Report - BioASQ Group 15

Hausberger Benedikt, Klaus Sebastian, Mayrhofer Michael, Katoch Suvi, and Necemer Michael

Technische Universität Wien

Repository: https://github.com/michi2402/airup

1 Introduction

This report provides an overview of the design, challenges, and performance of three different information retrieval systems we developed for Task 1 - BioASQ of the Advanced Information Retrieval course. We participated in Task 13b: Phase A competition for test batch 4. The task description can be found on the official webpage[1] of the competition. Details about our team members, including their contributions and the systems they developed, are provided in Table 1. We believe that we distributed the workload among group members fairly and effectively. To ensure balance, we paired two people to work collaboratively on tasks that carried more points, as these required additional effort and resources. Meanwhile, one member was primarily responsible for handling the task with fewer points. In addition, we had several meetings to ensure that everyone was on track with their tasks and to discuss the next steps.

Student Name	**Matr.-Number**	**Developed System**
Hausberger Benedikt	11908105	System 2 - Representation Learning
Klaus Sebastian	12120487	System 2 - Representation Learning
Mayrhofer Michael	12122564	System 3 - Reranking Model
Katoch Suvi	12120504	System 3 - Reranking Model
Necemer Michael	01360034	System 1 - Traditional IR

Table 1: BioASQ Group 15 - AIRup

Four systems were submitted to the BioASQ Task13b - Phase A competition for test batch 4: a traditional information retrieval model, a neural information retrieval model, and a reranker, which was applied on top of both retrieval models and submitted as two additional systems. In the official leaderboard these systems are listed by their description. A reference of system names and descriptions can be found in the following.

[1] https://participants-area.bioasq.org/general_information/Task13b/

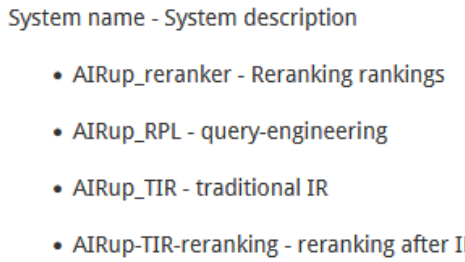

Fig. 1: BioASQ competition - submitted systems

In the remainder of our report we discuss key challenges faced, give an in-depth view on our systems developed including the design and architecture, our contributions, and results. We provide the code on our public repository on Github[2].

2 Key Challenges across all Systems

2.1 Query Preprocessing

A key challenge we encountered in all systems was to translate the question in natural language into a reasonable query, which we could then use to query the PubMed API[3] to create a body of somewhat relevant documents that we could further refine via our systems. Initially, we relied on simple rule-based methods to map natural language to Boolean expressions. To enhance this, we incorporated the NLTK[4] Python package. We evaluated this approach by comparing the documents contained in the training dataset with the documents the constructed query was able to retrieve. More precisely, we checked how many relevant documents we were able to retrieve. Although the NLTK approach worked better than our initial approach, the pre-processing was still missing most of them.

To further enhance the approach, we used the Word2Vec model provided on the BioASQ website. Using this model, we were able to extract synonyms or rather words used in a similar context to the previously extracted key terms. Each key term is connected to other key terms using a logical `AND`. For each key term, its synonyms are grouped together and connected via a logical `OR` as follows: (key_term1 `OR` synonym1_1 `OR` synonym1_2 `OR` ...) `AND` (key_term2 `OR` synonym2_1 `OR` synonym2_2 `OR` ...) `AND` ...

Since performing query expansion by combining key terms and their respective synonyms via a logical `OR` yielded promising results, we continued with the implementation of the actual systems, i.e. we used the same query preprocessing

[2] https://github.com/michi2402/airup
[3] https://www.ncbi.nlm.nih.gov/books/NBK25497/
[4] https://www.nltk.org/

steps for our systems to generate the corpus of relevant documents that are then refined using a specific system. On a side note, we also tried to use a BioBERT model to extract key terms. However, this approach did not show significant improvements, possibly due to insufficient fine-tuning.

