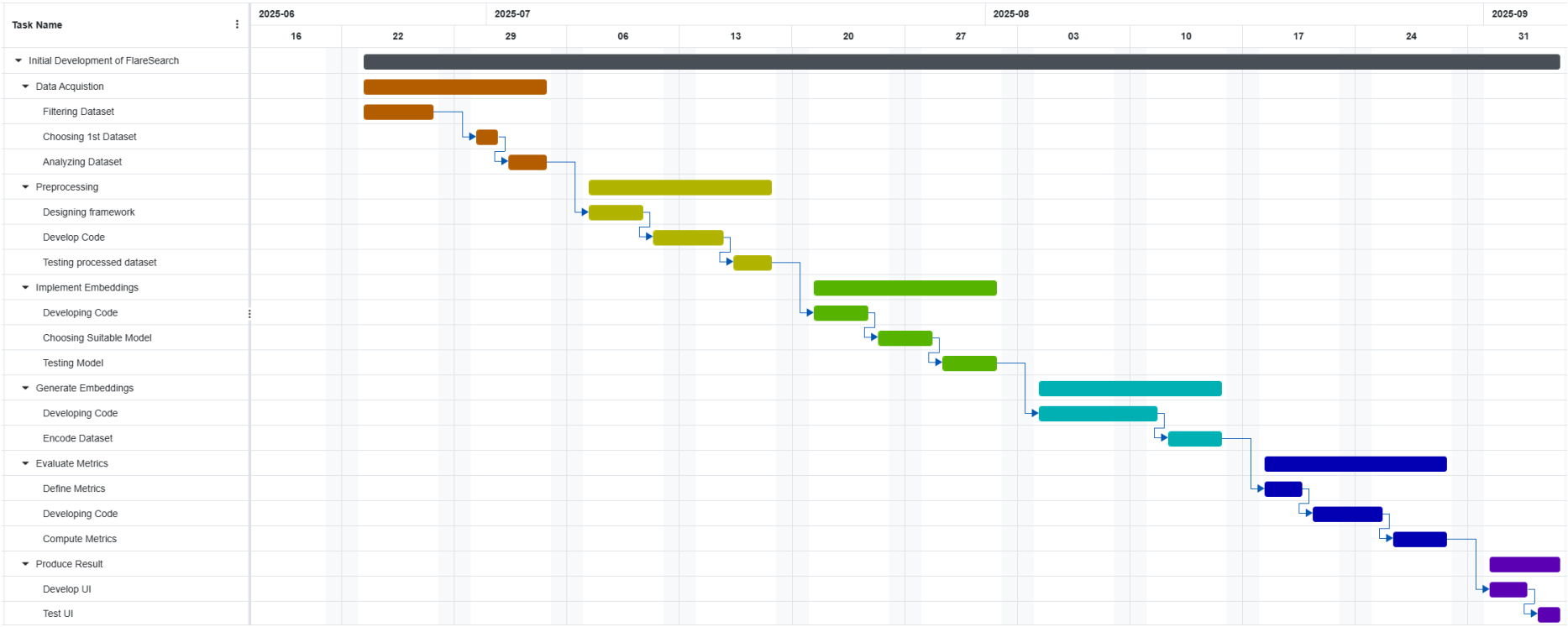

Figure 3.8.1: Gantt chart for the research and development of FlareSearch in FYP1

To ensure the flow of the experiment throughout the semester, **11 weeks** of the semester has been broken into **6 sprints**. Every sprint other than the last sprint length **2 weeks** to ensure that time is sufficient throughout the initial development of FlareSearch, which is a mini web application deliverable produced from this research. The last week of the semester are not included in the development as it is reserved for report writing.

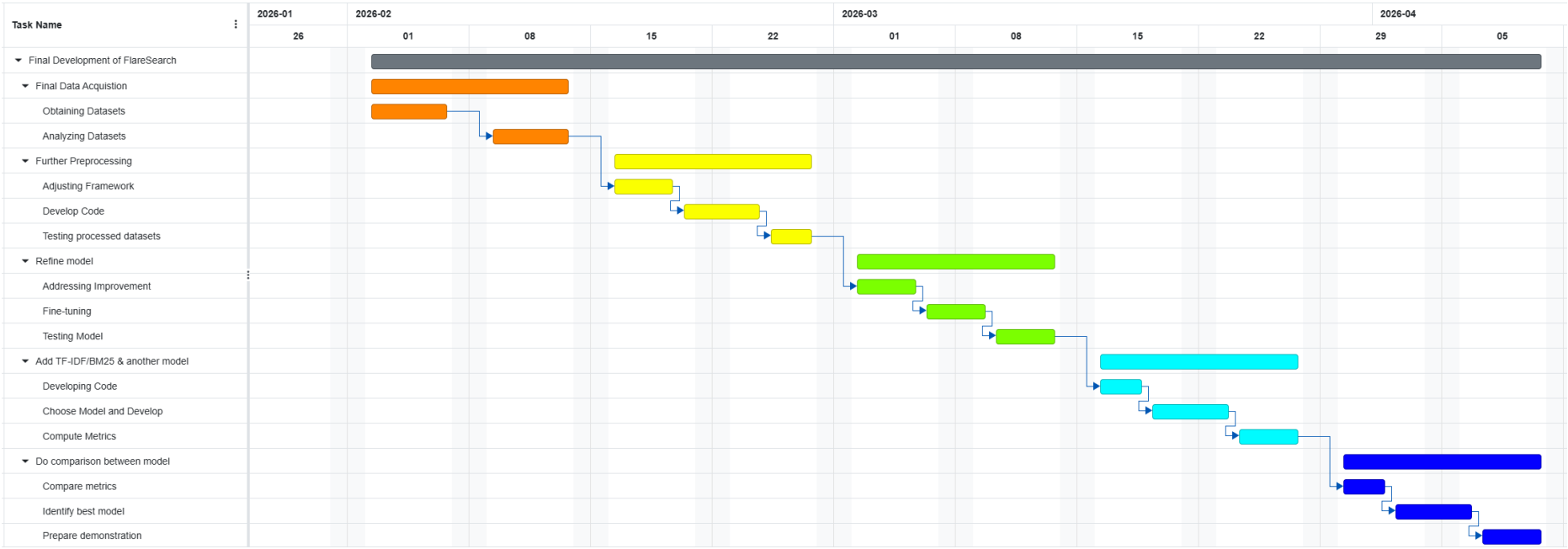

Figure 3.8.2: Gantt chart for the research and development of FlareSearch in FYP2

In FYP2, **10 weeks** from the semester has been broken into **5 sprints** and it lasts **2 weeks** per sprint to ensure the smoothness of the research. Hence, **around 3 weeks** before the submission deadline has been reserved to provide time for error resolving, perfecting system and report writing to ensure everything is well-documented and is good to go.

CHAPTER 4

System Design

This chapter describes in detail how the project was developed. The information provided should be sufficient for someone reading this chapter to rebuild the system.

4.1 System Block Diagram

Figure 4.1.1 shows the top-level system block diagram for FlareSearch. The system consists of two main layers: the data processing layer (offline, runs once) and the presentation layer (online, runs per query). All models share the same preprocessing and evaluation pipeline to ensure a fair comparison.

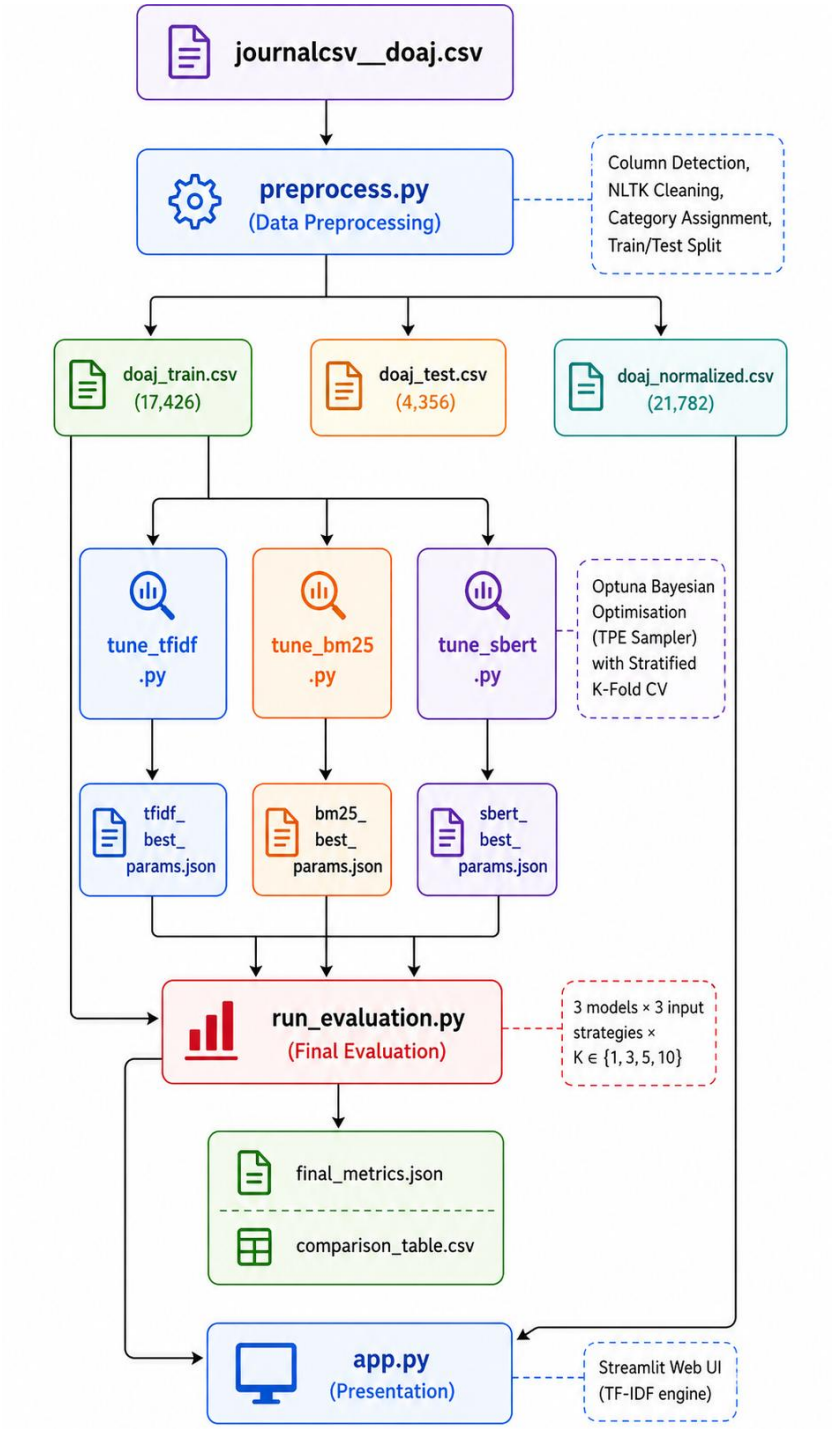

Figure 4.1.1: System Block Diagram

As shown in Figure 4.1.1, the system follows a pipeline architecture. Raw data enters the preprocessing module, which produces the normalised dataset and train/test splits. The three tuning scripts then operate independently on the training set to find the best hyperparameters for each model. The evaluation runner loads these parameters and produces the final benchmark results. Finally, the web application uses the best-performing model (TF-IDF) to serve real-time journal recommendations.

