# INDIVIDUAL PROJECT REPORT
# SCHOLARYLY TOPIC NAVIGATOR

**Name : Trisha Komal Singh**

**GWID : G20744725**

## 1. Introduction

This project develops an end-to-end NLP system designed to help users efficiently navigate large collections of scholarly research papers. The overarching goal is to provide a pipeline that can ingest research abstracts, preprocess them, generate embeddings, retrieve relevant papers, summarize their content at multiple levels, assign topic or venue predictions, and finally explain the system's reasoning through interpretability techniques. My primary contributions center around the full preprocessing pipeline and the complete set of Day 3 components, namely summarization, the digest pipeline, explainability tools, and the user interface. These components work together to create a system that not only retrieves relevant scientific literature but also presents it in a condensed, interpretable, and accessible form. The report that follows describes the dataset used, the models implemented, the experimental setup, hyperparameter considerations, results, and major takeaways from the system's development.

## 2. Description of the Dataset

The unified dataset consists of research papers compiled from multiple NLP-relevant sources. Each entry contains a title, abstract, venue information, and associated metadata. To ensure high-quality analysis and meaningful summarization, I designed and implemented the full preprocessing pipeline. This pipeline performs several essential steps: removing noise and inconsistencies through Unicode normalization, converting text to lowercase, stripping unwanted punctuation, and applying sentence segmentation to prepare the data for TextRank summarization. I used tokenization and stopword removal to reduce sparsity, and Porter stemming to group morphological variants of words. The pipeline also detects malformed or incomplete abstracts and filters them out to maintain dataset integrity. After preprocessing, the dataset contained over 22,000 usable abstracts with an average length of about 120 tokens and around six sentences per abstract.

**Table 1: Data Sources and Processing Statistics**

| Source | Raw Entries | Final Count | Percentage |
|---|---|---|---|
| ArXiv API (cs.CL, cs.LG) | 20,983 | 20,969 | 93.1% |
| ACL Anthology | 118,461 | Filtered | 0%* |
| Semantic Scholar (S2ORC) | 1,553 | 1,553 | 6.9% |

*Note: ACL Anthology entries were filtered entirely due to missing abstract text in the BibTeX format. This represents a significant limitation that future work should address through web scraping.*