2.2 Data quality and availability

Another challenge we faced was the limited amount of contextual information about the training data provided. We assumed that the documents and snippets in the training set were already pre-sorted by their relevance scores. However, in some cases, more than 10 documents or snippets were provided for certain questions, leading to uncertainty about the data. Specifically, we could not determine whether a larger number of relevant documents were intentionally included or if some of them shared identical relevance scores. In addition, we encountered inconsistencies in the training data provided. For example, some snippets had the *beginsection* key set to `section.0`, although they appeared in the abstract, while other snippets simply had the `abstract` value assigned. We decided to assign the value `abstract` in our results when encountering the described situation.

Another issue arose during the testing phase, where the PubMed API was occasionally unavailable. This was particularly problematic when experimenting with different query preprocessing parameters or setups, as discussed in Section 2.1, since access to the API was necessary in order to build the corpus of relevant documents.

3 System 1 - Traditional IR

3.1 Overview

The first system we had to implement was a traditional information retrieval system. After studying the material provided in the course, we decided to implement a `BM25` model. Initially, we implemented our own custom versions of both `BM25` and `BM25F`, with the code for these models provided on Github in the `traditional_IR/custom_models` folder. The `BM25F` approach was explored because we wanted to experiment with assigning different weights to specific fields in the document body. However, since the API only provided access to the title and abstract of documents, the field-weighting benefits of `BM25F` could not be fully utilized, leading us to discard this approach.

Although our custom `BM25` implementation delivered promising results, we decided to also assess a widely-used existing implementation. For this purpose, we chose the Python library rank-bm25[5], specifically leveraging the `BM25+` function. This variation of `BM25` addresses a key limitation of the original algorithm: the over-penalization of long documents. This decision was motivated by our observation that the abstracts of the retrieved documents varied in length. Upon

[5] https://pypi.org/project/rank-bm25

switching to `BM25+`, we observed noticeable improvements in retrieval performance compared to our own implementations. For the `BM25+` model, we used hyperparameters frequently recommended in the literature: `k1=1.5`, `b=0.75`, and `delta=1`. While our implementation included a train-test split and we originally planned to perform a grid search to optimize these hyperparameters, time constraints prevented us from completing this step. However, informal testing of a few alternate hyperparameter values did not yield better results than the defaults. As the literature suggested that text preprocessing techniques eventually improve retrieval results, we explored stemming and lemmatization. As discussed in Section 2.2, some questions in the training dataset had less than 10 documents or snippets attached. Therefore, we implemented a so-called drop-off function which acts as a filter that potentially removes documents or snippets from the results list.

3.2 Design and Architecture

In this section, we first provide an outline of the whole process of mapping a given query to a result list of documents or snippets. After that we go into more detail on the employed steps. Finally, we show the competition results of our traditional information retrieval approach.

3.2.1 Process In the following, we outline the core steps used to generate a list of documents with the highest relevance scores based on a given query using our traditional information retrieval approach.

1. Preprocess the query
2. Query PubMed API and retrieve documents based on the preprocessed query
3. For each document: apply text preprocessing, and add the preprocessed document to the `BM25+` corpus
4. Take original query: expand it by adding relevant synonyms, and lemmatize it
5. Score the documents in the corpus against the expanded query and take the 10 most relevant ones
6. Apply the drop-off function to filter out documents

Step 1 and 2, and the challenges we faced, were already discussed in Section 2.1. The process of generating a list of snippets was very similar. The only major difference was that in step 3 we first split a document into chunks (we chose chunking on sentence level), lemmatized those, and added them to the corpus. When scoring, we now obtained the chunks that matched the query the best. We applied our drop-off function on top of the retrieved chunks in order to further trim down the results.

3.2.2 Text preprocessing In step 3, we experimented with various text preprocessing techniques, specifically stemming and lemmatization, prior to incorporating the documents into the corpus for the `BM25+` function. Both stemming

and lemmatization are widely recognized for their potential to enhance information retrieval performance. Stemming, which reduces words to their root forms by removing suffixes, proved to be less effective in the biomedical domain. This was likely due to the high prevalence of technical terms, which were often incorrectly truncated, leading to a loss of semantic meaning. In contrast, lemmatization, which maps words to their base dictionary forms while preserving grammatical accuracy, demonstrated better performance. For lemmatization, we utilized the SciSpacy library[6], a Python package specifically designed to process biomedical text. More precisely, we employed its smallest model, `en_core_sci_sm`, to convert raw text into lemmatized tokens. Compared to general-purpose stemming or lemmatization tools, the SciSpacy-based approach yielded slightly improved results for the biomedical domain. Consequently, we adopted lemmatization with SciSpacy as our preprocessing method of choice. Additionally, as discussed in Section 2.1, we enriched the query to broaden the pool of potentially relevant documents by leveraging the PubMed API. This query enrichment step involved identifying synonyms for key query terms to increase recall. To further refine this process, we enriched the original query with synonyms in step 4, prior to applying lemmatization. This ordering ensured that documents containing these synonyms were rated higher during the retrieval process, ultimately improving the quality of the results.