4.2 Project Directory Structure

Table 4.2.1 lists all project files and their responsibilities. This structure ensures that each script has a single responsibility and can be executed independently.

Table 4.2.1: Project file structure

File	Purpose	Input	Output
preprocess.py	Data cleaning, normalisation, train/test split	journalcsv__doaj.csv	doaj_train.csv, doaj_test.csv, doaj_normalized.csv
utils.py	Shared utility functions	—	—
tune_tfidf.py	Optuna tuning for TF-IDF	doaj_train.csv	results/ tfidf_best_params.json
tune_bm25.py	Optuna tuning for BM25	doaj_train.csv	results/ bm25_best_params.json
tune_sbert.py	Optuna tuning for SBERT	doaj_train.csv	results/ sbert_best_params.json
run_ evaluation.py	Final evaluation	doaj_train.csv, doaj_test.csv, results/ _best_params.json	results/final_metrics.json, results/comparison_table.csv
eval_equal _distribution.py	Balanced evaluation	doaj_normalized.csv	results/ equal_distribution_metrics.json
eval_pe r_category.py	Per-category breakdown	doaj_train.csv, doaj_test.csv	results/ per_category_analysis.csv
app.py	Streamlit web application (TF-IDF engine)	doaj_normalized.csv, results/ tfidf_best_params.json	Web UI
requirements.txt	Python package dependencies	—	—

4.3 Data Preprocessing Pipeline

The preprocessing pipeline (preprocess.py) transforms the raw DOAJ CSV into a normalised, clean dataset ready for model training and evaluation. Figure 4.3.1 shows the preprocessing flowchart.

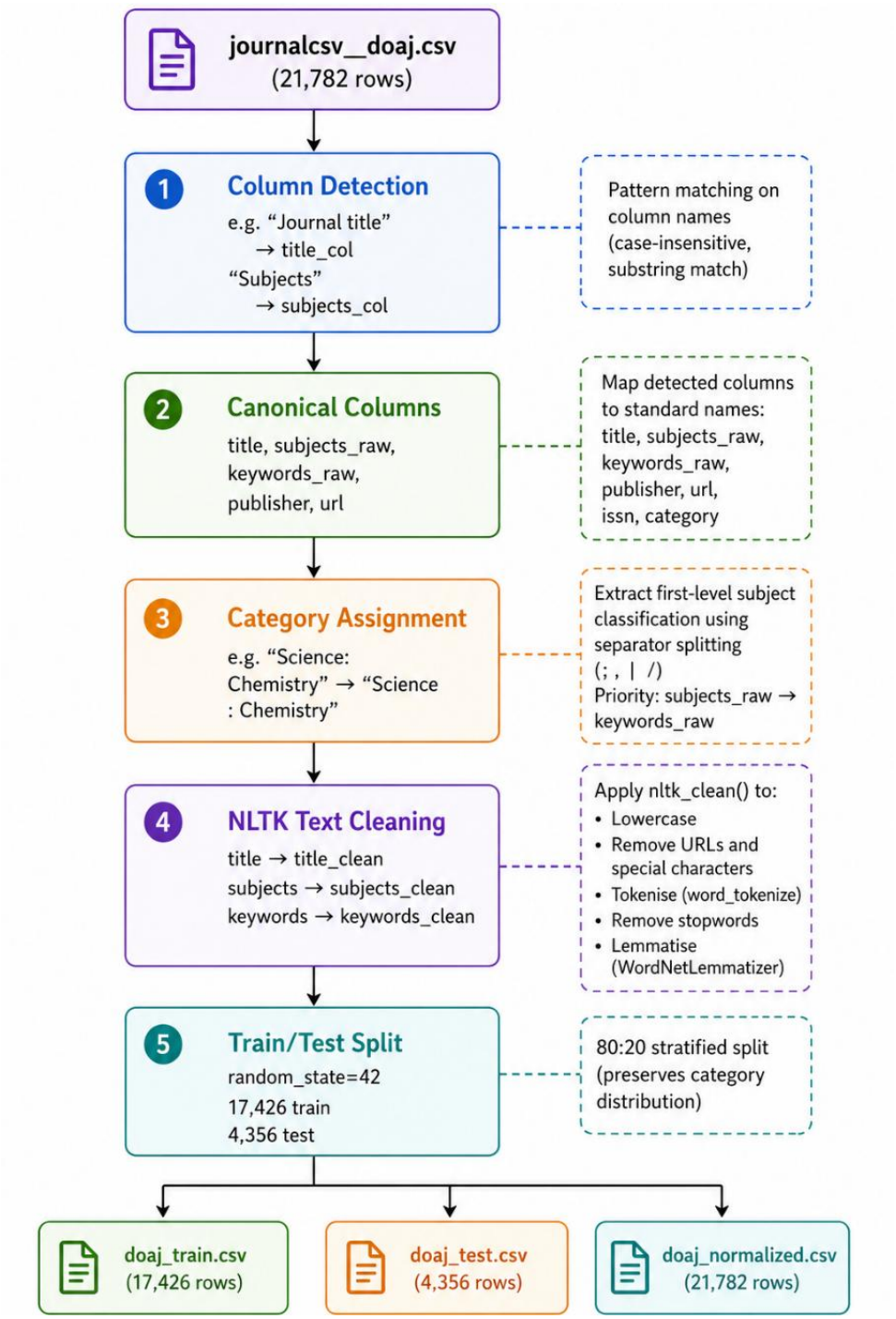

Figure 4.3.1: Preprocessing Flowchart

4.3.1 Column Detection

The script uses a candidate-based matching algorithm to detect columns from the raw CSV. For each target field (title, subjects, keywords, etc.), a list of candidate names is defined. The algorithm first attempts exact case-insensitive matching, then falls back to substring matching. For example, the title column candidates are ['journal title', 'title', 'paper title', 'name']. If the CSV contains a column named "Journal title", it would match on the first candidate.

This approach makes the preprocessing script robust to different CSV formats from different data sources, as column names may vary between publishers and datasets.

4.3.2 NLTK Text Cleaning

The nltk_clean() function in utils.py performs the following operations in sequence:

1. **Lowercase** — Convert all characters to lowercase to ensure case-insensitive matching.
2. **URL removal** — Remove any HTTP/HTTPS URLs using regex pattern http\S+.
3. **Whitespace normalisation** — Collapse multiple spaces into a single space.
4. **Tokenisation** — Split text into individual word tokens using NLTK's word_tokenize.
5. **Alphabetic filter** — Remove any token that contains non-alphabetic characters (e.g., numbers, punctuation).
6. **Stopword removal** — Remove English stopwords using NLTK's stopwords corpus (e.g., "the", "is", "in").
7. **Lemmatisation** — Reduce words to their base form using WordNetLemmatizer (e.g., "studies" → "study", "running" → "running").

4.3.3 Category Assignment

The **primary_cat** field is extracted from the subjects column using separator-based splitting. The algorithm checks for common separators (;, ,, |, /, -) and returns the first non-empty segment. If no separator is found and the string has 4 or fewer words, it is kept as-is; otherwise, the first 4 words are used.

For example, the DOAJ subjects field "Social Sciences: Economic theory. Demography | Economics" would produce a **primary_cat** of "Social Sciences: Economic theory. Demography".

4.3.4 Train/Test Split

The dataset is split using scikit-learn's train_test_split with stratified sampling on the **primary_cat** column and a fixed random_state=42. Stratified sampling ensures that each category appears in both the training and test sets in the same proportion as the original dataset. The 80:20 ratio was chosen to provide sufficient training data for TF-IDF and BM25 vocabulary building, while reserving enough test journals for statistically meaningful evaluation.

4.4 TF-IDF Model Design

4.4.1 Text Representation

The TF-IDF model represents each journal as a sparse vector in a high-dimensional vocabulary space. The scikit-learn TfidfVectorizer [18] is configured with the Optuna-tuned parameters:

- **ngram_range= (1, 3)** — Include unigrams, bigrams and trigrams. Character trigrams capture partial word matches (e.g., "bio" matching "biology" and "bioinformatics").
- **max_features= 3,000** — Restrict the vocabulary to the 3,000 most informative terms. This reduces noise from rare or overly specific terms.
- **sublinear_tf= True** — Apply logarithmic scaling to term frequency (1 + log(tf) instead of raw tf), which prevents long documents from being dominated by repeated terms.

- **min_df=5** — Ignore terms that appear in fewer than 5 documents, filtering out extremely rare terms.

4.4.2 Query Building

For each input strategy, the query text is constructed differently:

Table 4.4.2.1: Query of each input strategy

Strategy	**Query Text**
Title	title_clean
Title+Scope	title_clean + subjects_clean + keywords_clean
Title+Scope+Cat	title_clean + subjects_clean + keywords_clean + primary_cat

The query text is transformed into a TF-IDF vector using the fitted vectoriser (vectoriser.transform()), which maps the query into the same vocabulary space as the training documents.

4.4.3 Similarity Computation and Ranking

Cosine similarity is computed between the query vector and all training document vectors using scikit-learn's cosine_similarity function. This produces a similarity score between 0 and 1 for each training journal. The top-K journals with the highest scores are returned as recommendations.

The cosine similarity formula is:

$$\cos(q, d) = (q \cdot d)/(\left|\left|q\right|\right| \times \left|\left|d\right|\right|) \tag{4.1}$$

Where q is the query vector and d is the document vector. scikit-learn's implementation uses optimised sparse matrix operations, making this computation very fast (approximately 4 seconds for 4,356 queries against 17,426 documents).

4.5 BM25 Model Design

4.5.1 Text Representation

Unlike TF-IDF which uses vector space representations, BM25 operates directly on tokenised text. The training documents are split into word tokens by whitespace, and the BM25Okapi class builds an inverted index over the corpus. The two key parameters are:

- **k1 = 2.2247** — Controls term frequency saturation. Higher values give more weight to repeated terms. The tuned value of 2.2247 is higher than the typical default of 1.5, indicating that term repetition carries meaningful signal in journal metadata.
- **b = 0.9990** — Controls document length normalisation. A value near 1.0 means full normalisation, penalising longer documents. This was beneficial because journal scope texts varied significantly in length across categories.