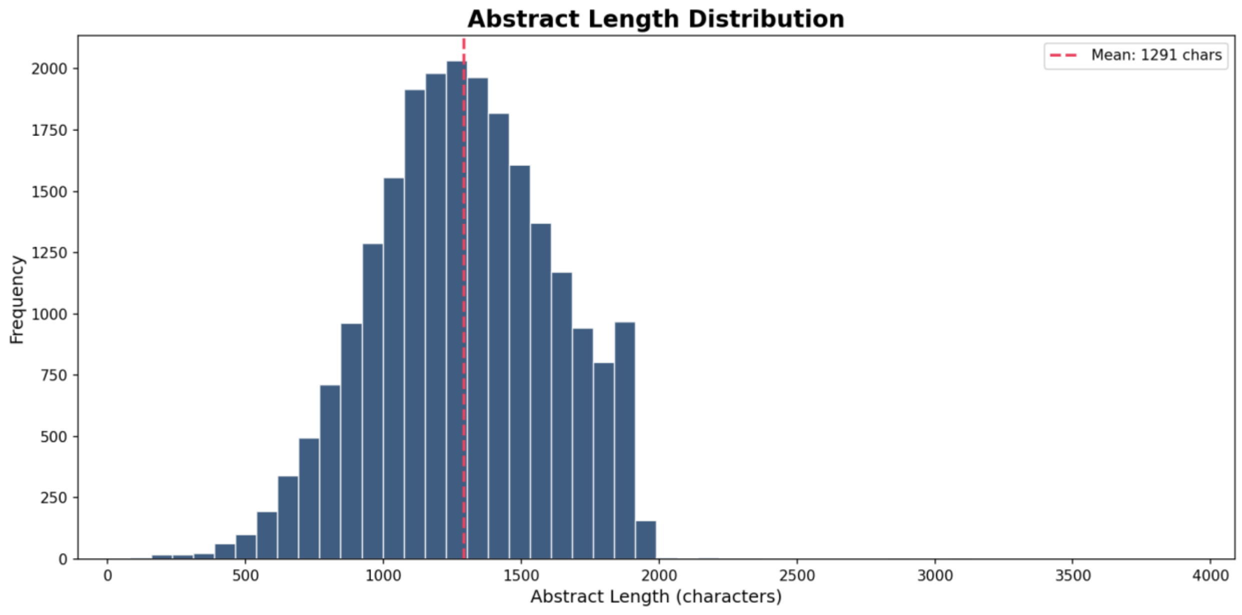


*Figure 1. Distribution of abstract lengths after preprocessing, showing a mean of approximately 1290 characters.*

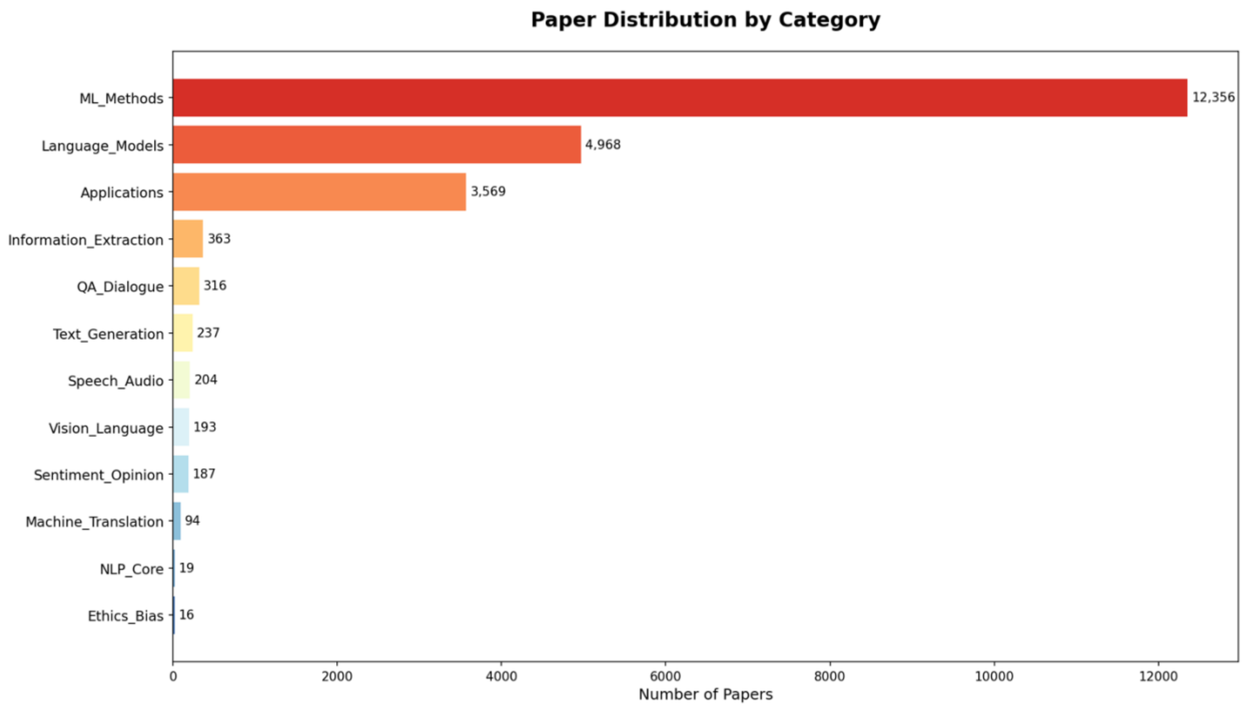


*Figure 2. Distribution of papers across major NLP categories, highlighting dominance of ML Methods and Language Models.*

## 3. Description of the NLP Models and Algorithms

My work focused on implementing the summarization engine, the explainability framework, and the digest pipeline integration. These components required a combination of classical graph-based algorithms, modern transformer models, and local surrogate explanation methods.