3.2.3 Scoring In step 5, we matched the corpus with the expanded query and computed relevance scores using the `rank-bm25` library. Notably, these relevance scores were not normalized. To explore potential improvements, we experimented with various normalization techniques, such as `max-normalization`. This method scales the scores to a range between 0 and 1, using the maximum score as the reference point for normalization. Despite the theoretical advantages of normalization, we observed a decline in performance when applying post-normalization filtering. Specifically, filtering based on normalized scores consistently produced worse results compared to working with the raw scores. As a result, we opted to reject normalization and retained the original scoring approach for the subsequent filtering step in our process.

3.2.4 Filtering results In step 6, we evaluated multiple criteria for filtering the results to improve the quality and relevance of the retrieved items. These criteria included the following:

- Relevance Threshold (`relevance_threshold`): This threshold is determined based on the highest relevance score in the result list. Items with relevance scores below this threshold are excluded, ensuring that only highly relevant documents are retained.
- Drop-Off Ratio (`drop_off_ratio`): This parameter removes items whose relevance scores decrease by a specified ratio compared to the previous item

[6] https://allenai.github.io/scispacy/

in the ranked list. This approach helps to filter out items that are significantly less relevant relative to higher-ranked results.
– Snippet Minimum Score (`snippet_min_score`): This criterion specifically targets snippet retrieval by excluding snippets with raw relevance scores below a predefined minimum value. This ensures that only snippets meeting a baseline relevance score are considered.

We conducted experiments with various parameter values to determine a good configuration. Based on our evaluation using the first 20 questions of the test set, the following parameter values were found to provide the best performance: `relevance_threshold = 0.7`, `drop_off_ratio = 0.12`, `snippet_min_score = 10`. This combination of filtering criteria enabled us to refine the results, achieving a reasonable balance between precision and recall. However, it is important to note that the evaluation was conducted on a limited subset of data, specifically the first 20 questions from the training set, and that the parameter `snippet_min_score` represents a raw, absolute value rather than a normalized threshold, see 3.2.3. The implementation of the drop-off function can be found on Github in `traditional_IR/utils.py`

3.3 Evaluation

The results of the traditional information retrieval approach for BioASQ Task 13b - Phase A are presented in Table 2 where we list the results separately for the document and the snippet retrieval task.

	Mean precision	**Recall**	**F-Measure**	**MAP**	**GMAP**
T_IR (Documents)	0.0217	0.0451	0.0231	0.0193	0.0000
T_IR (Snippets)	0.0135	0.0086	0.0100	0.0068	0.0000

Table 2: Results of the traditional information retrieval approach

When comparing the results of our traditional information retrieval approach to the official leaderboard, we are overall satisfied with the performance of our system. As outlined in Section 2.1, the primary challenge lay in constructing a robust corpus of potentially relevant documents through the API. In some cases, we observed that no retrieved documents overlapped with the relevant ones provided in the training data. This limitation highlights the inherent difficulty of the initial document retrieval step, particularly in ensuring comprehensive coverage of relevant materials. Given these circumstances and the foundational nature of

our approach, we are content with the results of our initial system. However, it is noteworthy that the recall for document retrieval is significantly higher than that for snippet retrieval, see Table 2. Although the snippet retrieval performance was strong for the subset of training questions discussed in 3.2.4, we suspect that the fixed raw score used as a threshold introduced limitations when applied to the test set. We suggest that the rigid nature of the snippet filtering might have excluded relevant snippets that fell below the fixed `snippet_min_score` value. However, those normalization steps we tried before applying filtering did not increase the results, as discussed in 3.2.3. However, we assume that improvements to the filtering functionality of the snippets should lead to better results as we found that for subsets of the training data it worked quite well.