4.5.2 Similarity Computation and Ranking

At query time, the test query is tokenised by whitespace, and BM25Okapi.get_scores() computes the BM25 score against every training document. The top-K documents are returned. Unlike TF-IDF which benefits from sparse matrix operations, BM25 scoring is implemented in pure Python, resulting in significantly longer computation times.

4.6 Sentence-BERT Model Design

4.6.1 Embedding Computation

The SBERT model encodes text into dense fixed-length vectors using pretrained transformer models. Two model variants were evaluated:

- **all-MiniLM-L6-v2 —** 22M parameters, 384-dimensional output, optimised for speed.
- **all-mpnet-base-v2 —** 110M parameters, 768-dimensional output, higher capacity.

Embeddings are computed in batches of 64 with L2 normalisation (normalize_embeddings=True), so that cosine similarity reduces to dot product. Four embedding sets are computed for the training data: title embeddings and scope embeddings, each stored as NumPy arrays.

4.6.2 Category Centroid Computation

To incorporate category information, a centroid vector is computed for each unique category. The centroid is the element-wise mean of all training journal scope embeddings belonging to that category:

$$centroid_c = \left(\frac{1}{|C|}\right) \times \sum_{\{d \in C\}} scope\_emb(d) \tag{4.2}$$

Where C is the set of all training journals in category c. This produces one centroid vector per category, representing the "average" scope of journals in that category.

4.6.3 Weighted Scoring

The similarity score for a test query q and a training document d is:

$$Score = w_{title} \times sim(title_q, title_d) + w_{scope} \times sim(scope_q, scope_d) + w\_cat \times sim(scope_q, centroid_{cat_d}) \tag{4.3}$$

The Optuna-tuned weights (w_title=0.1509, w_scope=0.1246, w_cat=0.7244) allocate 72.4% of the weight to the category centroid component. This means the model primarily classifies by category similarity, with title and scope acting as secondary signals.

4.7 Hyperparameter Tuning Design

Figure 4.7.1 shows the tuning pipeline flowchart that applies to all three models.

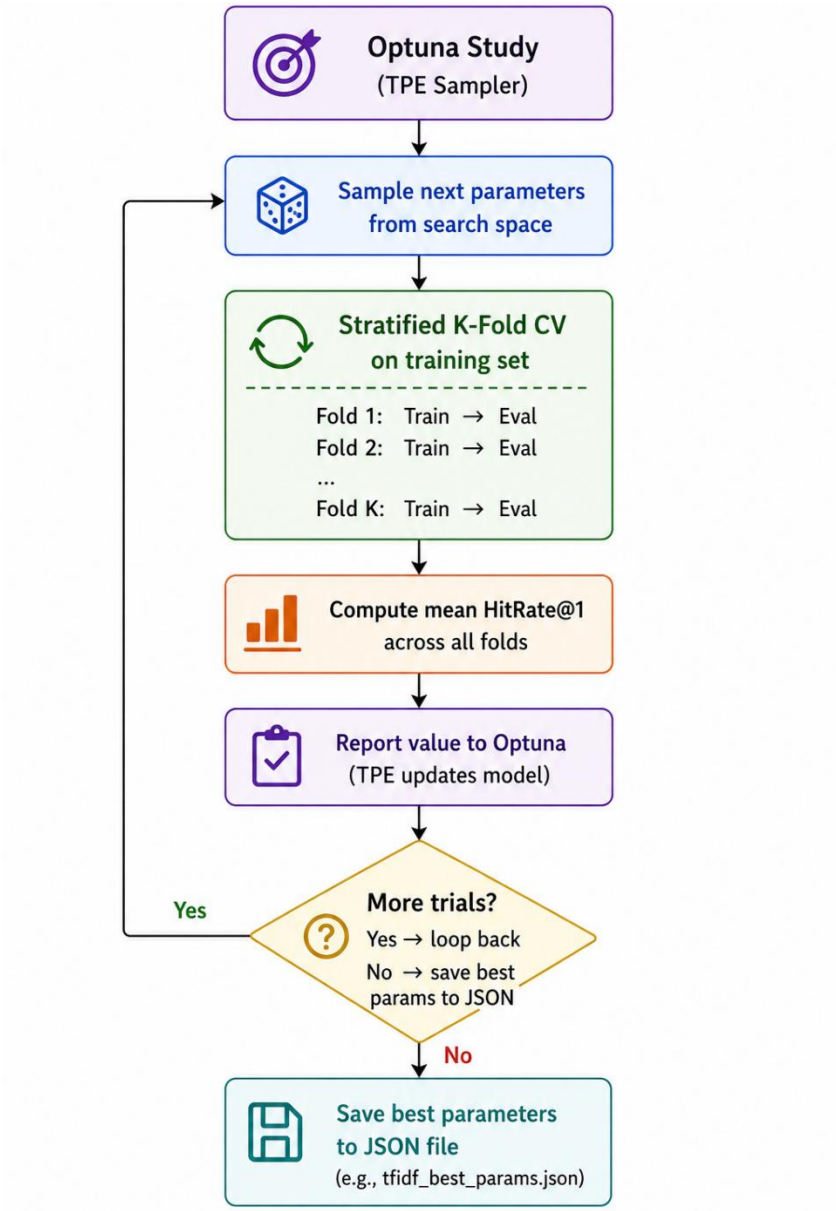

Figure 4.7.1: Hyperparameter Tuning Flowchart

Each tuning script follows this same pipeline. The only differences are the search space (Table 3.4.4.1), the number of trials, and the number of CV folds. The objective function builds the model with the sampled parameters, evaluates on validation folds, and returns the mean HitRate@1 to Optuna. The TPE sampler then uses this result to update its probabilistic model and sample better parameters in subsequent trials.

4.8 Evaluation Runner Design

The evaluation runner (run_evaluation.py) orchestrates the final benchmark. Figure 4.8.1 shows its flowchart.

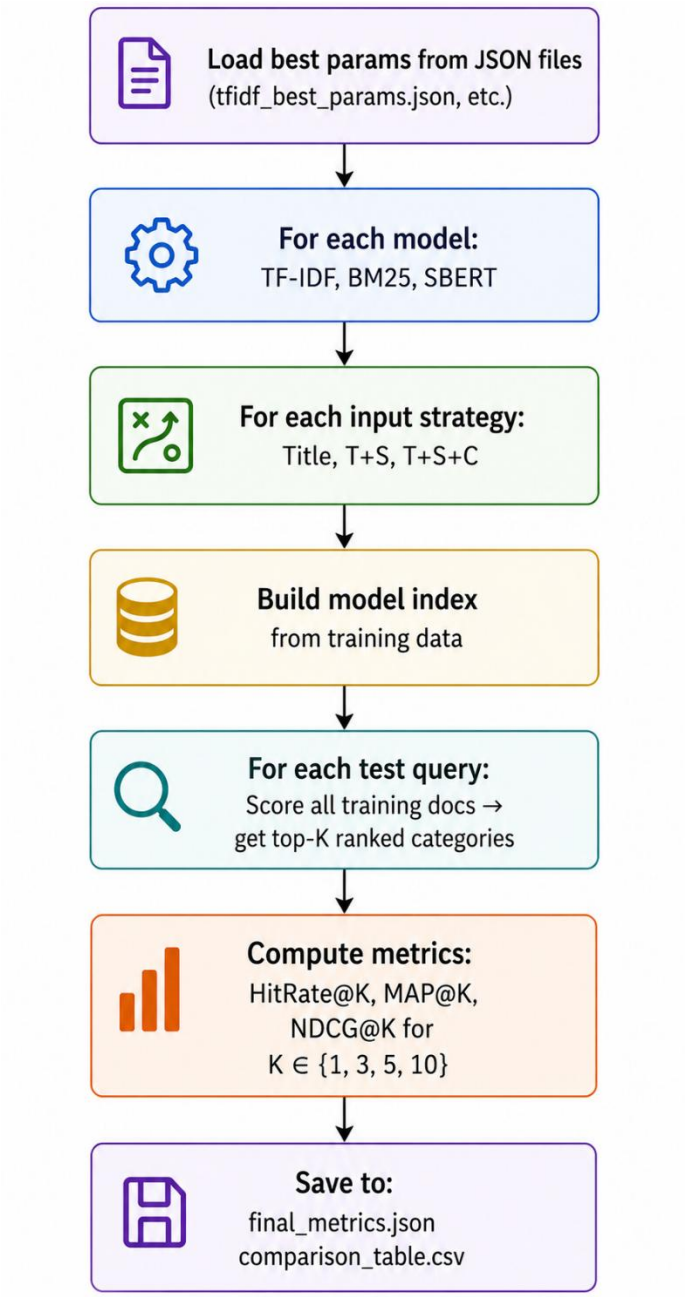

Figure 4.8.1: Evaluation Runner Flowchart

The runner executes 9 configurations in total (3 models × 3 input strategies). For each configuration, it builds the model with the best-tuned parameters, evaluates on the entire test set (4,356 queries), and records both the metrics and the wall-clock execution time. All results are saved to a single JSON file for consistency.

4.9 Presentation Layer Design

Based on the experimental results demonstrating that TF-IDF achieved the highest HitRate@1 (88.84%) and the fastest execution time (4.08 seconds), the FlareSearch web application was updated to utilise TF-IDF as its retrieval backend, replacing the Sentence-BERT model used in FYP1.

The web application (app.py) was built using Streamlit and accepts three inputs from the user: a manuscript title, an abstract, and an optional coarse category selection. The category dropdown presents the first-level subject classification from the DOAJ dataset (e.g., "Agriculture", "Medicine", "Technology").

When a user submits a query, the application performs the following steps:

1. **NLTK cleaning** — The user's title and abstract are cleaned using the same nltk_clean() function used in the training pipeline, ensuring vocabulary consistency.
2. **Category resolution** — If a coarse category is selected, the application maps it back to the full **primary_cat** string from the dataset and appends it to the cleaned query.
3. **TF-IDF transformation** — The cleaned query is transformed into a sparse vector using the pre-fitted TF-IDF vectoriser.
4. **Cosine similarity** — Similarity scores are computed against all journal vectors (or filtered candidates if a category is selected).
5. **Ranking and display** — Journals above a minimum similarity threshold are displayed in descending order of their score, showing the journal title, publisher, scope and URL.

The Optuna-optimised hyperparameters (ngram_max=3, max_features=3,000, sublinear_tf=True, min_df=5) are loaded automatically from the tuning results JSON file at startup.

4.10 System Components Interaction

Table 4.10.1 summarises how the key modules interact with each other through their inputs and outputs.