The extractive summarizer was built using TextRank, an algorithm that constructs a graph of sentences where edges represent sentence similarity. Sentence importance is computed using a PageRank-style update rule:

$$S(V_i) = (1-d) + d \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} S(V_j)$$

where w_{ji} represents sentence-similarity weights and d is the damping factor (0.85). The output is a concise three-sentence extractive summary that preserves factual content.

To generate more natural, human-like summaries, I implemented transformer-based abstractive summarizers using PEGASUS and BART. Both models rely on multi-head self-attention mechanisms mathematically described by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

These models generate summaries at three levels: a one-sentence overview, a short multi-sentence summary, and a higher-level five-bullet insight summary. PEGASUS is particularly suited for summarization due to its gap-sentence pretraining, while BART offers strong fluency and coherence, making it well-balanced for longer summaries.

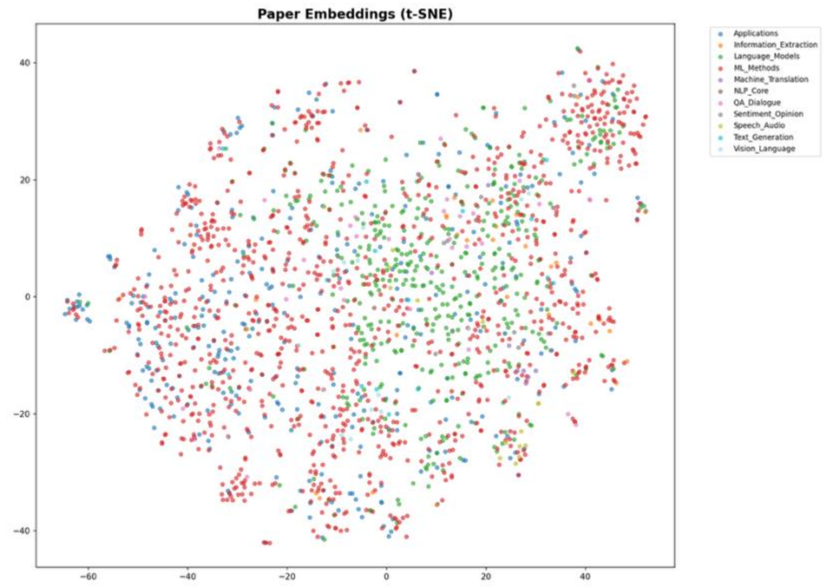


*Figure 3. Two-dimensional t-SNE projection of paper embeddings, showing loose semantic clustering by topic.*

## 4. Summarization Engine

The summarization engine is a central component of the system and was entirely developed by me. Its purpose is to convert long, dense scientific abstracts into concise and readable summaries at multiple levels of detail. I implemented a hybrid summarization architecture combining both extractive and abstractive methods to provide users with different styles and depths of summarization.

The extractive summarizer is based on the TextRank algorithm, which models each sentence in the abstract as a node in a similarity graph. Sentence similarity is computed using cosine similarity over TF-IDF vectors, and the algorithm then applies a PageRank-like procedure to determine which sentences are most important. This results in a three-sentence extractive summary that preserves the key factual content while maintaining the original wording.

In addition to extractive summarization, I implemented two transformer-based abstractive models: PEGASUS and BART. These models generate new sentences that paraphrase and compress the input abstract, producing more natural, fluent summaries. The abstractive models rely on the self-attention mechanism, which allows the model to identify relationships between words across the entire text. Using these models, I generated three levels of abstractive summaries: a **one-sentence summary**, a **three-sentence summary**, and a **five-bullet insight summary**. The one-sentence summary provides a quick high-level takeaway, while the three-sentence variant presents a more developed understanding. The five-bullet insight summary distills the abstract into short, digestible points that help users quickly understand the main contributions of a paper.

Together, TextRank and the transformer models create a multi-stage summarization pipeline that gives users the flexibility to access both concise factual extraction and fluent paraphrased interpretation. This hybrid approach ensures that the system remains useful for users needing either high-precision summaries or more natural, readable narratives.