4 System 2 - Representation Learning

4.1 Overview

The second system we chose to implement was a representation learning approach. After longer research, we decided to use a pre-trained BioBERT model, which we fine tuned in the process, as the core of our application. Around the model we also built a preprocessing system to increase the performance of the ranking. Since the backbone of the system is a vector representations of biomedical language and it refines them for the ranking the application falls under a representation learning system. The model does not rely on manually crafted features, instead it leans meaningful representation directly from the data. More detailed information of the design, the training data set and the evaluation will be analysed in the following sections.

4.2 Design and Architecture

For our representation learning approach, we used the pre-trained BioBERT model available on Hugging Face[7]. To adapt the model to our specific domain, we constructed a training dataset using the data provided by BioASQ. This dataset was then used to fine-tune the BioBERT model.

The following sections detail the preparation of the training data, the training parameters, and our approach for ranking the retrieved documents and snippets using the fine-tuned model.

4.2.1 Training Data The BioASQ dataset contained several thousand questions. For each question, the top-ranked documents and their corresponding top snippets were provided. We used this information to construct both positive and negative training examples. Since the top snippets for a question were extracted only from its top documents, we used these top snippets as positive examples.

To generate negative examples for each question, we randomly sampled snippets from the gold-standard answers of other, unrelated questions.

[7] https://huggingface.co/dmis-lab/biobert-v1.1

4.2.2 Training With the prepared dataset, we split the data into training and validation sets, using a 90% 10% split respectively. We employed cosine similarity loss as the objective function for training, with the following parameters:

- Batch size: 16
- Number of epochs: 2
- Warm-up steps: 100
- Learning rate: 2e-5

Disclaimer: These values are not reflected in the public repository, as they were used in a modified version of the training script that was not pushed and has since been lost.

4.2.3 Ranking Using the retrieval approach described in Section 2.1, we first obtained a set of 100 candidate documents for each question. These documents were then passed to our ranking module.

The abstracts of the retrieved documents were encoded using the fine-tuned BioBERT model. Each question was also encoded in natural language, and cosine similarity was computed between the question and each abstract. The top 10 documents with the highest similarity scores were selected.

For each of these top 10 documents, we further segmented the abstracts into smaller snippets and evaluated their similarity to the question using the same approach. Additionally, the titles of the top 10 documents were also encoded and ranked based on cosine similarity.

4.3 Evaluation

The representational learning model was evaluated and scored by the *BioASQ challenge 2025*. In the following table the scores of the document rankings and the snipped rankings are presented:

	Mean precision	**Recall**	**F-Measure**	**MAP**	**GMAP**
RL (Documents)	0.0276	0.0644	0.0317	0.0346	0.0000
RL (Snippets)	0.0092	0.0191	0.0122	0.0195	0.0000

Table 3: Results of the representational learning approach evaluated by BioASQ.[8]

[8] This naming of the evaluation sets are *query-engineering* in the *BioASQ* results list. For readability it was changed in the presented table.

As seen in Table 5, the score increased in comparison to the traditional IR approach described in Section 3. This effect is particularly visible in the documentation retrieval score. The mean precision and the recall indicate a better performance. When investigating the snippet retrieval the mean precision lags behind the traditional information retrieval approach we implemented, although the recall, f-measure and MAP is higher. The possible explanation for these results might be the key challenges we had with the *BioASQ* assignment. We had struggled with setting the preprocessing, described in Section 2.1, and the exact validation of the snippets, described in Section 2.2.

5 System 3

5.1 Overview

We began our investigation of the reranking model by reviewing the provided slides, examining the proposed architecture, and studying relevant techniques in both bi-encoder and cross-encoder designs through online tutorials, documentation, and blog posts. Early on, we realized that the suggested strategy—indexing every potential result in a vector database—was infeasible because we could only query PubMed's API. This constraint transformed the challenge: rather than building and querying our own embedding index, we had to treat PubMed as a black box and rely on its API responses to supply all candidate articles. Consequently, our reranker operates exclusively on the snippets and full-text documents returned by the traditional and neural IR systems (which we tweaked for this step to return more than only the 10 most relevant each) that feed into our pipeline.

5.2 Design and Architecture

5.2.1 Data Preprocessing For each question, we first extract its gold-standard snippets as positive example (label = 1) and organize them into a question-to-examples map. To create negative examples, we employ semantic sampling through question clustering (similar to *Topic Aware Sampling* in the slides).