Table 4.10.1: System component interactions

Step	Script	Depends On	Produces	Used By
1	preprocess.py	journalcsv__doaj.csv	doaj_train.csv, doaj_test.csv	All tuning, evaluation runner
2	tune_tfidf.py	doaj_train.csv, utils.py	tfidf_best_params.json	run_evaluation.py, app.py
3	tune_bm25.py	doaj_train.csv, utils.py	bm25_best_params.json	run_evaluation.py
4	tune_sbert.py	doaj_train.csv, utils.py	sbert_best_params.json	run_evaluation.py
5	run_evaluation.py	doaj_train.csv, doaj_test.csv, *_best_params.json, utils.py	final_metrics.json	Report writing
6	eval_equal_distribution.py	doaj_normalized.csv, *_best_params.json	equal_distribution_metrics.json	Report writing
7	eval_per_category.py	doaj_train.csv, doaj_test.csv, *_best_params.json	per_category_analysis.csv	Report writing
8	app.py	doaj_normalized.csv, tfidf_best_params.json, utils.py	Web UI	End user

Steps 1 through 7 are executed offline in sequence. Step 8 is the online serving layer that loads the artifacts produced by the offline pipeline.

4.11 How to Rebuild the System

To rebuild the system from scratch, execute the following commands in order:

Step 1: Install dependencies

pip install -r requirements.txt

Step 2: Preprocess the raw DOAJ CSV

python preprocess.py --infile journalcsv__doaj.csv --out_prefix doaj

Step 3: Tune hyperparameters (can run in parallel)

python tune_tfidf.py

python tune_bm25.py

python tune_sbert.py

Step 4: Run final evaluation

python run_evaluation.py

Step 5: Run additional experiments

python eval_equal_distribution.py

python eval_per_category.py

Step 6: Launch the web application

streamlit run app.py

CHAPTER 5
Experiment

5.1 Software Setup

The experiment was conducted using Python 3.13.7 with the following key packages: **scikit-learn, rank-bm25, sentence-transformers, optuna, torch** (with CUDA 12.x support), **pandas, numpy, and matplotlib**. A virtual environment (.venv) was created in the project directory to isolate dependencies.

5.2 Hardware Setup

All experiments were executed on a desktop machine with an Intel Core i5-14400F processor, 32GB DDR5 RAM, and an NVIDIA GeForce RTX 4060 8GB GPU. The GPU was used for SBERT embedding computation via CUDA, while TF-IDF and BM25 ran on CPU only.

5.3 Hyperparameter Tuning Process

The tuning process was executed sequentially for each model. Each tuning script prints progress information including trial numbers, sampled parameters, cross-validation scores, and the best result found so far.

5.3.1 Data Preprocessing

Before tuning, the raw DOAJ CSV was preprocessed using preprocess.py. Figure 5.3.1 shows the terminal output of the preprocessing step, including the detected columns, the number of rows processed, and the train/test split sizes

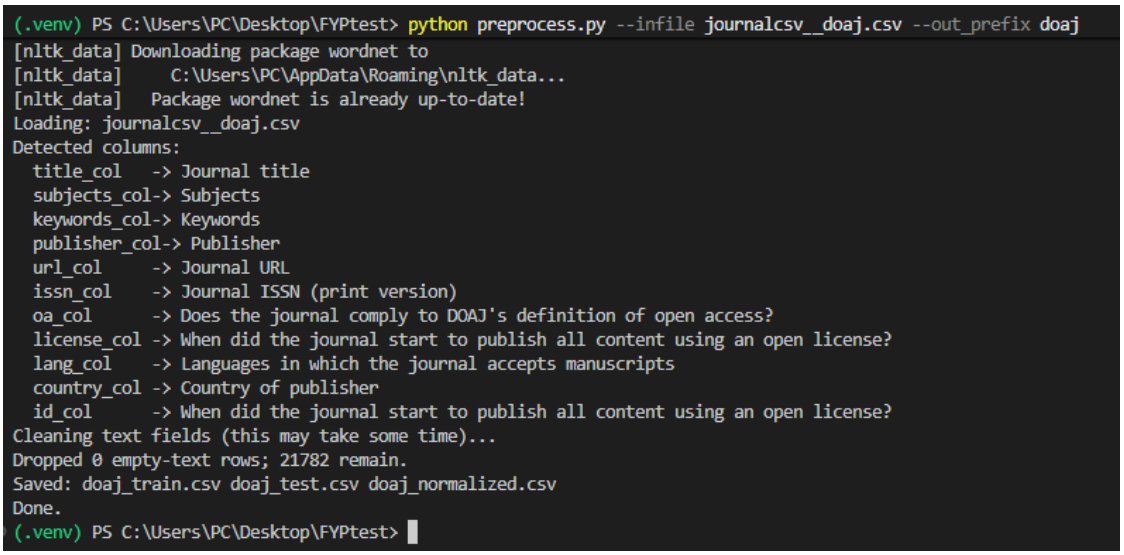

Figure 5.3.1.1: Terminal screenshot — python preprocess.py

As shown in Figure 5.3.1, the script automatically detected the title, subjects, keywords, publisher and URL columns from the raw CSV. After cleaning and category assignment, the dataset was split into 17,426 training journals and 4,356 test journals.

5.3.2 TF-IDF Tuning

TF-IDF tuning was executed with 50 trials and 5-fold stratified cross-validation. Figure 5.3.2 shows the terminal output during the tuning process. Each line represents one completed trial, showing the trial number, the sampled parameters, and the resulting CV HitRate@1.

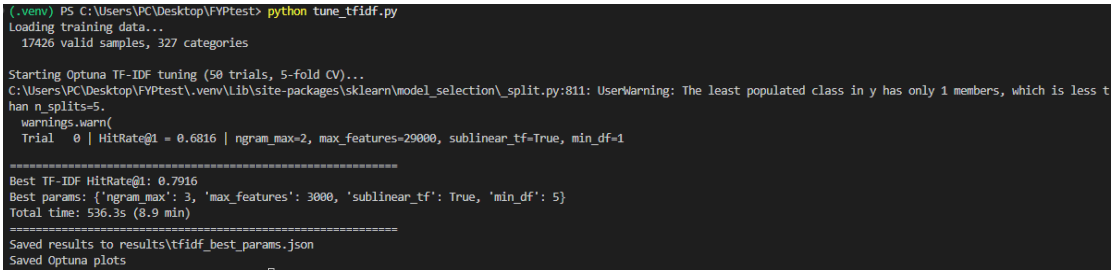

Figure 5.3.2: Terminal screenshot — python tune_tfidf.py

As shown in Figure 5.3.2, the optimiser explored a wide range of parameter combinations. The best configuration (ngram_max=3, max_features=3000, sublinear_tf=True, min_df=5) achieved a CV HitRate@1 of 79.16%. The entire process completed in approximately 9 minutes.

5.3.3 BM25 Tuning

BM25 tuning was executed with 30 trials and 3-fold stratified cross-validation. The number of folds was reduced from 5 to 3 due to the significantly longer computation time of the pure Python BM25 implementation. Figure 5.3.3 shows the terminal output during BM25 tuning.

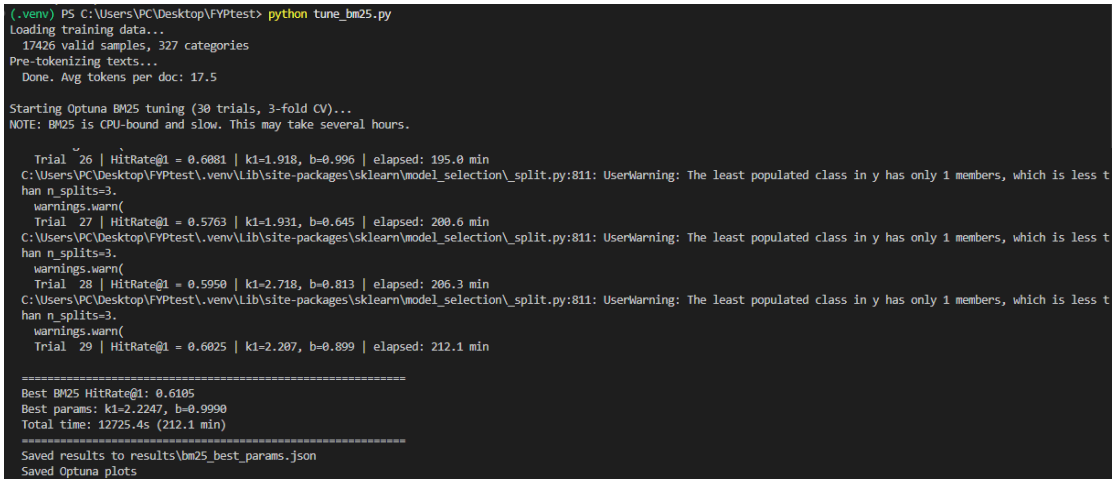

Figure 5.3.3: Terminal screenshot — python tune_bm25.py

As shown in Figure 5.3.3, the optimiser converged on k1=2.2247 and b=0.9990, achieving a CV HitRate@1 of 61.05%. The entire process completed in approximately 4 hours, making BM25 the most time-consuming model to tune.

5.3.4 SBERT Tuning

SBERT tuning was executed with 100 trials and 5-fold stratified cross-validation. GPU acceleration via CUDA was used for embedding computation. Figure 5.3.4 shows the terminal output during SBERT tuning.

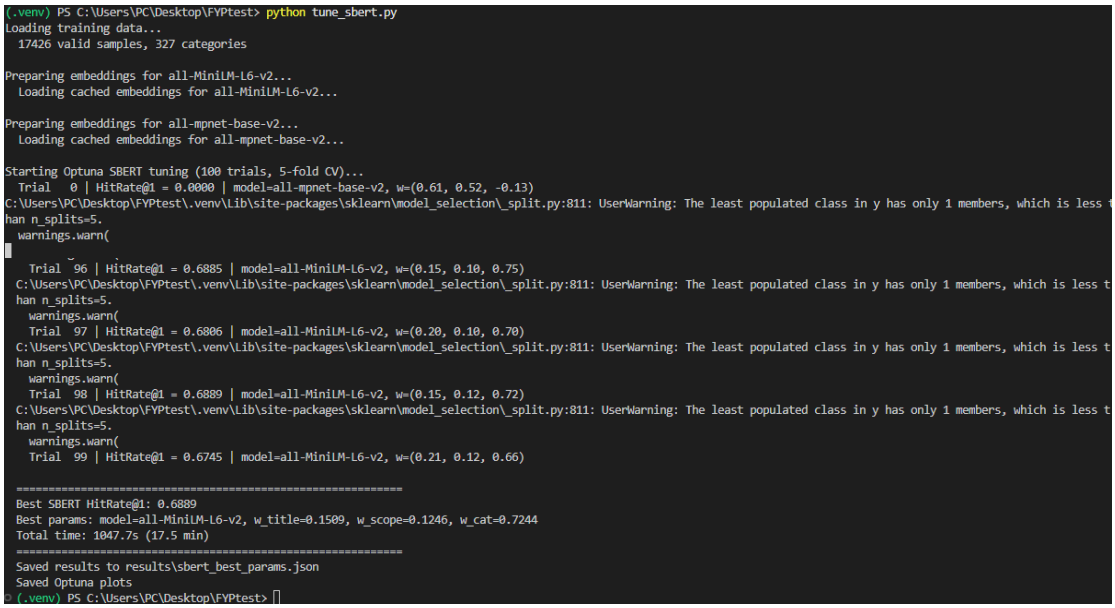

Figure 5.3.4: Terminal screenshot — python tune_sbert.py

As shown in Figure 5.3.4, the optimiser selected all-MiniLM-L6-v2 with w_title=0.1509, w_scope=0.1246, and w_cat=0.7244, achieving a CV HitRate@1 of 68.89%. The high weight on the category centroid (72.4%) indicated that category-level matching was the most discriminative signal for the SBERT model. The process completed in approximately 17 minutes.