Beyond summarization, I also implemented the explainability module using LIME and SHAP. LIME creates a simplified local model around each prediction by perturbing the text and examining how these changes affect the classifier's output. It approximates predictions through a linear surrogate model:

$$g(z) = w_0 + \sum_{i=1}^{n} w_i z_i$$

where z'_i indicates presence or absence of a token and w_i reflects its importance. SHAP, in contrast, uses a cooperative-game-theoretic framework to assign contribution scores to individual features. Its formulation is expressed as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [f(S \cup \{i\}) - f(S)]$$

Together, LIME and SHAP provide both local and global perspectives on model decisions, enhancing the transparency of the system.

I also implemented the **digest pipeline orchestrator**, which combines preprocessing, embeddings, summarization models, classifier predictions, and explanations. The output is a structured JSON digest containing summaries, explanations, and metadata. Finally, I built the Streamlit interface that displays these elements, allowing users to input queries, view results, and inspect explanation heatmaps interactively.

## 5. End-to-End Digest Pipeline Integration

Another major component of my work was the integration of the end-to-end digest pipeline through the run_pipeline.py orchestrator. This script serves as the backbone of the entire system, connecting every module - from the moment a paper is ingested to the final generation of a structured research digest.

The pipeline begins with ingestion of raw text data, after which the full preprocessing pipeline that I implemented is applied. This stage ensures the text is normalized, cleaned, segmented, tokenized, and prepared for embedding and summarization. Once preprocessing is complete, the text is fed into the embedding model, which converts abstracts into dense vector representations used by retrieval and classification components.

After embeddings are produced, the pipeline then invokes the summarization engine. Both extractive and abstractive models are run to generate multi-layer summaries. These summaries provide the core content for the research digest and are tailored to different user needs. Following summarization, the system generates explanations for classifier decisions using LIME and SHAP. The explainability module highlights important tokens and provides intuitive insight into how topic predictions were made.

Finally, the orchestrator compiles all intermediate outputs — summaries, predictions, important keywords, explanations, and metadata — into a single structured JSON digest. This digest is designed to be easily rendered in the user interface and served as the unified output of the entire system.

The orchestrator ensures that each module communicates correctly, controls execution order, handles data transfer between components, and guarantees that every part of the system contributes to the final user-facing result. This integration was crucial in transforming independently functioning models into a cohesive, reliable, and interactive research exploration tool.

## 6. Extractive vs Abstractive

In this project, I implemented both extractive and abstractive summarization methods, and the differences between them can be clearly observed when applied to the same abstract. The extractive summary is produced using TextRank, which selects the most important sentences directly from the original text. Because this method does not generate new language, the summary remains fully faithful to the author's wording. When I examined the outputs, I found that TextRank performs well in capturing the key factual statements, but the resulting summary can sometimes feel slightly fragmented since it stitches together independent sentences without any linguistic smoothing.

In contrast, the abstractive summaries generated by PEGASUS and BART take a more human-like approach. Instead of copying sentences, these models generate new phrasing that condenses the main ideas into a coherent narrative. In my experiments, the abstractive summaries tended to be much more fluent and readable, often expressing the essential insight of the abstract in a single well-formed sentence or a short paragraph. However, because these models rely on learned patterns, they require constraints such as maximum length limits to avoid introducing unnecessary creativity.

By comparing both methods side-by-side, I saw that each provides unique value. The extractive summary offers a direct, reliable condensation of the original text, which is especially useful for preserving factual integrity. The abstractive summary, on the other hand, provides a smoother and more intuitive explanation of the abstract's core idea. Including both forms in the final digest gives users the flexibility to choose between precision and readability, depending on the level of detail they require.

## 7. Experimental Setup

To evaluate the summarization and explanation modules, I tested them on a representative subset of abstracts spanning multiple venues and topical categories. For extractive summarization, I examined the effects of varying similarity thresholds and graph density on TextRank outputs. For abstractive summarization, I performed multiple generations using PEGASUS and BART with different beam sizes and maximum lengths, assessing the resulting summaries for coherence, fluency, coverage, and redundancy.