1. **Embedding**:
   Encode all questions with a SentenceTransformer to obtain dense vectors.
2. **Clustering**:
   Apply DBSCAN ($\epsilon = 0.3$, min_samples = 2, cosine metric) to group semantically similar questions.
3. **Distance matrix**:
   Compute pairwise cosine distances between question embeddings.
4. **Sampling negatives**:
   - *Close negatives*: snippets from questions in the same cluster
   - *Medium negatives*: from the 40th–60th percentile of distances
   - *Far negatives*: from the top 10% farthest questions
5. **Labeling**:
   All negative examples receive label = 0.

5.2.2 Train/Test Split After preprocessing, we split the questions into 90% train and 10% test sets using `train_test_split`, then convert each split into a Hugging Face `datasets.Dataset` for downstream training and evaluation.

5.2.3 Model Training As base model we used a HuggingFace cross-encoder. Thereby, we tried out different models with different hyperparameters. Finally we ended up with our centralized config in `config.py`:

- **Training parameters**:
  - batch size = 32
  - epochs = 3
  - logging every 20 steps
- **Negative-sampling settings**:
  - `AMOUNT_SAME_SIMILAR` = 2
  - `AMOUNT_MID_SIMILAR` = 1
  - `AMOUNT_FAR_SIMILAR` = 1
- **Base models**:
  - Cross-encoder base: `cross-encoder/ms-marco-MiniLM-L12-v2`
  - Sentence-transformer for question clustering: `all-MiniLM-L12-v2`

The training pipeline was designed according to the following schema:

- Tokenize question-snippet pairs with truncation and padding
- Initialize model from `ms-marco-MiniLM-L12-v2`
- Set `num_labels=1` for regression-style scoring
- Use accuracy on binary labels as metric via the `evaluate` library
- Run `Trainer.train()` and save the best model and tokenizer

5.2.4 Model Inference We treat a document's relevance as the sum of its snippets' relevance: each snippet–question pair in the evaluation set is scored by our fine-tuned cross-encoder and the parent score is inferred by them. Hence, the result generation looks like the following.

1. Tokenize all (question, snippet) pairs in batch
2. Forward pass to get scalar logits
3. Sort snippets by descending score
4. Aggregate scores by summing for parent documents
5. Output result in JSON submission format

5.3 Evaluation

5.3.1 Loss Monitoring Plot training vs. evaluation loss curves using Matplotlib.

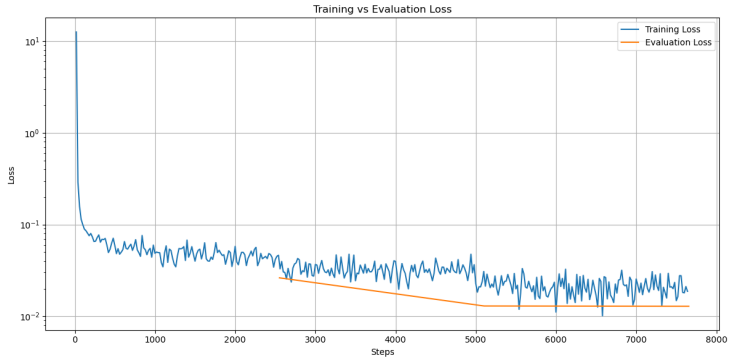

Fig. 2: Training vs Evaluation Loss

5.3.2 Quantitative Metrics Mean precision, recall, F-Measure, MAP and GMAP results scored by the submitted systems.
Scores achieved in the task of ranking documents by the reranker system built on top of the traditional IR system and the neural IR system, respectively:

	Mean precision	**Recall**	**F-Measure**	**MAP**	**GMAP**
RL (Documents)	0.0182	0.0157	0.0154	0.0110	0.0000
RL (Snippets)	0.0135	0.0086	0.0100	0.0113	0.0000

Table 4: Results of reranker built on traditional IR ('system 'reranking after IR') approach evaluated by BioASQ.

	Mean precision	Recall	F-Measure	MAP	GMAP
RL (Documents)	0.0335	0.0814	0.0404	0.0416	0.0001
RL (Snippets)	0.0191	0.0562	0.0259	0.0460	0.0000

Table 5: Results of reranker built on neural IR (system 'reranking rankings') approach evaluated by BioASQ.