5.4 Final Evaluation Execution

After tuning, the evaluation runner was executed to evaluate all three models across all three input strategies on the held-out test set. Figure 5.4.1 shows the terminal output of the evaluation runner, displaying the loaded parameters and the computed metrics for each model-strategy combination.

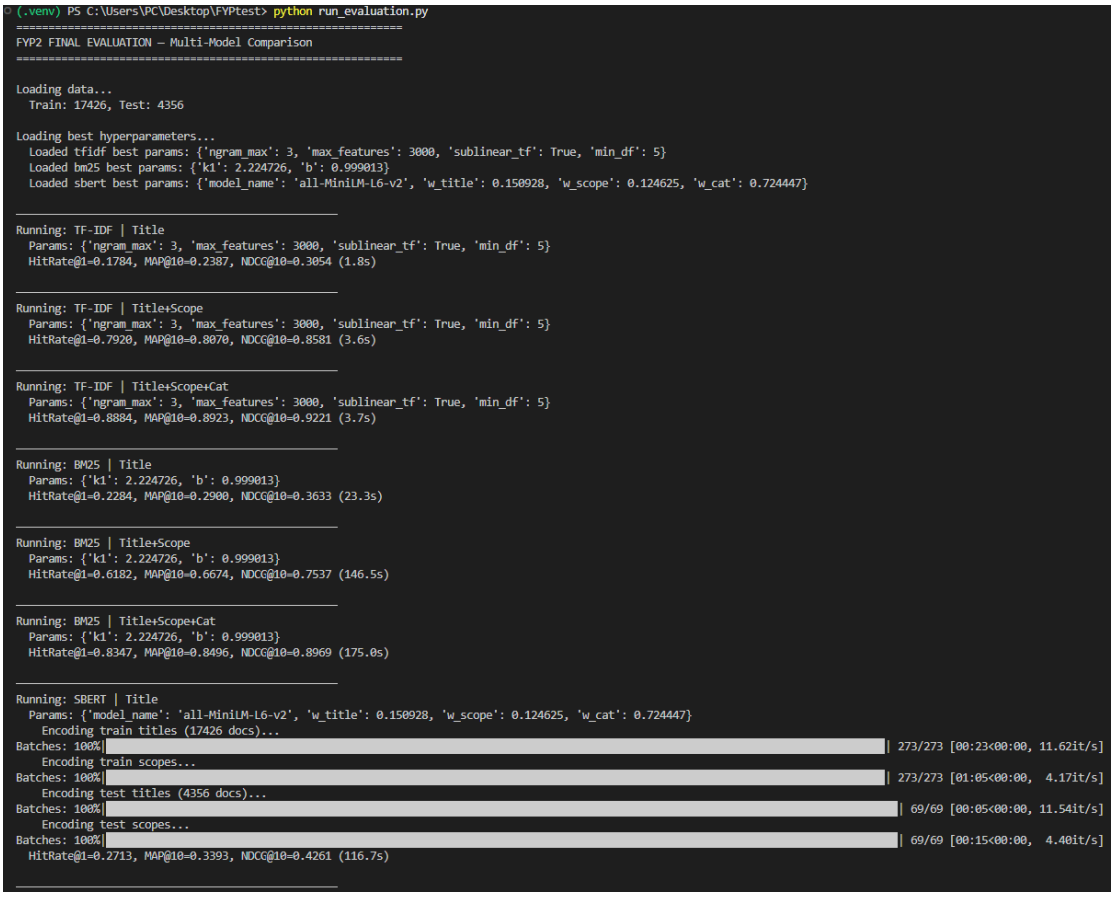

Figure 5.4.1: Terminal screenshot — python run_evaluation.py

As shown in Figure 5.4.1, the runner evaluated 9 configurations in total (3 models × 3 input strategies) and recorded HitRate@K, MAP@K, and NDCG@K at K ∈ {1, 3, 5, 10}. The full evaluation completed in approximately 10 minutes, with BM25 Title+Scope+Cat being the slowest individual configuration at 175 seconds. All results were saved to results/final_metrics.json and results/comparison_table.csv.

5.5 Equal Distribution Experiment

The equal distribution experiment was conducted to test whether the model rankings changed when category imbalance was removed. This experiment followed the B!SON methodology [19], where an equal number of journals per category was sampled to eliminate popularity bias.

The script sampled 20 training journals and 5 test journals per qualifying category (categories with at least 20 training and 5 test journals). This produced 122 qualifying categories out of 327 total, with 2,440 balanced training journals and 610 balanced test journals. Figure 5.5.1 shows the terminal output of the equal distribution experiment.

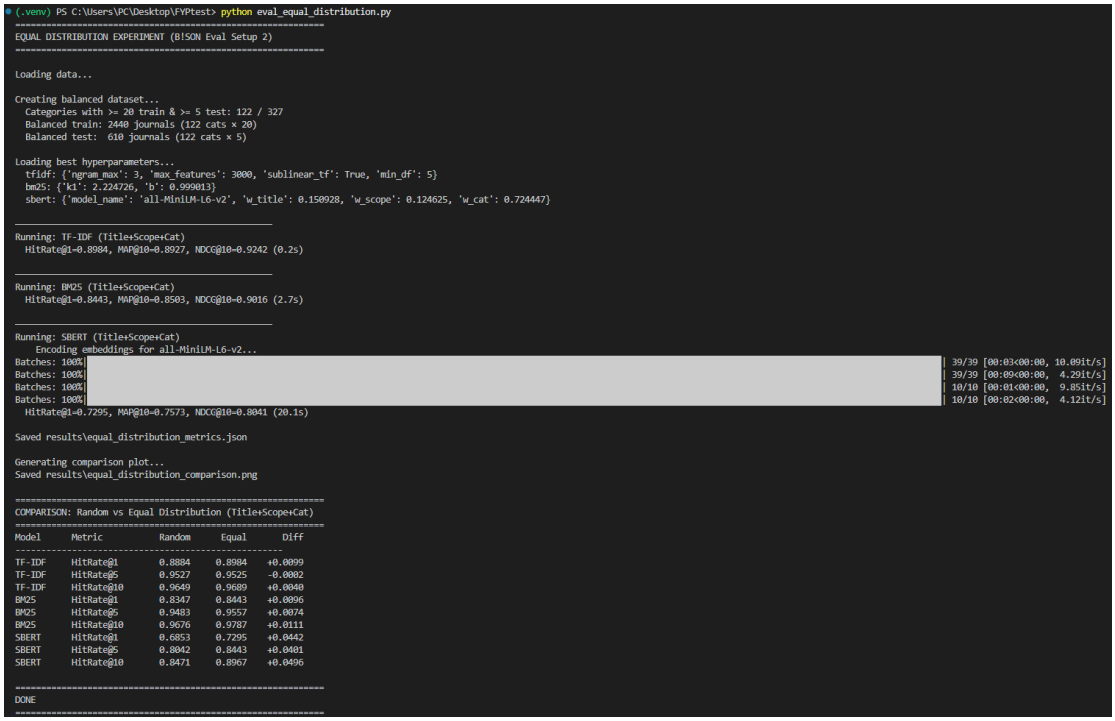

Figure 5.5.1: Terminal screenshot — python eval_equal_distribution.py

As shown in Figure 5.5.1, all three models showed slight improvements under the equal distribution setup compared to the random distribution. The results were saved to results/equal_distribution_metrics.json.

5.6 Per-Category Analysis

A per-category breakdown was conducted on the full test set to identify which subject areas were easiest and hardest for each model. Categories with fewer than 5 test samples were excluded, leaving 135 categories for analysis. Figure 5.6.1 shows the terminal output of the per-category analysis, displaying the top 10 easiest and bottom 10 hardest categories.

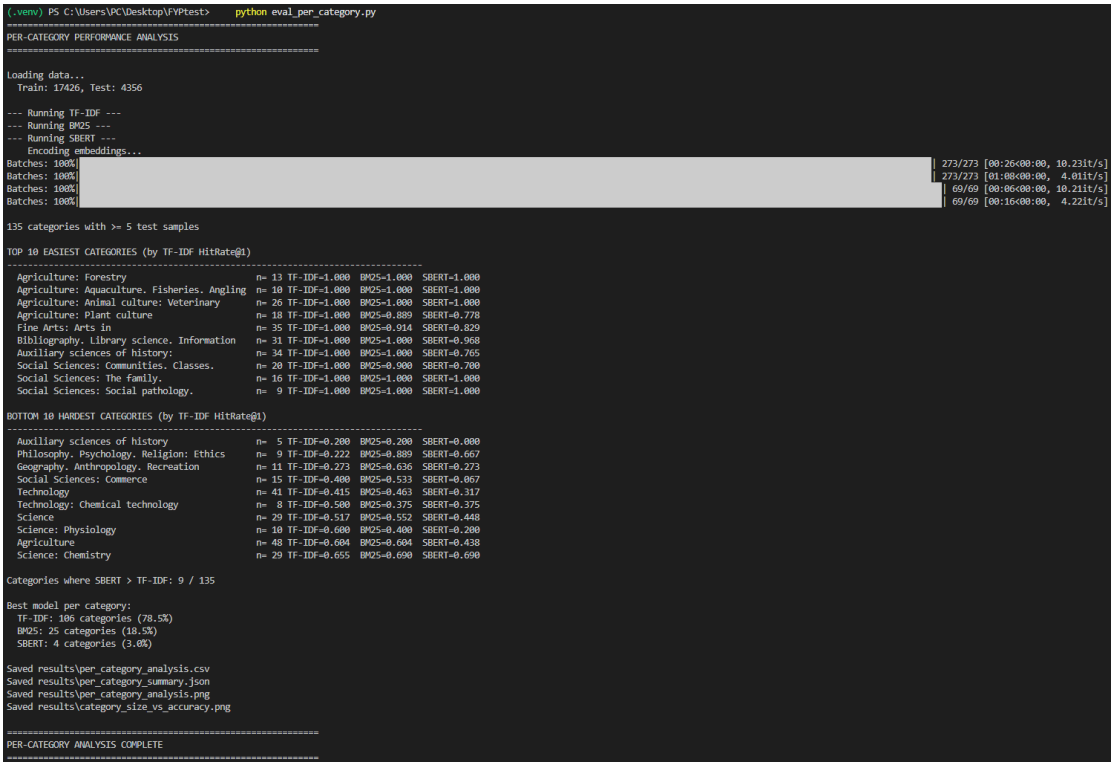

Figure 5.6.1: Terminal screenshot — python eval_per_category.py

As shown in Figure 5.6.1, the easiest categories (e.g., "Agriculture: Forestry") achieved 100% HitRate@1 across all models, while the hardest categories (e.g., "Technology", "Science") ranged from 20% to 50%. The script also reported the best model distribution: TF-IDF was the best model in 106 out of 135 categories (78.5%).