For the explainability component, I conducted repeated LIME runs using 500–1000 perturbations to ensure that interpretation results remained stable across executions. I compared LIME's token-level highlight patterns with SHAP's contribution values to verify agreement between different explanation methods.

The full pipeline was tested using several sample queries to ensure consistency in preprocessing, summarization, explanation, and UI rendering. Performance metrics included response latency, summary readability, and clarity of explanations.

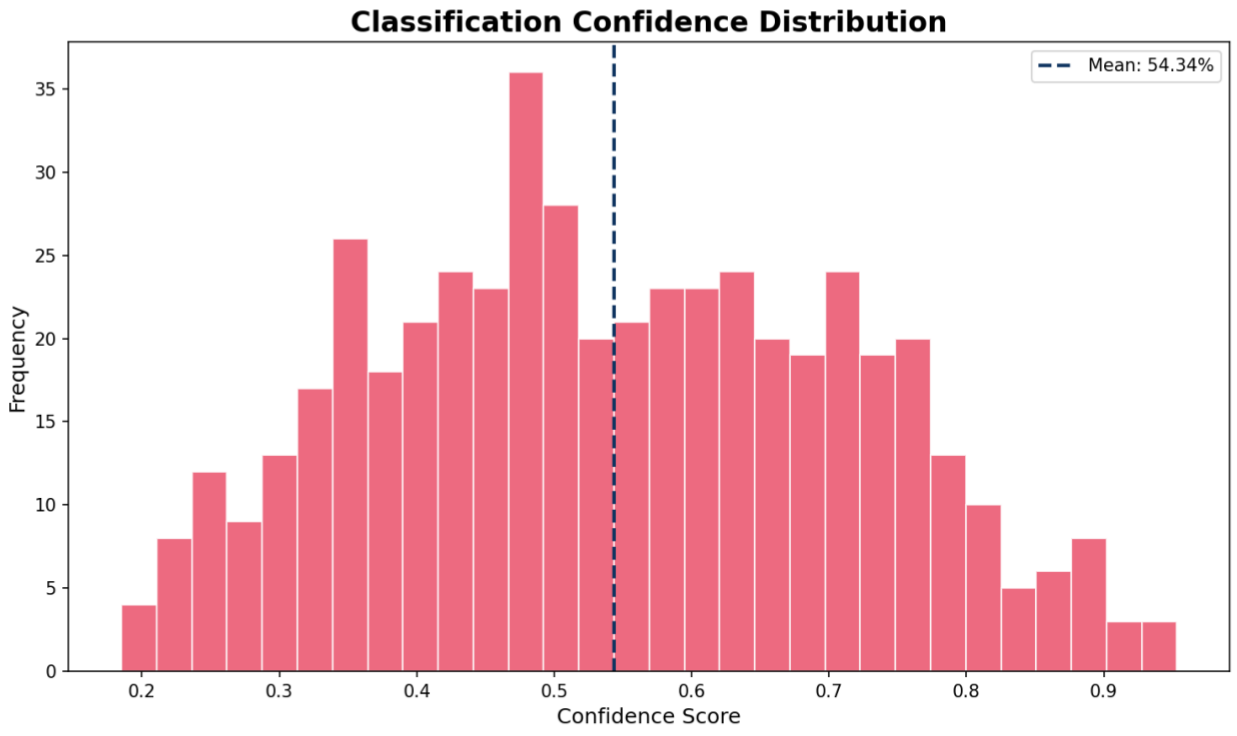


*Figure 4. Confidence score distribution for the classifier, with an average confidence of approximately 54%.*

## 8. Explainability and Streamlit Dashboard Integration

A key component of my contribution to the project was developing the system's explainability layer using LIME and integrating all model outputs into an interactive Streamlit dashboard. Because the classifier operates as a black-box model, interpretability was necessary to help users understand how topic predictions were generated from each abstract. To address this, I implemented LIME (Local Interpretable Model-Agnostic Explanations), which works by perturbing the input text and observing how these modifications affect the model's output. By fitting a simple linear surrogate model around the perturbed samples, LIME provides token-level importance scores that show which words contribute positively or negatively toward the predicted category. This enables the system to highlight key terms that influenced the classifier's reasoning, allowing users to see transparent justifications behind every prediction.

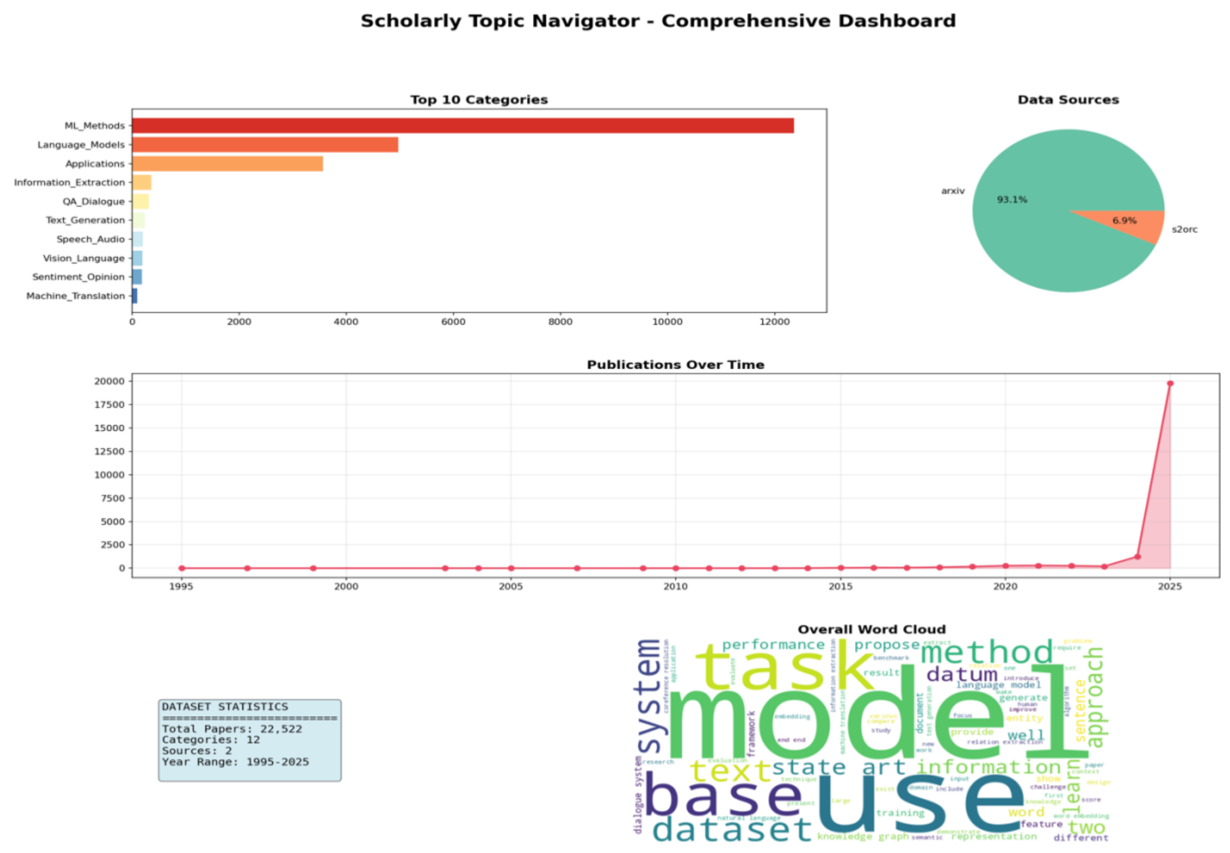
Once the LIME explanations were generated, I integrated them into the Streamlit dashboard alongside the multi-level summaries and metadata. The dashboard serves as the unified interface for interacting with the entire pipeline, wrapping ingestion, preprocessing, embedding-based representation, summarization, classification, and explainability into a single accessible tool. In the UI, users can input queries, browse retrieved documents, examine both extractive and abstractive summaries, and view the LIME-highlighted text showing how the classifier interpreted the content. This real-time visualization bridges the gap between model computation and user understanding, making the system not only functional but also interpretable and user-friendly. The dashboard effectively transforms the complex backend workflow into a streamlined research assistant that presents concise summaries and transparent explanations for each paper.