5.7 Implementation Issues and Challenges

The rank_bm25 library is implemented in **pure Python without vectorised operations**, making it significantly slower than TF-IDF (scikit-learn) and SBERT (PyTorch). To manage tuning time, the number of CV folds was **reduced from 5 to 3 for BM25**.

During initial development, the NDCG@K implementation normalised against an ideal DCG of 1.0 (assuming only one relevant document), which produced **values exceeding 1.0 when multiple relevant documents existed**. This was corrected by **computing the ideal DCG dynamically** based on the actual number of relevant documents in the candidate pool.

The initial evaluation runner **did not append the category string** to the query text for TF-IDF and BM25, meaning the Title+Scope+Cat strategy **produced identical results** to Title+Scope for these models. This was identified and corrected by updating the build_query_text function to include the **primary_cat** string when the category strategy was selected.

5.8 System Operation

The FlareSearch web application was launched using the command **streamlit run app.py** in the project directory. Upon startup, the application loaded the normalised journal dataset and built the TF-IDF index using the Optuna-optimised parameters. The index building process completed in approximately 1 second, compared to approximately 10 seconds required for loading the Sentence-BERT model in the FYP1 version.

Figure 5.8.1 shows the user interface of FlareSearch. The interface provides three input fields: Title, Abstract, and an optional Subject dropdown for category filtering.

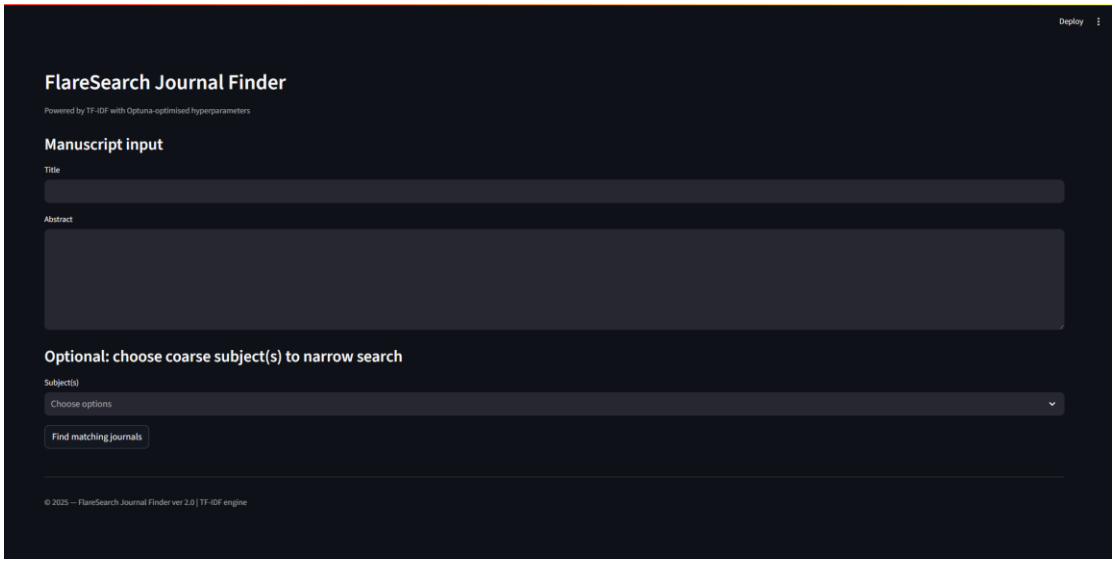

Figure 5.8.1: User Interface of FlareSearch

In the Subject section, a drop-down list appears to show the coarse categories available in the dataset, for users to choose and include as their input for a **more precise search**. When a user enters their manuscript details and presses "Find matching journals", the application **computes the TF-IDF cosine similarity** and displays the top matched journals **ordered by their similarity score**.

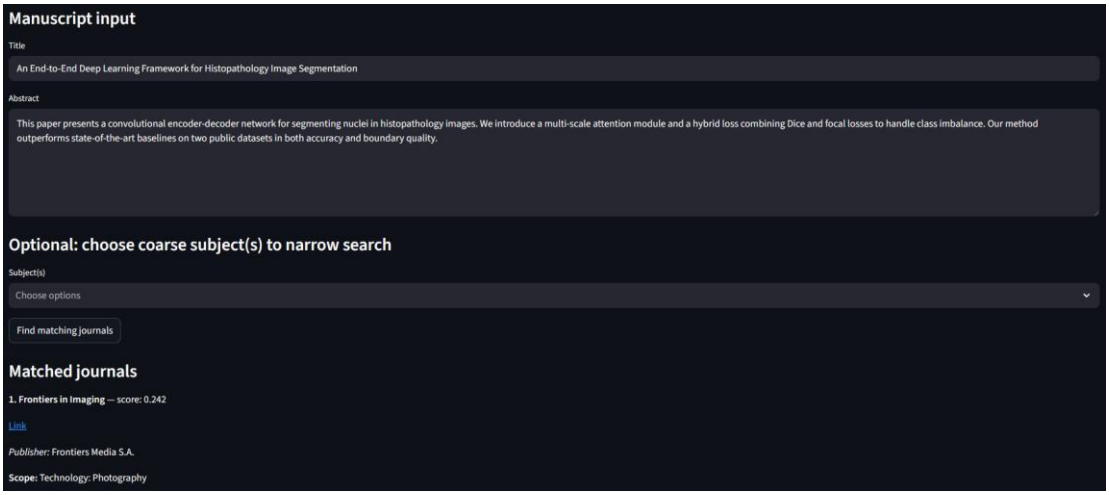

Figure 5.8.2: Matched results after entering manuscript details

In Figure 5.8.2, after entering manuscript information, the matched journals are shown in descending order of their similarity scores. Information on the journals shown includes the title, scope, publisher and the URL to the journal. The similarity scores displayed are raw **cosine similarity values ranging from 0 to 1**, where higher values indicate stronger textual overlap between the query and the journal scope.

A score threshold was set so that a journal is shown only when it meets the minimum similarity requirement. Therefore, **irrelevant results are filtered out** when the search is out of scope. Instead of showing poor matches, the system displays "No suitable journal found" when no journal exceeds the threshold. **The score will only be displayed in experimental process**; it will not be displayed to the user.

5.9 Web Deployment

To make FlareSearch accessible without local installation, the application was deployed as a public website using Streamlit Community Cloud. Streamlit Community Cloud is a free hosting service that automatically builds and deploys Streamlit applications directly from a GitHub repository.

5.9.1 Deployment Architecture

The deployment follows a simple architecture: the source code and data files are stored in a GitHub repository, and Streamlit Community Cloud pulls from the repository to build and serve the application. When a user visits the deployed URL, Streamlit Cloud runs app.py on its servers and streams the UI to the user's browser. No installation or setup is required from the end user.

5.9.2 Required Files

Table 5.9.1 lists the files that were uploaded to the GitHub repository for deployment. Only files required by the web application are included; training scripts, evaluation scripts, and experiment results are excluded as they are not needed at runtime.

Table 5.9.2.1: Files required for Streamlit Cloud deployment

File	Purpose	Size
app.py	Main Streamlit application	~10 KB
utils.py	Shared utilities (nltk_clean function)	~4 KB
doaj_normalized.csv	Full normalised journal dataset (21,782 journals)	~20 MB
results/ tfidf_best_params.json	Optuna-tuned TF-IDF hyperparameters	~10 KB
requirements.txt	Python package dependencies (pandas, numpy, scikit-learn, nltk, streamlit)	<1 KB
packages.txt	System-level dependencies for Streamlit Cloud (libgomp1)	<1 KB

.streamlit/config.toml	Streamlit theme and server configuration	<1 KB

The requirements.txt for deployment contains only the packages used by app.py (pandas, numpy, scikit-learn, nltk, streamlit), excluding packages used only for training and evaluation (sentence-transformers, optuna, rank-bm25, matplotlib). This reduces the deployment size and build time.

5.9.3 Deployment Steps

The deployment was performed using the following steps:

1. A GitHub repository was created and the files listed in Table 5.9.1 were pushed to the repository.
2. On the Streamlit Community Cloud dashboard (share.streamlit.io), the GitHub repository was linked by selecting the repository name and specifying app.py as the main file.
3. Streamlit Cloud automatically installed the dependencies from requirements.txt and packages.txt, then launched the application.
4. The application was assigned a public URL (https://flaresearch.streamlit.app) accessible from any browser.

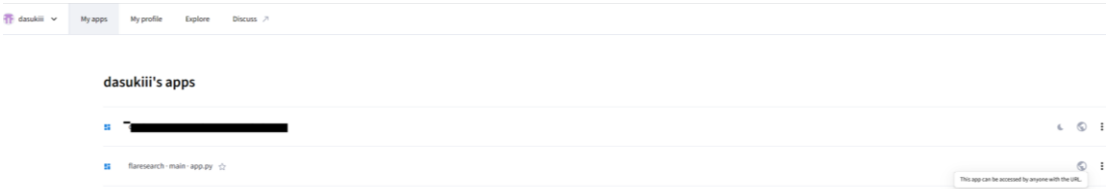

Figure 5.9.3.1: Screenshot of the deployed FlareSearch on Streamlit Cloud

CHAPTER 6

System Evaluation and Discussion

6.1 Performance Metrics

Three evaluation metrics used were HitRate@K, MAP@K and NDCG@K as defined in Section 3.3. All metrics were evaluated at K ∈ {1, 3, 5, 10} across three input strategies (Title, Title+Scope, Title+Scope+Cat) and three models (TF-IDF, BM25, SBERT).

6.2 Hyperparameter Tuning Results

The Optuna Bayesian optimisation framework [20] was executed for all three models. The best performing configurations are reported in Table 6.2.1.

Table 6.2.1: Best hyperparameters found by Optuna

Model	Parameter	Optimal Value	CV HitRate@1
TF-IDF	ngram_max	3	79.16%
	max_features	3,000	
	sublinear_tf	True	
	min_df	5	
BM25	k1	2.2247	61.05%
	b	0.9990	
SBERT	model	all-MiniLM-L6-v2	68.89%
	w_title	0.1509	
	w_scope	0.1246	
	w_cat	0.7244	

For TF-IDF, character trigrams (ngram_max=3) **consistently outperformed** unigrams and bigrams across all trials. A smaller vocabulary (max_features=3,000) performed better than larger vocabularies, suggesting that restricting the feature space

reduced noise from rare or irrelevant terms. This was an unexpected finding, as the FYP1 proposal tested max_features values of 5,000 to 20,000.

For BM25, the optimal parameters (k1=2.2247, b=0.9990) indicated a preference for high term frequency saturation and near-maximum document length normalisation. The **high b value** suggested that normalising document length was **beneficial** in this journal metadata retrieval task, where document lengths varied significantly across categories.

For SBERT, the optimiser allocated 72.4% of the weight to the category centroid component, with only 15.1% for title similarity and 12.5% for scope similarity. This indicated that the **category centroid was the most discriminative signal** available to SBERT. The all-MiniLM-L6-v2 model (22M parameters) was selected over all-mpnet-base-v2 (110M parameters), indicating that the larger model did not provide meaningful improvements for this task.

Figure 6.2.1a: TF-IDF optimization history

Figure 6.2.1b: BM25 optimization history

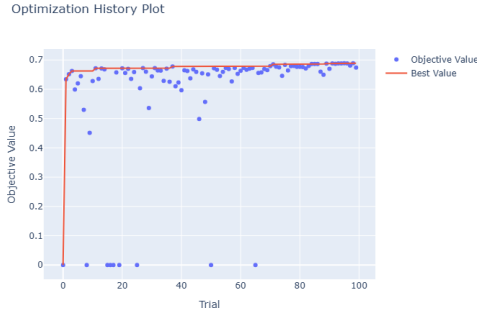

Figure 6.2.1c: SBERT optimization history

As shown in Figure 6.2.1, all three models exhibited clear convergence behaviour. For TF-IDF, the best value improved rapidly in the first 15 trials and plateaued at 79.16% by trial 34, confirming that 50 trials **were sufficient**. For BM25, the best value was found early (trial 0, at 60.0%) and improved gradually to 61.05% by trial 18, with the remaining trials producing **no further improvement**. For SBERT, the best value rose sharply within the first 5 trials and **stabilised around 68.89% by trial 20**, with the remaining 80 trials confirming that the search space had been thoroughly explored. The convergence of all three models **validated** that the Bayesian optimisation approach produced **reliable, near-optimal hyperparameter configurations**.

6.3 Testing Setup and Results

Figure 6.3.1: Model comparison result bar chart

6.3.1 Model Comparison — Random Distribution

The three models were evaluated on the held-out test set (4,356 queries) using their respective **optimal hyperparameters**. Table 6.3.1 presents the full results for the Title+Scope+Cat input strategy, which was the best-performing strategy for all models.

Table 6.3.1: Model comparison on random distribution (Title+Scope+Cat)

Model	HRate@1	HRate@3	HRate@5	HRate@10	MAP@10	NDCG@10	Time(s)
TF-IDF	88.84%	93.92%	95.27%	96.49%	89.23%	92.21%	3.71
BM25	83.47%	92.75%	94.83%	96.76%	84.96%	89.69%	175.04
SBERT	68.53%	77.46%	80.42%	84.71%	71.83%	76.29%	124.48

TF-IDF achieved the highest performance across most metrics, with a HitRate@1 of 88.84%, followed by BM25 at 83.47% and SBERT at 68.53%. This ranking **contradicted the initial expectation** stated in FYP1 report, where the expected performance ordering was BERT > Okapi BM25 > TF-IDF.

6.3.2 Effect of Input Strategy

Table 6.3.2 presents the HitRate@1 results across all input strategies.

Table 6.3.2: Effect of input strategy on HitRate@1

Model	Title Only	Title+Scope	Title+Scope+Cat	Improvement (Title → T+S+C)
TF-IDF	17.84%	79.20%	88.84%	+71.00%
BM25	22.84%	61.82%	83.47%	+60.63%
SBERT	27.13%	45.94%	68.53%	+41.40%

The inclusion of scope information (subjects and keywords) **dramatically improved** performance for all models. Adding the category label further boosted HitRate@1 by approximately 10–22 percentage points. The actual improvement from Title+Scope to Title+Scope+Cat ranged from 9.64% (TF-IDF) to 22.59% (SBERT), far **exceeding the 2% target** from the research objective.

6.3.3 Equal Distribution Results

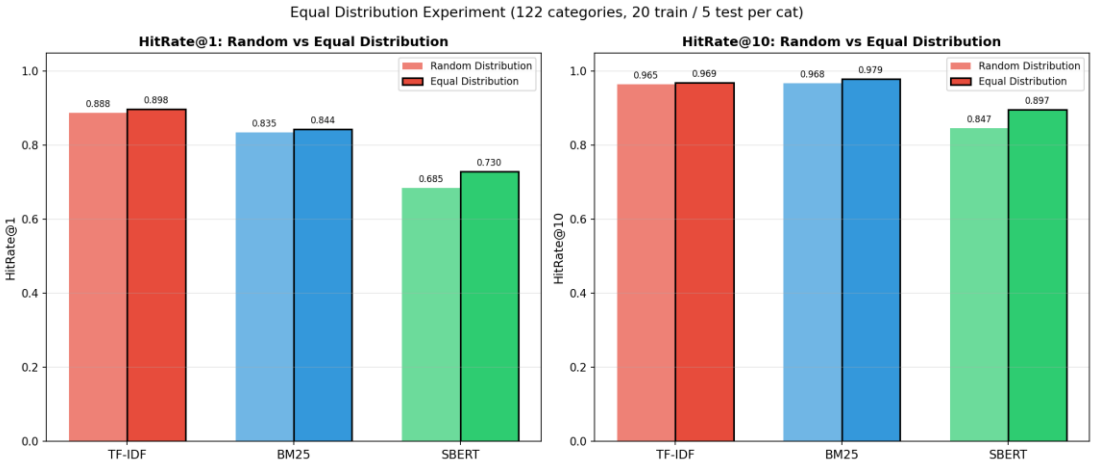

Figure 6.3.3.1: Equal distribution bar chart

Following the B!SON evaluation methodology [19], a balanced evaluation was conducted with 20 training journals and 5 test journals per qualifying category (122 categories, 2,440 train, 610 test). Table 6.3.3 compares the results.

Table 6.3.3: Random vs Equal distribution comparison (Title+Scope+Cat)

Model	HitRate@1 (Random)	HitRate@1 (Equal)	Difference
TF-IDF	88.84%	89.84%	+1.00%
BM25	83.47%	84.43%	+0.96%
SBERT	68.53%	72.95%	+4.42%

All three models showed **slight improvements** under the equal distribution setup. SBERT exhibited the largest improvement (+4.42%), suggesting that it was **disproportionately affected** by the **imbalanced category distribution** in the random setup. When popularity bias was removed, the performance gap between SBERT and the lexical models narrowed slightly, though TF-IDF still maintained a **clear lead**.

6.3.4 Per-Category Analysis

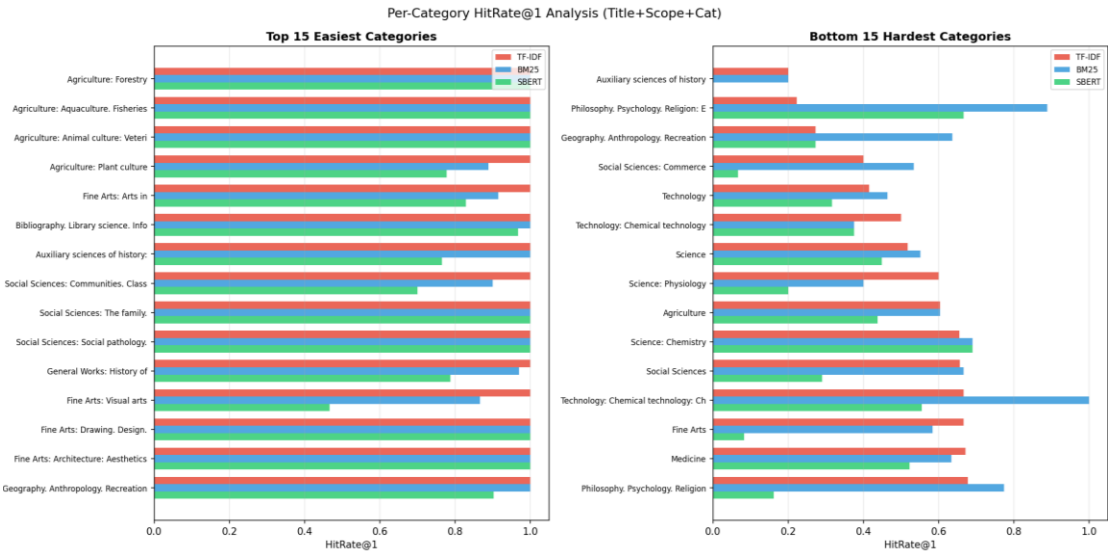

Figure 6.3.4.1: Per-Category analysis top & bottom preview

Figure 6.3.4.1 showed per-category breakdown of HitRate@1 was conducted across 135 categories with at least 5 test samples. Table 6.3.4 summarises the distribution of the best-performing model across categories.

Table 6.3.4: Best model distribution across categories

Best Model	No. of Categories	Percentage
TF-IDF	106	78.5%
BM25	25	18.5%
SBERT	4	3.0%

To provide a visual comparison of the **best and worst-performing categories**, Figure 6.3.4.1 presents a horizontal grouped bar chart of HitRate@1 for the top 15 easiest and the bottom 15 hardest categories. Each category is represented by three bars corresponding to TF-IDF, BM25 and SBERT, allowing for a direct comparison across models. The left panel shows categories where all three models achieved near-perfect accuracy, while the right panel highlights categories that **posed challenges** for all approaches.

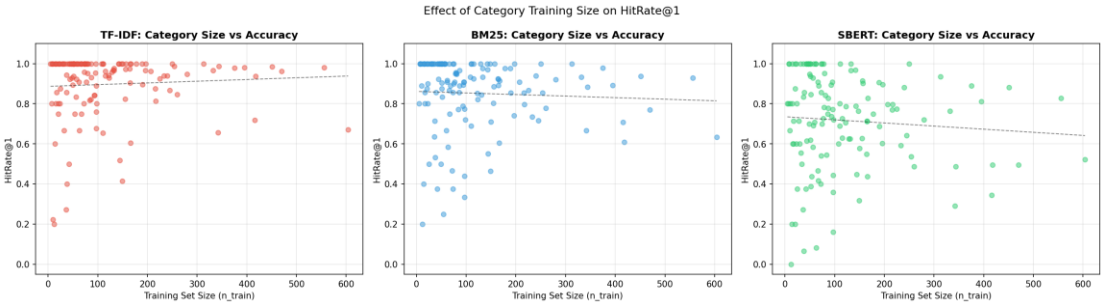

Figure 6.3.4.2: Category size vs accuracy scatter plot

As shown in Figure 6.3.4.2, no strong positive correlation was observed between category size and accuracy for any of the three models. Categories with as few as 10–20 training journals could achieve 100% HitRate@1 if their labels were specific, while large categories with over 100 training journals (e.g., "Medicine") did not necessarily achieve the highest accuracy. This indicated that how specific or broad the category label is, rather than the number of training journals, was the **primary factor determining retrieval difficulty**.