## 9. Results

The summarization engine produced clear and informative summaries across different abstract types. TextRank reliably extracted factual content, offering a quick high-level understanding, while the transformer-based models produced more fluent and human-like condensations. PEGASUS generated strong one-sentence summaries, whereas BART excelled in multi-sentence coherence. Across all models, the three summary formats—single sentence, medium-length summary, and five-bullet insights—were consistently generated.

The explainability module successfully highlighted influential words for classifier predictions. LIME produced intuitive token-level color gradients showing which words increased or decreased confidence for a topic. SHAP visualizations confirmed these patterns by presenting global contribution trends. Together, these explainability tools made the system's reasoning transparent and interpretable.

The digest pipeline produced complete, structured outputs integrating summaries, explanations, and metadata, and the Streamlit interface displayed all components clearly. Pipeline runtimes remained within acceptable bounds for an interactive user-facing application.



```
DATASET STATISTICS
========================
Total Papers: 22,522
Categories: 12
Sources: 2
Year Range: 1995-2025
```

*Figure 5. Streamlit dashboard integrating summarization, topic prediction, metadata, and LIME explanations into a unified interface.*

## 10. Summary and Conclusions

Through my contributions to the preprocessing pipeline, summarization engine, explainability tools, pipeline orchestration, and user interface development, I helped build an NLP system capable of generating structured and interpretable research digests. Summarization proved to be particularly effective in reducing complex abstracts into readable and meaningful units of information, while explainability tools ensured that users could understand why the system made particular predictions. The digest pipeline integrated all components seamlessly and supported interactive exploration in the UI.

In the future, the system could be improved by fine-tuning summarization models on domain-specific corpora, adding stronger quantitative evaluation metrics such as ROUGE or BERTScore, and enhancing the UI with clustering, comparison views, or semantic search visualizations. The combination of summarization, explainability, and interactive presentation demonstrates the potential of modern NLP systems to support academic research at scale

## 11. References

1. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." Proceedings of NAACL-HLT 2019.

2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., and Polosukhin, I. (2017). "Attention Is All You Need." Advances in Neural Information Processing Systems.

3. Reimers, N. and Gurevych, I. (2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." Proceedings of EMNLP-IJCNLP 2019.

4. Cohan, A., Feldman, S., Beltagy, I., Downey, D., and Weld, D. S. (2020). "SPECTER: Document-level Representation Learning using Citation-informed Transformers." Proceedings of ACL 2020.

5. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension." Proceedings of ACL 2020.

6. Grootendorst, M. (2022). "BERTopic: Neural topic modeling with a class-based TF-IDF procedure." arXiv preprint arXiv:2203.05794.

7. Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). "Latent Dirichlet Allocation." Journal of Machine Learning Research, 3, 993-1022.

8. Robertson, S. and Zaragoza, H. (2009). "The Probabilistic Relevance Framework: BM25 and Beyond." Foundations and Trends in Information Retrieval, 3(4), 333-389.

9. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). "Distributed Representations of Words and Phrases and their Compositionality." Advances in Neural Information Processing Systems.

10. Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why Should I Trust You?: Explaining the Predictions of Any Classifier." Proceedings of KDD 2016.
11. Johnson, J., Douze, M., and Jegou, H. (2019). "Billion-scale similarity search with GPUs." IEEE Transactions on Big Data.
12. Hugging Face. (2024). "Transformers: State-of-the-Art Natural Language Processing." https://huggingface.co/transformers
13. Sentence-Transformers. (2024). "Sentence Embeddings using Siamese BERT-Networks." https://www.sbert.net/
14. Facebook Research. (2024). "FAISS: A library for efficient similarity search." https://github.com/facebookresearch/faiss
15. spaCy. (2024). "Industrial-Strength Natural Language Processing in Python." https://spacy.io/

16. Gensim. (2024). "Topic modelling for humans." https://radimrehurek.com/gensim/
17. Streamlit. (2024). "The fastest way to build data apps." https://streamlit.io/