6.3.5 Execution Time Comparison

Table 6.3.5: Execution time per model (Title+Scope+Cat)

Model	Time (seconds)	Relative to TF-IDF
TF-IDF	3.71	1.0×

SBERT	124.48	33.6×
BM25	175.04	47.2×

TF-IDF was the fastest model, completing evaluation in approximately 4 seconds due to **highly optimised sparse matrix operations** in scikit-learn. SBERT required approximately 124 seconds, dominated by the **embedding computation** step. BM25 was the slowest at 175 seconds due to the **pure Python implementation** of the rank_bm25 library.

6.4 Discussion

The results revealed that TF-IDF, the simplest model, outperformed both BM25 and SBERT across most metrics, input strategies and evaluation setups. This finding **contradicted the initial hypothesis** that transformer-based models would yield superior performance. Several factors contributed to this outcome.

Journal metadata in the DOAJ dataset used consistent, **standardised academic terminology**. Journal titles and subject classifications used specific terms (e.g., "Medicine", "Forestry") that were repeated exactly across related journals. TF-IDF excelled at matching such exact term overlaps, whereas SBERT's strength in capturing paraphrases and synonyms was largely unnecessary in this context.

When the category label was appended to the query text for TF-IDF and BM25, these models **benefited from direct string matching on the category name**. For SBERT, the category was incorporated indirectly through a centroid similarity score, which was computed as the average embedding of all journals belonging to a category. This averaging produced a more diffuse signal compared to exact lexical matching, which explained a significant portion of the performance gap.

The journal metadata consisted of relatively short texts (titles and subject descriptors). Transformer models **typically demonstrate their advantages on longer documents** where word order, context, and semantics matter more. For short metadata, the **bag-of-words approach of TF-IDF captured sufficient information**.

The evaluation metric (same-category match) inherently favoured models that could **leverage category information directly**. This was a characteristic of the dataset design rather than a fundamental limitation of SBERT.

These findings aligned with observations from the B!SON framework [19], where the Elasticsearch BM25 baseline remained competitive with neural embedding approaches for journal-level metadata retrieval.

6.5 Objectives Evaluation

The objectives stated in Section 1.4 of the FYP1 report were evaluated as follows:

"Gather and normalize journal scope metadata from multiple sources to reach over ~10,000 journals." Achieved. The DOAJ dataset **contained 21,782 journals**, of which 21,782 were normalised and used (17,426 train, 4,356 test).

"Examine both keyword-based models (TF-IDF, BM25) and BERT models in retrieving manuscripts against journal scopes." Achieved. All three models were **implemented, tuned** with Bayesian optimisation, and **evaluated systematically**.

"Compare different processing pipelines and experiment with title-abstract-category as input; an accuracy increase of 2% is considered as success." Exceeded. The Title+Scope+Cat strategy **improved HitRate@1 by 9.64% to 22.59%** over Title+Scope, far exceeding the 2% target.

"Measure and compare the effectiveness of each method using ranking metrics mainly with Precision@K and Accuracy@K." Achieved with revised metrics. As discussed in Section 3.3, the metrics were updated to HitRate@K (equivalent to Accuracy@K), MAP@K and NDCG@K to provide a **more comprehensive evaluation** aligned with the B!SON framework.

6.6 Concluding Remark

The experimental results confirmed that all three models achieved high performance when provided with scope and category information, with TF-IDF being the most accurate (88.84% HitRate@1), fastest (3.71 seconds) and simplest model. The equal distribution experiment and per-category analysis further validated these findings.

CHAPTER 7

Conclusion and Recommendation

7.1 Conclusion

This project addressed the challenge of aligning manuscripts with suitable journals by implementing and benchmarking three retrieval models — TF-IDF, Okapi BM25, and Sentence-BERT — on the DOAJ journal metadata dataset containing 21,782 journals across 327 categories.

A key methodological improvement from FYP1 to FYP2 was the adoption of **Bayesian hyperparameter optimisation using the Optuna framework**, replacing the originally proposed manual grid search. This enabled **efficient search** over continuous parameter spaces and produced optimal configurations that would not have been discovered through the limited discrete grid originally proposed.

The evaluation framework was also strengthened by adopting **HitRate@K, MAP@K and NDCG@K as the primary metrics**, replacing the originally proposed Accuracy@K and Recall@K. Two evaluation setups (random and equal distribution) and three input strategies were tested to ensure the validity of the results.

The experimental results demonstrated that **TF-IDF** achieved the **highest HitRate@1 of 88.84%** with the Title+Scope+Category input strategy, outperforming BM25 (83.47%) and Sentence-BERT (68.53%). This finding contradicted the initial hypothesis that transformer-based models would yield superior performance, and was attributed to the **standardised vocabulary** and **short document lengths** in the DOAJ journal metadata.

The equal distribution experiment confirmed the validity of these findings across **balanced category distributions**. The per-category analysis revealed that TF-IDF was the **best-performing model in 78.5% of all analysed categories**, with specific category labels achieving near-perfect accuracy across all models while broad labels presented challenges for all approaches.

7.2 Recommendation

Based on the findings of this research, the following recommendations are proposed for future work:

The current evaluation used **journal-level queries** (matching journals to journals within the same category). By **evaluating on article-to-journal matching**, where actual manuscript abstracts are matched against journal scopes, it would provide a more **realistic assessment** of the system's practical utility. In such a setup, SBERT's **semantic understanding may prove more valuable** due to the greater vocabulary diversity in manuscript abstracts compared to standardised journal metadata.

Secondly, given that TF-IDF excelled at **lexical matching** while SBERT captured **semantic relationships**, a learned ensemble **combining both signals** could potentially yield performance improvements. A **fusion approach** that uses TF-IDF scores as a primary signal with SBERT re-ranking could leverage the strengths of both approaches.

Hence, the current research used only the DOAJ dataset. **Incorporating additional datasets** such as Web of Science and Scopus journal metadata, could test whether the findings generalize to other publishers and metadata formats.

Conducting a user study with real manuscripts from researchers would validate whether the offline metrics can actually translate into practical usefulness. This could involve a controlled experiment where researchers evaluate the top-5 recommended journals for relevance and suitability.

REFERENCES

[1] N. Abdullah, "Publishing Scholarly Articles: From Manuscript to Publication," *Nil*, vol. 7, no. 1, pp. 240–273, Oct. 2024, doi: 10.62810/jss.v7i1.19.

[2] A. Taofik, A. Benjamin, and O. Olusola "A Recommender System For Academic Publications Using Content-based filtering techniques" Presented at International Conference on Innovative Systems for Digital Economy | ISDE'2021 [Online]. Available: https://www.researchgate.net/publication/357240718_A_RECOMMENDER_SYSTEM_FOR_ACADEMIC_PUBLICATIONS_USING_CONTENT-BASED_FILTERING_TECHNIQUES

[3] Z. Hans. "'What's the best journal for my paper?' New tool can help" elsevier.com. https://www.elsevier.com/connect/whats-the-best-journal-for-my-paper-new-tool-can-help (accessed Apr. 24, 2025).

[4] Z. Zhang *et al.*, "Scholarly recommendation systems: a literature survey," *Knowledge and Information Systems*, vol. 65, no. 11, pp. 4433–4478, Jun. 2023, doi: 10.1007/s10115-023-01901-x.

[5] N. Kang, M. A. Doornenbal, and R. J. A. Schijvenaars, "Elsevier Journal Finder," *Recommending Journals for Your Paper*, pp. 261–264, Sep. 2015, doi: 10.1145/2792838.2799663.

[6] B. Nilesh. "How to Code BERT Using PyTorch – Tutorial With Examples." neptune.ai. https://neptune.ai/blog/how-to-code-bert-using-pytorch-tutorial (accessed Sep. 1, 2025).

[7] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, Apr. 2005, doi: 10.1109/tkde.2005.99.

[8] Deepjyoti and Mala, *A systematic review and research perspective on recommender systems*. 2022. doi: 10.1186/s40537-022-00592-5.

[9] GeeksforGeeks, "Understanding TF-IDF (Term Frequency Inverse Document Frequency)," *GeeksforGeeks*, Feb. 07, 2025. https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/ (accessed Apr. 27, 2025)

[10] GeeksforGeeks, “What is BM25 (Best Matching 25) Algorithm?,” *GeeksforGeeks*, Mar. 28, 2025. https://www.geeksforgeeks.org/what-is-bm25-best-matching-25-algorithm/ (accessed Apr. 28, 2025)

[11] J. Alammar, “The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning).” https://jalammar.github.io/illustrated-bert/ (accessed Sep. 1, 2025)

[12] M. K. Hemila and H. Rölke, "Recommendation System for Journals based on ELMo and Deep Learning," in *2023 10th IEEE Swiss Conference on Data Science (SDS)*, 2023, pp. 110-117.

[13] X. Li, B. Shao, and G. Bian, "A scholars' personality traits augmented multi-dimensional feature fusion scholarly journal recommendation model," *Appl. Soft Comput.*, vol. 163, p. 111888, 2024.

[14] X. Li, B. Shao, G. Bian, and X. Huang, "A journal name semantic augmented multi-dimensional feature fusion model for scholarly journal recommendation," *Inf. Process. Manag.*, vol. 60, no. 4, p. 103460, 2023.

[15] C. Liu, X. Wang, H. Liu, X. Zou, S. Cen, and G. Dai, "Learning to recommend journals for submission based on embedding models," *Neurocomputing*, vol. 508, pp. 242–253, 2022.

[16] Y. Li et al., "Cross-loop contrast of heterogeneous graphs for interdisciplinary journal recommendation," *Expert Systems With Applications*, vol. 265, p. 125949, 2025.

[17] “Metadata help – DOAJ.” https://doaj.org/docs/faq/ (accessed Sep. 2, 2025)

[18] “TFIDFVectorizer,” *Scikit-learn*. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (accessed Sep. 9, 2025)

[19] E. Entrup, A. Eppelin, R. Ewerth, J. Hartwig, M. Tullney, M. Wohlgemuth, and A. Hoppe, "Comparing different search methods for the open access journal recommendation tool B!SON," *International Journal on Digital Libraries*, vol. 25, no. 3, pp. 505–516, 2023, doi: 10.1007/s00799-023-00372-3.

[20] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of*

the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019, pp. 2623–2631, doi: 10.1145/3292500.3330701.

APPENDIX

Poster